Coursework Algorithm

Question 1.a

. Functionment of the procedure :

Each member of A will be compared to the following members of A . So, for each i, for each j>i, all the members aj will be compared to ai. If one of them is equal to ai, then, the program will return false. Otherwise, It will return true.

. PseudoCode of the procedure :

Distinction(Input: sequence A = <a1, . . . , aN>)
- for each element ai in <a1, . . . , aN>
  - For each element aj in <ai+1, . . . , aN>
    - If aj==ai
      Return false

  Return true

. Exponation of the space and time complexity :

- The procedure does not need the creation of any object to store some datas, so, the space complexity is O(1)
- The first element will be compared to the N-1 next elements, the second will be compared to the next N-2, the i element will be compared to the N-i next elements. So, the time complexity is in the worst case (when there is no duplicate) equivalent to : 1+2+...+ N-1= N(N-1)/2. So the time complexity is $O(N^2)$.

Finally, the Distinct procedure uses O(1) space and $O(N^2)$ time.

. Functionment of an other procedure :

An other procedure could be to sort the element in the input sequence A by using an insertion sort. Then, if there are two following integers which are equals, the procedure returns false otherwise it returns true.

. PseudoCode of the procedure :

Distinction(Input: sequence A = <a1, . . . , aN>)
- For each element ai in <a1, . . . , aN>
  - j=i
  - While j>1 and aj>ai

- Exchange aj and ai
- Decrement j
- For each element ai in <a1, . . . , aN-1>
  - If ai==ai+1
    Return False

Return True

. Exponation of the space and time complexity comparing to the previous procedure :

- In the same way as the previous procedure, the space complexity is O(1).
- The new procedure uses a selection sort to sort the element of the sequence. As seen in the lectures, a selection sort has a time complexity of $O(N^2)$ in the worst case (if the sequence is in a decreasing order). Then, finding two consecutive values has a time complexity of O(N) . Indeed, in the worst case, we have to compare all the following value in the array (so N-1 comparaison). So, the time complexity of this procedure is $O(N)+O(N^2)$ which is equivalent to $O(N^2)$
- Finally, the first and the second Distinct procedure use O(1) space and $O(N^2)$ time. However, the first Distinct procedure will be faster in the best case. Indeed, in the best case, if the first two values of A are the same, the first procedure will go through the loops once whereas the second procedure will sort them first which is longer.

Question 1.b

. PseudoCode of the procedure :

Distinction(Input: sequence A = <a1, . . . , aN>)

- Initialisation of an array HashTable
- For each element ai in <a1, . . . , aN>
  h= hascode(ai)
  - If T[h]!=null
    Return false

  T[h]=ai

Return true

. Functionment of the procedure :

An array HashTable is created. Then, each element in <a1,...,an> is put in the table according to the hash function. For each element ai, if the hash Table is already occupied, it means that there is a duplicate value and the procedure return false. Otherwise, we return true at the end of the procedure.

. Exponation of the space and time complexity :

- In the worst case (if all the values are distinct), all the value are stored in the hash table which needs N spaces. So, the space complexity is O(N).

- Each hash function runs in O(1). In the worst case(if all the values are distinct), the procedure hashes all the values and put it in the hash table one time. So the time complexity is O(N). Finally, the space and time complexity are O(N).

Question 2

. PseudoCode of the procedure :

```
Longest(Input: sequence A = <a0, . . . , aN-1>)
    Table_of_longest=array[N]
    Table_of_longest[0]=1
    i=1
    • While i<N
        • j=i-1
        • While j>=0
            • If ai>aj
                T[i]=T[j]+1
                Halt
            Decrement j
        • If j==-1
            Table_of_longest[i]=1
        Increment i
    longest=Table_of_longest[0]
    • For each element i in <1,...,N-1>
        • If Table_of_longest[i]>longest
            longest=T[i]
    Return longest
```

. Exponation of the procedure :

- Answer of the subproblem : If the length of the longest increasing sequence within A that finishes with Ai is known, for all i < j, what is the length of the longest sequence that finishes with Aj? :

To find the longest increasing sequence that finishes with Aj, it is necessary to find i such as i<j, Ai<Aj and for each p in <i+1,...,j-1>, Ap>Aj. Then, longest(j)=longest(i)+1.

- Answer of the problem :
It is possible to have an array of all the longest increasing sequence of all the subsequences of A answering to the subproblem. Then, the longest increasing sequence of A is the maximun of this array.

. Exponation of the time complexity :

To create the array of all the longest increasing sequences of all the subsequences of A, it is necessary to go to each value of A and in the worst case (if the longest increasing sequence for this subsequence of A is 1), it is then necessary to compare it to all the value of the subsequence. So, it takes 1+2+3+...+N-1=N(N-1)/2 repetitions in the worst case (if A is classified in a decreasing order).
So, the time complexity to create this array is $O(N^2)$.
Then, the time complexity to find the maximum of the array of all the longest increasing sequence is O(N) (It is necessary to make a comparaison for all the value of all the array once).
Finally, the time complexity is $O(N^2) + O(N)$ which is $O(N^2)$.