Meet Modi 2019435

1

DATE: / /
PAGE NO.:

ADA Mid-SEM

Ques 1.
Solution a) Psudo code
Let two numbers be a, b each of them with n digits

```
def Mul(a,b)
    if (a,b is 1 digit)
        return a*b;
    else:
        a = x*10^(n/2) + y      (here x is the first half of a
                                 and y is second half of a)
        b = z*10^(n/2) + w      (w is the second half b
                                 and z is first half of b)
    val = Mul(x, z)
    val1 = Mul(y, w)
    val2 = Mul(x-y, z-w)
    ans = val x 10^n + 10^(n/2)(val+val1 - val2) + val1

    return ans
```

Brief Description
In this algorithm we divide the ~~string~~ numbers in two halves here [(x,y), (z,w) and then Recursively call the Algorithm.

$(x\times 10^{n/2}+y)*(z\times 10^{n/2}+w) = xz\times 10^{n}+10^{n/2}(xw+yz) + yw$

we can get xz from val

" " " yw from val 1

for val 2 $(x-y)\times(z-w) = xz + yw - xw - yz$

$(xw+yz) = xz + yw - (x-y)(z-w)$

$= val + val1 - val2$

This can be replaced in the equation and hence the answer

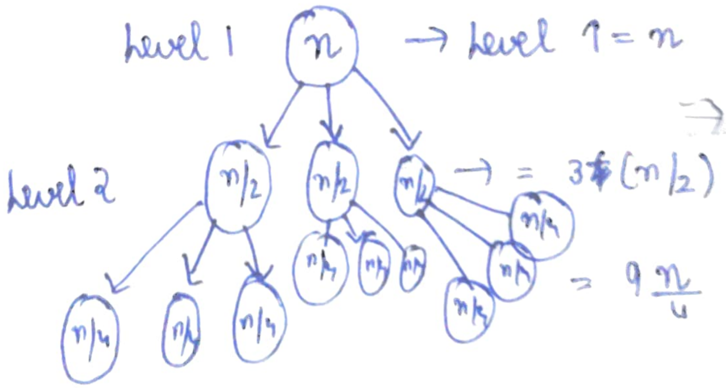
b) Recurrence Relation - $3T(n/2) + O(n)$

because the algorithm is recursively called on (x, z), (y, w), $(x-y)$, $(z-w)$, and each of $x, z \dots$ have $n/2$ bits and hence recursion is called for $n/2$ bits for three times. so $3T(n/2)$. $O(n)$ because the division of numbers into two halves each with $n/2$ bits takes $O(n)$ and Multiplication of val $\times 10^n$ also takes $O(n)$ as it is equal to shifting n bits hence $O(n)$ time

$$T(n) = 3T(n/2) + O(n)$$

c) Solving Recurrence

$$T(n) = 3T(n/2) + O(n)$$



As it can be observed that work increases at each level and hence the most significant work is done at all the leaves

Total work leaves = c * no. of leaves

no of leaves = $a^{\log_b n}$

$= 3^{\log_2 n}$

Total work = $c \times 3^{\log_2 n} = c\, n^{\log_2 3}$ (logarithmic property)

$T(n) = O(n^{\log_2 3})$ Ans.

Meet Modi
2019435

Rough

$\frac{9n}{25}$ $\frac{12n}{25}$ $\frac{12n}{25}$ $\frac{16n}{25}$

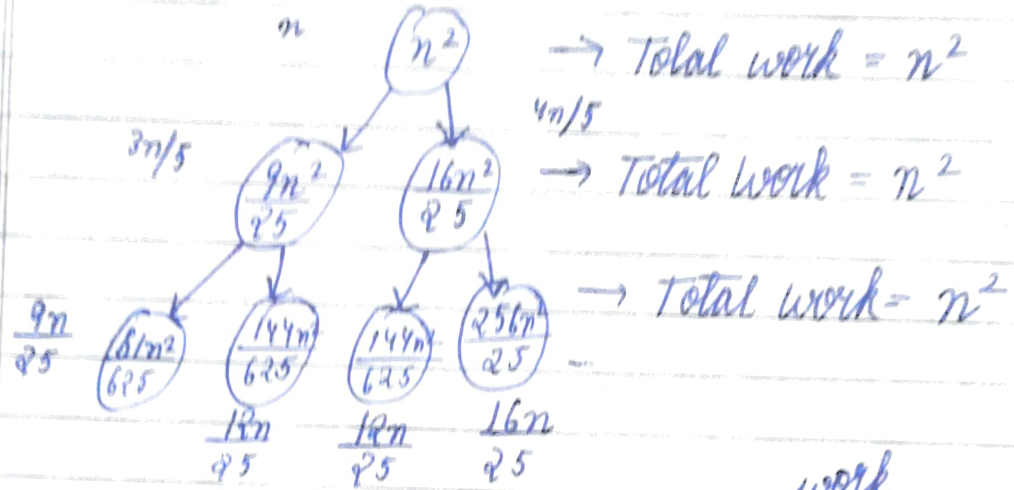
$\begin{array}{r} 81 \\ 144 \\ \hline 225 \\ 144 \\ \hline 369 \end{array}$ $\begin{array}{r} 369 \\ 256 \\ \hline 625 \end{array}$ $\begin{array}{r} 16 \\ 16 \\ \hline 96 \\ 16\times \\ \hline 256 \end{array}$

③

Ques 2.
Solution

$T(n) = T(3n/5) + T(4n/5) + n^2 \quad T(1) = 1$

Reccurrence tree



→ As it can be observed that the ~~level~~ work at each level remains constant hence the total work is n^2 X number of levels

number of levels = $(\log_b n + 1)$
here there can two b's $\frac{5}{3}$, $\frac{5}{4}$ but as n is same for both of them the levels for $\frac{5}{4}$ will be greater than $\frac{5}{3}$ as $\frac{5}{3} > \frac{5}{4}$ so n will reach 1 later for this value

levels = $(\log_{5/4}(n) + 1)$

$T(n) = n^2 * (\log_{5/4}(n) + 1)$

$T(n) = O(n^2 * \log_{5/4}(n))$

The tightest possible asymptotic behaviour of $T(n) =$
$= n^2 \log n$

Ques 3.
Solution n take home assignments
$d_h = [1 \ldots\ldots n] \rightarrow$ deadline for each$_i$ assignment
$P_h = [1, \ldots\ldots n] \rightarrow$ number of days for each assignment
Suggested algorithm – non-decreasing order of deadline and then work on them.

Counter Example.
Suppose the array for deadline is $= [10, 11, 12]$
Corresponding days $= [9, 3, 5]$

here the deadline is sorted already in the non-decreasing order $\rightarrow$ so the student will work on deadline 1 with $d_h = 10$ and $P_h = 9$ so he has already spent ~~on~~ 9 days on this deadline. for the 11 Assignment deadline 3 days is required but $9 + 3 = 12 \geq 11$ hence he cannot complete this deadline further for 12^{th} deadline, he/she requires 5 days and hence $9 + 5 = 14 > 12$ and hence ~~the~~ he/she cannot complete this deadline also

Meet Modi 2019435

→ but in this case optimally he can complete 2 assignments as if he starts from 2^{nd} assignment with 11days deadline he will only take 3 days and for 12^{th} he will have 5 days
$3+5 = 8 < 12$ hence he can complete that also
So he can complete at most two deadlines ~~Optimal Solution~~ and hence $1 < 2$ and
2 is optimal Solution

Ques 4.
Solution → Given n balls in a row with value v_i
→ To do → pick maximum subset of ball so that 2 two are consecutive
for $K = 2$, this means that no two consecutive balls can be selected

2 2 3 2 2

Subproblem (i)
opt(i) → Max subset that can be taken from the subarray $(i \to n)$

Recurrence
opt(i) = { Max (opt$(i+1)$, opt$(i+2)$ + value i) }
Base Case
if $(i > n)$ → (length of array)
return 0

In this algorithm, there can be two possible cases ⑥

1) when the ball is Selected :- then the value of ball is added and then we jump to Index i+2 due to the Constraint and hence

2) when the ball is not Selected → the optimum Solution is index+1 and score is not added.

Proof of Correctness of Recurrence

~~If~~ let us assume that opt(i) is optimal Solution for the Subproblem

Case 1

when ball is not Selected

$opt(i) = opt(i+1)$

Claim - opt(i+1) is the optimal Solution for opt(i) Subproblem

let us assume that claim is not true hence the Solution opt(i+1) is not optimal and there exists another Solution $*opt(i+1) > opt(i+1)$. In this Case $*opt(i+1) + 0 > opt(i+1) + 0$. also $opt(i+1) + 0 = opt(i)$

so $*opt(i+1) + 0 > opt(i)$ but this cannot be possible as no value is added hence $opt(i) = *opt(i+1)$. This leads to a Contradiction and hence above claim is true

Case 2: when ball is Selected

→ $opt(i) = val_i + opt(i+2)$

claim – $opt(i+2)$ gives optimal solution for $opt(i)$ subproblem

let us assume claim is not true, hence there exists another $^* opt(i+2)$ such that $^* opt(i+2) > opt(i+2)$ hence $^* opt(i+2) + val_i > opt(i+2) + val_i$

but $opt(i) = val_i + opt(i+2)$

so $^* opt(i+2) + val_i > opt(i)$ but this is a contradiction hence the above claim is correct

→ with the help of above claims, the recurrence is true

→ The subproblem $opt(1)$ i.e subarray $1 \to n$ gives ~~best~~ optimal solution

Pseudo code

```
int[] arr;
int[] val
arr[n] = val[n]
arr[n-1] = max( val[n-1], val[n])
arr[n-2] = max(val[n-2] + val[n], val[n-1])
for (i = n-3 ; i>0 ; i--)
    arr[i] = max (arr[i+1], val[i] + arr[i+2])

Return arr [1]
```

→ Time complexity

The time complexity of above algorithm is $O(n)$ as there is only 1 for loop filling the array rest is in $O(1)$ time

Question 5

Solution → ~~sort the array on the basis of the night~~

Sub problem

opt(i) = represents the maximum profit or money he can get within the array (0.... i)

Recurrence

1) there are two possibilities, that Jatin will perform on a particular night or not

If he works at a night energy is decreased by Value w_i and earns profit (V_i)

Base Case

```
if (index > n)
        return 0
else
    val1 = opt(i+1, energyi + 1), Val2 = Integer.minValue
    if (energy >= Wi([i]))
        Val2 = Vali + opt(i+1, energy - Wi + 1)

return max(val1, Val2)
```

Time Complexity

→ The time complexity is $n *$ (Size of W_i)
as there can be $n \times W_i$ possible combinations
of n and W_i (Energy)