Laborator 1

1) A n-size vector `a` is circular-sorted if, for some index i between 0 and n-1, the vector `{a[i],a[i+1],…,a[n-1],a[0],a[1],…,a[i-1]}` is sorted.

   Example: `a[] = {4,5,6,0,1,2,3}` is circular sorted (take i=3).

   a. Write a function **`int`** `minCirc(vector<`**`int`**`>& a)` which takes as input a circular-sorted vector and outputs the minimum element of a.

      Complexity: O(log(n)), where n = `a.size()`

   b. Write a function **`int`** `firstCirc(vector<`**`int`**`>& a,` **`int`** `e)` which takes as input a circular-sorted vector and an integer `e`. It outputs either -1 if `e` is not present in the vector, or the least index i such that `a[i] = e`.

      Complexity: O(log(n)), where n = `a.size()`

2) A vector is k-almost sorted if it could be transformed into a sorted vector by moving every elements from at most k positions.

   Example: a 0-almost sorted vector is a sorted vector. The vector `a[] = {1,0,3,2,5,4}` is 1-almost sorted. The vector `b[] = {3,2,1,4,5,6,9,8,7}` is 2-almost sorted.

   Write a function **`int`** `lastAlmost(vector<`**`int`**`>& a,` **`int`** `k,` **`int`** `e)` which takes as input a vector `a`, an integer `k` such that `a` is `k`-almost sorted, and an integer value `e`. It outputs either -1 if `e` is not present in the vector, or the largest index i such that `a[i] = e`.

   Complexity: O(k*log(n)), where n = `a.size()`

3) A vector is alternate-sorted if the sub-vector of all elements at even positions (i.e., `{a[0],a[2],a[4],…}`) is sorted by increasing values and the sub-vector of all elements at odd positions (i.e., `{a[1],a[3],a[5],…}`) is sorted by decreasing values.

   Write a function **`int`** `existsAlternate(vector<`**`int`**`>& a,` **`int`** `e)` which takes as input an alternate-sorted vector `a` and an integer value `e`. It ouputs -1 if `e` is not present in the vector, or any index i such that `a[i] = e`.

   Complexity: O(log(n)), where n = `a.size()`

4) A vector is bitonic if it can be partitioned into a vector sorted by increasing values and a vector sorted by decreasing values. The peak of the vector is its largest element.

   Example: a[] = {1,2,3,4,5,4,3,2,1} is bitonic and its peak equals 5.

   a. Write a function int peak(vector<int>& a) that takes as input a bitonic vector. Its output is any position in the vector that is equal to its peak.

      Example: if the input is a[] = {1,2,3,4,5,4,3,3,1} then the output must be 4.

      Complexity: O(log(n)) where n = a.size().

   b. Write a function int existsBitonic(vector<int>& a, int e) that takes as value a bitonic vector a and an integer value e. It ouputs -1 if e is not present in the vector, or any index i such that a[i] = e.

      Complexity: O(log(n)), where n = a.size()

5) Given a vector `a`, a threshold-sum query on a requires as input a triple `(i,j,e)`. Its output consists of the sum of all elements greater than `e` between positions `i` and `j`.

   Example: if `a[] = {0,3,44,5,11,2,1,6,78,9}` and the query input is `(3,7,4)`, then the query only considers the sub-vector `a[3,7] = {5,11,2,1,6}`. It outputs 55+11+6=72.

   We define a query as follows:

   ```
   struct query { int i,j,e; };
   ```

   Write a function

   ```
   vector<int> threshold-sum(vector<int>& a, vector<query>& q);
   ```

   Its output is a vector of size `q.size()` whose entries are the outputs for each query in vector q.

   Complexity: O(N√N *log(N)), where N = `a.size() + q.size()`

6) **`Implement MergeSort, HeapSort and Quicksort.`**

   `Link:` https://www.infoarena.ro/problema/algsort

7) **`Implement RadixSort.`**

`Link:` https://www.infoarena.ro/problema/radixsort