Design and Analysis of Algorithms Assignment

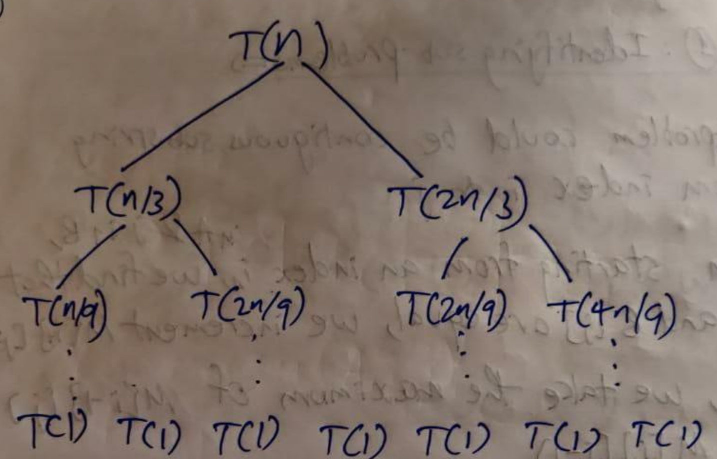
Question 1:

1. Consider another variation of the binary search algorithm so that it splits the input not only into two sets of almost equal sizes, but into two sets of sizes approximately one-third and two-thirds. Write down the recurrence for this search algorithm and the asymptotic complexity of this algorithm.

Answer to Question 1:

Homework Assignment

①



So from this recurrence tree of the binary search algorithm, we can see that a problem of size n has been divided into subproblem of size n/3 and 2n/3.

The worst case ~~occurs~~ occurs when the key to be searched lies in the rightmost index ie. n-1. This means the ~~a~~ recursion skews right ie. the recurrence relation is:

$$T(n) = \underbrace{T(2n/3) + c}$$

↳ cost of comparison.

PTO

$a = 1 \quad b = \frac{3}{2} \quad f(n) = c$

$n^{\log_b a} = 1 \text{ (or) } (n^{\log_{3/2} 1})$

$\Rightarrow$ Case 2

as $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$ for $k = 1$.

Hence, $T(n) = \Theta(\log n)$ as per Master Method.

Question 2:

2. Two different divide and conquer algorithms A and B have been designed for solving the problem π. A partitions π into 5 subproblems each of size n/3 . Here n is the input size of π.it takes a total of $\theta(n^2)$ time for the partition and combine steps. B partitions π into 10 sub problems each of size n/4 . Here n is the input size of π. It takes a total of $\theta(n^{1.8})$ time for the partition and combine steps. Which algorithm is preferable? Why?

Answer to Question 2:

②

A: $T(n) = 5T(n/3) + f_1(n) \Rightarrow \theta(n^2)$

B: $T(n) = 10T(n/4) + f_2(n) \Rightarrow \theta(n^{1.8})$

A: $n^{\log_b a} = n^{\log_3 5} \approx n^{1.465}$

B: $n^{\log_4 10} = n^{\log_4 10} \approx n^{1.661}$

A:

$af(n/b) = 5f_1(n/3)$

$= \frac{5}{9} f_2(n)$

$\delta = \frac{5}{9} \approx 0.56$

B:

$f(n) = \Omega(n^{1.6+0.2})$

$\Rightarrow \theta(n^{1.8})$

$af(n/b) = 10f_2(n/4)$

$= \frac{10}{48} f(n)$

$\delta = \frac{10}{4.8} \approx 0.625$

Hence, Algorithm A is preferrable as it computes in lesser time in average case, because δ is smaller

Question 3:

3. Design and analyze a divide and conquer MAXMIN algorithm that uses ceil(3n/2) – 2 comparisons for any n.

Answer to Question 3:

(3)

Base case: $n=2$

$$\Rightarrow \left\lceil \frac{3n}{2} \right\rceil - 2 = \left\lceil \frac{3(2)}{2} \right\rceil - 2 = 1$$

Algorithm maxmin (A, i, j)

if $i=j$

then return (A[i], A[j])

else if $i+1=j$

if $A[i]<A[j]$

then return (A[i], A[j])

else

return (A[j], A[i])

else

$m \leftarrow \left\lfloor \frac{j-i}{2} \right\rfloor$

if $m \% 2 = 1$

then $m \leftarrow m+1$

P.T.O

PTO

min1, max1 ← MaxMin (A, i, i+m-1)
min2, max2 ← MinMax (A, i+m, j)
min ← (min1 < min2) ? min1 : min2
max ← (max1 > max2) ? max1 : max2
return (min, max)

As per this algorithm, the recursion tree will have $\lceil \frac{n}{2} \rceil$ leaves and $\lceil \frac{n}{2} \rceil - 1$ inner nodes.

If $n \% 2 = 0$, then all $\frac{n}{2}$ leaves are 2 element sequences and require only one comparison.

If $n \% 2 = 1$, then one leaf is a single element that requires no comparison.

In the leaves we perform $\lfloor \frac{n}{2} \rfloor$ comparisons, and in the inner nodes we perform 2 comparisons, adding up to $2\lceil \frac{n}{2} \rceil - 2$.

Upon summing it up, we get

$$\lfloor \frac{n}{2} \rfloor + 2\lceil \frac{n}{2} \rceil - 2 = \lceil \frac{3n}{2} \rceil - 2 \text{ comparisons.}$$

Question 4:

4. Give pseudocode to reconstruct an LCS from the completed cost table and the original sequences X = <x1, x2,..., xm> and Y = <y1 , y2, ... , yn> in O(mn) time.

Answer to Question 4:

4

```
Algorithm LCS (X, Y, i, j)
    if (L[i,j] ≤ 0)
        if (X[i] = '\0' || Y[j] = '\0')
            L[i,j] = 0
        else if (X[i] = Y[j])
            L[i,j] =
                1 + LCS(X, Y, i-1, j-1)
        else
            L[i,j] =
                max{LCS(X, Y, i-1, j),
                    LCS(X, Y, i, j-1)}
    return L[i,j]
```

Complexity : O(mn)

m : size of x

n : size of y

Question 5:

5. Give an $O(n^2)$ time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers. If the array is {3,2,8}, {3,8} or {2,8} are monotonically increasing subsequences.

Answer to Question 5:

⑤ {3, 2, 8}

$$\begin{array}{c|cccc} & 0 & 3 & 2 & 8 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 1 & 1 \\ 2 & 0 & 1 & 1 & 2 \\ 8 & 0 & 1 & 1 & 2 \end{array}$$

{2,8}

For 2 different strings of sizes m and n,
The complexity of LCS is O(mn).

⇒ for same length, $O(n \cdot n) = O(n^2)$

Modified Algorithm:

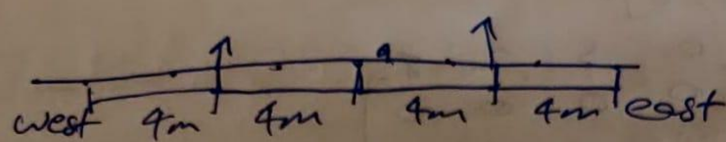
```
Algorithm LCS(X, Y, i, j)
  if(L[i,j] < 0)
    if (X[i] = '\0' & Y[j] = '\0')
        L[i,j] ← 0
    elseif(X[i] < Y[j])
        return 1 + LCS(X, Y, i-1, j-1)
  else return max{LCS(X,Y,i-1,j),
                  LCS(X,Y,i,j-1)}
```

Question 6:

6. Let us consider a long, quiet country road with houses scattered very sparsely along it. (Picture the road as a long line segment, with an eastern endpoint and a western endpoint. The houses are points along the line.) Further, lets suppose that despite the bucolic setting, the residents of all these houses are avid cell phone users. You want to place cell phone base stations at certain points along the road, so that every house is within four miles of one of the base stations. Give an efficient greedy algorithm that achieves this goal, using as few base stations as possible. Note that a station covers an interval of eight miles.
   1. At what point do you place the next base station?
   2. Show that your greedy algorithm is optimal using a swapping argument.

Answer to Question 6:

6



Greedy Approach :

→ Start at the western end

→ Move east until you go 4 miles from west

→ Place a base station there

→ Delete all houses covered by this base station

→ Iterate among remaining houses.

Question 7:

7. Let A[0..n − 1] be an array of n real numbers. A pair (A[i], A[j]) is said to be an inversion if these numbers are out of order, i.e., i < j but A[i] > A[j]. Design an O(n log n) algorithm for counting the number of inversions. Demonstrate that your algorithm is correct.

Answer to Question 7:

7

Given A[0, 1, 2, ..., n-1]

To get O(nlogn), merge sort has to be used.

So this is simply a modification to merge sort.

Algorithm:

```
Algorithm invCount (A)
    if(len(A) < 2)
        return 0
    mid = (len(A) + 1)/2
    L~~eft~~ = A[0 ... mid]
    R = A[mid+1 ... len(A)-1]
    return invCount(L) + invCount(right) +
              merge(A, L, R)
```

Upon every comparison, increment the count

Question 8:

8. The height of a binary tree can be determined using a divide and conquer strategy. Give a divide and conquer strategy to find the height of a binary tree. Write the recurrence relation of your solution and compute the running time.

Answer to Question 8:

8

Algorithm height(root)
if root is None
return 0
return 1+max(height(root.left), height(root.right))

Recurrence:

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

$\Rightarrow$ Case 2 $k=0 \Rightarrow T(n)$ is $\Theta(n \log n)$

Question 9:

9. The Bellman Ford algorithm solves the shortest path problem with negative weights. It applies dynamic programming. Explain the five steps in dynamic programming as applied in this algorithm with examples.

Answer to Question 9:

(9)

Bellman Ford Algorithm:

Subproblem:

Path from start vertex to some other vertex on graph G.

$D(i, G)$ = Length of shortest path from source to destination using at most i edges.

Guess:

Relax on each vertex atmost $(m-1)$ times

$$\text{if } (d[u] + c(u,v) < d(v))$$
$$d(v) = d[u] + c(u,v)$$

Relate Subproblems

we need to find smaller subproblems that can be solved and made use of later.

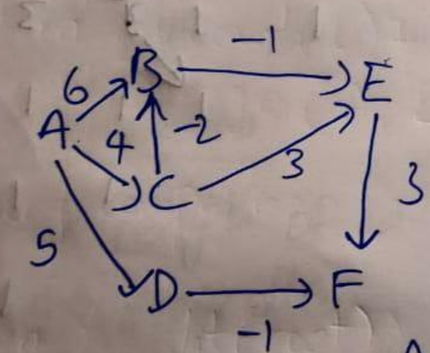
Recursion and Memoize:

Recursion:

$$D(i,s) = \min\{D(i-1,z), \min\{D(i-1),y\} + w(y,z)\}$$

A 2-D DP table could be used to keep track of shortest path

Ex:



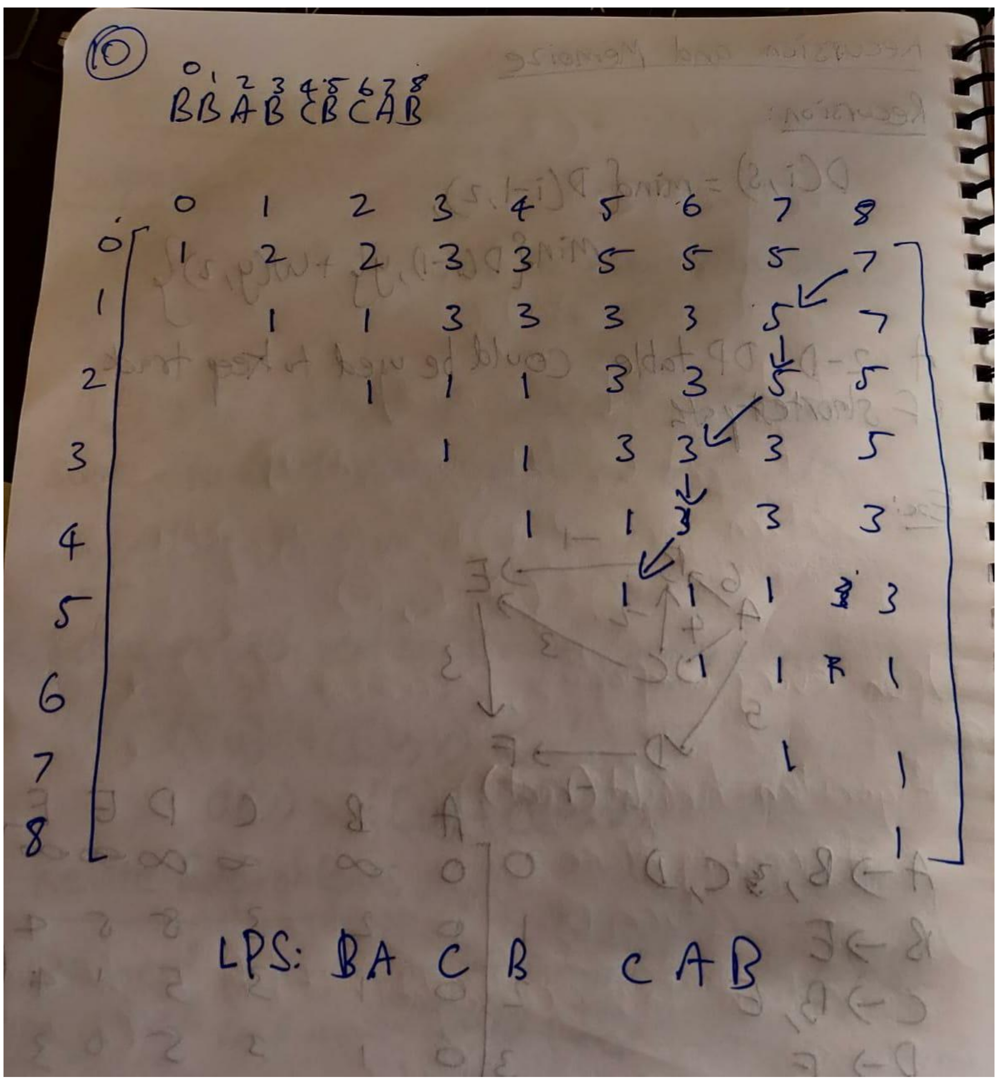
A → B, C, D
B → E
C → B, E
D → F
E → F

$$\begin{array}{c|c} & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & \infty & \infty & \infty & \infty & \infty \\ 0 & 2 & 3 & 5 & 5 & 4 \\ 0 & 1 & 3 & 5 & 1 & 4 \\ 0 & 1 & 3 & 5 & 0 & 3 \\ 0 & 1 & 3 & 5 & 0 & 3 \\ 0 & 1 & 3 & 5 & 0 & 3 \end{bmatrix} \end{array}$$

Question 10:

10. Given a sequence, given an efficient algorithm to find the length of the longest palindromic subsequence in it. For example, if the given sequence is "BBABCBCAB", then the output should be 7, and "BABCBAB" is the longest palindromic subsequence in it.

Answer to Question 10:



PTO

Bottom-up approach:

if (input [i] == input [j])
  T[i] [j] = T[i+1] [j-1] +2
else
  T[i][j] = Max{ T[i+1] [j], T[i] [j-1]}

~~Sep~~ Subproblem:

Finding max length of palindromic subsequence at each length upto last character

Relating subproblems:

Taking max length of a palindromic subsequence of previous subproblem

(or)

if first and last character are equal then 2+ previous subproblem

Solution:

Last element of first row gives length of LPS.