Assignment - 2

1) a) $T(n) = 2T(n/3) + 1$

from Masters Theorem; we can say $a = 2, b = 3$

$c = \log_3^2$ $\quad d = 0$

So here $c > d$ which we can say

$$T(n) = n^{\log_3^2}$$

b) $T(n) = 7T(n/7) + n$

from Master's theorem we can deduce $a = 7, b = 7$

$c = \log_7^7 = 1, \quad d = 1$

So here $c = d$, which concludes

$$T(n) = n^1 \times \lg n = n \lg n$$

c) $T(n) = T(n-1) + 2$

Using iteration Method to deduce -

$T(n-1) = T(n-1-1) + 2$

$= T(n-2) + 2$

$T(n) = T(n-2) + 2 + 2 = T(n-2) + 4$

$T(n-2) = T(n-3) + 2$

$T(n) = T(n-3) + 2 + 2 + 2 = T(n-3) + 6$

Now for i^{th} iteration it will be

$$T(n) = T(n-i) + 2i \longrightarrow (1)$$

Now let's say $n - i = 1$

$$i = n - 1$$

Substitute i value in (1)

$$T(n) = T(n - (n-1)) + 2(n-1)$$
$$= T(1) + 2n - 2$$
$$= 1 + 2n - 2$$
$$= 2n - 1$$

$$\Rightarrow \Theta(n)$$

Therefore $T(n) = \Theta(n)$

(2)

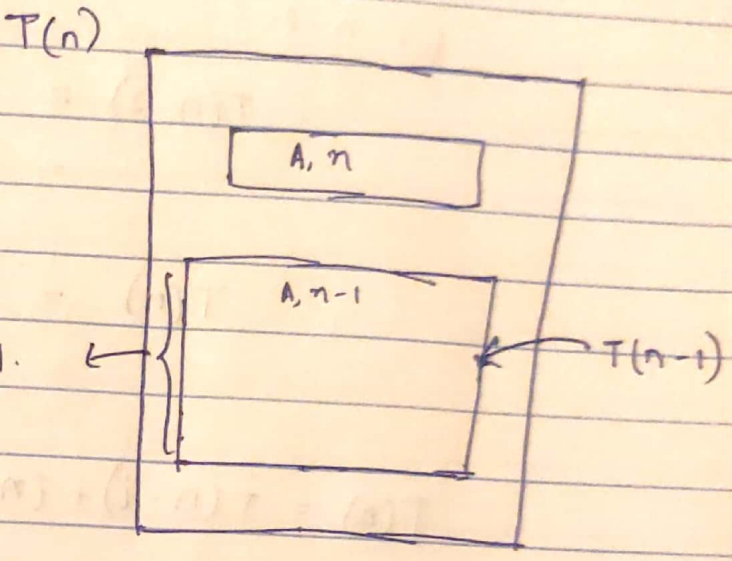
Pseudo code for Recursive Insertion Sort.

```
function insertionSort (A, n-1)
        Insert (A, n).

last = A[n]
j = n-1
while (j > 0 and A[j] > last)
        A[j+1] = A[j]
        j -= 1
A[j+1] = last
```

Recurrence Relation:

let's consider T(n) be the worst case running time of an algorithm, the base case for the Algorithm is when the array is null or has only one element. And also for each recursion there will be n-1 iterations which will give T(n-1) and also for each insertion iteration will can a constant time of n-1 iterations.



Therefore we can say that the recursive relation can be.

$$T(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ T(n-1) + n-1 & \text{if } n > 1 \end{cases}$$

Now we can use iteration Method to figure out the running time complexity.

$$T(n) = T(n-1) + n-1$$

for $n-1$

$$T(n-1) = T(n-1-1) + n-1-1$$
$$= T(n-2) + n-2$$

$$T(n) = T(n-2) + n-2 + n-1$$
$$= T(n-2) + 2n-3$$

for $n-2$

$$T(n-2) = T(n-2-1) + n-2-1$$
$$= T(n-3) + n-3$$

$$T(n) = T(n-3) + n-3 + 2n-3$$
$$= T(n-3) + 3n-6$$

$$T(n) = T(n-i) + (n-i+1) + (n-i+2) + \cdots (n-1)$$

for $n-i = 1$

$$T(n) = T(1) + (2+3+\cdots+n-1)$$
$$= \frac{n(n-1)}{2}$$
$$= (n^2-n) \times \frac{1}{2}$$

$$T(n) = \textcircled{H} n^2$$

③ let's Consider a set of points $P = P_1, P_2, P_3 \cdots P_n$

~~Initialize~~

for each point i in P

Initialize slopes[]

for all points j in P except P:

Calculate slopes of i and j and the add it to slopes array

(In Total n-1 slopes)

Sort these n slopes ($n \log n$)

Analysis

The Algorithm runs for n points which is $O(n)$

for each point calculate slope with all points which is $O(n) * O(n) = O(n^2)$

using Sort for n points which is $O(n \lg n) * O(n)$

$= O(n^2 \lg n)$

for comparison to check for the same slopes it will take $O(n) * O(n) = O(n^2)$

So, total time $= O(n^2) + O(n^2 \log n) + O(n^2)$

$= \underline{O(n^2 \log n)}$

④ Algorithm

n, x → input integers

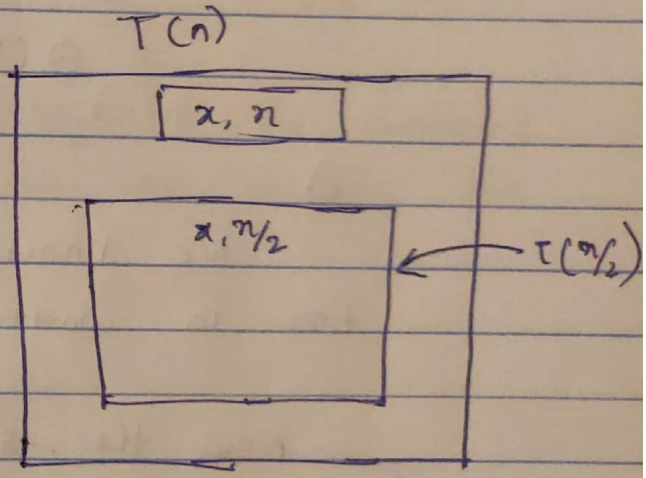
```
Power (x, n)

    if n == 0    return 1
    else
        K ← Power (x, n/2)
        if n%2 == 0   return K*K
        else return x * K * K
```

Run time Analysis

let T(n) be the worst case running time

we can say all the comparison operation can be done in a constant time which can be O(n).



And for every iteration T(n) will become $T(n/2)$.

So,

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n/2) + O(1) \end{cases}$$

$$T(n) = T(n/2) + O(1)$$

Applying Master's Theorem.

$$a = 1,\ b = 2,\ d = 0,\ c = \log_2 1 = 0$$

so, $c = d$ therefore $T(n) = n^{(0)} \times \lg n$

$= \underline{\lg n}$

Hence $\underline{T(n) = \lg n}$

(5)

a)

we know the standard running time for the insertion sort is $\Theta(n^2)$

for K element the running time will be $\Theta(K^2)$

To sort n/K sublists, each of size K it takes $\Theta(K^2) \times \frac{n}{K}$

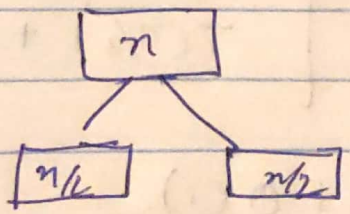
$= \Theta(nK)$

b)

we know that the Merging of can take $\Theta(n)$ time in worst case

Now the sublists we have are n/K

let's consider there are n nodes



So,

The time complexity for Merging is levels is $\Theta(\log n)$

Hence for n/K it will be $\Theta(\log \frac{n}{K})$

Therefore the total running time for Merging the sublists will be

$$O(n \log(n/k))$$

c)

The Modified (running) Algorithm running time is $O(nk + n \lg(n/k))$

The Standard running time of the Algorithm is $n \lg n$

So, let's consider $k = \log n$.

we can get

$$O(nk + n \lg(n/k)) = O(n \log n + n \lg(n/\log n))$$

from this we can say that $n \log n$ is the dominant term of the equation, Hence we can conclude the running time

$$O(n \log n)$$

(6)

a) let's assume that the bottom left most point is the Ghostbuster.

Sort all the remaining points from this point which are present based on Grahams Algorithm

let's keep track of the original count of the Ghosts and Ghostbusters. So that when we visit the points we can also keep track of the points visited

so that when we get the same number of ghosts and busters we can draw a line. Thus the algorithm we take $O(n \log n)$ time.

b)

For the given points let's follow the process as Mentioned above, By doing this we can get same number of ghosts and ghostbusters on the same side of a line. Now for checking the streams we have to do the process recursively on each ~~size~~ side of the points.

So, we can say that there will be $n/2$ iterations to find the pairs

which is we have to sort for $n/2$ iterations As we know the sorting will take $n \log n$ time

$$T(n) = (n \log n) \times \frac{n}{2}$$

$$= \frac{n^2}{2} \log n$$

So, we can say that the time Complexity can be $O(n^2 \log n)$