BT challenge Oanță Robert-Christian

Estimated solving time: 1h

Actual solving time : 1.5h

**Algorithm summary:**

Alg1 takes two parameters a list a[] and b which represents an index.

If the index b is equal or greater than list a’s length than it will return -1.

The algorithm purpose is to return the element from index b of the sorted version of list a[] (list starts from 0) . Basically, it returns element from index b from sorted a.

The trick is that list a[] is not fully sorted for achieving this. Usually parts of the list are sorted in such a way that the element from index b is on the right position.

For example if we have a = [3,7,2,1,5,6,10,11], sorted a is [1, 2, 3, 5, 6, 7, 10, 11]

So if b is 2 for example we get the element from index 2 which is 3.

If b is 6 we get element from index 6 which is 10. But here for example the actual list in memory after the sorting and search is {2,1,3,5,6,7,10,11} so as you can see it is not sorted entirely.

**Functions description:**

**Alg1**:

Input: list of integer a[] , integer number b

Output: integer number ( element from list or -1 )

Description: has a counter c which goes from 0 to list’s length – 1, also receives in the e variable the return value from alg2 which represents a postion, once that position is equal to our intial requested position (b) it returns the element.

**Alg2**:

Input: list of integers a[], integer number b, integer number c, integer number d

Output: integer number f

Description: calls alg3 when it is necessary for a switch comparing the current element with an element e which represents a pivot element from the list at index d, f element counts how many time the current elem is smaller than e, when that happens it is time for a switch ( calls alg3) . It works kind of like a selection sort algorithm

**Alg3**:

Input: list of integers a[], integer number b, integer number c

Output: nothing

Description: checks if b is different than c and switches places of numbers based on a xor statment

**Complexity analysis:**

The algorithm works very similiar to know algorithms like selection sort or quick sort

Upper bound complexity is O(length of the array at power 2), let’s call it O(n^2) , it also represents the **worst case** when the algorithm need to sort the entire list in order to return the needed element.

The **best case** would be O(n) when the algorithm need to pass the list only once in order to get the position of the necessary element.

The **average case** would be O(n logn ).