# - CSE 321 -Homework 2

23.11,2020

(1) Apply insertion sort to the following sample array: Array = 36, 5, 3, 11, 7, 5, 2}

Show each step of the algorithm. At each step, you are required to not only create the array with its new order, but also explain why you created that sequence

already sorted

A [O] is already sorted. Take A[1] which is 5' and compare with already sorted elements 526 so swap

5 6 3 11 7 5/2 already sorted

Take first unsorted element which is 3' Compare with 6; 3 26 so swap.

53611752

But not enough. Compare 3 with another already sorted 5'. 325 so swap

3 5 6 11 7 5 2 already sorted

3 is at the beginning of the array. So go to the unserted element '11'. Compare with 6; 11>6 so not swap

3 5 6 11 7 5 2 already sorted

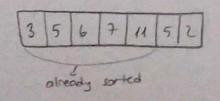
Take first unorted element which is Compare with 11; 7 × 11 so swap.

3 5 6 7 11 5 2

compare 7 with 6. 776 so not swap. we are done with 7.

already sorted.

Take first unsorted element which is 15%. compare with 11; 5211 so swap



35675112
3 5 6 5 7 11 2
3 5 5 6 7 11 2 already sorted
3 5 5 6 7 11 2 already sorted
3 5 5 6 7 2 11
35562711
3 5 5 2 6 7 11
3 5 2 5 6 7 11
3255671
2 3 5 5 6 7 11 already sorted.

Compare 5 with 7. 527 so swap. compare 5 with 6. 5 16 so swap. Compare 5 with other 5. 5 \$5 so not supp. Take first unsorted element which is 2. Compare 2 with already sorted elements. Compare 2 with 11, 2<11 so sup. compare 2 with 7. 227 so swep. Compare 2 with 6. 2 < 6 so swap. Compare 2 with 5. 225 so sug. Compare 2 with other 5. 2 < 5 10 swap Compare 2 with 3. 2 ×3 so swap. 2 is at the beginning of the array.

#### Steps of the insertion sort algorithm &

- 1) Take array of unsorted elements.
- 2) mark the first element as a sorted and rest of unsorted.
- 3) Repeat step 4-5-6 until the unsorted elements finished.
- (1) Take first unsorted element
- 5) compare that element with sorted elements. Swap this element until it arrives at the correct sorted position.
- b) move the marker one position ahead.
- 7) Stop.

2) what is the time complexity of the following programs? Explain in detail by providing your reasoning.

a) function (int n) {

if (n==1)

return;

for (int i=1); (ix=n); i++) {

for (int 5=1; J <= n; J ++) {

printf (" x ");

break;

}

This for loop works

only 1 times in each
time. Because there
is a break statement
inside. It breaks the cyle.

In above code "\*" will print n times. Because inner for loop works only a time when its called than break it. Therefore, the existence of inner loop does not change anything.

So, time complexity of above code is O(n).

for (int 
$$i=n/3$$
;  $i < = n$ ;  $i++$ )  $\int n$  times  
for (int  $j=1$ ;  $j+n/3 < = n$ ;  $j++$ )  $\int n$  times  
for (int  $k=1$ ;  $k < = n$ ;  $k=k < 3$ )  $log_3 n$  times  
count ++; )  $o(n)$ 

# The first for loop is running a times.

$$i = 1, 2, 3, \dots \frac{9}{3} \implies \frac{9}{3}$$
 times

# The second for loop is running n times. 
$$J + \frac{9}{3} <= n$$

$$J = \frac{2n}{3} = \frac{2n}{3} = \frac{2n}{3}$$
times.  $J <= \frac{3n-n}{3}$ 

# The third for loop is running loggn times.

Loop body runs O(1) (constat) time.

k values increases 
$$3^{t}$$

$$3^{t} = n$$

$$t = log_{3}n$$

3t

So, time complexity of above code is O(n.n.loggn)

3) We have an unordered array in which we are looking for pairs whose multiplication yields the desired numbers. For example, let our array be \$1,2,3,6,5,43 and let the desired number be 6. In this case, our pairs will be (1,6) and (2,3). Provide an algorithm that solves this problem with time complexity O(n(logn)), Express your algorithm as pseudocade (write actual code using Python) and prove that its complexity is O(n(logn)).

### Pseudocode

QuickSort (A, low, high)

if (low & high)

pivot = Partition (A, low, high)

QuickSort (A, low, pivot-1)

QuickSort (A, pivot+1, high)

for QuickSort.

We know that QuickSort

Algorithm average time

complexity is O(nlogn)

Partition (A, low, high)

pivot = A[low]

left = low

for i=low+L to high

if (A[i] < pivot) then

swap (A(i), A[left])

left = left + L

swap (pivot, A[left])

return (left)

part3.py

python code

Find Product Pair (A, site, numb)

begin = 0

end = site-1

while beginzend

product = A[begin]\* A[

After sorting, this function takes O(n) time to execute.

product = A[begin] \* A[end]

if (product = = numb)

print (A[begin], A[end])

if (product < numb)

begin++

else
end--

Scarneu with Cams

- First of all; we have an unordered array, we should sort this array increasingly. We can choose any sorting algorithm which time complexity is O(nlagn)
- I choose Quick Sort also I can choose Merge Sort or HeapSort as well.
- After sorting I write Find ProductPair() function to find pairs whose multiplication yields the desired number.

#### In Find Product Pair() ?

Using two integers begin and "end" to index the current pair of numbers. Initialize begin = 0 and end = size-1.

Then compare the product of A[begin] and A[end] with desired number:

If it is equal, we found the pairs.

If it is lower, we must find a bigger product, so increase begin with 1 lf it is higher, we must find a smaller product, so decrease end with 1.

- Find Product Pair () function time complexity is O(n).

Because it has one while loop inside. Statements inside the while loop have constant time complexity. It just increments, decrements...

So one loop -> O(n)

- we have two parts of the algorithms

1) sorting — O(nlogn)

2) Finding pairs in the sorted array -> O(n)

(nlogn name n) = O(nlogn) = O(nlogn)

(4) You are given a binary search tree (BST) with n nodes. You are required to merge this tree with another n-node BST. What is the time complexity of this process? Explain in detail by providing your reasoning.

- We have 2 BST with n nodes and we want to merge them into one BST.

1) First of all, perform the inorder traversal for both BST; it creates two sorted arrays.

In order to analyse the time complexity of inorder traversal, we have to think how many nodes visited. If free has a nodes, then each node is visited only once. Because of that the complexity is O(n)

2) Merge two arrays in one array.

We traverse both arrays and choose the smaller element. In the end, we copy the rest of the elements from two arrays. So the time complexity becomes O(n)

3) Convert sorted array to BST

Fivery node will be created so there will be a calls. Each call have O(1) runtime. Time complexity will be O(n).

- So all three steps takes linear time.

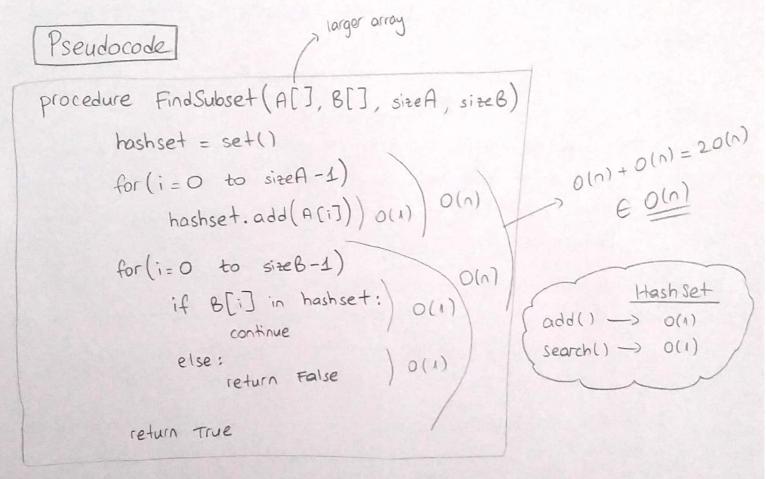
At the beginning we have 2 BST with n nodes.

So we can say that every steps take O(n+n) time to execute.

O(2n)

 $O(2n) + O(2n) + O(2n) = 30(2n) \in O(n)$ 

(5) Suppose that you are given two arrays, where one of them is larger than the other one. Propose a linear time algorithm that finds, if exists, the elements of the small array in the tig array. Express your algorithm using pseudocode and calculate the worst case complexity of the algorithm.



- To reach linear time complexity, I used thash Set.
- + First of all, add all elements of A[] (which is larger array) to the hashset. Adding takes constant time in hashset but we iterate all elements of the A[] so it takes O(n) time.
- After that we iterate over B[] and search for each element of B[] in the hashset. If the element is found than continue, if not than return false. Search takes constant time, so it takes O(n).
- At the end, if there is no returning false than it means that larger array contains the element of the small array, so return true.

O(n) + O(n) = 20(n) € O(n)

## worst case complexity of the algorithms

- We can write the algorithm that takes quadratic time.  $O(n^2)$  we can use two loops.

The outer loop takes all the elements of the smaller array. The inner loop searches if larger array contains smaller array elements linearly.

O(sizeA sizeB) 
$$\leftarrow$$

for  $j=0$  to sizeB

for  $j=0$  to sizeA

 $=$ 
 $=$ 
 $=$ 

Constant

So we have two loops, one of them traverses sizeB times, other one traverses sizeA times.