*All your algorithms must be written in pseudo code, and justified.  
A comparison will be considered as an elementary operation in O(1)*

1. Let a[] be a vector of size n and i an index between 0 and n-1. Propose an algorithm in O(n) time in order to output the largest element between two indices i and j (i < j). **/1**
2. Let a[] be a vector. The *Cartesian tree* of a[] is a binary rooted tree whose nodes are the element of a[]. It is recursively defined as follows: let a[j] be the largest element in the vector. Then, the root of the tree equals a[j], the left subtree is the Cartesian tree of the sub-vector a[0..j-1] (all element before a[j]) and the right subtree is the Cartesian tree of the sub-vector a[j+1..n-1] (all elements after a[j]).  
     
   a) Draw the Cartesian tree of the following vector a[] = {19,1,5,33,4,2,55,11}. **/1**

b) A right path is a rooted binary tree where all the left subtrees are empty. Characterize the vectors whose Cartesian tree is a right path. **/1**

c) For every index i, we define left[i] (resp., right[i]) as follows:

- if a[i] is larger than all elements in {a[0],a[1],...,a[i-1]} (resp., larger than all elements in {a[i+1],...a[n-1]}) then we set left[i] = -1 (resp., we set right[i] = -1).

- otherwise, left[i] is the largest index j < i such that a[j] > a[i] (resp., right[i] is the smallest index j > i such that a[j] > a[i]).

Write an O(n)-time algorithm in order to compute the vectors left[] and right[] – Hint: use a stack. **/2**  
  
d) Deduce from the previous question 2)b) an O(n)-time algorithm in order to construct the Cartesian tree of a[]. **/2**

1. a) Recall the definition of a max-heap. **/1**  
   b) Let a[] be a vector of size n. Consider the following algorithm in order to sort this vector:  
    i) Construct the Cartesian tree of a[].

ii) Add the root of the Cartesian tree into a max-heap.

iii) While the max-heap is not empty, select (and remove from the heap) its maximum element e. Add e into the first available position into the vector (starting from i = 0), then add in the max-heap all the children of e into the Cartesian tree.  
  
What is the complexity of this above algorithm? Is this optimal? **/2**