CSE-321 Introduction to Algorithm Design, Fall 2020
Homework 2 – Question 1

```
void insertionSort(int arr[], int size)
{
    int i, key, j;
    for (i = 1; i < size; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

The insertion algorithm is like that above. So I apply the rules step by step in this question.

Our array is {6,5,3,11,7,5,2}.

Step 1 - a)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
6	5	3	11	7	5	2

Step 1 - b)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
6	5	3	11	7	5	2

Now, according to code ; key = 5, i = 1, j = 0. We compare the index 0 and index 1.

In while loop, j >= 0 and 6 > 5. So we have to put index j to the index j+1.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
6	6	3	11	7	5	2

Again in while loop, j is not greater or equal than 0 because j = -1. So loop finished and index j+1 = 0 must be the key which is 5.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
5	6	3	11	7	5	2

Step 2- a)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
5	6	3	11	7	5	2

Now, according to code ; key = 3, i = 2, j = 1. We compare the index 1 and index 2.

In while loop, j >= 0 and 6 > 3. So we have to put index 1 to the index 2.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
5	6	6	11	7	5	2

Step 2 – b)

Now key = 3, j = 0. Again in while loop, j >= 0 and 5 > 3. So we have to put index 0 to the index 1.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
5	5	6	11	7	5	2

Step 2 – c)

Again in while loop, j is not greater or equal than 0 because j = -1. So loop finished and index i+1 = 0 must be the key which is 3.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	11	7	5	2

Step 3- a)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	11	7	5	2

Now, according to code ; key = 11, i = 3, j = 2. We compare the index 2 and index 3.

In while loop, j >= 0 but 11 is not bigger than 6. So the loop finished and index 3 is again 11.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	11	7	5	2

Step 4- a)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	11	7	5	2

Now, according to code ; key = 7, i = 4, j = 3. We compare the index 3 and index 4.

In while loop, j >= 0 and 11 > 7. So we have to put index 3 to the index 4.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	11	11	5	2

Step 4 – b)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	11	11	5	2

Now key = 7, j = 2. Again in while loop, j >= 0 but 6 is not greater than 7. So loop finished and index j+1 = 3 must be the key.which is 7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	7	11	5	2

Step 5- a)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	7	11	5	2

Now, according to code ; key = 5, i =5, j = 4. We compare the index 4 and index 5.

In while loop, j >= 0 and 11 > 5. So we have to put index 4 to the index 5.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	7	11	11	2

Step 5 – b)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	7	11	11	2

Now key = 5, j = 3. Again in while loop, j >= 0 and 7 > 5. So we have to put index 3 to the index 4.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	7	7	11	2

Step 5 – c)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	7	7	11	2

Now key = 5, j = 2. Again in while loop, j >= 0 and 6 > 5. So we have to put index 2 to the index 3.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	6	7	11	2

Step 5 – d)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	6	6	7	11	2

Now key = 5, j = 1. Again in while loop, j >= 0 but 5 is not greater than 5. So loop finished and index j+1 which is 2 must be the key.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	7	11	2

Step 6- a)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	7	11	2

Now, according to code ; key = 2, i =6, j = 5. We compare the index 5 and index 6.

In while loop, j >= 0 and 11 > 2. So we have to put index 5 to the index 6.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	7	11	11

Step 6 – b)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	7	11	11

Now key = 2, j = 4. Again in while loop, j >= 0 and 7 > 2. So we have to put index 4 to the index 5.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	7	7	11

Step 6 – c)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	7	11	11

Now key = 2, j = 3. Again in while loop, j >= 0 and 6 > 2. So we have to put index 3 to the index 4.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	6	7	11

Step 6 – d)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	6	7	11

Now key = 2, j = 2. Again in while loop, j >= 0 and 5 > 2. So we have to put index 2 to the index 3.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	5	6	7	11

Step 6 – e)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	6	7	11

Now key = 2, j = 1. Again in while loop, j >= 0 and 5 > 2. So we have to put index 1 to the index 2.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	5	6	7	11

Step 6 – f)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	6	7	11

Now key = 2, j = 0. Again in while loop, j >= 0 and 3 > 2. So we have to put index 0 to the index 1.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	3	5	5	6	7	11

Step 6 – g)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	5	5	6	6	7	11

Again in while loop, j is not greater or equal than 0 because j = -1. So loop finished and index 0 must be the key which is 2.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
2	3	5	5	6	7	11

Step 7)

In the for loop, i = 7 and it is not less than size which is 7. So sorting finished and the last array is below.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
2	3	5	5	6	7	11

Emirhan UZUN

171044019

Homework 2 Question 1

Emirhan Uzun
171044019

CSE - 321 INTRODUCTION TO ALGORITHMS
HOMEWORK #2

1) The answer of first question was answered above.

2-a) When the n is equal to 1, the first if statement executes and function will finish. And this if has just 1 statement which takes constant time. So the best case $B(n) = \Theta(1)$

When the n is bigger than 1, loops start to execute. Outer loop will be executing n times but inner loop executes 1 time for every step. Because it executes, prints "*" and then breaks. That means print one time star and exit the loop. So worst case will be $(n * 1) + 1$. n means first loop execution times, *1 means for inner loop execute time for every turn, +1 for checks the if statement.

Worst case $w(n) = \Theta(n*1+1) = \Theta(n)$

For average case, assume that the n will be positive integers. And the probability of $n=1$ is very low. Because n can be from 1 to very big numbers. So first if statement executes in very low probabilities. Therefore I think average case is equal to worst case.

$$A(n) \cong w(n) = \Theta(n)$$

2-b)

```
void function (int n){
    int count = 0
①   for (int i = n/3 ; i<=n ; ++i)
②       for (int j = 1 ; j + n/3 <= n ; ++j)
③           for (int k = 1 ; k<=n ; k = k*3)
                count++ ; }
```

The 1^{st} loop runs $2n/3$ times and that would be n times as n grows.

The 2^{nd} loop runs approximately $2n/3$ times (For example if n is 600, loop runs 400 times, if $n=1200$ loop runs 900 times etc.) and that would be also n times.

The 3^{rd} loop runs like that: $1, 3^1, 3^2, \ldots 3^k = n$ so k would be $\log_3 n$.

This function's worst, best and average case are equal each other I think. Because all of them just depend on n and it has not condition etc. So the complexity will be $n.n.\log_3 n$ which is $\Theta(n^2 \log_3 n)$

Emirhan UZUN
171044019

3- Pseudocode

```
procedure findPairs (L[1:n], desiredNumber)
    mergeSort (L) (or L.sort() it doesn't matter because sort time comp is O(nlgn))
    lowIndex = 0
    highIndex = lengthOfList - 1
    while (low < high)
        if L[lowIndex] * L[highIndex] is equal to desiredNumber
            print ("({}, {})".format (L[lowIndex], L[highIndex])
        endif
        if L[lowIndex] * L[highIndex] is less than desiredNumber
            low = low + 1
        endif
        else   high = high - 1
    endwhile
end
```

In question, you would like to provide an algorithm with $O(n\log(n))$. In my algorithm, mergeSort time complexity is $n.\log(n)$ for all cases which are best, average and worst. The while loop turns n time maximum. In while loop, the statements take constant time. So the time complexity of whole program is $\max(O(n), O(n\log n)) = O(n\log(n))$.

Emirhan UZUN
171044019

4- Now we have 2 Binary Search Tree with n nodes. Let's say these names are T1 and T2. If we want to put T1 nodes to T_2, then best case will be like this;

Best Case: If the height of T2 tree is logn, then insert an element to this tree is also logn. Because we eliminate half of nodes for every step. And T1 has n nodes. So we will do the insert operation to each node, we will apply $n\log n$ steps. So the time complexity of best case is $B(n) = \Theta(n\log n)$

Worst Case: If the height of T_2 tree is n, then insert an element to this tree is also n. Because we eliminate one by one of nodes for every step. On the other hand T1 has n nodes. So we will do insert operation to each node, we will apply n^2 steps. So the time complexity of worst case is $W(n) = \Theta(n^2)$

Emirhan UZUN
171044019

5) It is linear time complexity with hashing. We put first and second arrays into two sets. Let's say bigger array size n and smaller array size is m. Therefore n>m. After put operations, we check the elements of second array exist the bigger array. Let's look my pseudocode.

```
procedure findCommonElements (LA[1:n], LB[1:n])
    setA = set()
    setB = set()

    for i in range (n)                  → O(n) times
        setA.add (LA[i])
    endfor
    for i in range(m)                   → O(m) times
        setB.add (LB[i])
    endfor.

    if ( len (LA) < len(LB))            (I wrote above that assume n>m
        for x in setA                    but maybe m>n in actual code.
            if x in setB                 So I wrote this condition)
                print (x)               → O(min((len (LA)), (len(LB)))
    endif
    else
        for x in setB
            if x in setA                → O(min((len (LA), (len(LB)))
                print(x)
    endelse
end
```

→ The first 2 loops time complexity is O(n) and O(m) in order. After that if statement takes smaller length times. Because it turns length times and the other statements are constant times. So the time complexity is max (O(n), O(m)) = O(n) (we assume n>m). If m>n, that means O(m)