Linear Time Computation of the Maximal Linear and Circular Sums of Multiple Independent Insertions into a Sequence

RICARDO C. CORRÊA
Universidade Federal Rural do Rio de Janeiro
Departamento de Ciência da Computação
Av. Governador Roberto Silveira S/N
26020-740 Nova Iguaçu, RJ, Brazil
correa@ufrrj.br

PABLO M. S. FARIA
Universidade Federal do Ceará
Departamento de Computação
Campus do Pici, Bloco 910
60440-554 Fortaleza, CE, Brazil
pmsf@lia.ufc.br

Abstract
The maximal sum of a sequence $A$ of $n$ real numbers is the greatest sum of all elements of any linearly contiguous and possibly empty subsequence of $A$. It can be computed in $O(n)$ time by means of Kadane’s algorithm. Letting $A(x \rightarrow p)$ denote the sequence which results from inserting a real number $x$ between elements $A[p-1]$ and $A[p]$, we show how the maximal sum of $A(x \rightarrow p)$ can be computed in $O(1)$ worst-case time for any given $x$ and $p$, provided that an $O(n)$ time preprocessing step has already been executed on $A$. In particular, this implies that, given $m$ pairs $(x_0, p_0), \ldots, (x_{m-1}, p_{m-1})$, we can compute the maximal sums of sequences $A(x_0 \rightarrow p_0), \ldots, A(x_{m-1} \rightarrow p_{m-1})$ optimally in $O(n+m)$ time, improving on the straightforward and suboptimal strategy of applying Kadane’s algorithm to each sequence $A(x_i \rightarrow p_i)$, which takes a total of $\Theta(n \cdot m)$ time. Our main contribution, however, is to obtain the same time bound for the more complicated problem of computing the greatest sum of all elements of any linearly or circularly contiguous and possibly empty subsequence of $A(x \rightarrow p)$. Our algorithms are easy to implement in practice, and they were motivated by and find application in a buffer minimization problem on wireless mesh networks.

1 Introduction
Let a sequence of $n$ elements be denoted by $A = \langle A[0], \ldots, A[n-1] \rangle$ and its size by $|A| = n$. The aim in this paper is to provide efficient algorithms to answer certain insertion-related queries on a sequence of numbers. For a given sequence $A$ of $n$ real numbers, a query takes as arguments a real number $x$ and an index $p \in \{0, \ldots, n\}$, and returns the “score” of sequence $A(x \rightarrow p)$, the latter being the sequence which results from inserting $x$ between elements $A[p-1]$ and $A[p]$. By the “score” of a sequence $B$ of real numbers we mean the greatest sum of all elements of any contiguous, possibly empty subsequence $S$ of $B$. The focus in this paper is on independent queries, that is, given a fixed sequence $A$, we want to answer a number of unrelated queries on the same sequence $A$.

1.1 Definitions and Results
For $n \in \mathbb{N}$, $n > 0$, we use the additive group modulo $n$ to handle the indices of elements of sequences of size $n$. In this sense, $[i : j]$, where $i, j \in \mathbb{Z}$, denotes the range of indices given by the mapping of $\langle i, i + 1, \ldots, j \rangle$ to $\langle i \mod n, (i+1) \mod n, \ldots, j \mod n \rangle$. We write $(i : j)$ for the mapping of $\langle i, i + 1, \ldots, j \rangle \setminus \{i\}$, which corresponds to the subsequence obtained by removing the first occurrence of $i \mod n$ from $[i : j]$. Similarly,
[i : j] stands for the sequence obtained by removing the last occurrence of j mod n from [i : j]. We also write (i : j) = (i : j) ∩ [i : j]. The circular shift of (0, . . . , n − 1) starting at i is denoted by [i : i − 1]. If i = j, then ([i : j], [i : j]), and ([i : j]) are the empty range. In the general case, A[i : j], A(i : j), or A[i : j] are sequences constituted by all elements of A indexed by indices in [i : j], (i : j), or [i : j], respectively. These are subsequences of a sequence, if the corresponding range does not contain any repeated element. If i = j, then A(i : j) = A(i : j) = A(i : j) = ∅ are the empty sequence.

The concatenation of sequences A and B of sizes respectively n and m is denoted by AB = ⟨A[0], . . . , A[n − 1]⟩ ⟨B[0], . . . , B[m − 1]⟩ = ⟨A[0], . . . , A[n − 1], B[0], . . . , B[m − 1]⟩. The sequence A[0 : p]⟨x⟩ A[p : n] which results from the insertion of an element x into position p ∈ [0 : n] of A is denoted by A(x−p). We may also use the abbreviation A[p] when element x is clear by the context. If A is a sequence of real numbers, then its sum is sum(A) = n−1 i=0 A[i], which equals zero if A = ∅. Moreover, the maximal linear sum of a sequence is denoted by MLS(A), which is the greatest sum of a subsequence A[i : j], with 0 ≤ i ≤ j ≤ n, of A. Note that MLS(A) ≥ 0 due to the empty subsequence. An O(n) algorithm to compute MLS(AB) can be obtained by simply applying the Kadane’s linear time algorithm (described in Section 2) to AB. Nevertheless, the extension of such an algorithm to solve the MLSII problem takes Ω(nm) time. Denoted by MCS(A), the maximal circular sum of A is defined as the greatest sum of a subsequence A[i : j], with 0 ≤ i ≤ j ≤ n, of A. Note that MCS(A) ≥ MLS(A) since only subsequences A[i : j] with i ≤ j are considered in MLS(A).

Let a query be a pair (x, p), x ∈ R, and p ∈ [0 : n]. We consider in this paper two optimization problems defined on a sequence A of n real numbers and m queries (x0, p0), . . . , (xm−1, pm−1). The MAXIMAL LINEAR SUMS OF INDEPENDENT INSERTIONS (MLSII) is defined as the problem of determining the maximal linear sums of the resulting sequences A(x0−p0), . . . , A(xm−1−pm−1). Similarly, the MAXIMAL CIRCULAR SUMS OF INDEPENDENT INSERTIONS (MCSII) problem asks for the maximal circular sums of the resulting sequences. In this paper, we give an O(n + m) time algorithm to solve the MLSII problem that generalizes the algorithm given in [1] in order to handle any number m of queries in a sequence of n numbers, each query concerning an arbitrary insertion position p in the sequence. Our previous algorithm can only handle the fixed number of n queries in the sequence, one for each insertion position p ∈ [0 : n]. Our second contribution in this paper is an O(n + m) time algorithm to solve the MCSII problem. This second algorithm turned out not to be a straightforward extension of the first one, having demanded several scattered technical observations. In special, we show that it is related to the problem of finding the minimal sum of a subsequence in some cases.

Our algorithms to solve the MLSII and MCSII problems are divided in two phases. In the first phase, a preprocessing step is performed taking as argument the sequence A and computing the summary of A, which consists of several arrays storing special aggregate information about specific families of subsequences of A. The second phase is constituted by m instantiations of a query-answering step that takes as arguments a query (x, p), i ∈ [0 : m], and the summary produced by the preprocessing step to compute MLS(A(x−p)), in the linear case, or MCS(A(x−p)), in the circular case. The O(n + m) time complexity to solve the MLSII and MCSII problems stems from the fact that the preprocessing step takes O(n) time and the query-answering one takes O(1) time.

1.2 Motivation and Applications

The algorithms proposed in this paper were motivated by a buffer minimization problem in wireless mesh networks, as follows. In a radio network, interference between nearby transmissions prevents simultaneous communication between pairs of nodes which are sufficiently close to each other. One way to circumvent this problem is to use a time division multiple access (TDMA) communication protocol. In such a protocol, the communication in the network proceeds by the successive repetition of a sequence of transmission rounds, in each of which only noninterfering transmissions are allowed to take place. Such protocols can also be used in the particular case of a wireless mesh network, where each node not only communicates data relevant to itself, but also forwards packets sent by other nodes, thus enabling communication between distant parts of the network [2, 3]. In this case, each node stores in a buffer the packets that it must still forward, and optimizing buffer usage in such networks is a current research topic [4, 5].

In order to analyze the use of buffer space in a given set of m nodes of a network, we represent a sequence
of \( n \) transmission rounds by an \( m \times n \) matrix \( R \), defined as follows: \( R[i,j] = +1 \) if node \( i \) receives a packet at round \( j \); \( R[i,j] = -1 \) if at round \( j \) node \( i \) forwards a packet; and \( R[i,j] = 0 \) otherwise. The positive value of \( R[i,j] \) stands for the additional memory space required by node \( i \) to store the packet received at round \( j \), whereas the negative value accounts for the memory space released by node \( i \) when a packet is sent. Since the same sequence of rounds is successively repeated in the network, it can be verified that the greatest number of packets that node \( i \) will ever need to store simultaneously corresponds to the maximal circular sum of row \( i \) of \( R \). The use of buffer for the whole set of \( m \) nodes is then given by \( \text{score}(R) = \sum_{i=0}^{m-1} \text{MCS}(R_i) \), where, for \( i \in [0 : m] \), \( R_i \) denotes the sequence corresponding to the \( R \)'s row of index \( i \). This leads us to the following \( \text{NP} \)-Hard problem: given a matrix \( R \) representing a sequence of transmission rounds, find a permutation \( R' \) of the columns of \( R \) which minimizes \( \text{score}(R') \). The algorithms introduced in the present paper provide a valuable tool for the development of heuristics to this problem.

The concept of maximal sum subsequence and its generalizations for two dimensions are currently known to have several other applications in practice, for example in Pattern Recognition [6, 7], Data Mining [8], Bioinformatics [9, 10, 11], Health and Environmental Science [12], Medicine [13], and Strategic Planning [14]; consequently, it is plausible that other applications of our algorithms be found in the future.

### 1.3 Related Work

We are not aware of any previous attempt to solve the main problem tackled in this paper – i.e. computing \( \text{MCS}(A^{(x \to p)}) \) and \( \text{MCS}(A^{(x \to p)}) \) for any given \( x \) and \( p \), for a fixed sequence \( A \). However, there is a large number of works related to subsequence sums. The basic problem of finding a maximal sum subsequence of a sequence \( A \) of \( n \) numbers was given an optimal and very simple solution by Joseph B. Kadane around 1977. The algorithm, which takes \( O(n) \) time, was discussed and popularized by Gries [15] and Bentley [6]. This one-dimensional problem can be generalized for any number \( d \) of dimensions. The two-dimensional case consists in finding a maximal sum submatrix of a given \( m \times n \) matrix of numbers, and it can be solved in \( O(m^2 \cdot n) \) time, with \( m \leq n \) [16]. Asymptotically slightly faster algorithms do exist but are reported not to perform well in practice except for very large inputs [7, 12]. More recently, the two-dimensional case has also been explored in the direction of convex but not necessarily rectangular shapes [13, 17]. The problem of computing \( \text{MCS}(A[i:j]) \), for any given \( i \) and \( j \), \( 0 \leq i \leq j \leq n \), and a sequence \( A \), has been tackled in [18], where a two-phase algorithm consisting of a linear preprocessing time plus constant time per query is presented.

Another direction of generalization of the original problem which has been explored is that of finding multiple maximal sum subsequences instead of just one. In the ALL MAXIMAL SCORING SUBSEQUENCES problem, one must find a set of all successive and nonintersecting maximal sum subsequences of a given sequence \( A \) of \( n \) numbers; this problem can be solved in \( O(n) \) sequential time [9], \( O(\log n) \) parallel time in the EREW PRAM model [19] and \( O(|A|/p) \) parallel time with \( p \) processors in the BSP/CGM model [20]. A different problem, concerning the maximization of the sum of any set of \( k \) nonintersecting subsequences, is considered in [11]. In the \( k \) MAXIMAL SUMS problem, on the other hand, one must find a list of the \( k \) possibly intersecting maximal sum subsequences of a given sequence of \( n \) numbers, which can be done in optimal \( O(n + k) \) time and \( O(k) \) space [21]. In the related SUM SELECTION problem, one must find the \( k \)-th largest sum of a subsequence of a given sequence \( A \) of \( n \) numbers, which can be done in optimal \( O(n \cdot \max\{1, \log(k/n)\}) \) time [22].

With motivations from Bioinformatics, a further direction of research concerns problems where one or more measures of subsequences are constrained. For example, algorithms have been devised that, given a sequence of real numbers, compute the greatest sum among all subsequences subject to a length lower bound [23], a length upper bound [10, 24], both length bounds [10] or average bounds [25] (the average of a sequence \( A \) being \( \text{sum}(A)/|A| \)). Optimal algorithms have also been devised for the length constrained versions of the \( k \) MAXIMAL SUMS problem and the SUM SELECTION problem [22], and parallel and correct by construction algorithms have been devised for the one- and two-dimensional cases of the original maximal sum problem [26]. Optimization measures other than subsequence sum have also been investigated. For example, online linear-time algorithms have been devised for the problems of finding the longest or shortest subsequence subject to a sum lower bound (or, equivalently, a sum upper bound) [27]. Optimal algorithms
were also developed for the case of both length and average constraints in the contexts of maximizing or minimizing subsequence length, finding all valid subsequences (with respect to the given constraints), etc [28]. Optimization measures (“score functions”) of the form \( f(\ell, s) \), where \( \ell \) is the length and \( s \) is the sum of a subsequence, were also studied [29, 30].

To the best of our knowledge, the column permutation problem defined in the previous subsection has not yet been considered in the literature. The closest related and already studied problem that we know of is the following variation of it for only one row: given a sequence \( A \) of \( n \) real numbers, find a permutation \( A' \) of \( A \) which minimizes \( \ MLS(A') \). This problem was found to be solvable in \( O(\log n) \) time in the particular case where \( A \) has only two distinct numbers [31]; the same paper also mentions that the case where \( A \) may have arbitrary numbers can be shown to be strongly NP-hard by reduction from the 3-PARTITION problem. Such a reduction has actually been presented recently, together with an \( \ O(n \log n) \) algorithm which has an approximation factor of 2 for the case of arbitrary input numbers and \( 3/2 \) for the case where the input numbers are subject to certain restrictions [32].

Problems about insertion-related operations in a sequence of numbers in connection with the concept of maximal subsequence sum seem to have been considered only in recent papers by the present authors, in which restricted versions of the linear case of the MLSII problem were dealt with. Precisely, in a first paper we considered the problem of, given a sequence \( A \) of \( n \) real numbers and a real number \( x \), finding an index \( p \in [0 : n] \) which minimizes \( \ MLS(A^{(p \rightarrow p)}) \), and we showed that this can be done by means of a relatively simple linear time algorithm [32]. Later we generalized this result by considering the problem of, given sequences \( A \) and \( X \) of \( n \) and \( n + 1 \) real numbers respectively, computing \( \ MLS(A^{(X[p \rightarrow p]}) \) for all \( p \in [0 : n] \), and for this problem we also gave an \( O(n) \) time algorithm [1].

### 1.4 Structure of the Paper

The remaining of this paper is structured as follows. Section 2 introduces our approach as well as some notation for both linear and circular cases. The preprocessing and query-answering algorithms for the linear case are then presented in Sections 3 and 4, respectively. Sections 5 and 6 then present our preprocessing and query-answering algorithms for the circular case, and Section 7 presents our concluding remarks, closing the paper.

## 2 General Approach

Let \( A \) be a sequence of \( n > 0 \) real numbers. If \( i, j \in [0 : n) \) and \( h \in [i : j] \), then we say that \( A[i : h) \) and \( A(h : j] \) are respectively a proper prefix and a proper suffix of \( A[i : j] \). Additionally, \( A[i : h) \) itself and \( A[i : h] \) are prefixes of \( A[i : j] \), as well as \( A(h : j) \) and \( A[h : j] \) are suffixes of \( A[i : j] \). We give next an overview of the general approach we adopt in the rest of the paper for the linear and circular cases. In special, we point out the essential differences between these two cases.

### 2.1 Linear Case

Consider a query \( (x, p) \). In order to compute \( \ MLS(A^p) \) fast, we need to find out quickly how the insertion of \( x \) relates to the sums of the subsequences of \( A \). This means that, in a certain sense, we need to previously know the “structure” of \( A \). A simple and useful aspect of such a structure of \( A \) is as follows. Let a linear interval partition of \( A \) be a division into nonempty subsequences \( I_0, \ldots, I_{\ell-1} \), with \( I_k = A[\alpha_k : \beta_k] \), \( \alpha_k \leq \beta_k \), for all \( k \in [0 : \ell) \), such that \( A = I_0 \ldots I_{\ell-1} \) and the following holds for every interval \( I_k \) and \( j \in [\alpha_k : \beta_k] \):

1. if \( k < \ell - 1 \), then \( I_k \) is maximal with respect to the following property: every proper prefix has nonnegative sum and every nonempty proper suffix has negative sum, i.e.
   
   \[
   \text{sum}(A[\alpha_k : j]) \geq 0 \text{ and } (j \neq \alpha_k) \Rightarrow \text{sum}(A[j : \beta_k]) < 0;
   \]
2. if \( k = \ell - 1 \), then \( I_k \) is maximal with respect to the following property: every proper prefix has nonnegative sum, \( i.e. \)
\[
\sum(A[\alpha_k : j]) \geq 0.
\]

An example of linear interval partition is given in Fig. 1(a).

Lemma 1 Sequence \( A \) can be uniquely linearly partitioned into intervals.

Proof. The statement trivially holds for \( A[0 : 0] \). By induction, assume that the sequence \( A[0 : n'] \), \( n > n' > 0 \), is uniquely linearly partitioned into intervals \( I_0, \ldots, I_{\ell-1} \). It follows that intervals \( I_0, \ldots, I_{\ell-2} \) are the unique manner to satisfy condition 1 for the subsequence \( I_0 \ldots I_{\ell-2} \). If \( \sum(I_{\ell-1}) \geq 0 \), then \( I_{\ell-1}[A[n']] \) would be the unique way to also satisfy 2 for \( A[0 : n'] \). Otherwise, conditions 1 and 2 are uniquely satisfied by \( I_{\ell-1} \) and \( A[n'] \), respectively. Thus, either \( I_0, \ldots, I_{\ell-2}, I_{\ell-1}[A[n']] \) or \( I_0, \ldots, I_{\ell-2}, I_{\ell-1}, A[n'] \) forms the unique linear interval partition of \( A[0 : n'] \).

□

The definition above is equivalent to that in [32], as shown below. This characterization is more convenient to devise an algorithm to compute the linear interval partition, as we discuss later in this section.

Lemma 2 A partition of \( A \) into subsequences \( I_0, \ldots, I_{\ell-1} \) is the linear interval partition of \( A \), for some \( \ell \geq 1 \), if and only if, for every \( I_k, j \in [\alpha_k : \beta_k] \Rightarrow \sum(A[\alpha_k : j]) \geq 0 \) and \( (k < \ell - 1) \Rightarrow \sum(A[\alpha_k : \beta_k]) < 0 \).

Proof. Let \( k \in [0 : \ell - 1] \). First, assume that \( I_0, \ldots, I_{\ell-1} \) is the linear interval partition of \( A \). To show that \( k < \ell - 1 \) yields \( \sum(A[\alpha_k : \beta_k]) < 0 \), consider \( \ell > 1 \) (otherwise there is nothing to prove). If \( k = \ell - 2 \), then the claim holds since \( I_{\ell-1} \) is maximal with respect to \( \sum(A[\alpha_{\ell-1} : \beta_{\ell-1}]) \geq 0 \). Hence, \( k < \ell - 2 \) and \( \sum(A[\alpha_{k+1} : \beta_{k+1}]) < 0 \). In this situation, \( \sum(A[\alpha_k : \beta_k]) \geq 0 \) would lead \( I_kI_{k+1} \) to contradict the maximality of the proper prefixes and proper suffixes of \( I_k \) and \( I_{k+1} \).

Conversely, assume interval \( I_k \), \( k < \ell - 1 \), and index \( j \in [\alpha_k : \beta_k] \) such that \( \sum(A[\alpha_k : j]) \geq 0 \) and \( \sum(A[\alpha_k : \beta_k]) < 0 \) hold. Since \( A[\alpha_k : j] \) is a proper prefix of \( I_k \) and \( \sum(I_k) = \sum(A[\alpha_k : j]) + \sum(A[j : \beta_k]) \), we get \( \sum(A[j : \beta_k]) < 0 \). In addition, \( I_k \) is maximal with respect to this property because (i) either \( k = 0 \) or every suffix of \( I_{\ell-1} \) is negative, and (ii) \( \sum(I_k) < 0 \). Item (i) implies that \( I_k \) cannot begin before \( \alpha_k \), whereas (ii) implies that it cannot end after \( \beta_k \).

□

An additional way to summarize relevant aspects of the structure of \( A \) is by means of aggregate information on sets of subsequences. In this sense, let the maximal linear suffix sum of a subsequence \( A[0 : j] \), for \( j \in [0 : n] \), be defined as \( MLS_A(j) = \max\{\sum(A[i : j]) : 0 < i \leq j \} \). Note that the empty subsequence is not considered in this definition, as illustrated again in Fig. 1(a). Even though the number of subsequences that have to be considered in the definition of \( MLS(A) \) is quadratic in \( n \), Kadane’s algorithm computes the maximal sum of \( A \) in linear time thanks to the aggregate information in \( MLS_A(i) \) for all \( i \in [0 : n] \). This is accomplished by a single left-to-right sweep of \( A \) by observing that \( MLS_A(0) = A[0] \) and that
\[
MLS_A(i + 1) = \max\{A[i + 1], A[i + 1] + MLS_A(i)\} = \sum(A[\alpha_k : i + 1]),
\]
where \( k \) is such that \( i + 1 \in [\alpha_k : \beta_k] \). Analogously, let the maximal linear prefix sum of a subsequence \( A[i : n] \), for \( i \in [0 : n] \), be defined as \( MLP_A(j) = \max\{\sum(A[i : j]) : i \leq j < n\} \).

The general approach to handle the insertion of \( x \) in order to compute \( MLS(A^p) \) fast is as follows. First, we define a summary consisting of a description of the structure of \( A \). For this purpose, let the maximal linear suffix sum for an \( S \subseteq \{0, \ldots, n - 1\} \) be given by \( MLS_A(S) = \max\{MLS_A(i) : i \in S\} \), which leads to \( MLS_A(A) = \max\{0, MLS_A(A)\} \) and \( MLS_A(A^p) = \max\{0, MLS_A(A^p)\} \). The summary is constituted by a set of arrays storing information, like values and indices, about maximal linear prefix or suffix sums of selected subsets \( S \subseteq [0 : n] \), computationally computable in \( O(n) \) time during the preprocessing step. Then, the query-answering step is performed in constant time by taking a partition of \( [0 : n] \) into a constant number of suitable subsets \( S \subseteq [0, \ldots, n] \), depending on the characteristics of the query \((x, p)\), such that each \( MLS_A(S) \) can be computed in constant time using a constant amount of information from the summary.
For the sake of illustration, let us consider the insertion corresponding to query \((x, p)\). An example of such a situation is depicted in Fig. 1(b). Note the use of index \(k\) in the figure: henceforth in this paper, \(I_k\) denotes the interval such that \(p \in [\alpha_k : \beta_k]\), if \(p < n\), or such that \(k = \ell - 1\), if \(p = n\). It is also shown in the figure that the maximal linear suffix sums can change in the resulting sequence, producing a linear interval partition of \(A^p\) distinct from that of \(A\). The partition of \([0 : n]\) used in the query-answering step consists in \([0 : p] \cup [p : p] \cup (p : \beta_k + 1) \cup (\beta_k + 1 : n]\), leading to

\[
\mathcal{MC}(A^p) = \max\{0, \mathcal{MC}(A^p([0 : p])), \mathcal{MC}(A^p(p)), \mathcal{MC}(A^p((p : \beta_k + 1])), \mathcal{MC}(A^p((\beta_k + 1 : n]))\}. \tag{1}
\]

We postpone the analyses of the linear case until the next two sections.

\[
\begin{array}{cccccccccccc}
2 & -5 & 4 & -21 & 22 & 3 & -5 & 4 & 5 & -1 & -3 & 5 & 16 & -2 & 8 & 18 \\
\hline
A
\end{array}
\]

(a) A sequence, its linear interval partition, and maximal linear suffix sums.

\[
\begin{array}{cccccccccccc}
\hline
A^p
\end{array}
\]

(b) Sequence resulting from the insertion of \(x = 12\) at position \(p = 8\).

Figure 1: Insertion in the linear case. Elements have equal colors iff they belong to the same interval in \(A\). Above each element is its maximal linear suffix sum in the corresponding sequence; the greatest such values are written in bold for each sequence.

### 2.2 Circular Case

A possible approach to compute the maximal circular sum of a sequence is to handle the subsequences of \(A[0 : 2n]\) with size at most \(|A|\). Indeed, since the length-constrained maximal sum problem can be solved in linear time by Lin et al.’s algorithm \([10]\) or by Mu’s algorithm \([24]\) (see Subsection 1.3), then these algorithms can also be used to compute \(\mathcal{MC}(A)\) in \(\Theta(|A|)\) time. However, in the context of the MCSII problem, computing each value \(\mathcal{MC}(A^p([x : p]))\) to answer the query \((x_i, p_i), i \in [0 : n]\) by means of either Lin et al.’s or Mu’s algorithm takes \(\Theta(2^{(n + 1)} = \Theta(n)\) time per query, and therefore \(\Theta(n^2m)\) time in total. A faster algorithm for the MCSII problem is obtained with an approach similar to the one adopted to the linear case. This approach, introduced next, is based upon an appropriate redefinition of the interval partition. The entire summary is described in Section 5 and the query-answering algorithm is left to Section 6.

Let a circular interval partition of \(A\) be a division of \(A\) into nonempty subsequences \(I_0, \ldots, I_{\ell-1}\), with \(I_k = A[\alpha_k : \beta_k]\) for all \(k \in [0 : \ell]\), such that \(A\) is a suitable circular shift of \(I_0 \ldots I_{\ell-1}\) and the following holds for every interval \(I_k\) and \(j \in [\alpha_k : \beta_k]\):

1. \(I_k\) is maximal with respect to the following property: every proper prefix has nonnegative sum and every nonempty proper suffix has negative sum, i.e.

   \[
   \text{sum}(A[\alpha_k : j]) \geq 0 \quad \text{and} \quad (j \neq \alpha_k) \Rightarrow \text{sum}(A[j : \beta_k]) < 0.
   \]

The circular interval partition of the example in Fig. 1 is given in Fig. 2(a). A novelty in the circular case is that the interval \(I_{\ell-1}\) is not a special case with respect to \(I_0, \ldots, I_{\ell-2}\) and, furthermore, it may happen that the condition established in Lemma 2 does not hold. This motivates the definition of \(A\) as regular if there exists a circular shift \(A'\) of \(A\) such that its linear interval partition satisfies \(\text{sum}(A'[\alpha_k : \beta_k]) < 0\), for every \(k \in [0 : \ell]\). If the sequence \(A\) is not regular, then it is irregular. An example of irregular sequence is shown
in Fig. 3. Interestingly, in order to compute length-constrained maximal sum subsequences, Lin et al. used a different (and, in a sense, opposite) partition scheme, which, roughly speaking, divides the sequence into minimal segments of positive sum [10].

**Lemma 3** A is irregular iff sum \(A[\alpha_k : \beta_k]) \geq 0\), for every circular interval partition \(I_0, \ldots, I_{\ell - 1}\) of \(A\) and for every \(k \in [0 : \ell]\).

**Proof.** Assume that \(A\) is irregular. For the sake of contradiction, let \(I_0, \ldots, I_{\ell - 1}\) be a circular interval partition of \(A\) and \(I_k\) and \(I_{k+1}\), with sum modulo \(\ell\), be two consecutive intervals such that \(sum(A[\alpha_k : \beta_{k+1}]) \geq 0\) and \(sum(A[\alpha_{k+1} : \beta_k]) < 0\). For all \(j \in [\alpha_k : \beta_{k+1}]\), it holds that \((j \neq \beta_{k+1}) \Rightarrow sum(A[\alpha_k : j]) \geq 0\) and \((j \neq \alpha_k) \Rightarrow sum(A[j : \beta_{k+1}]) < 0\), which makes \(I_k\) and \(I_{k+1}\) to contradict the maximility of \(I_k\) and \(I_{k+1}\).

Conversely, if \(A\) is regular, then the linear interval partition of a suitable circular shift of \(A\) gives a circular interval partition \(I_0, \ldots, I_{\ell - 1}\) of \(A\) with \(sum(A[\alpha_k : \beta_k]) < 0\), for every \(k \in [0 : \ell]\).

The complement of \(A\), denoted by \(\bar{A}\), is the sequence \((-A[n-1], \ldots, -A[0])\). Clearly, \(\bar{A} = A\). The following lemma indicates that solving the MCSIII problem is equivalent to solve the analogous problem defined with respect to the minimum sum subsequence.

**Lemma 4** A is regular iff \(\bar{A}\) is irregular.

**Proof.** If \(A\) is regular, let \(I_0, \ldots, I_{\ell - 1}\) be the linear interval partition of \(A\) such that \(sum(A[\alpha_k : \beta_k]) < 0\), for every \(k \in [0 : \ell]\). It follows that \(I_{\ell-1}, \ldots, I_0\) is a circular interval partition of \(\bar{A}\) such that \(sum(\bar{A}[\alpha_k : \beta_k]) \geq 0\), for every \(k \in [0 : \ell]\). Thus, \(\bar{A}\) is irregular by Lemma 3.

Conversely, if \(\bar{A}\) is regular and the same argument applies since \(\bar{A} = A\).

**Lemma 5** Sequence \(A\) can be uniquely circularly partitioned into intervals.

**Proof.** If \(A\) is regular, then its unique, by Lemma 1, linear interval partition is also the unique circular interval partition. Otherwise, \(\bar{A}\) is regular by Lemma 4. Hence, by Lemma 1, the unique linear interval partition of \(\bar{A}\) is the unique circular interval partition of \(\bar{A}\), whose complement gives a circular partition of \(A\).

We close this section with an additional remark about the “structure” of irregular sequences. Let the maximal circular suffix sum at \(j \in [0 : n]\), illustrated in Fig. 2(a), be defined as \(MCS_A(j) = \max\{MCS_A(i) \mid i \in [j+1 : n]\}\). As depicted in Fig. 2, the insertion of \(x\) may also affect the maximal circular suffix sums at elements \(A[0], \ldots, A[p - 1]\). Although each term \(MCS_A(i)\) takes all elements of \(A\) into account, these elements are combined differently for each index \(i\). Thus, computing \(MCS_A(0), \ldots, MCS_A(n - 1)\) within the same time bound is more complicated than computing \(MCS_A(0), \ldots, MCS_A(n - 1)\). The maximal circular suffix sum for an \(S \subseteq \{0, \ldots, n - 1\}\) is given by \(MCS_A(S) = \max\{MCS_A(i) \mid i \in S\}\). The subsets \(S\) used to compute \(MCS_A(A)\) are specified in Section 6. According to the following lemma, the maximal circular sum at an element \(A[j]\) of an interval \(I_i\) of \(A\) depends on the elements of all other intervals when \(A\) is irregular (as in Fig. 3). This contrasts with the linear case, as well as with the circular case when \(A\) is regular, where the maximal sum at \(A[j]\) equals \(sum(A[\alpha_1 : j])\) and thus only depends on elements of \(I_i\).

**Lemma 6** If \(A\) is irregular, \(k \in [0 : \ell - 1]\), and \(i \in [\alpha_k : \beta_k]\), then \(MCS_A(i) = sum(A[\beta_k : i])\).

**Proof.** Given \(k\) and \(i\), let \(i' \in [\alpha_k : \beta_k]\) be such that \(A[i' : i]\) is the maximal subsequence of \(A\) having \(MCS_A(i) = sum(A[i' : i])\). We show that \(i' = \beta_k + 1\). For this purpose, we first consider the case when \(i \neq \beta_k\). The subsequence \(A[i + 1 : \beta_k]\) is a proper suffix of \(I_k\), which leads to \(sum(A[i+1 : \beta_k]) < 0\). For this reason, \(A[i+1 : \beta_k]\) cannot be a prefix of \(A[i' : i]\). We conclude that \(i' \not\in [i+1 : \beta_k]\) in this case.

On the other hand, we claim that \(\beta_k + 1 \in [i' : i]\). By the sake of contradiction, assume the contrary. Considering that \(\beta_k + 1 = \alpha_{k+1}\), with sum modulo \(\ell\), we conclude that \(A[\beta_k + 1 : i']\) is a concatenation of intervals with a prefix of \(I_{k+1}[i']\). By Lemma 3, \(sum(A[\beta_k + 1 : i']) \geq 0\). It turns out that \(sum(A[\beta_k + 1 : i']) \geq sum(A[i' : i])\) contradicts the definition of \(i'\).

\(\square\)
Figure 2: Insertion in the circular case. Elements have equal colors iff they belong to the same interval in $A$. Outside each element is its maximal circular sum in the sequence; the greatest such values are written in bold for each sequence.

3 Preprocessing Algorithm for the Linear Case

We give a complete description of the summary for the linear case in Table 1. A function that appears in this table is the maximal gap given by

$$MG_A([i : j]) = \max \{ \text{sum}(A[i : j_1]) - \text{sum}(A[i : j_2]) : i \leq j_2 \leq j_1 \leq j \}.$$ 

For the sake of illustration, a partial description of the summary for the example in Fig. 1(b) is given by

- $A[i]$: 2 -7 4 -25 22 -19 -8 4 1 -6 -3 5 11 -18 8 10
- $MLS[i]$: 2 -5 4 -21 22 3 -5 4 5 -1 -3 5 16 -2 8 18
- $PRVMLS[i]$: 0 2 2 4 4 22 22 22 22 22 22 22 22 22 22 22
- $K[i]$: 0 0 1 1 2 2 2 3 3 3 4 5 5 5 6 6 6
- $SFMLS[i]$: 2 -5 4 -21 22 3 -5 5 5 -1 -3 16 16 -2 18 18
- $MG[i]$: 0 0 0 0 0 0 0 0 1 0 0 0 11 0 0 10 0

Of all this data, Kadane’s algorithm can be easily made to compute arrays $K$, $MLS$, $PRVMLS$, $IMPF_S$, and $IS$ still based on Lemma 2. Moreover, arrays $SFMLS$, $MG$, and $INXTMS$ can be easily computed by means of a right-to-left traversal of array $MLS$ by taking the interval partition of $A$ into account. In special, note that the computation of $MG$ uses $\text{sum}(A[\alpha_k : j_1]) = MLS_A(f_1)$ and $\text{sum}(A[\alpha_k : j_2]) = MLS_A(j_2)$. With regard to array $IRS$ (which is defined only for $i < \ell - 1$), note that, for any $i \in [0 : \ell - 1)$, if $IRS[i] = sum(A(\beta_1 : j))$ and $j \in [\alpha_{i'} : \beta_i')$, with $i' > i$, then, by the definitions of arrays $IRS$ and $IMPF_S$,

$$IRS[i] = (\sum_{x=\ell}^{i'-1} \text{sum}(I_x)) + IMPFS'[i'].$$ 

Thus, for all $i \in [0 : \ell - 1)$,

$$IRS[i] = \max_{i < i' < \ell} \left( \sum_{x=i+1}^{i'-1} \text{sum}(I_x) \right) + IMPFS'[i']$$

$$= \begin{cases} 
IMPFS[i + 1], & \text{if } i = \ell - 2; \\
\max \{IMPFS[i + 1], \text{sum}(I_{i+1}) + IRS[i + 1]\}, & \text{if } i < \ell - 2.
\end{cases}$$
Figure 3: An irregular sequence and the corresponding arrays OMCS and MCS, where $MCS[i] = MCS_A(i)$ for all $i \in [0 : n)$. The numbers above sequence $A$ are the indices of its elements.

We can therefore compute array IRS backwards in $O(\ell) = O(n)$ time by means of arrays IMPFS and IS.

**Theorem 1** The preprocessing algorithm computes the summary in Table 1 in $O(n)$ time for any sequence $A$ of size $n$.

### 4 Query-answering Algorithm for the Linear Case

Given a query $(x, p)$, we show in this section how to compute $\mathcal{MLS}(A^p)$ according to (1) in $O(1)$ time given that the summary described in Section 3 has already been computed. We analyze the cases $x \geq 0$ and $x < 0$ separately, since, as pointed out in [32], in each case the insertion of $x$ has different effects on the maximal sums at the elements of $A$. In both cases, it is clear that $\mathcal{MLS}_{A^p}(\lfloor p : \beta_k \rfloor) = x + SFMLS[p]$.

Thus, we discuss below the remaining ranges indicated in (1) assuming that $p < n$. For all $i \in [p : n)$, let $i^o$ denote the index of element $A[i]$ into sequence $A^p$, that is, $i^o = i + 1$.

#### 4.1 Handling a Nonnegative $x$

It is easy to see that the linear sums of the subsequences of $A^p$ ending at elements $A[p], \ldots, A[\beta_k]$ all increase by $x$. Thus,

$$\mathcal{MLS}_{A^p}(p : \beta_k + 1) = x + SFMLS[p].$$

Handling the subsequences of $A^p$ ending at elements $A[\beta_k + 1], \ldots, A[n - 1]$ to compute $\mathcal{MLS}_{A^p}(\lfloor \beta_k + 1 : n \rfloor)$ when $x \geq 0$ is more complicated. In this regard, an important observation is that, although the linear sums at the elements of $A$ to the right of $I_k$ may increase by different amounts with the insertion of $x$ into $A$, they do so in a regular manner. More precisely, the maximal linear suffix sums at interval $I_{k+1}$ increase by $\max\{0, x + \text{sum}(I_k)\}$, and, for all $i \in \lfloor k + 1 : \ell - 1 \rfloor$, if the linear suffix sums at interval $I_i$ increase by some value $y$, then the linear suffix sums at interval $I_{i+1}$ increase by $\max\{0, y + \text{sum}(I_i)\}$. By exploiting this regularity, we get the following result.
Table 1: Summary for the linear case. \( \ell \) denotes the size of the interval partition of \( A \). Arrays indexed with \( p \) (instead of \( i \)) make (direct or indirect) use of \( k = K[p] \) in their definitions. Some arrays are not defined for every index.

<table>
<thead>
<tr>
<th>Arrays with indices in the range ([0 : n])</th>
</tr>
</thead>
<tbody>
<tr>
<td>( K[i] ) = ( \begin{cases} \ell - 1, &amp; \text{if } i = n \ j \in [0 : \ell) &amp; \text{such that } i \in [\alpha_j : \beta_j], &amp; \text{if } i &lt; n \end{cases} )</td>
</tr>
<tr>
<td>( \text{PRVMLS}[i] = \mathcal{MLS}_A([0 : i]) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arrays with indices in the range ([0 : n])</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{MLS}[i] = \mathcal{MLS}_A(i) )</td>
</tr>
<tr>
<td>( \text{SFMLS}[p] = \mathcal{MLS}_A([p : p]) )</td>
</tr>
<tr>
<td>( \text{MG}[p] = \mathcal{G}_A([p : p]) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arrays with indices in the range ([0 : \ell])</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{IMPFS}[i] = \mathcal{MLP}_I(0) )</td>
</tr>
<tr>
<td>= ( \text{max}{\text{sum}(A[0 : \beta_i]) : j \in [\alpha_i : \beta_i]} )</td>
</tr>
<tr>
<td>( \text{IS}[i] = \text{sum}(I_i) )</td>
</tr>
<tr>
<td>( \text{INXTMS}[i] = \mathcal{MLS}_A([\beta_i : n]) )</td>
</tr>
<tr>
<td>( \text{IRS}[i] = \mathcal{MLP}_A(\beta_i + 1) )</td>
</tr>
</tbody>
</table>

Lemma 7 If \( x \geq 0 \) and \( k < \ell - 1 \), then
\[
\mathcal{MLS}_A^\prime((\beta_k + 1 : n]) = \max\{\text{INXTMS}[k], x + \text{IS}[k] + \text{IRS}[k]\}.
\]

Proof. Let us first show that, for all \( i \in (\beta_k : n)\), \( \mathcal{MLS}_A^\prime(i^\circ) = \max\{\mathcal{MLS}_A(i), \text{sum}(A[\alpha_k : j^\circ])\} \). Indeed, if \( \mathcal{MLS}_A^\prime(i^\circ) \neq \mathcal{MLS}_A(i) \), then there exists \( j \in [0 : p) \) such that \( \text{sum}(A[p : j^\circ]) = \mathcal{MLS}_A^\prime(i^\circ) \). Note that it cannot be the case that \( j < \alpha_k \): since the sum of every suffix of every interval of \( A \) different from \( I_{\ell - 1} \) is negative, \( j < \alpha_k \) implies \( \text{sum}(A[\alpha_k : j^\circ]) > \text{sum}(A[p : j^\circ]) \), a contradiction with our choice of \( j \). Thus, \( j \in [\alpha_k : p] \). Note also that it cannot be the case that \( \text{sum}(A[p : j]) > 0 \), since this again implies that \( \text{sum}(A[\alpha_k : j^\circ]) > \text{sum}(A[p : j^\circ]) \). Thus, \( \text{sum}(A[p : j]) \leq 0 \). Now, either \( \alpha_k = j \), which implies that \( \text{sum}(A[p : j]) = \text{sum}(A[\alpha_k : j]) \), or \( \alpha_k < j \), which, since \( A[p : j] \) is a proper prefix of \( I_k \), implies that \( \text{sum}(A[p : j]) = 0 \) and again that \( \text{sum}(A[p : j]) = \text{sum}(A[\alpha_k : j]) \), as desired.

The previous paragraph implies that the statement of the lemma holds when we substitute the inequality sign “\( \leq \)” for the equality sign “\( = \)” in it. To see that the converse inequality (“\( \geq \)” also holds, first note that there are \( i, i' \in (\beta_k : n) \) such that \( \mathcal{MLS}_A(i) = \text{INXTMS}[k] \) and \( \text{sum}(A[\beta_k : i']) = \text{IRS}[k] \). Now, since \( x \geq 0 \), then \( \mathcal{MLS}_A^\prime(i') \geq \mathcal{MLS}_A(i) = \text{INXTMS}[k] \). Moreover, \( \mathcal{MLS}_A^\prime(i') \geq \text{sum}(A[\alpha_k : i']) = x + \text{sum}(I_k) + \text{IRS}[k] \). Therefore, the second inequality also holds, and so does the lemma.

4.2 Handling a Negative \( x \)

Inserting \( x \) into interval \( I_k \) clearly does not change the linear suffix sums at elements to the right of \( I_k \), that is, \( \mathcal{MLS}_A(i^\circ) = \mathcal{MLS}_A(i) \) for all \( i \in [\beta_k + 1 : n] \). Thus, if \( k < \ell - 1 \), then
\[
\mathcal{MLS}_A^\prime((\beta_k + 1 : n]) = \text{INXTMS}[k].
\]

Consequently, what remains to be shown is how to compute \( \mathcal{MLS}_A^\prime((p : \beta_k + 1]) \) in \( O(1) \) time if \( p < n \), which we do in the rest of this section.
The difficulty here is that the linear suffix sums at elements \( A[p] \), \( \ldots, A[\beta_k] \) may decrease by different amounts. We prove this in detail next using the function
\[
\text{dec}(i) = \begin{cases} 
\min\{-x, \text{sum}(A[\alpha_k : i])\}, & \text{if } i = p \\
\min\{\text{dec}(i - 1), \text{sum}(A[\alpha_k : i])\}, & \text{if } i > p.
\end{cases}
\]

Lemma 8 If \( x < 0 \), then, for all \( i \in [p : \beta_k] \),
\[
\text{MLS}_{A^p}(i^\circ) = \text{MLS}_A(i) - \text{dec}(i).
\] (2)

Proof. Firstly, if \( p = \alpha_k \), then, since \( x < 0 \), \( \text{MLS}_{A^p}(i^\circ) = \text{sum}(A^p[\alpha_k : i^\circ]) = \text{MLS}_A(i) \), as desired. Now suppose that \( p > \alpha_k \), which means that \( \text{MLS}_A(p - 1) \geq 0 \). We prove (2) by induction on \( i \). If \( i = p \), then there are two cases. First, if \( \text{dec}(p) + x = 0 \), then \( \text{MLS}_{A^p}(p) = x + \text{MLS}_{A^p}(p - 1) = x + \text{sum}(A[\alpha_k : p]) \geq 0 \), which implies that \( \text{MLS}_{A^p}(i^\circ) = A[i] + \text{MLS}_{A^p}(p) = A[i] + x + \text{MLS}_A(p - 1) \). Thus (2) stems from \( \text{MLS}_A(i) = A[i] + \text{MLS}_A(i - 1) \). Otherwise, \( \text{dec}(i) = \text{sum}(A[\alpha_k : p]) \) implies that \( \text{MLS}_{A^p}(p) = 0 \) and, consequently, (2) holds. Now suppose that (2) holds for \( i < \beta_k \). If \( \text{MLS}_{A^p}(i^\circ) = 0 \), then, by induction hypothesis, \( \text{dec}(i) = \text{sum}(A[\alpha_k : i]) \). Moreover, \( \text{MLS}_{A^p}(i + 1^\circ) = A[i + 1] = \text{MLS}_A(i + 1) - \text{MLS}_A(i) \). Hence, (2) also holds for \( i + 1 \) due to \( \text{dec}(i + 1) = \min(\text{dec}(i), \text{sum}(A[\alpha_k : i])) = \text{sum}(A[\alpha_k : i]) \). The case when \( \text{MLS}_{A^p}(i^\circ) > 0 \) is similar.

The key to compute \( \text{MLS}_{A^p}(p : \beta_k + 1) \) quickly is the following result.

Lemma 9 If \( x < 0 \) and \( p < n \), then
\[
\text{MLS}_{A^p}(p : \beta_k + 1) = \max\{x + \text{MLS}_A([p : \beta_k]), MG[p]\}.
\] (3)

Proof. We say that function \( \text{dec}(i) \) has a discontinuity at \( i > p \) if \( \text{dec}(i) < \text{dec}(i - 1) \). The proof is by induction on the number of discontinuities of \( \text{dec}(i) \) in \([p : \beta_k]\). If there is no such discontinuity, then \( \text{MLS}_{A^p}(i^\circ) = \text{MLS}_A(i) - \text{dec}(p) \), for all \( i \in [p : \beta_k] \), by Lemma 8. Since \( \text{dec}(p) + x = 0 \), if \( \text{sum}(A[\alpha_k : p]) + x \geq 0 \), or \( \text{dec}(p) = \min\{\text{sum}(A[\alpha_k : i]) : i \in [p : \beta_k]\} = \text{MLS}_A([p : \beta_k]) - MG[p] \) otherwise, equality (3) holds. Now assume that the number of discontinuities in \([p : \beta_k]\) is \( d + 1 \) and that (3) holds whenever the number of discontinuities is \( d \geq 0 \). Let \( i \) be the largest index of a discontinuity in \([p : \beta_k]\). By the induction hypothesis, \( \text{MLS}_{A^p}(p : i) = \max\{x + \text{MLS}_A([p : i]), MG_A([p : i])\} \). Since there is a discontinuity at \( i \), we can write \( \text{MLS}_{A^p}(i : \beta_k + 1) = MG_A([i : \beta_k]) \). Equality (3) results from \( \text{MLS}_{A^p}(p : \beta_k + 1) = \text{MLS}_{A^p}(p : i) + \text{MLS}_{A^p}(p : \beta_k + 1) \).

Gathering the results of this section, we get the following.

Theorem 2 Given a summary as defined in Section 3, the query-answering algorithm computes \( \text{MLS}(A^{(x \to p)}) \) in \( O(1) \) worst-case time for any sequence \( A \), \( x \in R \), and \( p \in [0 : n] \).

5 Preprocessing Algorithm for the Circular Case

We describe in this section the preprocessing algorithm for the summary for the circular case depicted in Table 2. What is common between the circular and linear cases is that, whenever two elements \( A[i] \) and \( A[j] \) belong to the same interval \( I \), the maximal (circular) sums at these elements equal the sums of subsequences of \( A \) which begin at a common element \( A[h] \) – which, however, may not be the first element of \( I \). This indicates that, for each element \( A[i] \) of \( A \), it is important to know an “origin” element \( A[h] \) such that \( \text{MCS}_A(i) = \text{sum}(A[h : i]) \). We therefore define, as an element of the summary of the circular case, \( \text{OMCS} \) as the array such that \( \text{OMCS}[i] = i - \max\{j \in [0 : n] : \text{sum}(A[i : j]) = \text{MCS}_A(i)\} \), for all \( i \in [0 : n] \) such that \( \text{MCS}_A(i) \) is given by a nonempty subsequence. Intuitively, \( \text{OMCS}[i] \) is, with respect to \( i \), the circularly leftmost index \( j \in [0 : n] \) such that \( \text{sum}(A[j : i]) = \text{MCS}_A(i) \). This vector for the example of Fig. 3(a) is given by
The statement in Lemma 6 yields that if \( A \) is irregular, then the circular interval partition has the following characteristics, for all \( i \in [0 : n) \). First, if \( OMCS[i] \neq i + 1 \), then \( OMCS[i + 1] = OMCS[i] \). Second, \( OMCS[h] = OMCS[i] \) for all \( h \in [i : OMCS[i] - 1] \). A direct consequence is that if \( i = OMCS[i] \) for some \( i \in [0 : n) \), then \( A \) has only one interval.

The first step of the preprocessing algorithm is to compute the array \( MCS \) for all \( i \in [0 : n) \), as follows. The algorithm begins with an initially empty queue and then performs \( n \) successive insertions: for all \( i \in [0 : n) \), insertion \( i \) first adds \( A[i] \) to all already inserted elements and then enqueues \( A[i] \), as indicated below:

\[
\begin{align*}
0 & \rightarrow [A[0]] \\
1 & \rightarrow \left[ \sum_{i=0}^{1} A[i], A[1] \right] \\
2 & \rightarrow \left[ \sum_{i=0}^{2} A[i], \sum_{i=1}^{2} A[i], A[2] \right] \\
& \cdots \\
n-1 & \rightarrow \left[ \sum_{i=0}^{n-1} A[i], \sum_{i=1}^{n-1} A[i], \cdots, \sum_{i=n-2}^{n-1} A[i], A[n-1] \right].
\end{align*}
\]

Clearly, the final state of the queue that results from these \( n \) operations is the set of all subsequences ending at \( A[n-1] \). It turns out that the greatest element in the queue equals \( MCS_A(n-1) \), so the preprocessing algorithm peeks this value and sets \( MCS[n-1] \) to it. To obtain the remaining elements of the array \( MCS \), the algorithm performs the following operations on the queue: for all \( i \in [0 : n-2) \), (1) remove the oldest element, (2) add \( A[i] \) to all elements, (3) enqueue \( A[i] \), and (4) peek the greatest element and set \( MCS[i] \) to it. As an example, note that the state of the queue after iteration \( i = 0 \) is

\[
\left[ \left( \sum_{i=1}^{n-1} A[i] \right) + A[0], \left( \sum_{i=2}^{n-1} A[i] \right) + A[0], \ldots, \left( \sum_{i=n-2}^{n-1} A[i] \right) + A[0] \right],
\]

and that the greatest element of the queue in this state is actually \( MCS_A(0) \).

The computation of the array \( MCS \) as performed above involves the following special queue operations:

- **PUSH** the insertion operation takes as arguments not only a new element, but also a real number \( d \) which must be added to all elements currently stored in the queue before the new element is inserted;
- **POP** remove the least recently inserted element;
- **PEEK** return, without removing, the element with greatest key with ties broken by choosing the least recently inserted element.

The implementation of these operations involves the following structures. A linked list \( Q \) stores all elements already inserted with \( PUSH \) and not yet removed with \( POP \), sorted in a FIFO order. To give fast access to the maximum element, a doubly linked list \( L \), sorted in a nondecreasing order, is maintained containing the current maximum element and all other \( v \) of \( Q \) such that \( all \ w \in Q, v \leq w \), have been inserted in \( Q \) before \( v \). This property is preserved by removing from \( L \) all elements strictly smaller than \( w \) when \( w \) is inserted in \( Q \) with \( PUSH \). In addition, a variable \( D \) stores the sum of the parameter \( d \) corresponding to all \( PUSH \) operations performed so far.

Every element \( v \) in the queue is associated with a pointer to the next element in \( Q \), pointers to the previous and next elements in \( L \), and the value \( D(v) \) that equals \( D \) when \( v \) has been inserted with \( PUSH \). The \( POP \) operation is trivially performed in constant time by simply removing the first element in \( L \) from both \( L \) and \( Q \). The constant time execution of the \( PEEK \) operation is also trivial by simply returning the last
The rest of the data that the preprocessing algorithm needs to compute is the following. The circular array \( A \) is the circular equivalent of array \( K \), which can easily be computed by replacing array \( K \) with \( K[p] \) for every index \( p \). Clearly, all this can be done in \( O(n) \) time.

We conclude that array \( MCS \) can be computed by means of \( O(n) \) queue operations and that this takes \( O(n) \) time. Since the \texttt{peek} operation of a queue always refers to the oldest greatest element, we can get the array \( OMCS \) directly as a by-product of our computation of array \( MCS \). For this purpose, it is sufficient to supply index \( i \) as a satellite data every time \( A[i] \) is enqueued. To finalize this first part of the preprocessing algorithm, a suitable circular left-shift is performed in \( A \) and, in order that arrays \( MCS \) and \( OMCS \) remain consistent, we can either compute them from scratch again or perform the same left-shift on them. Clearly, all this can be done in \( O(n) \) time.

Arrays from Table 1

\[
\begin{array}{ll}
K, \text{IMPFS and IS} \\
PRVMLS, \text{INXTMS, and MG}, \text{if } A \text{ is regular} \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Arrays with indices in the range ([0 : n])</th>
</tr>
</thead>
<tbody>
<tr>
<td>( MCS[i] ) &amp; ( = \mathcal{MCS}_A(i) )</td>
</tr>
<tr>
<td>( OMCS[i] ) &amp; ( = i - \max{j \in [0 : n) : \text{sum}(A[i - j : i]) = \mathcal{MCS}_A(i)} )</td>
</tr>
<tr>
<td>( PFMCS[p] ) &amp; ( = \mathcal{MCS}_A([\alpha_k : p]) )</td>
</tr>
<tr>
<td>( PFRMCS[p] ) &amp; ( = \mathcal{MCS}_A([0 : n) \setminus [\alpha_k : p]) )</td>
</tr>
<tr>
<td>( PFRMMCS[p] ) &amp; ( = \mathcal{MCS}_A([0 : n) \setminus [\alpha_k : p]) )</td>
</tr>
<tr>
<td>( PFSF[p] ) &amp; ( = \min{0} \cup {\text{sum}(A[i : j]) : \alpha_k &lt; i \leq j &lt; p} )</td>
</tr>
<tr>
<td>( SFMCS[p] ) &amp; ( = \mathcal{MCS}_A([p : \beta_k]) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arrays with indices in the range ([0 : \ell])</th>
</tr>
</thead>
<tbody>
<tr>
<td>( IMCS[i] ) &amp; ( = \mathcal{MCS}_A([\alpha_i : \beta_i]) )</td>
</tr>
<tr>
<td>( IOMCS[i] ) &amp; ( = \max{IMCS[j] : i \neq j \in [0 : \ell]} )</td>
</tr>
<tr>
<td>( IRS[i] ) &amp; ( = \text{the greatest sum of a prefix of } A[\alpha_i+1 : \beta_i+(\ell-1)] )</td>
</tr>
</tbody>
</table>

Table 2: Summary for the circular case. Those indexed with \( p \) (instead of \( i \)) make use of \( k = K[p] \) in their definitions. Some arrays are not defined for every index (see Section 5).

The rest of the data that the preprocessing algorithm needs to compute is the following. The circular interval partition of \( A \) can be determined in \( O(n) \) time by using sequence \( A \) and arrays \( MCS \) and \( OMCS \). Moreover, since \( \mathcal{MCS}(A^{(x \rightarrow n)}) = \mathcal{MCS}(A^{(x \rightarrow 0)}) \) is true for any \( x \), then the query-answering algorithm can treat an insertion into position \( p = n \) as one into position \( p = 0 \), which implies that, in the circular case, no array needs to have indices in the range \([0 : n]\). If \( A \) is regular, then the required arrays from Table 1 can be computed exactly as described in Section 3 (with array \( MCS \) replacing array \( MLS \)). What remains to be shown is how to compute the arrays of Table 2. We do not further discuss, however, the computation of arrays \( PFMCS \) (which is defined only for \( p > \alpha_k \)), \( SFMCS \) and \( IMCS \), which can easily be computed by simple traversals of array \( MCS \).

Array \( IRS \) is the circular equivalent of array \( IRS \) of Table 1 and needs to be computed iff \( \ell > 1 \). Note that, given an interval \( I_i \), if \( IRS[i] = \text{sum}(A[\beta_i : j]) \) and \( j \in [\alpha_i' : \beta_i'] \), then certainly \( \text{sum}(A[\alpha_i' : j]) = \text{IMPFS}[\alpha_i'] \). It follows that \( IRS[0] \) equals the greatest element of the queue

\[
\left[ \sum_{i=1}^{\ell-2} IS[i] + \text{IMPFS}[\ell - 1], \ldots, \sum_{i=1}^{2} IS[i] + \text{IMPFS}[3], \sum_{i=1}^{1} IS[i] + \text{IMPFS}[2], \text{IMPFS}[1] \right],
\]
which can be built by means of the following \(\ell - 1\) push operations: for all \(i\) from \(\ell - 1\) to 1, add \(IS[i]\) to all current elements and then enqueue IMPFS[\(i\)]. Moreover, by removing the oldest element from the queue with pop and then performing the push operation in question with index \(0 = (\ell - 1) + 1\), we get a queue whose greatest element equals IRCS[\(\ell - 1\)]. By repeating this procedure for indices \(\ell - 2, \ldots, 1\), we finally obtain array IRCS in \(O(\ell)\) time. The computation of the array IOMCS is similar, with each push operation defined by the new element IMCS[\(i\)] and without changing the elements currently stored in the queue.

Array PFSF is defined only for \(p > \alpha_k\). Letting \(A^- = (-A[0], -A[1], \ldots, -A[n - 1])\), we have:

**Observation 1** \(PFSF[p] = -\text{MLS}(A^-(\alpha_k : p))\) for all \(p \in [0 : n]\) such that \(p > \alpha_k\).

**Proof.** If \(A(\alpha_k : p)\) has no negative elements, then \(PFSF[p] = 0 = -\text{MLS}(A^-(\alpha_k : p))\), as desired. If, on the other hand, \(A(\alpha_k : p)\) has at least one negative element, then \(\text{MLS}(A^-(\alpha_k : p)) > 0\). Let \(i, j \in (\alpha_k : p)\) be such that \(\text{sum}(A^-[i : j]) = \text{MLS}(A^-(\alpha_k : p))\). Now, the definition of array PFSF implies that \(PFSF[p] \leq \text{sum}(A[i : j]) = -\text{sum}(A^-[i : j]) = -\text{MLS}(A^-(\alpha_k : p))\). To conclude the proof, suppose by contradiction that \(PFSF[p] < -\text{MLS}(A^-[i : j])\). Thus, there are \(i', j' \in (\alpha_k : p)\) such that \(\text{sum}(A[i' : j']) < -\text{sum}(A^-[i : j])\), which implies that \(\text{sum}(A^-[i' : j']) > \text{sum}(A^-[i : j])\), contradicting our choice of \(i\) and \(j\). \(\square\)

This observation immediately implies that, for every interval \(I_i\) of \(A\), we can compute \(PFSF[j]\) for all \(j \in (\alpha_i : \beta_i)\) by means of a single run of Kadane’s algorithm with \(A^-(\alpha_i : \beta_i)\) as input. Since this takes \(O(|I_i|)\) time, it follows that we can compute array PFSF in \(O(n)\) time. Array PFRMMS must be computed iff \(A\) is irregular. For any \(p\), if \(\ell = 1\), then array PFRMMS can be computed by means of a right-to-left run of Kadane’s algorithm on \(A\). If \(\ell > 1\), we have:

**Lemma 10** If \(\ell > 1\), then \(PFRMMS[p] = \text{MLS}(A[p : \beta_k](\text{IRCS}[k]))\) for all \(p \in [0 : n]\).

**Proof.** We first claim that \(PFRMMS[p] \leq \text{MLS}(A[p : \beta_k](\text{IRCS}[k]))\). Indeed, let \(B = A[p : \beta_{k+(\ell - 1)}]\); \(PFRMMS[p]\) is defined as \(\text{MLS}(B)\). Since the first element of \(I_{k+1}\) is nonnegative by definition, there are \(i, j \in [p : \alpha_k]\) such that \(\text{MLS}(B) = \text{sum}(S)\), where \(S = A[i : j]\). Given such \(i\) and \(j\), either \(S\) encompasses element \(A[\beta_k + 1]\) (that is, \(i \in [p : \beta_k + 1]\) and \(j \in (\beta_k : \alpha_k)\)) or not. In the latter case, either \(S\) is a subsequence of \(A[p : \beta_k]\) or \(S\) is a subsequence of \(A(\alpha_k+1 ; \beta_{k+(\ell - 1)})\), in which case the sum of any subsequence of \(A(\alpha_{k+1} ; \beta_{k+(\ell - 1)})\) is not greater than \(\text{IRCS}[k]\). In both situations, the claim holds. In the former case, \(S = S'A[\beta_k : j]\), where \(S'\) is either \(\emptyset\), if \(i \notin [p : \beta_k]\), or \(A[i : \beta_k]\), otherwise. Since \(\text{sum}(A[\beta_k : j]) \leq \text{IRCS}[k]\), our claim also holds in this case.

Finally, to see that \(PFRMMS[p] \geq \text{MLS}(A[p : \beta_k](\text{IRCS}[k]))\), note that, for every maximal sum subsequence \(S\) of \(A[p : \beta_k](\text{IRCS}[k])\), it is easy to find a subsequence \(S'\) of \(B\) such that \(\text{sum}(S') = \text{sum}(S)\). \(\square\)

Thus, for every interval \(I_i\) of \(A\), we can compute \(PFRMMS[j]\) for all \(j \in [\alpha_i : \beta_i]\) in \(O(n)\) time by means of a single right-to-left run of Kadane’s algorithm on sequence \(I_i(\text{IRCS}[i])\).

Array PFRMMS also needs to be computed iff \(A\) is irregular. Its definition immediately implies that, for all \(p \in [0 : n]\),

\[
PFRMMS[p] = \begin{cases} 
  \text{MCS}[p], & \text{if } p = \beta_k \text{ and } \ell = 1; \\
  \max\{\text{MCS}[p], \text{IOMCS}[p]\}, & \text{if } p = \beta_k \text{ and } \ell > 1; \\
  \max\{\text{MCS}[p], \text{PFRMMS}[p + 1]\}, & \text{if } p < \beta_k.
\end{cases}
\]

This directly implies a right-to-left computation of the array in \(O(n)\) time.

**Theorem 3** The preprocessing algorithm computes the summary in Table 2 in \(O(n)\) time for any sequence \(A\) of size \(n\).
6 Query-answering Algorithm for the Circular Case

The computation of $\mathcal{MCS}(A^p)$ in $O(1)$ time for the summary described in Table 2 depends on the sign of $x$ and on the type of sequence $A$. In what follows, if $p = n$, we for convenience set $p$ to zero since $\mathcal{MCS}(A^{(x \to n)}) = \mathcal{MCS}(A^{(x \to 0)})$. Moreover, we adopt again the notation $i^o$ to denote the index of element $A[i]$ into sequence $A^p$, for all $i \in [0 : n]$.

6.1 Handling a Nonnegative $x$ and a Regular Sequence

If $A$ is regular, then $\mathcal{MCS}_A(i) = \mathcal{MLS}_A(i)$ for all $i$ by definition. It turns out that the linear and circular sums are affected similarly by the insertion of $x \geq 0$. Roughly speaking, if the maximal circular sums in an interval $I_p$ increase by some value $z$, then the maximal circular sums in interval $I_{p+1}$ increase by $\max\{0, z + \text{sum}(I_p)\}$. The partition of $[0 : n]$ used in this case is $([0 : \alpha_k) \cup ([\beta_{k+1} : n]) \cup [\alpha_k : p) \cup [p : p] \cup (p : \beta_k + 1)$, which gives

$$\mathcal{MCS}(A^p) = \max\{0, \mathcal{MCS}_{A^p}([0 : \alpha_k) \cup ([\beta_{k+1} : n]), \mathcal{MCS}_{A^p}([\alpha_k : p)), \mathcal{MCS}_{A^p}(p), \mathcal{MCS}_{A^p}((p : \beta_k + 1))\}.$$ 

The essential difference in the circular sums is that interval $I_0$ is also affected by interval $I_{k-1}$. We therefore need an extension of our results for the linear case, which is provided below.

**Lemma 11** If $A$ is regular and $x \geq 0$, then $\mathcal{MCS}_{A^p}(p) = \mathcal{MCS}_A(p - 1) + x$. Moreover, for all $i \in [0 : n)$, $\mathcal{MCS}_{A^p}(i^o)$ equals:

1. $\max\{\mathcal{MCS}_A(i), x + \text{sum}(A[\alpha_k : i])\}$, if $i \notin [\alpha_k : \beta_k]$, 
2. $\max\{\mathcal{MCS}_A(i), \max\{x + \text{sum}(A) - \text{sum}(A(i : j)) : j \in (i : p)\}\}$, if $i \in [\alpha_k : p)$, and 
3. $\mathcal{MCS}_A(i) + x$, if $i \in [p : \beta_k]$.

**Proof.** The statement about $\mathcal{MCS}_{A^p}(p)$ follows directly from definition. Let $i \in [0 : n)$. If $i \in [p : \beta_k]$, then $\mathcal{MCS}_{A^p}(i^o) \geq \text{sum}(A[\alpha_k : i^o]) = \mathcal{MCS}_A(i) + x = \mathcal{MCS}_A(i) + x$. Statement 3 stems from the fact that there is no $j \in [0 : n)$ such that $\text{sum}(A[j : i]) > \mathcal{MCS}_A(i)$.

Suppose that $i \notin [p : \beta_k]$ and let $I_{k'}$ be the interval of $A$ such that $i \in [\alpha_{k'} : \beta_{k'}]$. Since $A[\alpha_{k'} : i] = A^p[\alpha_{k'} : i^o]$, we get $\mathcal{MCS}_{A^p}(i^o) \geq \mathcal{MCS}_A(i)$. We only need to consider the case when the subsequence of $A^p$ that gives $\mathcal{MCS}_{A^p}(i^o)$ contains $x$. Thus, we show statements 1 and 2 assuming that $\mathcal{MCS}_{A^p}(i^o) > \mathcal{MCS}_A(i)$. First suppose $k \neq k'$, which corresponds to statement 1. Since intervals have negative nonempty proper suffixes and nonnegative proper prefixes, we get $\mathcal{MCS}_{A^p}(i^o) = \text{sum}(A[\alpha_k : i^o]) = x + \text{sum}(A[\alpha_k : i])$. On the other hand, if $k = k'$, then $i \in [\alpha_k : p)$ and $\mathcal{MCS}_{A^p}(i^o)$ equals the greatest sum of a suffix of $A^p[i+1 : i]$ which includes $x$, that is, $\mathcal{MCS}_{A^p}(i^o) = \text{sum}(A[j : i]) = x + \text{sum}(A) - \text{sum}(A(i : j))$ for some $j \in (i : p)$, which proves statement 2.

The lemma above immediately implies the following.

**Corollary 1** If $A$ is regular and $x \geq 0$, then $\mathcal{MCS}(A^p)$ is the maximum among

1. $x + \text{MCS}[p - 1]$,
2. $x + \text{SF MCS}[p]$,
3. $\max\{\text{IOMCS}[k], x + \text{IS}[k] + \text{IRCS}[k]\}$, if $\ell > 1$, and 
4. $\max\{\text{PF MCS}[p], x + \text{sum}(A) - \text{PFSF}[p]\}$, if $p > \alpha_k$.
6.2 Handling a Negative \( x \) and a Regular Sequence

In this case, the following observation implies \( \mathcal{MCS}(A^p) = \mathcal{MLS}(A^p) \).

Observation 2 If \( A \) is regular and \( x < 0 \), then \( \mathcal{MCS}_{A^p}(i) = \mathcal{MLS}_{A^p}(i) \) for all \( i \in [0 : n] \).

Proof. Suppose by contradiction that, for some \( i \), \( \mathcal{MCS}_{A^p}(i) \neq \mathcal{MLS}_{A^p}(i) \). It follows that \( \mathcal{MCS}_{A^p}(i) > \mathcal{MLS}_{A^p}(i) \), and thus that \( \text{sum}(A^p[j : i]) = \mathcal{MCS}_{A^p}(i) \) for some \( j \in (i : n] \). Since \( x < 0 \) and \( \mathcal{MCS}_A(n-1) < 0 \), it follows that \( \text{sum}(A^p[j : n]) < 0 \), which implies that \( \mathcal{MCS}_{A^p}(i) < \text{sum}(A^p[0 : i]) \leq \mathcal{MLS}_{A^p}(i) \), a contradiction.

A consequence of this observation is that, since the summary for the circular case includes, when \( A \) is regular, the arrays used to handle negative values of \( x \) in the linear case, then we can compute \( \mathcal{MLS}(A^p) \), and thus \( \mathcal{MCS}(A^p) \), in \( O(1) \) time exactly as shown in Section 4.

6.3 Handling a Nonnegative \( x \) and an Irregular Sequence

When \( A \) is irregular, we know by Lemma 6 that inserting \( x \geq 0 \) into interval \( I_k \) increases the maximal circular sums at the elements of every interval \( I_k' \neq I_k \) by exactly \( x \). Moreover, the maximal circular sums at the other elements of \( A^p \) can be computed exactly as when \( A \) is regular, which leads to the following result.

Lemma 12 If \( A \) is irregular and \( x \geq 0 \), then \( \mathcal{MCS}(A^p) \) is the maximum among:

1. \( x + \text{MCS}[p-1] \),
2. \( x + \text{SFMCS}[p] \),
3. \( x + \text{IOMCS}[k] \), if \( \ell > 1 \), and
4. \( \max\{\text{PFMCS}[p], x + \text{sum}(A) - \text{PFSF}[p]\} \), if \( p > \alpha_k \).

Proof. The term of 1 equals \( \mathcal{MCS}_{A^p}(p) \). Thus, to prove the lemma, it is enough to show that the terms 2, 3, and 4 equal the maximal value of \( \mathcal{MCS}_{A^p}(\beta^i) \) for all \( i \) in, respectively \([p : \beta_k]\), \([0 : n] \setminus [\alpha_k : \beta_k]\), and \([\alpha_k : p-1]\).

Given \( i \in [p : \beta_k] \) we have \( \text{OMCS}[i] = \beta_k + 1 \) by Lemma 6. Thus, by definition of \( \text{OMCS}[i] \) and since \( x \) is an element of \( S = A^p[(\beta_k + 1)^i : i^0] \), we have \( \mathcal{MCS}_{A^p}(i^0) = \text{sum}(S) = x + \text{sum}(A(\beta_k : i)) = x + \text{MCS}_A(i) \).

The definition of \( \text{SFMCS} \) then implies that \( \max\{\mathcal{MCS}_{A^p}(i^0) : i \in [p : \beta_k]\} = x + \text{SFMCS}[p] \), as desired.

If \( i \in [0 : n] \setminus [\alpha_k : \beta_k] \), that is, if \( i \in [\alpha_{i,j} : \beta_j] \) for some interval \( I_i \neq I_k \), then the argument is analogous to the previous one, the difference being that, if \( \beta_j + 1 = p \), then the subsequence which gives \( \mathcal{MCS}_{A^p}(i^0) \) does not begin at \( A[\beta_j + 1] \), but instead at \( A^p[p] = x \), that is, \( \mathcal{MCS}_{A^p}(i^0) = \text{sum}(A^p[p : i^0]) = x + \text{sum}(A(\beta_j : i)) = x + \text{MCS}_A(i) \).

Finally, given \( i \in [\alpha_k : p] \), first note that, in contrast with the previous cases, subsequence \( A^p[(\beta_k + 1)^i : i^0] \) does not include \( x \). Now, either \( \mathcal{MCS}_{A^p}(i^0) = \text{sum}(A(\beta_k : i)) \) or not. If not, then, by definition of \( \text{OMCS}[i] = \beta_k + 1 \), we have \( \mathcal{MCS}_{A^p}(i^0) = \text{sum}(S') \), for some subsequence \( S' \) of \( A^p \) which ends at \( A[i] \) and includes \( x \), that is, \( S' = A^p[j : i] \) for some \( j \in (i : p] \). Note then that \( \text{sum}(S') = x + \text{sum}(A) - \text{sum}(A[j : j]) \).

The definitions of arrays \( \text{PFMCS} \) and \( \text{PFSF} \) then imply that \( \max\{\mathcal{MCS}_{A^p}(i^0) : i \in [\alpha_k : p]\} = \max\{\text{PFMCS}[p], x + \text{sum}(A) - \text{PFSF}[p]\} \), as desired.

6.4 Handling a Negative \( x \) and an Irregular Sequence

As depicted in Fig. 4, if \( A \) is irregular and \( p \neq \alpha_k \), then Lemma 6 implies that the insertion of \( x < 0 \) into \( I_k \) does not affect the maximal circular sum at any \( A[i] \) such that \( i \in (\alpha_k : p) \), that is, \( \mathcal{MCS}_{A^p}(i^0) = \mathcal{MCS}_A(i) \).

The same is true of any \( A[i] \) such that \( i \in [\alpha_k : \beta_{k'}] \), where \( k' = k - 1 \), if \( p = \alpha_k \). It follows that

\[
\mathcal{MCS}_{A^p}(\alpha_k : p) = \text{PFMCS}[p], \quad \text{if } p \neq \alpha_k;
\]

\[
\mathcal{MCS}_{A^p}(\alpha_{k'} : \beta_{k'}) = \text{IMCS}[k'], \quad \text{if } p = \alpha_k.
\]
which ends in $\ell$ changes the maximal circular sum at Figure 4: The elements of an irregular sequence $A$ in an insertion of $x < 0$. Subsequence $A[\text{OMCS}[:i]]$, whose sum is $\text{MCS}_{A}(i)$, is colored green/orange when $x$ is inserted outside/inside it.

Suppose that $p \neq \alpha_{k}$ and let $i \in [0:n \setminus \alpha_{k} : p)$. As depicted in Fig. 4, the insertion of $x$ potentially changes the maximal circular sum at $A[i]$, because $x$ is inserted (circularly) between $A[\text{OMCS}[:i]]$ and $A[i]$. In this case, independently of whether $i \in [p : \beta_{k}]$ or not, $\text{MCS}_{A^p}(i^{o})$ equals either (1) $\text{MCS}_{A}(i) + x$ – which, from the definition of $\text{OMCS}[:i]$, is the greatest sum of a subsequence of $A^p$ which ends in $A[i]$ and includes $x$ – or the greatest sum of a suffix of $A^p[p^{o} : i^{o}] = A[p : i]$ – which is the greatest sum of a subsequence of $A^p$ which ends in $A[i]$ and does not include $x$. It follows that

$$\text{MCS}_{A^p}([0:n \setminus \alpha_{k}^{o} : p^{o})] = \max\{x + PFRMMS[p], PFRMMS[p]\}.$$  

The analysis for the case when $p = \alpha_{k}$ and $i \notin \alpha_{k} : \beta_{k}$ is similar to the previous one: $\text{MCS}_{A^p}(i^{o})$ equals either (1) $\text{MCS}_{A}(i) + x$ or (2) the greatest sum of a suffix of $A[p : i]$. Moreover, the latter term actually equals $\text{sum}([A[p : i])$, since, by definition, every prefix of $A[\alpha_{k} : i]$ has nonnegative sum. It follows that, if $\ell > 1$, then

$$\text{MCS}_{A^p}([0:n \setminus \alpha_{k}^{o} : \beta_{k}^{o}]) = \max\{x + IOMCS[k'], IRCS[k']\}.$$  

To conclude the argument, we claim that $\text{MCS}_{A^p}(p) < \text{MCS}(A^p)$. Indeed, suppose by contradiction that $\text{sum}(A^p[i : p]) \geq \text{MCS}(A^p)$ for some $i \in [0:n]$. Since $\text{MCS}(A^p) \geq 0$ and $x < 0$, then $i \neq p$, which implies that $\text{sum}(A^p[i : p]) > \text{MCS}(A^p)$, contradicting the definition of $\text{MCS}(A^p)$. Thus, we conclude that $\text{MCS}(A^p)$ is the maximum between zero and $\max\{\text{MCS}_{A^p}(i^{o}) : i \in [0:n]\}$. Finally, since terms $PFMCS[p]$ and $IMCS[k']$ are certainly nonnegative, then our previous considerations imply the following result.

**Lemma 13** If $x < 0$ and $A$ is irregular, then $\text{MCS}(A^p)$ equals:

1. $\max\{PFMCS[p], x + PFRMMS[p], PFRMMS[p]\}$, if $p \neq \alpha_{k}$, and
2. the maximum among $\text{IMCS}[k'], x + IOMCS[k']$, and $IRCS[k']$, where $k' = k - 1$, if $p = \alpha_{k}$. (Include the two last terms if $\ell > 1$.)

Gathering the results of this section, we finally get the main result of this section.
Theorem 4 Given a summary as defined in Section 5, the query-answering algorithm computes $\text{MCS}(A^{(x \rightarrow p)})$ in $O(1)$ worst-case time for any $x \in \mathbb{R}$ and $p \in [0 : n]$. 

7 Concluding Remarks

In this paper we have considered the problem of, for a fixed sequence $A$ of $n$ real numbers, answering queries which ask the value of $\text{MLS}(A^{(x \rightarrow p)})$ or $\text{MCS}(A^{(x \rightarrow p)})$ for given $x \in \mathbb{R}$ and $p \in [0 : n]$. We showed that, after an $O(n)$ time preprocessing step has been carried out on $A$, both kinds of queries can be answered in constant worst-case time. This is both an optimal solution to the problem and a considerable improvement over the naive strategy of answering such queries by means of Kadane’s algorithm (or a variation of it, in the circular case), which takes $\Theta(n)$ time per query. This problem has applications in the context of finding heuristic solutions to an NP-hard problem of buffer minimization in wireless mesh networks. Given the generality of these kinds of queries and the multiplicity of applications of the maximal sum subsequence concept, we would not be surprised to see other applications of our algorithms in the future. An interesting related problem is then that of, given an $m \times n$ matrix $A$, inserting $k$ size-$m$ columns $C_0, \ldots, C_{k-1}$ successively and cumulatively into $A$ in order to minimize $\text{score}(A') = \sum_{i=0}^{m-1} \text{MCS}(A'[i, 0], \ldots, A'[i, n-1])$ for every matrix $A'$ resulting of each insertion. By using the algorithms presented in this paper to insert one column at a time, one can carry out $k$ successive insertions in $O(m(n+1) + m(n+2) + \ldots + m(n+k)) = O(kmn+mk)$ time. However, since the input to the problem has size $O(mn+mk)$, we leave it as an open problem whether substantially more efficient algorithms exist.

References


