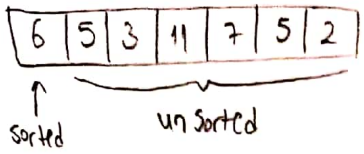
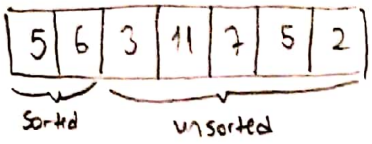
Introduction to Algorithm Design - CSE 321

Harun ALBAYRAK - 171044014 - Homework #2

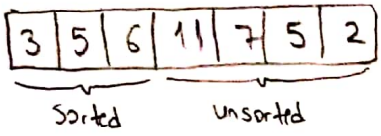
Q1) Array = {6,5,3,11,7,5,2}



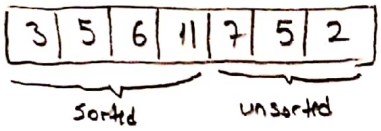
→ This is the first version of the array.



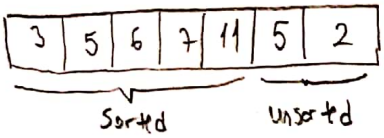
→ Because 5 is less than 6, array's 0-index is 5.



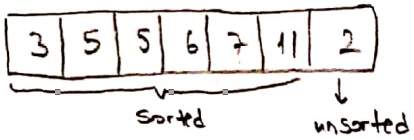
→ Because of $3<5<6$, the array has been created with this order.



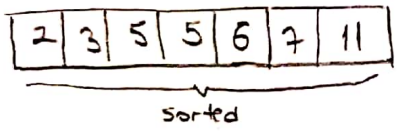
→ Berause 11 is greater than first three element, array's 3rd index is 11.



→ Berause of $6<7<11$, New element of the 3rd index of the array is 7.



→ The first 6 element have sorted by ascending order.



→ Insertion sort has completed. All elements have sorted by ascending order. And 2 is the least number of the array, therefore it is in 0 index.

Q2) →

CamScanner ile tarandı

Q2)

a)
```
function (int n){
    if (n == 1)                          ----> 1
        return;
    for (int i=1; i<=n; i++){            ----> n+1
        for (int j=1 ; j<=n; j++){       ----> 1
            printf ("*");                ----> 1
            break;
        }
    }
}
```

$n+4 \longrightarrow O(n)$

Time complexity of this function is O(n). Because the inner loop includes "break". When the inner loop executed, the loop will be exited. Therefore, the inner loop will be executed only 1 time, whenever the outer loop executed. The outer loop will be executed n times. Hence, the function's time complexity is O(n).

b)
```
void function(int n){
    int count = 0;
    For (int i= n/3 ; i <=n ; i++)
        for (int j=1; j+n/3 <=n ; j++)
            for(int k=1; k <= n; k = k*3)
                count ++;
}
```

This algorithm is $O(n^2 \log n)$ because the innermost loop's time complexity is $O(\log n)$, middle loop's time complexity is $O(n)$ and outermost loop's time complexity is $O(n)$. In innermost, whenever k enter the loop, it is multiplied by 3. Therefore, the innermost loop's time complexity will be $O(\log n)$. And the others loop are executed n times, so these loops take place $O(n)$ time. Because they are all intertwined, the function's time complexity will be $O(n^2 \log n)$

CamScanner ile tarandı

Q3)

```
from bisect import bisect_left

def BinarySearch(arr,x):                    ---> Its time complexity is O(log n)
    i = bisect_left(arr,x)
    if i != len(arr) and arr[i] == x:
        return i
    else
        return -1

def question3(array, number):
    array.sort()                            ---> O(nlog n)
    listx = []                              ---> O(1)
    listy = []                              ---> O(1)
    for x in range(0,len(array)):           ---> O(n)
        m = number / array[x]               ---> O(1)
        res = BinarySearch(array,m)         ---> O(log n)
        if res != -1:                       ---> O(1)
            listx.append(array[x])          ---> O(1)
            listy.append(array[y])          ---> O(1)
                                                  } O(nlog n)
    for x in range(0, len(listx)):          ---> O(n)
        print("Pair:(", listx[x], ",", listy[x], ")")  ---> O(1)
```

array = [1,2,3,6,5,4]
question3(array, 6) → desired number

+

O(nlog n) + O(nlog n) + O(n) => O(nlog n)

This algorithm is O(n log n). The sort algorithm's time complexity is O(nlog n). the loop's time complexity is O(n) and the binary search is O(log n). Since the binary search within the loop, the loop's time complexity will be O(n log n). Also, printing the elements of the pairs is O(n). Therefore, (nlogn + nlogn + n) is (2n log n + n), so the algorithm's time complexity will be O(n log n).

CamScanner ile tarandı

Q4) The time of this process is $O(n+n)$

1) Firstly, we store the binary search tree in an array. This step is $O(n)$

2) Secondly, we store other binary search tree in an another array. This step is also $O(n)$.

3) These arrays that created in step 1 and step 2 are sorted arrays. We need to convert an array these two arrays. This step takes place $O(n+n)$ time.

4) And we need to create an tree using the array we have created in 3rd step. This step also takes place $O(n+n)$ time.

Q5)

```
def question5(array1, array2):
    min_arr = []                                   → O(1)
    max_arr = []                                   → O(1)
    myset = set()                                  → O(1)
    if len(array1) > len(array2):                  → O(1)
        min_arr = array2                           → O(1)
        max_arr = array1                           → O(1)
    else:                                          → O(1)
        min_arr = array1                           → O(1)
        max_arr = array2                           → O(1)
    for x in min_arr:                              → O(n)
        myset.add(x)                               → O(1)
    for y in max_arr:                              → n
        if (y in myset):                           → Average O(1), worst O(n)
            print(y, "was found the both array.")  → O(1)
```

Average = $O(n)$

Worst = $O(n^2)$

The algorithm takes place in linear time ($O(n)$). The first loop takes place $O(n)$ time. The second loop's average case complexity is $O(n)$ but worst case complexity will be $O(n^2)$ since when we lookup the set, we may encounter collisions in the set because python set is also a hashset. When we encounter collisions, the worst case of looking up the set will be $O(n)$. And total worst case will be $O(n^2)$.

CamScanner ile tarandı