ASSIGNMENT-2

1

Name:- NIHAR MUNIRAJU.
ID :- 2072857
email :- nmuniraj@depaul.edu.

1) Masters Theorem is a methord for solving recurrence relations of the form :- $T(n) = aT(n/b) + f(n)$.
where n = size of input; a = number of subproblems in the recursion; n/b = size of each subproblem. All the subproblems are assumed to have the same size; $f(n)$ = cost of the work done outside the recersive call, which includes the cost of dividing the problems & cost of merging the solns.

$\Rightarrow$ Here $a \geq 1$ & $b \geq 1$ are constants & $f(n)$ is asymptotically posiite fn.
Then the time complexibity of a recursive relation is given by :-

(1) If $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$

(2) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} * \log n)$.

(3) If $f(n) = \Omega(n^{\log_b a + \epsilon})$, then $T(n) = \Theta(f(n))$.

where $\epsilon > 0$ is a constant.

(1) Solution $T(n) = 2T(n/3) + 1$.

Here $a = 2$

$\Rightarrow \frac{n}{b} = n/3$

$f(n) = 1$.

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

$\Rightarrow$ Since $O(1) = 1 = O(n^0)$;

& $\log_3 2 = 0.6309$.

$\Rightarrow T(n) \approx O(n^{0.6309})$.

(2) Solution :- $T(n) = 7T(n/7)+n$.

Here, $a=b=7$

$\Rightarrow f(n) = n$.

Then, $T(n) = n^{\log_7 7}$

From the properties w.k.T n'

$\therefore f(n) = n = n$.

$\Rightarrow T(n) = n . \log_7^n \quad \therefore \Rightarrow O(n \log n)$.

(3) Solution :- $T(n) = T(n-1)+2$.

As Given from the Question $T(n) = O(1)$ & $n = O(1)$.

$\Rightarrow T(n) = T(n-1)+2$.

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

$= T(n-3)+4)$

From this we get the form $T(n-x)+2x$.

put $x = n-1$.

$\Rightarrow T(n-n+1)+2(n-1)$.

$\Rightarrow T(1)+2n-2$.

$\Rightarrow 1+2n-2$.

$T(n) = O(n)$.

Q2:-

Pseudo Code:-

```
void rec_insertionsort (int arr[], int n){
// First Base case to be Implemented.
  if (n <= 1)
      return
  // Let's sort all first n-1 elements.
  rec_insertionsort (arr, n-1).
   int val = arr[n-1]
   int pos = n-2.
  while (pos >= 0 && arr[pos] > val){.
   arr[pos+1] = arr[pos].
        pos = pos-1.
      }.
    // Insert the Last element in its
     correct Sorted Array.
    arr[pos+1] = val
    }.
```

⇒ The Runing Time Complexity for this recuresive Insertion Sort is $O(n^2)$.

Q3 ④

⟹ For More than 3 points to be Collinear they have to be on the surface of the points defined by other two points.

⟹ As we know all ~~pai~~ pairs of two points joining these two & other points which lie on the same surface.

Uses $O(N^3)$ time complexity of Algorithm.

Algorithm Pseudo Code:—

```
all_collinear_points = empty set.
for point X1 in all points:
  for point X2 in all points besides X1:
      N = unique line joining X1 & X2
      M = Empty Set.
     for point X in all points:
   if X lies on N:
        add X to M
   if size of M is >=3:
        add M to all_collinear_points.
 print all_collinear_points.
```

⟹ ∴ Lets use this way of an Algorithm to get $O(n^2 \log n)$

as X compute the slope joining X & X0. And the List N-1 slopes.

⟹ If all these points have the same slope of adjoining in the sorted list. Then the same slope X0 in fact is Collinear.

⟹ If we redo the same with other points also we get the points to be colinear in $O(n^2 \lg n)$.

Q4) An Algorithm that takes as Input a positive Integer n and a number x, and computes x^n by performing $O(\lg n)$ multiplication.

Soln Algorithm:-

```
// Input n and x
int power (int n, unsigned int x)
{
int temp;
if (n==0)
    return 1;
temp = power (x, n/2);
if (n%2 ==0)
    return temp* temp;
else
    return x *temp* temp;
}.
```

⇒ The Time Complexity for performing requires $O(\lg n)$.

1. Insertion Sorting Sublists:-

⇒ The Sorting each list takes $ak^2 + bk + c$ for some constants a, b and c. We have n/k Sublists for each list.

So, $\frac{n}{k}(ak^2+bk+c)$.

$= ank + bn + \frac{cn}{k} = n/k \cdot \theta(k^2)$.

$= \theta(nk)$.

2. Merging Sublists:-

⇒ Sorting a sublists of length k then,

$$T(a) = \begin{cases} 0 & \text{if } a=1 \\ 2T(a/2)+ak & \text{if } a=2^p, \text{ if } p>0 \end{cases}$$

⇒ Since Merging one Sublist is trivival & merging a sublists & splitting them into two groups of a/2 lists and recursisively combining the two resuts we get ak steps, Since have two arrays of each length $\frac{a}{2}k$.

Now we substitue n/k for a: we get

$T(n/k) = \frac{n}{k} k \lg \frac{n}{k}$

$= n\lg(n/k)$.

This is exactly when n/k is a power of 2, the overall Time complexity is $\theta(n \lg(n/k))$

3. The Largest Value of K :-

⇒ If we substitute $k = \lg n$.

$$\Rightarrow \Theta\left(n\lg n + n\lg \frac{n}{\lg n}\right)$$

$$\Rightarrow \Theta(n\lg n).$$

⇒ If $K = f(n) > \lg(n)$, the complexity of time will be $\Theta(nf(n))$. which is larger running time than of the merge sort.

Q6

⇒ We will have to find the surface of points to where the ghosts can move and use ~~largest~~ Quicksort, like a divide and conqueror method of sorting these ghosts by Ghostbusters.

⇒ The Ghostbusters have to use the points from the farthest points to the closest points and aligning them on a straight line of points to use Quicksort.

⇒ The Ghostbusters will have to choose a midpoint where the farthest and closest point needs to be in sync in every angle of the surface they move in a straightline.

⇒ We sort the points using the midpoint everytime the ghost moves and use recurrisive calls before and after the movement of the ghost to get them on a straight line.

⇒ As we halt and Eliminate by divide and Conqueor the average time complexibility will be O(nlgn time) and also doing this way we can find the ghosts in a single stream and would be easily possible to eradicate them.

(b) ⇒ Running Dijkstras algorithm on a dense graph using binary Heap.

⇒ By using Dijkstras algorithm on a graph of surface with n nodes and m edges, using a Binary Heap we get $O(m \log n)$.

⇒ A dense graph of Surface is one where $m = \theta(n^2)$. So, Dijksthas Algorithm would take time $O(n^2 \lg n)$ in this case to close no stream cross while eradicating the ghosts.

⇒ Similarly, we can use Prims Algorithm on a Binary Heap Sort