Chongqing University of technology
Md.Anower Hossain(an hao ming)
Student ID: 62017010084
Teacher: Henry

Sub: Algorithm's

1. Solve the following recurrence relation and give a Θ bound for each them.

1.1 T (n) = 2T (n/3) + 1

T(n)= aT $(\frac{b}{a})$ + $\int(x)$

$\int(x) = \theta(n^k log^p n)$) a≥1, b>1.

Two things here 1. $\log_b a$

2. K

Case 1: $\log_b a >$ k then $\theta(n^{\log_b a})$

Case 2: $\log_b a =$ k $\begin{cases} and\ if & p > -1 \text{ then } \theta(n^k log^{p+1} n) \\ and\ if & p = -1 \text{ then } \theta(n^k loglogn) \\ and\ if & p < -1 \text{ then } \theta(n^k) \end{cases}$

Case 3: $\log_b a <$ k $\begin{cases} and\ if\ p \geq 0\ then\ \theta(n^k log^p n) \\ and\ if\ p < 0\ then\ \theta(n^k) \end{cases}$

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

so the equation is T (n) = 2T (n/3) + 1

Here a = 2; b = 3; and $\int(x) = \theta(1)$

$= \theta(n^0)$

$= \theta(n^0 log^0 n)$

$\therefore k = 0$ And P = 0

Therefore

$\log_b a = \log_3 2$

= 0.631 > k

$\therefore \Theta(n^{\log_b a})$

Finally $\Theta(n^{\log_3 2})$

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

T(n)= a$T(\frac{b}{a})+\int(x)$

$\int(x) = \theta(n^k log^p n)$) a≥1 , b>1.

Two things here 1. $\log_b a$

2. K

Case 1: $\log_b a$ > k then $\theta(n^{\log_b a})$

Case 2: $\log_b a =$ k $\begin{cases} and\ if & p > -1 \text{ then } \theta(n^k log^{p+1} n) \\ and\ if & p = -1 \text{ then } \theta(n^k loglogn) \\ and\ if & p < -1 \text{ then } \theta(n^k) \end{cases}$

Case 3: $\log_b a <$ k $\begin{cases} and\ if\ p \geq 0\ then\ \theta(n^k log^p n) \\ and\ if\ p < 0\ then\ \theta(n^k) \end{cases}$

Solution: - $T(n) = 4T(n/2) + n^2$

so the equation is

$T(n) = 4T(n/2) + n^2$ compare with $T(n) = aT(\frac{b}{a}) + \int(x)$

Here a = 4; b = 2; and $\int(x) = \theta(n^2)$

$= \theta(n^2)$

$= \theta(n^2 log^0 n)$

$\therefore k = 2$ And P = 0

Therefore

$\log_b a = \log_2 4$

$= \log_2 2^2$

$= 2 \log_2 2$

$= 2$

$\log_b a = k$

And $k > -1$

$\therefore \Theta(n^k log^{p+1} n)$

$\Theta(n^k log^1 n)$

Finally $\Theta(n^2 log\, n)$

1.3 $T(n) = 8T(n/3) + n^3$

Using master theorem compare with $T(n) = aT(\frac{b}{a}) + \int(x)$

Here a = 8; b = 3 and $\int(x) = n^3$

$= \theta(n^3 log^0 n)$ $[(n^k log^p n)]$

K = 3 and p = 0;

Therefore

$\log_b a = \log_3 8$

$= \log_2 2^3$

$= 3 \log_2 2$

$= 3$

$\log_b a = k$

And $k > -1$

$\therefore \quad \Theta(n^k log^{p+1} n)$

$\Theta(n^k log^1 n)$

Finally $\Theta(n^3 \log n)$

1.4 $T(n) = T(n-1) + 2$

Solution:-

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + 2 & n > 0 \end{cases}$$

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

$= [T(n-2) + 2] + 2$

$T(n) = T(n-2) + 4$ ..... (II)

$= [T(n-4) + 2] + 4$

$T(n) = T(n-4) + 6$ ..... (III)

.

.

.

. Continue for k times.....

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

$n-k = 0$

$\therefore n = k$

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

$= T(0) + n$

$= 1 + n$

Finally $\Theta(n)$

1.5 $T\ (n) = 2T\ (n/3) + nlogn$

T (n) = a$T\ (\frac{b}{a}) + \int(x)$

$\int(x) = \theta(n^k log^p n)$) a≥1, b>1.

Two things here 1. $\log_b a$

2. K

Case 1: $\log_b a >$ k then $\theta(n^{\log_b a})$

Case 2: $\log_b a =$ k $\begin{cases} and\ if & p > -1 \text{ then } \theta(n^k log^{p+1} n) \\ and\ if & p = -1 \text{ then } \theta(n^k loglogn) \\ and\ if & p < -1 \text{ then } \theta(n^k) \end{cases}$

Case 3: $\log_b a <$ k $\begin{cases} and\ if\ p \geq 0\ then\ \theta(n^k log^p n) \\ and\ if\ p < 0\ then\ \theta(n^k) \end{cases}$

Solution: - $T\ (n)$ = 2$T\ (n/3)$ + n$logn$

so the equation is

$T\ (n)$ = 2$T\ (n/3)$ + n$logn$ compare with $T(n)$= a$T(\frac{b}{a})+ \int(x)$

Here a = 2; b = 3; and $\int(x) = \theta(\text{n}logn)$

$= \theta(n^1 log^1 n)$

$\therefore k = 1$ And P = 1

Therefore

$\log_b a = \log_3 2$

= 0.63

$\therefore \log_b a < k$

$\Theta(n^1 log^1 n)$

Finally Θ (n$logn$)

2.1 solves problems by dividing them into five sub problems of half the size, recursively solving each sub problem, and then combining the solutions in linear time.

Solution: - we can get the equation is $T(n) = 5T(n/2) + \theta(n)$

T (n) = a$T(\frac{b}{a}) + \int(x)$

$\int(x) = \theta(n^k log^p n)$) a≥1, b>1.

Two things here 1. $\log_b a$

2. K

Case 1: $\log_b a >$ k then $\theta(n^{\log_b a})$

Case 2: $\log_b a =$ k $\begin{cases} and\ if & p > -1 \text{ then } \theta(n^k log^{p+1} n) \\ and\ if & p = -1 \text{ then } \theta(n^k loglogn) \\ and\ if & p < -1 \text{ then } \theta(n^k) \end{cases}$

Case 3: $\log_b a <$ k $\begin{cases} and\ if\ p \geq 0\ then\ \theta(n^k log^p n) \\ and\ if\ p < 0\ then\ \theta(n^k) \end{cases}$

Using master theorem the equation is

$T(n) = 5T(n/2) + \theta(n)$ compare with $T(n)$= a$T(\frac{b}{a}) + \int(x)$

A = 5; b = 2; and $\int(x) = \theta(n^1)$ so k = 1

Therefore

$$\log_b a = \log_2 5$$

$$= 2.332$$

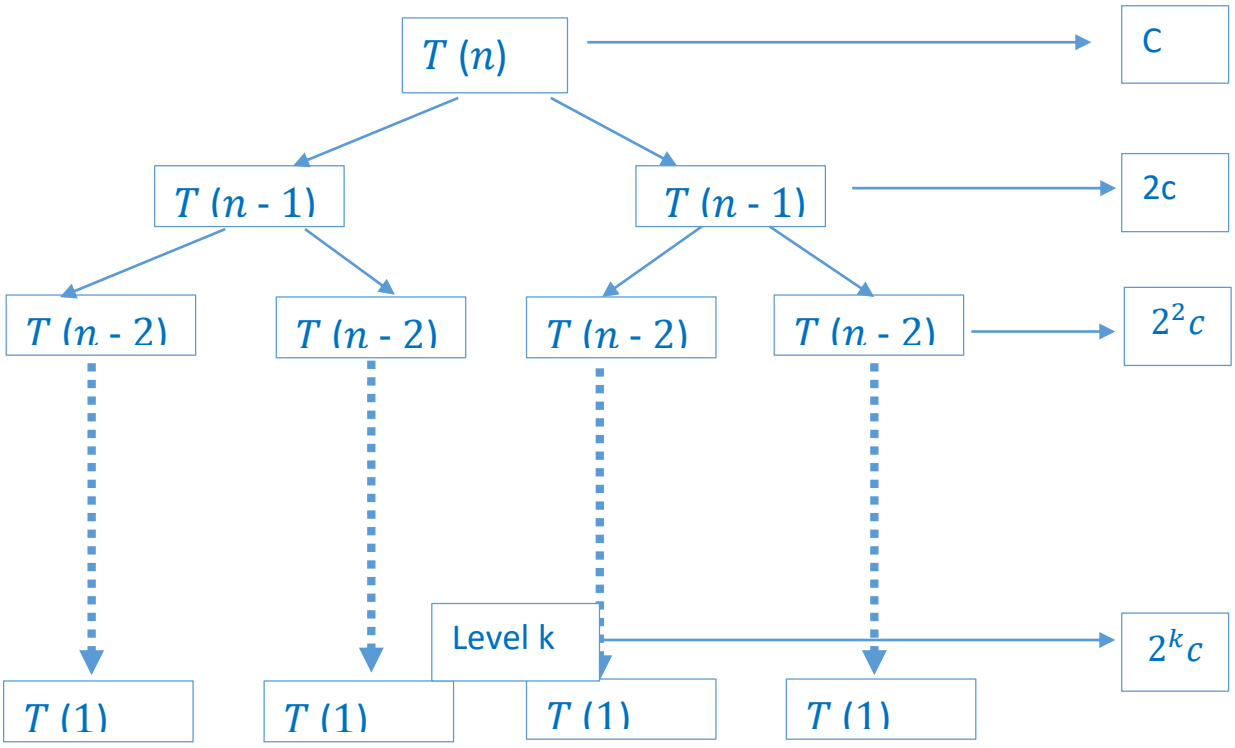
$\therefore \log_b a > k$

$\Theta(n^{\log_b a})$

Finally $\Theta(n^{\log_2 5})$

2.2 solves problems of size n by recursively solving two sub problems of size n -1 and then combining the solutions in constant time.

Solution: - we can get the equation is $T(n) = 5T(n/2) + c$



$T(n) = \sum_{i=0}^{n-1}(2)^k * C$

$= C * \frac{2^n - 1}{2-1}$

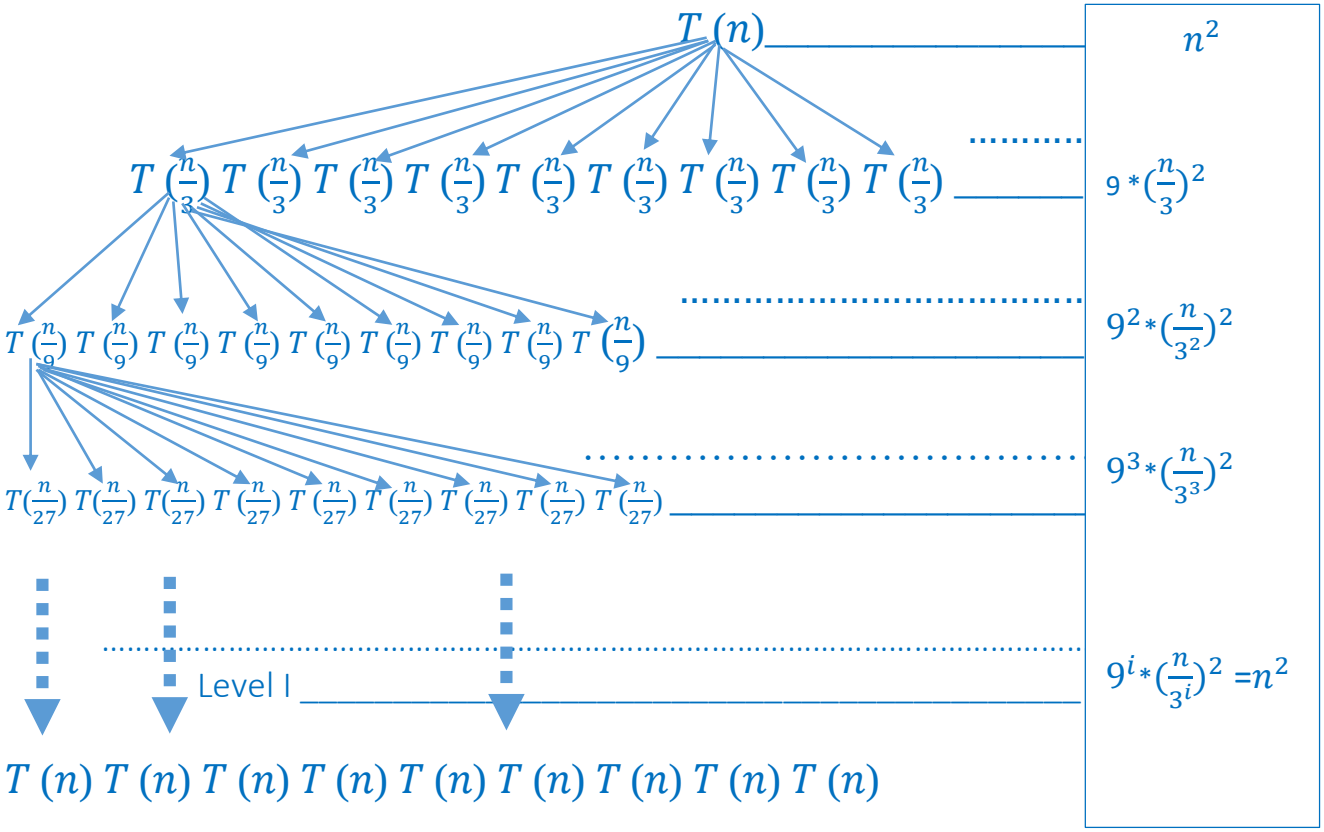
$= C * 2^n - 1$

$\therefore\ \Theta(2^n - 1)$

Finally $\Theta(2^n)$

2.3 Solves problems of size n by dividing them into nine sub problems of size n/3, recursively solving each sub problem, and then combining the solutions in O(n2) time.

Solution: - we can get the equation is $T(n) = 9T(n/3) + O(n^2)$



$$T(n) = \sum_{i=0}^{\log_3 n} n^2$$

$= n^2 \log_3 n$

Finally $\Theta(n^2 logn)$

Which one I will chose:

For algorithm 2.1: $T(n) = 5T(n/2) + n$ $\quad \Theta(n^{\log_2 5})$

For algorithm 2.2: $T(n) = 9T(n/3) + c$ $\quad \Theta(2^n)$

For algorithm 2.3: $T(n) = 9T(n/3) + n^2$ $\quad \Theta(n^2 logn)$

Since algorithm 2.2 runs in exponential time, it will be the slowest. So we need to compare
Algorithm 2.1 and algorithm 2.3.

$$\lim_{n\to\infty} \frac{n^{\log_2 5}}{n^2 \log_3 n}$$

$$= \lim_{n\to\infty} \frac{n^{\log_2 5-2}}{\log_3 n}$$

$$= \lim_{n\to\infty} \frac{n^{2.32-2}}{\log_3 n}$$

$$= \lim_{n\to\infty} \frac{n^{0.32}}{\log_3 n}$$

$$= \lim_{n\to\infty} \frac{0.32n^{-0.68}}{\frac{1}{nln3}} \quad [\text{L Hospitals Ruls}]$$

$$= \lim_{n\to\infty}(0.32n^{-0.68})nln3$$

$$= \infty$$

Thus $n^2 \log_3 n = O(n^{\log_2 5})$

In conclusion, based on the asymptotic behavior of the runtime of these algorithms, I will chose Algorithm 2.3 that has runtime $\Theta(n^2 \log_3 n)$ to solve the problem.

3. Ak-way merge operation. Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements. Merge the first two arrays, then merge in the third, and merge in the fourth, and so on. What is the time complexity of this algorithm in terms of *k* and n?

Solution: - We were called the merge procedure for the resulting arrays with the next input array one by one. The marge procedure for first two array would take time 2n the marge procedure for the resulting array and the 3rd input array would require 3n comparisons in the worst case and so on. If we take the comparison performed time is c then we can write the following function

T(n,k) = 2nc + 3nc + 4nc + 5nc + ……..+ knc

= nc (2 + 3 + 4 + 5 + …… + k)

= nc {(1 + 2 + 3 + 4 + 5 + ……. + k) – 1}

= nc $\{\frac{k(k+1)}{2} - 1\}$

= nc $(\frac{k^2+k-2}{2})$

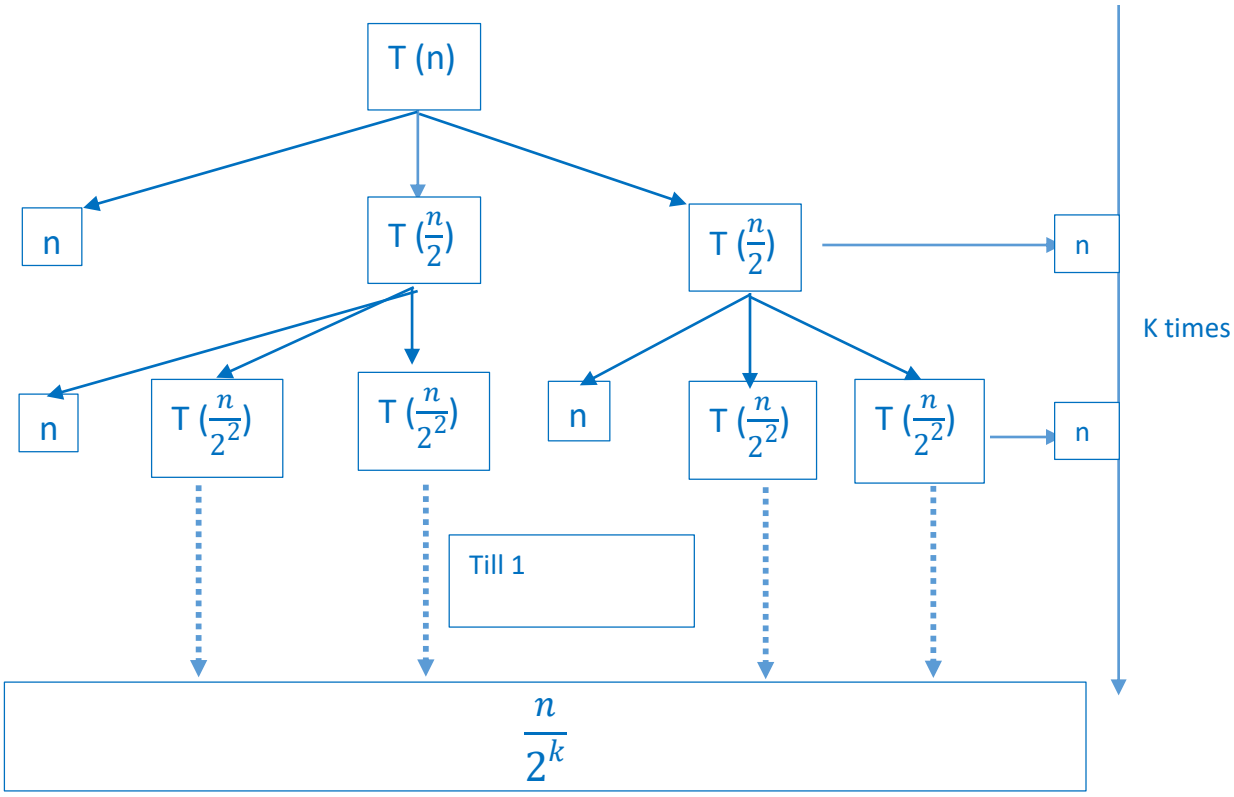
= $\frac{1}{2}$ c$k^2n + \frac{ckn}{2} - cn$

So T (n) = O (k^2n)

3.2 Give a more efficient solution to this problem using divide and conquer, and what is the complexity?

Solution: - for more efficient solution to this problem using divide and conquer we split the input list of arrays $[A_1 \ldots\ldots\ldots\ldots A_k]$ into two parts of the same size $[A_1 \ldots\ldots\ldots\ldots A_{k/2}]$ and $[A_{k/2} \ldots\ldots\ldots\ldots\ldots A_k]$. Each of these part we split in two sub parts again. Continue the process until we reach a list with a single array $[A_i]$ from this stage we can start merging the sub parts to acquire the sorted array .make this recursive until merge all the array. so we can write the time complexity is the following .

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + n & n > 1 \end{cases}$$



Assume $\frac{n}{2^k} = 1$

$n = 2^k$

$\log n = \log 2^k$

$\log n = k \log 2$

$\therefore k = n\log_2 n$ this the equivalent of being at the $\log_2 n$ level of merge sort n. so we just continue upwards and the process takes O(logn)

4. Given a sorted array of distinct integers $A[1,\cdots,n]$, you want to find out whether there is an index I for which $A[i]=i$. Given a divide and conquer algorithm that runs in time $O(\log n)$.

Solution: - Algorithm: If the array has just one integer, then we check whether

A [1] = 1 with one comparison. Otherwise divide the list into two parts, the first half and the second half, as equally as possible. Consider the largest element $A[m]$ of the left half. We compare A [m] with m.

- If $A[m] = m$, then the answer is yes and we are done.
- If $A[m] > m$, then we can throw away the right half and continue recursively in the left half. Indeed, then for every integer $k \geq 0$ using the fact that the integers are distinct and sorted

$$A\,[m + k] \geq A[m] + k > m + k.$$

- If $A[m] < m$, then we can throw away the left half and continue recursively in the right half. Indeed, then for every integer $k \geq 0$ using the fact that the integers are distinct and sorted

$$A\,[m - k] \leq A[m] - k > m + k.$$

Thus for the number of comparisons we get the following recursion:

$$T(n) = \begin{cases} 1 & n = 1 \\ T\left(\frac{n}{2}\right) + 1 & n > 1 \end{cases}$$

$T(n) = aT\left(\frac{b}{a}\right) + \int(x)$

$\int(x) = \theta(n^k log^p n)$) a≥1, b>1.

Two things here 1. $\log_b a$

2. K

Case 1: $\log_b a > k$ then $\theta(n^{\log_b a})$

Case 2: $\log_b a = k \begin{cases} and\ if & p > -1 \ \text{then}\ \theta(n^k log^{p+1} n) \\ and\ if & p = -1 \ \text{then}\ \theta(n^k loglogn) \\ and\ if & p < -1 \ \text{then}\ \theta(n^k) \end{cases}$

Case 3: $\log_b a < k \begin{cases} and\ if\ p \geq 0\ then\ \theta(n^k log^p n) \\ and\ if\ p < 0\ then\ \theta(n^k) \end{cases}$

Solution: - $T(n) = T\left(\frac{n}{2}\right) + 1$

so the equation is $T(n) = T\left(\frac{n}{2}\right) + 1$ for using the master theorem compare with T (n) = $aT\left(\frac{n}{b}\right) + \int(x)$

Here a = 1; b = 2; and $\int(x) = \theta(1)$

$= \theta(n^0)$

$= \theta(n^0 log^0 n)$

$\therefore k = 0$ And P = 0

Therefore

$\log_b a = \log_2 1$

= 0

$\log_b a$ = k

Case 2:

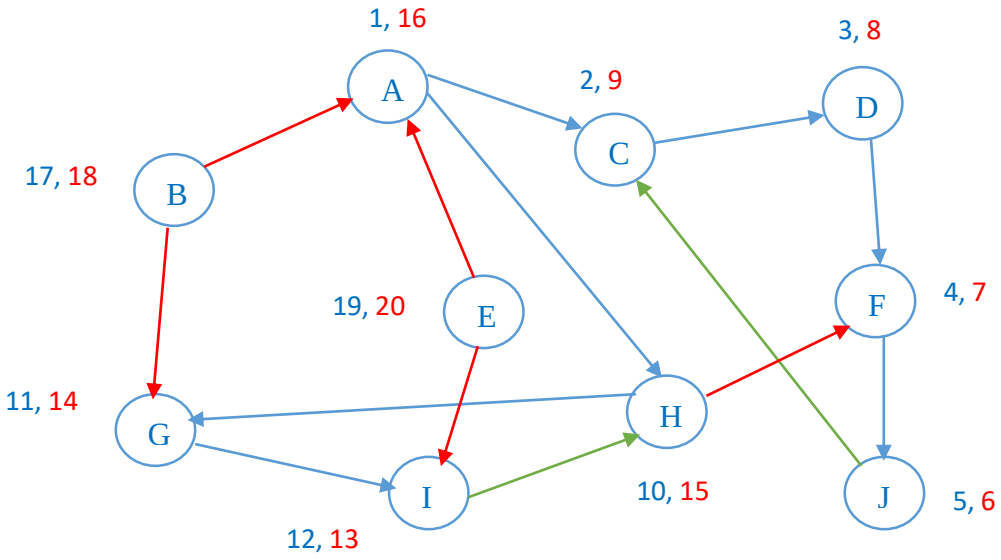
$\therefore \Theta(n^k log^{p+1} n)$

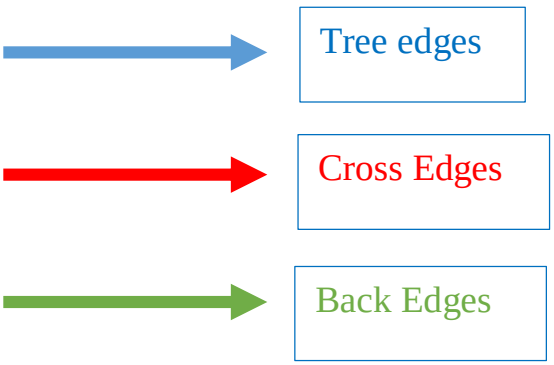
$\Theta(n^0 log^{0+1} n)$

Finally $\Theta(logn)$

5.1 Give the pre and post number of each vertex. Classify each edges as a tree edge, forward edge, back edge or cross edge.

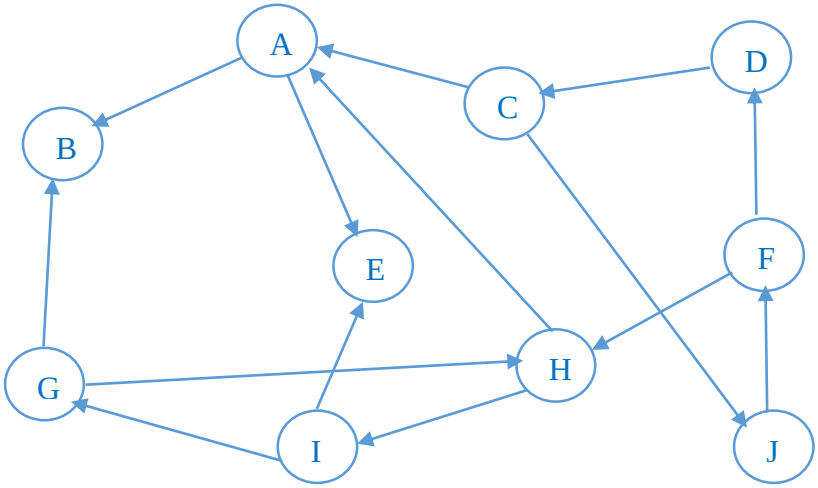
Solution: -



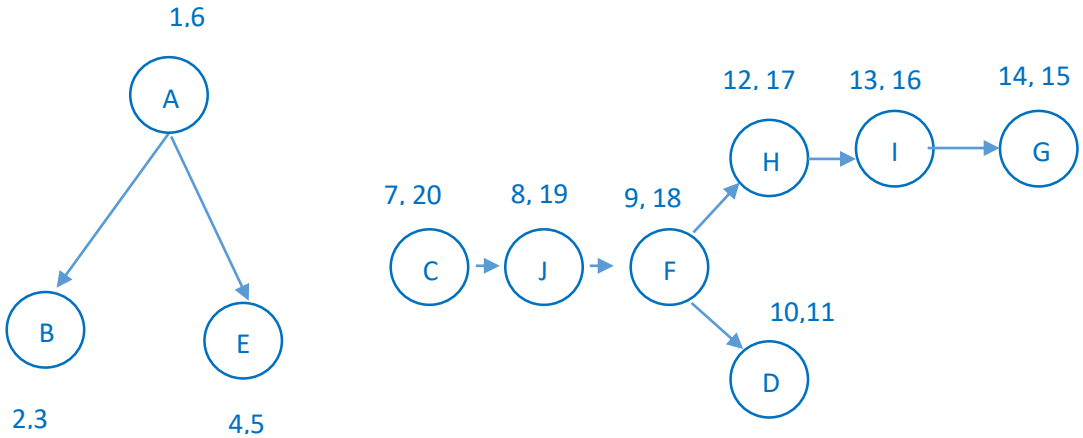


5.2 When doing DFS on G^R, what are the strongly connected components found?

Solution: - The G^R Graph is



Run the DFS on the above graph and the result is below:



DFS algorithms calls explores twice on the graph. Here is G^R and thus there will be two strongly connected components. The order of strongly connected components can be found by looking at the order of the post numbers.

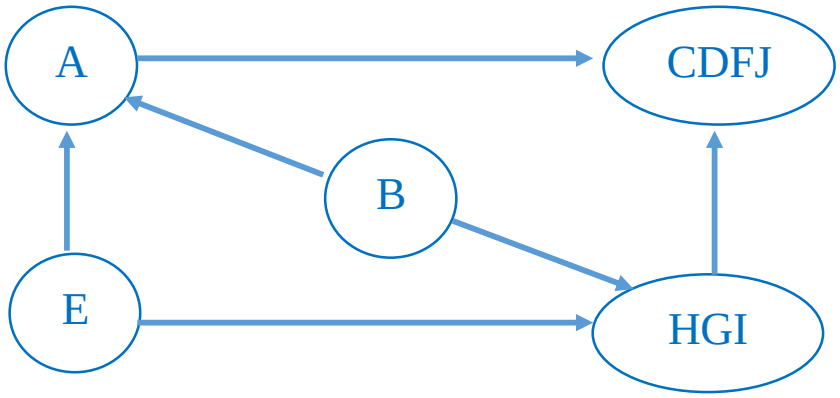
Here C and then add J, F, H, I, G, D. finally the SCC is {C, J, F, H, I, G, D}

And the sub graph A has the height post number then add E and B so the SCC is for sub graph is {A, E, B}

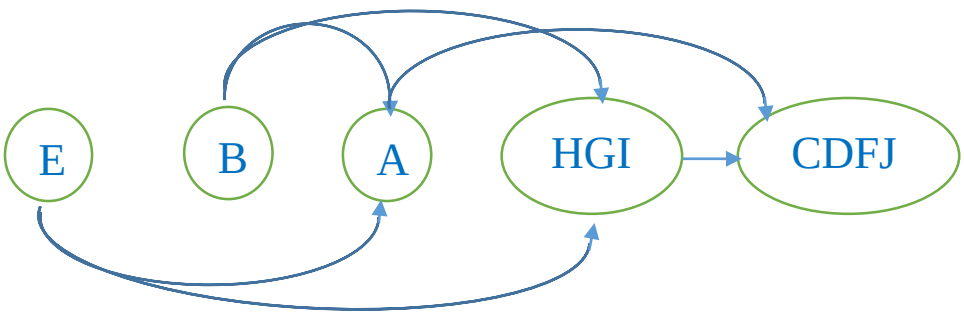
Therefore the strongly connected components found in the order:

{A B E}, {H G I, C D F J}

5.3 Draw the "meta graph"



Another



6.1 Show the tables for knapsack with repetition and for the knapsack without repetition.

Solution: -

- Table for knapsack with repetition

Item	Weight	Value
1	6	$30
2	3	$14
3	4	$16
4	2	$9

K (w) = $\max_{i:w_i \le w} \{K\ (w - \ w_i) + \ v_i\}$

ω	0	1	2	3	4	5	6	7	8	9	10
$K(\omega)$	0	0	9	14	18	23	30	32	39	44	48

K(ω) = max{k(3-3)+v3,k(3-2)+v2}={0+14,0+9} =14

K(ω) = max{k(4-3)+v3,k(4-2)+v2}={0+14,9+9} =18

- Table for knapsack without repetition

K(w, j) = max { K(w-wj, j-1) + vj , K(w, j-1) }

ω	0	1	2	3	4	5	6	7	8	9	10
J = 0	0	0	0	0	0	0	0	0	0	0	0
J = 1	0	0	0	0	0	0	30	30	30	30	30
J = 2	0	0	0	14	14	14	30	30	30	44	44
J = 3	0	0	0	14	16	16	30	30	30	44	46
J = 4	0	0	9	14	16	23	30	30	39	44	46

K(ω,j) = max{k(w,j-1),k(w-wj,j-1)+vj} ={30,30+14}={44}

K(ω,j) = max{k(w,j-1),k(w-wj,j-1)+vj}={44,30+16} ={46}

6.2 Write the algorithm of knapsack without repetition using dynamic programming.

Solution: -

```
Knapsack(v, w, n ,W)
{
        for( w = 0 to W) V[0,w] = 0;
        for(i = 1to n)
          for(w = 0 to W)
          if(w[i] ≤ w)
                V[i, w] = max{V[i-1,w],v[i] + V[i-1, w-w[i]]};
          else
                V[i,w] = V[i – 1,w];
        return V[n,W];
}
```

Clearly time complexity O(nW)

7.1 Devise an algorithm in a pseudo code that takes a sequence $x[1, \cdots, n]$ and returns the length of the longest palindromic subsequence using a 2-dimensional table. Its running time should be O (n^2)

Solution: -

```
class Main
{
        public static String longestPalindrome(String X, String Y, int m, int n, int[][] T)
        {
          if (m == 0 || n == 0) {
                return "";
          }
          if (X.charAt(m - 1) == Y.charAt(n - 1))
          {
                return longestPalindrome(X, Y, m - 1, n - 1, T) + X.charAt(m - 1);
          }

          if (T[m - 1][n] > T[m][n - 1]) {
                return longestPalindrome(X, Y, m - 1, n, T);
          }
          return longestPalindrome(X, Y, m, n - 1, T);
        }
        // Function to find length of LCS of substring X[0..n-1] and Y[0..n-1]
```

```
        public static int LCSLength(String X, String Y, int n, int[][] T)
        {
          for (int i = 1; i <= n; i++)
          {
                for (int j = 1; j <= n; j++)
                {
                      // if current character of X and Y matches
                      if (X.charAt(i - 1) == Y.charAt(j - 1)) {
                            T[i][j] = T[i - 1][j - 1] + 1;
                      }
                      // else if current character of X and Y don't match
                      else {
                            T[i][j] = Integer.max(T[i - 1][j], T[i][j - 1]);
                      }
                }
          }
          return T[n][n];
        }
        public static void main(String[] args)
        {
          String X = "ACGTGTCAAAATCG";
          String Y = new StringBuilder(X).reverse().toString();
          int[][] T = new int[X.length() + 1][X.length() + 1];
          System.out.println("The length of Longest Palindromic Subsequence is "+
LCSLength(X, Y, X.length(), T));
          System.out.println("The Longest Palindromic Subsequence is " +
longestPalindrome(X, Y, X.length(), X.length(), T));
        }
}
```

7.2 Show its running time is $O(n^2)$.

Solution: - We build a 2-dimensional table T, where the entry T [i, j] stores the longest palindrome in the string x_i . . . . x_j . Now we write a recursive definition of T [i, j]. If symbols x_i and x_j are same, we can assume that they are part of the longest palindrome. So, T [i, j] will be equal to T [i + 1, j − 1]. If the two symbols are different, then they both cannot be part of any palindrome. So, T [i, j] will be maximum of T [i + 1, j] and T [i, j − 1].
Thus, we get

$$T[i,j] = \begin{cases} T[i+1, j-1\,] & if\ x_i = x_j \\ \max(T[i+1,j], T[i,j-1]) & otherwise \end{cases}$$

The base case happens when i = j, in which case the answer is 1.The for loops are:

for i = 1 to n ⟶ n + 1

T [i, i] = 1; ⟶ n

for i=n-1 down to 1 ⟶ n + 1

for j = i+1 to n ⟶ n * (n + 1)

T [i, j] = as mentioned above ⟶ n * n

The algorithm returns T [1, n].

$\therefore 2n^2 + 4n + 2$

Then Finally O (n^2)

*** END ***