2. HW

1) 6, 5, 3, 11, 7, 5, 2

Insertion Sort

6 5 3 11 7 5 2

Compare first and second element each other, sort increasingly

5 6 3 11 7 5 2

Compare 3 and 6, 3 is smaller than 6 change place

5 3 6 11 7 5 2

Compare 3 and 5, 3 is smaller change their positions

3 5 6 11 7 5 2

Compare 6, 11, 11 greater than 6 no changing. The other elements also smaller than 6, so not changing them

3 5 6 \ 11 7 5 2

Compare 11 and 7, 7 smaller one change its position

3 5 6 7 11 5 2

Compare 6 and 7, 7 bigger, no need to compare other elements

3 5 6 7 11 5 2

Compare 11 and 5, 5 is smaller change positions

3 5 6 7 5 11 2

Compare 7 and 5, 5 is smaller, change positions

3 5 6 5 7 11 2

Compare 6 and 5, 5 is smaller, change positions

3 5 5 6 7 11 2

Compare 5 and 5, they are equal, no need to change positions.

3 5 5 6 7 11 2

Compare 11 and 2 , 2 is smaller then 11 change positions

3 5 5 6 7 2 11

Compare 7 and 2 , 2 is smaller then 2 , change positions.

3 5 5 6 2 7 11

Compare 2 and 6 , 2 is smaller then 2 , change positions

3 5 5 2 6 7 11

Compare 5 and 2, 2 is smaller then 2, change positions.

3 5 2 5 6 7 11

Compare 5 and 2 , 2 is smaller then 2, change positions.

3 2 5 5 6 7 11

Compare 3 and 2, 2 is smaller then 3, change positions

2 3 5 5 6 7 11

3)

{1, 2, 3, 6, 5, 4}

find (1,1), (2,3).

Merge sort time complexity is O(n. logn)

size = len (array)

for i to n do{

searching_one = array[i]

mod = target % searching_one

division = target / x

if mod = 0

pair = Binary Search (array, i+1, n-1, division)

}

}

Time complexity O(n. logn) + O(n) * O(logn)

= O(n.logn) worst case

4)

```
merged_trees(tree1, tree2):
    convert tree1 and tree2 to ordered lists.
    //create new tree
    tree3 = new tree()
        i=0
        j=0
        while (i<m && j<n)
            if (tree1[i] < tree2[j]):
                tree3.Add(tree1[i])
                i++
            else:
                tree3.Add(list2[i])
                j++
        while (j<m)
            tree3.Add(tree1[i])
            i++
        while (j<n)
            tree3.Add(tree2[j])
            j++
        return list3
```

Let say tree1's height is n, so it has 2^n-1 nodes. Tree2's height is m, so it has 2^m-1 nodes.

First of all convert tree's to ordered lists. $O(n)$ and $O(m)$ times.

Then merging that lists $O(m+n)$ times. That's why worst case of the program $O(n)+O(m)+O(m+n) \Rightarrow O(m+n)$

5) Finding small array element in big array

Can be use hash table

s = [1, 2, 3, 4] = larger (m) , [1, 5] = smaller one (n)

```
find_elements(larger_array[], smaller_array[]){
    Hash_table = new HashTable(larger_array[])
    for i to len(smaller_array) do
        eg = smaller_array[i].hashCode()
        if hash_table.get(eg) == ""
            return false
        else
            return true
```

→ Creating hash table is size of array which m $O(m)$.

In code we have for loop, which runs n times

$O(m) + O(n) = O(m+n)$, also worst case same (

2) Time Complexity?

a)

inner loop runs 1 times, because of the break statements.

Outer loop runs n times.

So, in total code runs $n \times 1 = n$ times. Time complexity of the code is $O(n)$

b) for ($i = n/3$; $i \leq n$; $i{+}{+}$)

for ($j = 1$; $\frac{j+n}{3} \leq n$; $j{+}{+}$)

for ($k=1$, $k <= n$, $k = k*3$)

works $\log_3 n$ → because of increasing rate

$\frac{j+n}{3} \leq n \Rightarrow j+n \leq 3n \Rightarrow j \leq 2n$

so new loop

$j=1$; $j \leq 2n$; $j{+}{+}$

→ It runs $2n$ times

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

In total $O(\frac{2n}{3})$ $O(2n)$ $O(\log_3 n)$

$O(n \; n \; \log_3 n) = O(n^2 \log_3 n)$