Declaration and statement of authorship

I, bearing Registration Number 106119100, agree and acknowledge that:

1. The assessment was answered by me as per the instructions applicable to each assessment, and that I have not resorted to any unfair means to deliberately improve my performance.

2. I have neither impersonated anyone, nor have I been impersonated by any person for the purpose of assessments.

Signature of the Student : Rpandey 11/05/21

Full Name : RAJNEESH PANDEY

Roll No. : 106119100

Sub Code : CSPC42

Mobile No. : 8290968008

Instructions: **Do not include this in the declaration**

1. Either print the declaration or
   Write in hand on a separate sheet of paper with
   - ✓ Write your Registration Number (Roll No.)
   - ✓ Sign against Signature of the Student with date
   - ✓ Full Name (in Capital Letters)
   - ✓ Roll Number
   - ✓ Sub Code
   - ✓ Mobile Number
2. Scan the document and save it in PDF format
3. **Upload along with the Answer Sheet as first page.**
4. **Without this declaration, the answer sheet will not be evaluated.**

NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI

End-Sem-Algos

CSPC-42

10619100

Rajneesh

11/05/21

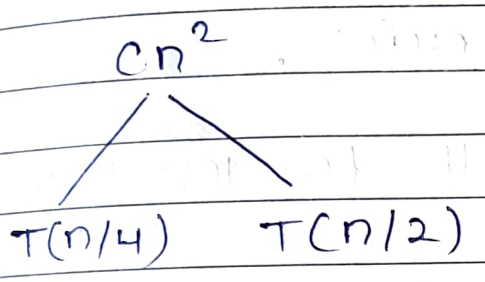
Question (1)

1) (a) $T(n) = T(n/4) + T(n/2) + cn^2$

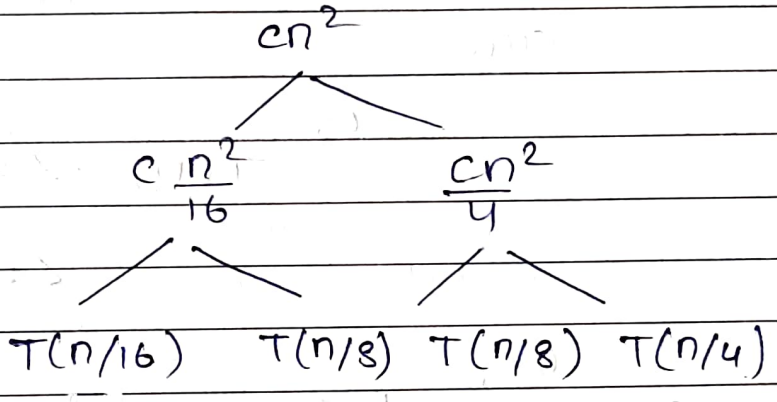
As, $T(1) = c$

$T(0) = 0$

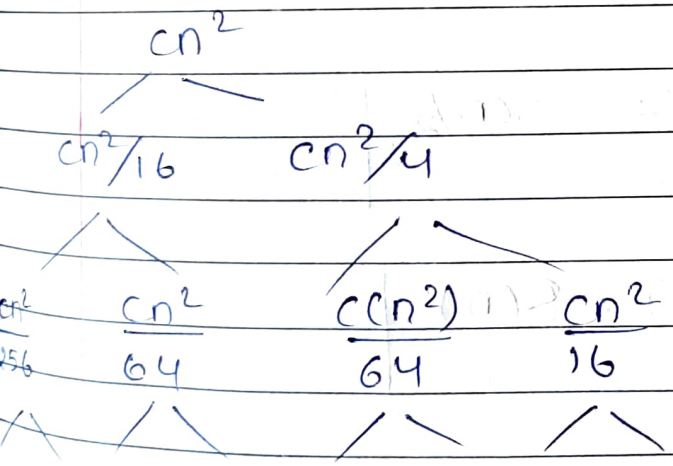
Recursion Tree for this recursive solution



now, further breakdown the T(n/4) & T(n/2)



Now,



for T(n) = sum of leaves nodes level by level

$$T(n) = c\left(n^2 + \frac{5n^2}{16} + \left(\frac{25n^2}{256}\right) + \text{----}\right)$$

It's GP with ratio 5/16.

Upper bound, take sum upto infinite

$$\boxed{T(n) = \frac{n^2}{(1 - 5/16)} \Rightarrow O(n^2)}$$

1) (b) As,

$n <= 3$, then $T(n) = n$

else

$$T(n) = T(n/3) + cn$$

now,

$$T(n) = cn + T(n/3)$$
$$= cn + cn/3 + T(n/9)$$
$$= cn + cn/3 + cn/9 + T(n/27)$$

Sum of infinite GP series.

The value of $T(n)$ will be less than the sum

so,

$$T(n) <= cn\,(1/(1-1/3))$$
$$<= 3cn/2$$

Or,

$$\boxed{cn <= T(n) <= 3cn/2}$$

Therefore $\boxed{T(n) = \Theta(n)}$

Question (2)

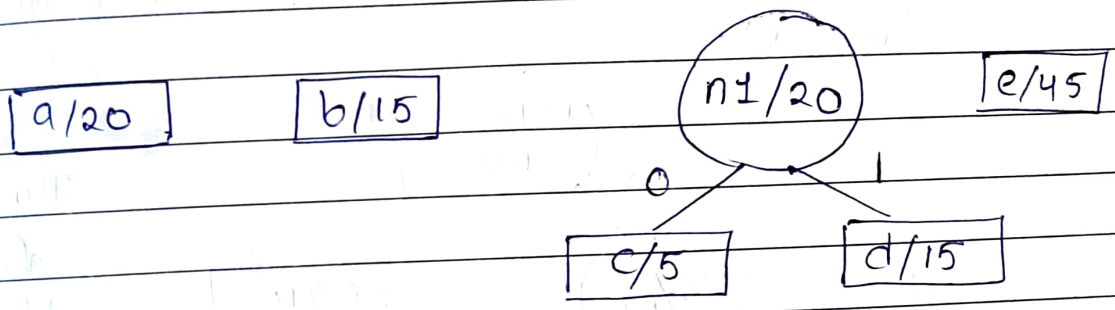
2) (a) Let $X = \{a/20, b/15, c/5, d/15, e/45\}$ be, the alphabet and its frequence distributio

In first step of Huffman coding we merge c and d.

because,

→ we start and select two symbols with the lowest frequency.

so,



now, Alphabet is now

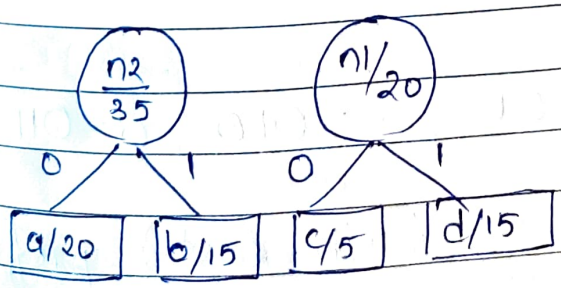
$A_1 = \{a/20, b/15, n1/20, e/45\}$.

→ Repeating the above step until only two values remain.

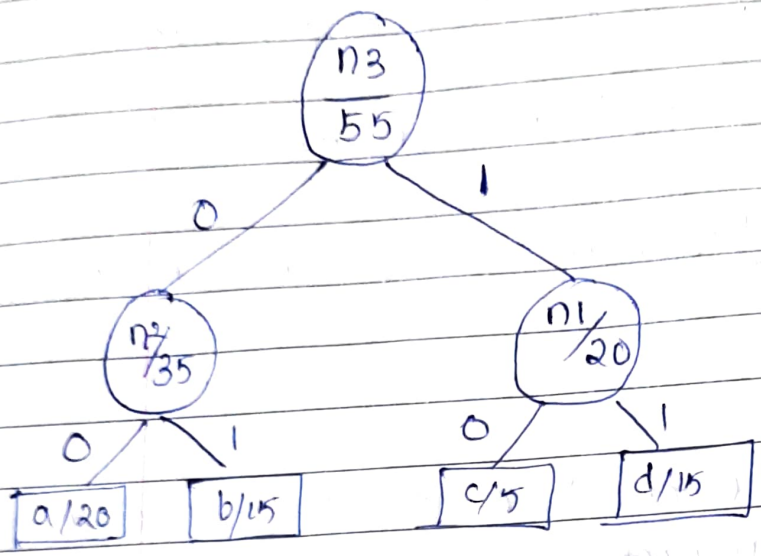
Now,

Algorithm merges a and b.

(could also merge n1 and b)

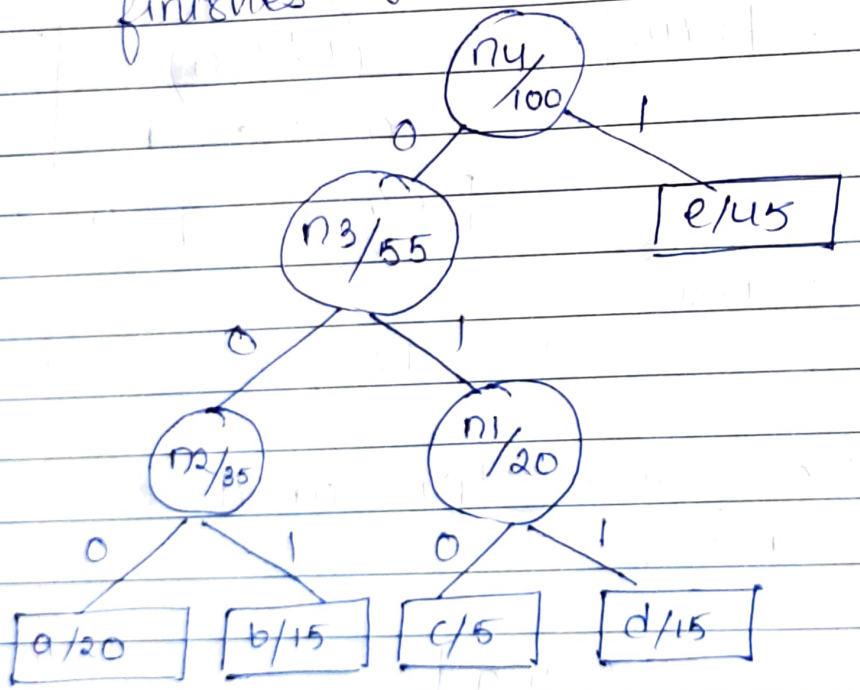


Now, $A_2 = \{n2/35,\ n1/20,\ e/45\}$



Now
$A_3 = \{n3/55,\ e/45\}$

now, merge e & n3. and algorithm finishes



This is final Huffmann Tree.

Huffman code

a=000 , b=001 , c=010 , d=011 , e=1

This is the optimum (min-cost) prefix code for this distribution.

(2) (b)

Greedy Algorithm works step-by-step, and always choose the step which provide immediate profit/benefit.

It chooses the "local optimal solution", without thinking about future consequences. This may not always lead to the optimal global solution, because it does not consider all the data.

Algorithm:

getOptimal (Item, arr[], int n)

1) Initializes empty result : result = { }
2) while (All items are not considered)

// we make a greedy choice to select an item

i = selectAnItem()

// If i is feasible, add i to the result

If (feasible (i))

result = result ∪ i

3) return result.

Advantage:

- Greedy approaches is easy to implement
- Typically have less time complexity
- Greedy algorithms used for optimization purpose or finding

close to optimization in case of NP Hard problem.

Disadvantage :

The local optimal solution may not always be global optimal.

Application :

(1) Finding Optimal solution
   (a) Activity selection
   (b) Fractional knapsack
   (c) Job sequencing
   (d) Huffman coding

(2) Finding close to the optimal solution of NP-Hard problems like TSP.

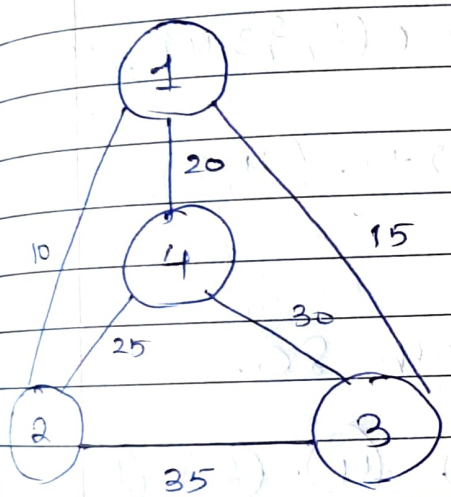
Question (3)

3) A) Traveling Salesman Problem (TSP) is NP-hard problem.

we can use, naive, Dynamic Programing or Approx. MST to solve TSP

but, naive and DP solution are infeasible

Now, solving using DP (dynamic Programming)



	1	2	3	4
1	0	10	15	20
2	10	0	35	25
3	15	35	0	30
4	20	25	30	0

S: subset of the graph which is not yet traversed

dist (i,1) : distance from i to 1

If size of S is 2, then S must be {1, i}

$C(s,i) = dist(1,i)$

Else if size of S is greater than 2.

$C(s,i) = \min\{C(S-\{i\}, j) + dis(i,j)\}$.

where j belongs to S, $j \neq i$

and $j \neq 1$

lets staut from node 1.

$C(4, phi) = 20$, $C(3, phi) = 15$, $C(2, phi) = 10$

$C(2, \{3\}) = d(2,3) + C(3, phi) = 50$
$C(2, \{4\}) = d(2,4) + C(4, phi) = 45$
$C(3, \{2\}) = d(3,2) + C(2, phi) = 45$
$C(3, \{4\}) = d(3,4) + C(4, phi) = 50$

Finally

$C(1, \{2,3,4\}) = \min(d(1,2) + C(2, \{3,4\}), d(1,3) + C(3, \{2,$
$\rightarrow d(1,4) + C(4, \{2,3\}))$

$= \min(90, 80, 95)$
$= 80$

So, an optimal tour has length 80.

and tour is 1 — 3 — 4 — 2 — 1

there are at most $O(2n * n)$ time complexity, much less than $O(n!)$

3) B) Given that,

weight [] = {1, 2, 3}

value [] = {6, 10, 12}

capacity ⇒ W = 5

Solving this 0/1 knap sack problem using Dynamic programming

for, $1 \le i \le n$ and $0 \le j \le w$

$DP[i][j] = \max(DP[i-1][j] + val[i] + DP[i-1][j-wt[i-1]])$

and

for, $i = 0$ and $0 \le j \le w$

$DP[0][j] = 0$.

and for $wt[i-1] > j$ $\quad DP[i][j] = DP[i-1][j]$

Now, W = 5 and n = 3.

$DP[0][j] = 0 \quad 0 \le j \le 5$

$DP[1][1] = \max(DP[0][1], 6 + DP[0][0])$

$= 6$

$DP[1,2] = \max(DP[0][2], 6 + DP[0][1])$

$= 6$

$DP[1][3] = \max(0, 6+0) = 6$

$DP[1][4] = \max(0, 6+0) = 6.$

$DP[1][5] = \max(DP[0][6], 6 + DP[0][4])$

$= \max(0, 6+0) = 6.$

DP[2][1] = max(6, 10−∞) = 6.

DP[2][2] = max(6, 10+0) = 10

DP[3][1] = max(6, 12−∞) = 10.

DP[3][4] = max(16, 22) = 22

hence,

DP[3][5] = max(16, 12+10) = 22.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	6	6	6	6	6
2	0	6	10	16	16	16
3	0	6	10	16	18	22

Hence the max wt. in knapsack = 22

= DP[3][5]

Question (4)

4) (A) we have given sorted array and we need to find x.

So, **Algorithm**:

Generate a random number t since range of number in which we want a random number is [start, end]

Hence,

t = t % (end − start + 1)

Then,

t = start + t

Example:

Array $A = \{6, 7, 10, 15, 23\}$

search for $x = 15$

(i) Range $R = [0, 4]$

$n = (1024 \% (4-0+1)) + 0$

$n = 4$

$A[4] > x \Rightarrow$ search in left

(ii) Range $R = [0, 3]$

$n = (222 \% (3-0+1)) + 0$

$n = 2$

$A[2] < x \Rightarrow$ search in right

(iii) Range $R = [2, 3]$

$n = (6172 \% (3-3+1)) + 3$

$\Rightarrow n = 3$

$A[3] == x$

$\therefore$ x is present at index 3 in Array A.

(4) (B)

Pseudo-code :

```
Randomized-Quicksort (A,p,r) {
    if (p<r)
    { q = RPartition(A,p,r)
      Randomized-Quicksort (A,p,q-1)
      Randomized-Quicksort (A,q+1,r)
    }
}

RPartition (A,p,r) {
    i = random-int (p,r);
    swap (A[i], A[r]);
```

```
x = A[r]
i = p-1
for ( j=p ; j<=r-1 ; j++) {
    if (A[x] <= x {
            i++
            swap [ A[i], A[j] )
    }}
swap( A[i+1], A[r])
return i+1 ;
}
```

Analysis of Randomized-Algo.

$$T(N) = T(N-i) + cN, \quad N>1$$
$$+ T(i) \quad : \text{otherwise}$$

So,

$$T(N-1) = T(N-2) + c(N-1) + T(0)$$

$$T(N) = T(1) + cN$$

the time to sort the i.e equal to sorting left and right partition and the time of building the partition.

the partitions.

Pivot in random place (on average)

$T(N) = 2T(N/2) + cN$

Dividing by N, we get

$$\frac{T(N)}{N} = \frac{T(N/2)}{(N/2)} + c$$

Using Telescopic Series, we get:

$$\frac{T(N)}{N} + \frac{T(N/2)}{(N/2)} + \frac{T(N/4)}{(N/4)} + \cdots\cdots$$

$$= \left(\frac{N/2}{N/2}\right) + \frac{T(N/4)}{(N/4)} + \frac{T(1)}{1} + c\log N$$

After crossing equal terms, we get:

$$\frac{T(N)}{N} = T(1) + c \times \log N$$

$$T(N) = N + N \times c \times \log N$$

$$= O(N \log N)$$

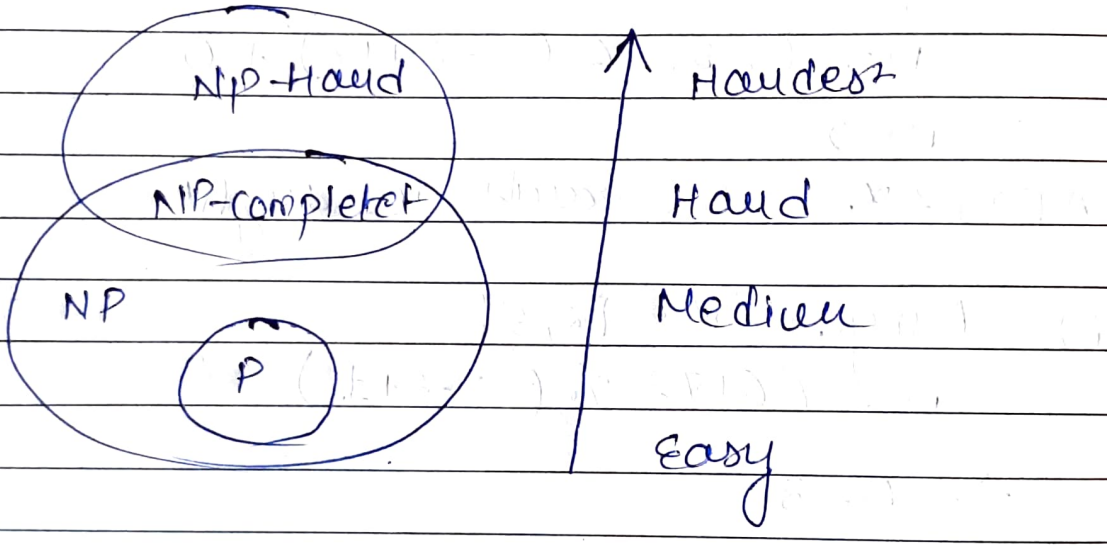
(average case Time Complexity)

Question (5)

P → "Polynomial" time
NP → "Non-deterministic Polynomial time"

There is also NP-Hard and NP-complete sets.

Easy → P , Medium → NP , Hard → NP-complete
Hardest → NP- Hard.



1. P (Polynomial Algorithm)

$T(n) = O(C * n^K)$ where $C > 0$ $K > 0$

In general, for polynomial-time algorithm K is expected to be less than n.

Example
→ Dijkstras, Bellman-Ford, Floyd warshall
→ linear & binary search

2. NP-Algorithms

we expect these algorithm to have an exponential complexity.

$T(n) = O(C_1 * k^{C_2 * n})$ $C_1 > 0$, $C_2 > 0$ $k > 0$

for example.

1) Graph Isomorphism
and 2) Integer Factorization.

the NP algorithm can't be solved in polynomial time.

3) NP-complete.

These belongs to NP set but are Hardest to solve
there exist a polynomial-time algo. that transform it to NP-complete problem

example
TSP, knapsack, Graph coloring

4) NP-Hard

most complex Algor. for computerscience to solve them.

exg. k-means.
TSP.
graph colouring