SRI MANVITH
VADDEBOYINA
1231409457

CSCI 570 HW 5

1.

a) $T(n) = 4T(n/2) + n^2 \log n$

$a = 4$

$b = 2$

$f(n) = n^2 \log n$

$n^{\log_b^a} = n^{\log_2^4} = n^2$

Case 2: $T(n) = \Theta(n^2 \log^2 n)$

b) $T(n) = 8T(n/6) + n \log n$

$a = 8$

$b = 6$

$f(n) = n \log n$

$n^{\log_b^a} = n^{\log_6^8}$

Case 1: $T(n) = \Theta(n^{\log_6^8})$

c) $T(n) = \sqrt{6000}\, T(n/2) + n^{\sqrt{6000}}$

$a = \sqrt{6000}$

$b = 2$

$f(n) = n^{\sqrt{6000}}$

$n^{\log_b^a} = n^{\log_2^{\sqrt{6000}}}$

Case 3: $T(n) = \Theta(n^{\sqrt{6000}})$

d) $T(n) = 10T(n/2) + 2^n$

$a = 10$ $\quad f(n) = 2^n$

$b = 2$

$n^{\log_b^a} = n^{\log_2^{10}}$

Case 3: $T(n) = \Theta(2^n)$ as exponential grows faster than polynomial for large 'n'

e) $T(n) = 2T(\sqrt{n}) + \log_2^n$.

let us take $n = 2^k$

$T(2^k) = 2T(2^{k/2}) + k$

$\downarrow$

$P(k) = 2P(k/2) + k$ — (1)

Solving this equation (1)

$k^{\log_b^a} = k^{\log_2^2} = k^1$

$f(k) = k$

Case 2: $P(k) = \Theta(k \log k)$

Replacing k with $\log_2^n$

$T(n) = \Theta(\log_2^n \log \log_2^n)$

2. Our goal is to find the person with more than $n/2$ cards if any.

We can solve this problem using Divide and Conquer approaches.

We initially divide the given set of 'n' cards into two halves of equal size. We then try to find if there is any person with more than $n/2$ cards recursively. If we find a person with the above condition, we return that card and person.

1. After the sub problems are recursively solved, we then merge them into one to find the person in the whole set of cards.

2. If in the each subproblem that is divided into half, there is no person as per the condition, then the set of cards does not contain any person as per the requirement.

3. If both of the halves contain same person, then we can choose any of the two cards and return the representative or the person with more than $n/2$ cards.

4. If only one of the half contains a person satisfying the condition, then we need to check if the person is found in the whole set.

5. If each of the sets return a different person, then we need to check the person satisfying the condition on the whole set.
This we can do by comparing the person's card by traversing the whole set and checking the count of same person.

$$T.C = T(n) \leq 2T(n/2) + O(n)$$

$2T(n/2)$ → 2 sub problems

$O(n)$ → traversing in last steps 4 & 5.

$a=2, b=2, f(n)=n. \Rightarrow n^{\log_2^2} = f(n) \Rightarrow$ Case 2

$T(n) = \theta(n \log n)$

3. Given 'n' lines $L_1, L_2, \ldots L_n$.

Sort the arrays using slopes (increasing slopes).

→ Divide the set of lines into two sets with equal sizes (based on median slope). Solve for the subproblems recursively and combine.

$$L = \{L_1, L_2, \ldots L_n\}$$

$$L_l = \{L_1, L_2, \ldots L_{n/2}\} \qquad L_r = \{L_{n/2+1}, \ldots L_n\}$$

Also lets compute the intersection points of consecutive lines in set.

$L_l = \{L_1, L_2, L_3 \ldots L_{n/2}\}$ $\quad$ $L_r = \{L_{n/2+1}, \ldots L_n\}$

$A = \{a_1, a_2, \ldots a_{n/2}\}$ $\quad$ $B = \{b_1, b_2, \ldots b_{n/2}\}$

The x- coordinates of $a_1, a_2, \ldots a_{n/2}$ will be increasing order, because the lines with larger slope will be visible later than the lines with smaller slope.

Combine :

$L_l = \{L_1, L_2, \ldots L_{n/2}\}$ $\quad$ $L_r = \{L_{n/2+1}, L_{n/2+2} \ldots L_n\}$

L_1, L_2 visible earlier; $L_{n/2}$ visible later; $L_{n/2+1}$ visible earlier; $L_{n/2+2} \ldots L_n$ visible later

We will find first line in L_l which is below a particular line in L_r

Using two pointer approach

```
i = 1  j = 1
while (i < n/2 & j < n/2)
    if  L_i slope > L_j slope
            return i & j
    else  L_i slope < L_j slope
            i++
```

Then the combined list would be

$\{L_1, L_2, \ldots L_i\} \cup \{L_j, L_{j+1} \ldots L_n\}$

Recurrence relation:

$$T(n) = 2T(n/2) + \Theta(n) \rightarrow \text{combine step}$$

$a = 2, \quad b = 2 \quad n^{\log_2^2} = n \quad f(n) = \Theta(n)$

Case (ii) $f(n) = \Theta(n) = \Theta(n^{\log_2^2})$

$$T(n) = n \log n.$$

4. Compute x^a in $O(n)$ time.

We can solve this problem using Divide and Conquer approach.

Any x^a can be written as a product of two if the value of 'a' is even as we can divide into '2' halves and if 'a' is odd we can write as a product of 3 i.e 2 subproblems and a value 'x'.

$$x^a = \begin{cases} x^{a/2} * x^{a/2} & a \equiv \text{even} \\ x^{a/2} * x^{a/2} * x & a \equiv \text{odd.} \end{cases}$$

From above, we can say that the black box is called atmost 3 times to Compute the value of x^a.

$$T(n) \leq T(n-1) + 3$$

↓ sub problem of size lesser than 'a' value 1.

↳ 3 calls atmost

Solving $T(n)$,

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

~~$T(n-k) \leq T(n-k) + 3$.~~

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

$T(n) = T(n-3) + 9$

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

Substitute $k = n$.

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

$$T(n) = T(0) + 3n.$$

$$T(n) = n +$$

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

5. J-similar(a,b) = (i) a=b

(ii) J-similar (a_1, b_1) & J-similar (a_2, b_2)

(or)

J-similar (a_1, b_2) & J-similar (a_2, b_1)

Prove that only strings of equal length will be J-similar

Case (i) a length is even, b length is odd, then $a \neq b$ and second case also does not hold because for odd lengths it is not valid.

Case (ii) a length is even, b length is even and (a.length ≠ b.length)

then

a) $a \neq b$ [both are not equal]

b) a_1 a_2 a_1.length = a.length/2

b_1 b_2 b_1.length = b.length/2

for (a_1, b_1) to be J-similar then either $a_1 = b_1$ [this is not true because both are of different lengths]

(or)

$a_1 = b_2$ [this is not true because both are of different lengths]

J-similar of two strings is valid if both are of equal length.

Algorithm

For a particular string $a(a_1 a_2)$ is J-similar with $a'(a_2 a_1)$

We compute lexicographical smallest string a which is J-similar to a,

function to find lexicographically small string

```
func(a)
    if a.length is odd : return a
    a1 = a(0, a/2); a2 = a(a/2, a)
    a1' = fun(a) ; a2' = fun(a)
    if a1' > a2'
        return a2' + a1'
    else
        return a1' + a2'
```

for a → Compute func(a) if func(a) is similar to
b → Compute func(b) func(b)

⇓

a is J-similar to b

Recurrence relation for function

$T(n) = 2T(n/2) + O(n)$ → Concatenating string

$a=2, b=2 \Rightarrow n^{\log_2^2} = n$

Case (ii) $T(n) = \theta(n \log n)$

Total T.C = $O(n \log n) + O(n)$ → Compare.

6. Given, the array is sorted and we need to find an index i, such that arr[i] is equal to i.

a) FIXED-POINT (array, l, h):

```
WHILE l < h:
    MID = (l+h)/2
    IF array[mid] == mid:
        RETURN mid, array[mid]
    ENDIF
    ELSE IF array[mid] < mid
        FIXED-POINT(array, mid+1, h)
    ENDELSEIF
    ELSE
        FIXED-POINT (array, l, mid-1)
    ENDELSE
ENDWHILE
RETURN -1
```

T.C = $O(\log n)$ ⇒ similar to Binary search

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

$a = 1$

$b = 2$

$f(n) = 1$

$n^{\log_b^a} = n^0 = 1 = f(n)$

Case 2; $T(n) = \Theta(n^0 \log n)$

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

c) After we find the FIXED POINT P, as the array is sorted, now our goal is find if the point P is unique in O(1) time.

Let us say we found point P at index k, then we check the values of k-1 and k+1 and determine if point P is unique. If it is unique then we can say that we identified unique fixed point as all elements to left will be less and all elements to right will be more than the value of k.

i.e. $i < j$ array[j] < j ⇒ left
$j > i$ array[j] > j ⇒ right

=

This will take O(1) time.