KIIT Deemed to be University
Online End Semester Examination(Autumn Semester-2020)

Subject Name & Code: Design & Analysis of Algorithms (CS-2012)
Applicable to Courses: CSE, IT, CSCE, CSSE, ECS

Full Marks=50 **Time:2 Hours**

DAA SOLUTION & EVALUATION SCHEME

SECTION-A
(Answer All Questions. Each question carries 2 Marks)

Time:30 Minutes **(7×2=14 Marks)**

Question No	Question Type	Question	Answer Key
Q.No: 1	MCQ	Given two functions as $f(n)=n^3+n\log n$, $g(n)=n^3+n^2\log n$. Indicate whether f(n) is O, Ω or Θ of g(n). Note that zero, one or more of these relations may hold for this pair of list of all correct ones. A. O, Ω , Θ B. O C. Ω D. O, Ω	A
	MCQ	Given two functions as $f(n)=\sqrt{\log n}$, $g(n)=\log\sqrt{n}$. Indicate whether f(n) is O, Ω or Θ of g(n). Note that zero, one or more of these relations may hold for this pair of list of all correct ones. A. O, Ω , Θ B. O C. Ω D. O, Ω	B
	MCQ	Given two functions as $f(n)=2^n+1$, $g(n)=2^{n/2}+1$. Indicate whether f(n) is O, Ω or Θ of g(n). Note that zero, one or more of these relations may hold for this pair of list of all correct ones. A. O, Ω , Θ B. O C. Ω D. O, Ω	C
	MCQ	Given two functions as $f(n)=2^n+n$, $g(n)=2^{2n}+n$. Indicate whether f(n) is O, Ω or Θ of g(n). Note that zero, one or more of these relations may hold for this pair of list of all correct ones. A. O, Ω , Θ B. O C. Ω D. O, Ω	B

Q.No:			
Q.No: 2	**MCQ**	Consider the following function: int fun(int n) { int i, j, k = 0; for (i = 1; i <= n; i++) for (j = 1; j <= n; j = j * 2) k = k + 1/n; return k; } What is the returned value of the above function? A. $\Theta(1)$ B. $\Theta(\log n)$ C. $\Theta(n)$ D. $\Theta(n^2)$	**B**
	MCQ	Consider the following function: int fun(int n) { int i, j, k = 0; for (i = n; i<n; i=i/2) for (j = 1; j <= n; j++) k = k + 1/n; return k; } What is the returned value of the above function? A. $\Theta(1)$ B. $\Theta(\log n)$ C. $\Theta(n)$ D. $\Theta(n^2)$	**A**
	MCQ	Consider the following function: int fun(int n) { int i, j, k = 0; for (i = 1; i>=1; i--) for (j = 1; j <= n; j++) k = k + n/2; return k; } What is the returned value of the above function? A. $\Theta(1)$ B. $\Theta(\log n)$ C. $\Theta(n)$ D. $\Theta(n^2)$	**D**
	MCQ	Consider the following function: int fun(int n) { int i, j, k = 0; for (i = n; i>=1; i--) for (j = 1; j <= n; j++) k = k + 1/n; return k; } What is the returned value of the above function? A. $\Theta(1)$ B. $\Theta(\log n)$ C. $\Theta(n)$ D. $\Theta(n^2)$	**C**

Q.No: 3	**MCQ**	What is the contents of the array just before the final merge procedure of merge sort is applied to the array {1, 3, 6, 5, 7, 8, 4}. The division point in merge sort is (lb+ub)/2, where lb is the lower bound & ub is the upper bound of the array. A. {1, 3, 5, 6, 4, 7, 8} B. {1, 3, 6, 4, 5, 7, 8} C. {1, 3, 6, 4, 7, 5, 8} D. {1, 3, 4, 5, 6, 7, 8}	**A**
	MCQ	What is the contents of the array just before the final merge procedure of merge sort is applied to the array {1, 3, 6, 7, 5, 8, 4}. The division point in merge sort is (lb+ub)/2, where lb is the lower bound & ub is the upper bound of the array. A. {1, 3, 5, 6, 4, 7, 8} B. {1, 3, 6, 4, 5, 7, 8} C. {1, 3, 6, 7, 4, 5, 8} D. {1, 3, 4, 5, 6, 7, 8}	**C**
	MCQ	What is the contents of the array just before the final merge procedure of merge sort is applied to the array {1, 3, 4, 5, 6, 7, 8}. The division point in merge sort is (lb+ub)/2, where lb is the lower bound & ub is the upper bound of the array. A. {1, 3, 5, 6, 4, 7, 8} B. {1, 3, 6, 4, 5, 7, 8} C. {1, 3, 6, 7, 4, 5, 8} D. {1, 3, 4, 5, 6, 7, 8}	**D**
	MCQ	What is the contents of the array just before the final merge of merge sort is applied to the array {1, 3, 4, 6, 8, 7, 5}. The division point in merge sort is (lb+ub)/2, where lb is the lower bound & ub is the upper bound of the array. A. {1, 3, 5, 6, 4, 7, 8} B. {1, 3, 4, 6, 5, 7, 8} C. {1, 3, 6, 4, 5, 7, 8} D. {1, 3, 4, 7, 6, 5, 8}	**B**
Q.No: 4	**MCQ**	Quick Sort uses A. Divide & Conquer Strategy B. Back Tracking C. Greedy Approach D. Dynamic Programming	**A**
	MCQ	LCS uses A. Divide & Conquer Strategy B. Back Tracking C. Greedy Approach D. Dynamic Programming	**D**
	MCQ	Activity Selection Problem uses A. Divide & Conquer Strategy B. Back Tracking C. Greedy Approach D. Dynamic Programming	**C**
	MCQ	Dijkestra's algorithm uses A. Divide & Conquer Strategy B. Back Tracking C. Greedy Approach D. Dynamic Programming	**C**

Q.No: 5	**MCQ**	Given items as {value, weight} pairs {{60,20},{50,25},{20,5}}. The capacity of knapsack=10. Find the maximum value output assuming items to be divisible and nondivisible respectively. A. 35, 20 B. 50, 20 C. 90, 80 D. 100, 80	**A**
	MCQ	Given items as {value, weight} pairs {{60,20},{50,25},{20,5}}. The capacity of knapsack=15. Find the maximum value output assuming items to be divisible and nondivisible respectively. A. 35, 20 B. 50, 20 C. 90, 80 D. 100, 80	**B**
	MCQ	Given items as {value, weight} pairs {{60,20},{50,25},{20,5}}. The capacity of knapsack=30. Find the maximum value output assuming items to be divisible and nondivisible respectively. A. 35, 20 B. 50, 20 C. 90, 80 D. 100, 80	**C**
	MCQ	Given items as {value, weight} pairs {{60,20},{50,25},{20,5}}. The capacity of knapsack=35. Find the maximum value output assuming items to be divisible and nondivisible respectively. A. 35, 20 B. 50, 20 C. 90, 80 D. 100, 80	**D**
Q.No: 6	**MCQ**	What is the minimum number of scalar multiplications needed to evaluate a matrix-chain product with the sequence of dimensions <2, 2, 2, 2, 2>? A. 24 B. 32 C. 40 D. 64	**A**
	MCQ	What is the minimum number of scalar multiplications needed to evaluate a matrix-chain product with the sequence of dimensions <2, 2, 2, 2, 2, 2>? A. 24 B. 32 C. 40 D. 64	**B**
	MCQ	What is the minimum number of scalar multiplications needed to evaluate a matrix-chain product with the sequence of dimensions <2, 2, 2, 2, 2, 2, 2>? A. 24 B. 32 C. 40 D. 64	**C**

	MCQ	What is the minimum number of scalar multiplications needed to evaluate a matrix-chain product with the sequence of dimensions <4, 4, 4, 4, 4>? A. 32 B. 64 C. 192 D. 256	**C**
Q.No: 7	**MCQ**	Let X be a problem that belongs to the class NP. Then which one of the following is TRUE? A. There is no polynomial time algorithm for X. B. If X can be solved deterministically in polynomial time, then P = NP. C. If X is NP-hard, then it is NP-complete. D. X may be undecidable	**C**
	MCQ	__________ is the class of decision problems that can be solved by non-deterministic polynomial algorithms? A. NP B. P C. Hard D. Complete	**A**
	MCQ	How many conditions have to be met if an NP- complete problem is polynomially reducible? A. 1 B. 2 C. 3 D. 4	**B**
	MCQ	A problem which is both ________ and __________ is said to be NP complete. A. NP, NP hard B. P, P complete C. Hard, Complete D. NP, P	**A**

SECTION-B

(Answer Any Three Questions. Each Question carries 12 Marks)

Time: 1 Hour and 30 Minutes **(3×12=36 Marks)**

Question No		Question
Q. No: 8	a)	**Write the algorithm for insertion sort and use step count method to analyze its time complexity.**

Evaluation Scheme

- Insertion sort algorithm : 6 Marks
- Analysis of time complexity: 6 Marks

Answer/Solution

Line No.	Insertion Sort Algorithm	Cost	Times
1	INSERTION-SORT(A)	0	
2	{	0	
3	for j←2 to length[A]	c1	n
4	{	0	
5	key←A[j]	c2	n-1
6	//Insert A[j] into the sorted sequence A[1..j-1]	0	
7	i←j-1	c3	n-1
8	while(i>0 and A[i]>key)	c4	$\sum_{j=2}^{n} tj$
9	{	0	
10	A[i+1]←A[i]	c5	$\sum_{j=2}^{n} (tj-1)$
11	i←i-1	c6	$\sum_{j=2}^{n} (tj-1)$
12	}	0	
13	A[i+1]←key	c7	n-1
14	}	0	
15	}	0	

Analysis of Time Complexity of Insertion Sort Algorithm

To compute T(n), the running time of INSERTION-SORT on an input of n values, we sum the products of the cost and times column, obtaining

$T(n)=c1n + c2(n-1) + c3(n-1) + c4\sum_{j=2}^{n} tj + c5\sum_{j=2}^{n} (tj-1) + c6\sum_{j=2}^{n} (tj-1) + c7(n-1)$ ………… (1)

Best case Analysis:

- Best case occurs if the array is already sorted.
- For each j=2 to n, we find that A[i]≤key in line number 8 when i has its initial value of j-1. Thus tj=1 for j=2 to n and the best case running time is

  $T(n)=c1n + c2(n-1) + c3(n-1) + c4(n-1) + c7(n-1) = O(n)$

Worst case Analysis:

- Worst case occurs if the array is already sorted in reverse order
- In this case, we compare each element A[j] with each element in the entire sorted subarray A[1..j-1], so tj=j for j=2 to n.
- The worst case running time is

$T(n)=c1n + c2(n-1) + c3(n-1) + c4(n(n+1)/2-1) + c5(n(n-1)/2) + c6(n(n-1)/2) + c7(n-1)=O(n^2)$

	b)	**Consider the following function** **int fun(int n)** **{** **if (n<=1)** **return 1;** **else** **return fun(n-1) + fun(n-2);** **}** **a) What does the above function compute?** **b) Executing the function for n=6 results in the function being recursively invoked with the arguments n=1, 2, 3, 4 and 5. Draw a recurrence tree to illustrate this fact.** **c) How many additions are performed to compute fun(6)?** **d) Assuming that each addition taken constant time, write a recurrence relation for the running time of fun(n) & solve the recurrence.**

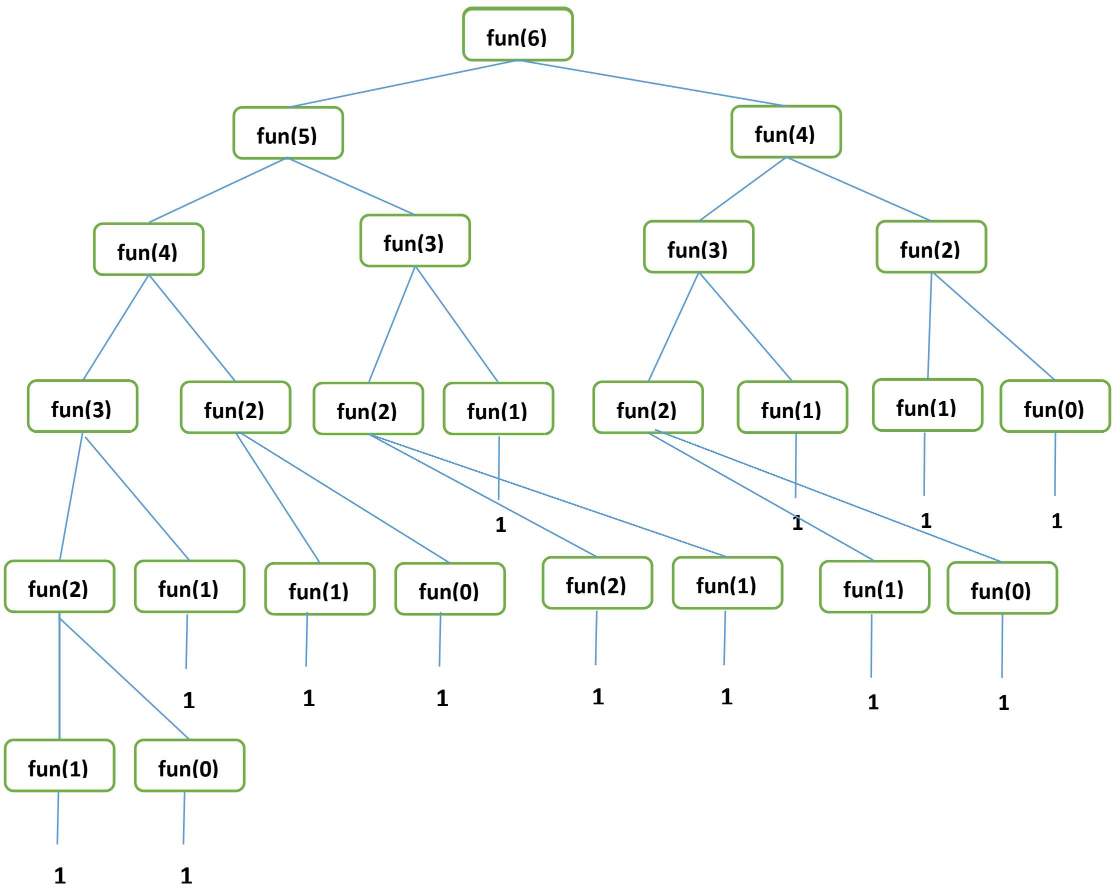
Evaluation Scheme

- Part-a: Mentioning the task of function fun(): 2 Marks
- Part-b: Drawing recurrence tree for function fun(6): 4 Marks
- Part-c: Mentioning number of additions performed by calling fun(6): 2 Marks
- Part-d: Writting a recurrence relation for the running time of fun(n) & solving the recurrence: 4 Marks

Answer/Solution

a) The function fun() computes the (n-1)th fibanacci number in the fibanacci sequence starting from 1, 1, 2, 3,……

b) Recurrence Tree for fun(6)



c) Total number of additions are performed to compute fun(6) = 12

d) Assuming that each addition taken constant time, the recurrence relation for the running time of fun(n) is as follows:

$$T(n) = T(n-1) + T(n-2) + 1 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\text{(I)}$$

Now the recurrence is solved as follows:

Establishing a lower bound by approximating that T(n-1) ~ T(n-2) , though T(n-1) ≥ T(n-2), hence lower bound, the recurrence eq-1 becomes

$T(n) = T(n-2) + T(n-2) + 1$
$= 2T(n-2) + 1$
$= 2\{2T(n-4) +1\} + 1$
$= 2^2T(n-4) +3$
$= 2^2\{2T(n-6) +1\} + 3$
$= 2^3T(n-6) + 7$
$= 2^3\{2T(n-8) + 1\} + 7$
$= 2^4T(n-8) + 15$
…...
…...
…...
$= 2^iT(n-2*i) + 2^i-1$

To find out the value of i for which: $n - 2*i = 0 \Rightarrow i = n/2$
$= 2^{n/2}T(0) + 2^{n/2}-1$
$= 2^{n/2}x1 + 2^{n/2}-1$
$= 2^{n/2} + 2^{n/2}-1 \sim O(2^{n/2})$

Establishing an upper bound by approximating that T(n-2) ~ T(n-1) , though T(n-1) ≥ T(n-2), hence upper bound, the recurrence eq-1 becomes

$T(n) = T(n-1) + T(n-1) + 1$
$= 2T(n-1) + 1$
$= 2\{2T(n-2) +1\} + 1$
$= 2^2T(n-2) +3$
$= 2^2\{2T(n-3) +1\} + 3$
$= 2^3T(n-3) + 7$
$= 2^3\{2T(n-4) + 1\} + 7$
$= 2^4T(n-4) + 15$
…...
…...
…...
$= 2^iT(n-i) + 2^i-1$

To find out the value of i for which: $n - i = 0 \Rightarrow i = n$
$= 2^nT(0) + 2^n-1$
$= 2^nx1 + 2^n-1$
$= 2^n + 2^n-1 \sim O(2^n)$

Hence, **the time complexity of function fun() in worst case = $O(2^n)$**

c) **Deduce the running time T(n) in asymptotic Θ-notation for all of these recurrences using the method mentioned against each case:**

(i) $T(n) = 3T(n/3) + n$ if $n > 3$ and $T(1) = 1$ (Use Master Theorem)
(ii) $T(n) = \sqrt{n}\, T(\sqrt{n}) + n \log n$, if $n > 2$ and $T(2) = 2$
(Use repeated Recurrence Relation)
(iii) $T(n) = T(2n/3) + T(n/3) + n$, if $n > 1$ and $T(1) = 1$ (Use Recurrence Tree)

<u>Evaluation Scheme</u>

- Solving each recurence by the specific procedure as directed: 4 Marks
- Solving each recurrence other than specified method: 2 Marks

Answer/Solution

(i) $T(n) = 3T(n/3) + n$ if $n > 3$ and $T(1) = 1$ (Use Master Theorem)

Solution to the recurrence => $T(n)= \Theta(n \log n)$

Explanation

<table>
<tr><th colspan="2">Type:-1 (Master Theorem as per CLRS)</th></tr>
<tr><th>Master Theorem</th><th>Solution of recurrence
$T(n) = 3T(n/3) + n$</th></tr>
<tr><td>The Master Theorem applies to recurrences of the following form:
$T(n) = aT(n/b) + f(n)$
where $a \ge 1$ and $b > 1$ are constants and f(n) is an asymptotically positive function. T(n) is defined on the non-negative integers by the recurrence.
T(n) can be bounded asymptotically as follows: There are 3 cases:
a) Case-1: If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
b) Case- 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
c) Case-3: If $f(n) = \Omega(n^{\log_b a+\epsilon})$ with $\epsilon > 0$, and : $af(n/b) \le cf(n)$, then $T(n) = \Theta(f(n))$, for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$</td><td>Given, a=3, b=3, f(n)=n
<u>Step-1 (Guess)</u>
$n^{\log_b a}=n^{\log_3 3}=n$, Comparing $n^{\log_b a}$ with f(n), f(n) is found same as $n^{\log_b a}$. So case-2 of master theorem is guessed.

<u>Step-2</u>
As per case-2 of master theorem,
So the solution is
<u>$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n \log n)$</u></td></tr>
<tr><th colspan="2">Type:-2 (Master Theorem)</th></tr>
<tr><th>Master Theorem</th><th>Solution of recurrence
$T(n) = 3T(n/3) + n$</th></tr>
<tr><td>If the recurrence is of the form
$T(n) = aT(n/b) + n^k \log^p n$,
where $a \ge 1$, $b > 1$, $k \ge 0$ and p is a real number, then comapre a with b^k and conclude the solution as per the following cases.

Case-1: If $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$

Case-2: If $a = b^k$, then
a) If $p > -1$, then
$T(n) = \Theta(n^{\log_b a}\log^{p+1} n)$
b) If $p = -1$, then
$T(n) = \Theta(n^{\log_b a}\log\log n)$
c) If $p < -1$, then $T(n) = \Theta(n^{\log_b a})$

Case-3: If $a < b^k$, then
a) If $p \ge 0$, then $T(n) = \Theta(n^k \log^p n)$
b) If $p < 0$, then $T(n) = \Theta(n^k)$</td><td>Here, a=3, b=3, k=1, p=0
$b^k = 3^1 = 3$
Comparing a with b^k, we found a is same as b^k., so this will fit to case-2.
Now p=0, so case-2.a solution is the recurrence solution that is
If $p > -1$, then
$T(n) = \Theta(n^{\log_b a}\log^{p+1} n)$
$T(n)= \Theta(n \log n)$</td></tr>
</table>

(ii) $T(n) = \sqrt{n}\, T(\sqrt{n}) + n \log n$, if $n > 2$ and $T(2) = 2$
(Use repeated Recurrence Relation)

Explanation

$T(n) = \sqrt{n}\, T(\sqrt{n}) + n \log n$

$$T(n) = \sqrt{n}\, T(\sqrt{n}) + n \log n$$

$$= n^{\frac{1}{2}}\, T(n^{\frac{1}{2}}) + n \log n$$

$$= n^{\frac{1}{2}} \left\{ n^{\frac{1}{2^2}}\, T(n^{\frac{1}{2^2}}) + n^{\frac{1}{2}} \log n^{\frac{1}{2}} \right\} + n \log n$$

$$= n^{\frac{1}{2}+\frac{1}{2^2}}\, T(n^{\frac{1}{2^2}}) + n^{\frac{1}{2}+\frac{1}{2}} \log n^{\frac{1}{2}} + n \log n$$

$$= n^{\frac{1}{2}+\frac{1}{2^2}}\, T(n^{\frac{1}{2^2}}) + n \log n^{\frac{1}{2}} + n \log n$$

$$= n^{\frac{1}{2}+\frac{1}{2^2}} \left\{ n^{\frac{1}{2^3}}\, T(n^{\frac{1}{2^3}}) + n^{\frac{1}{2^2}} \log n^{\frac{1}{2^2}} \right\} + n \log n^{\frac{1}{2}} + n \log n$$

$$= n^{\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}}\, T(n^{\frac{1}{2^3}}) + n^{\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^2}} \log n^{\frac{1}{2^2}} + n \log n^{\frac{1}{2}} + n \log n$$

$$= n^{\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}}\, T(n^{\frac{1}{2^3}}) + n \log n^{\frac{1}{2^2}} + n \log n^{\frac{1}{2}} + n \log n$$

$$= n^{\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}}\, T(n^{\frac{1}{2^3}}) + n \log n \left\{ \frac{1}{2^2} + \frac{1}{2^1} + \frac{1}{2^0} \right\}$$

$$\vdots$$

$$= n^{\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}+\cdots+\frac{1}{2^i}}\, T(n^{\frac{1}{2^i}}) + n \log n \left\{ \frac{1}{2^{i-1}} + \cdots + \frac{1}{2^2} + \frac{1}{2^1} + \frac{1}{2^0} \right\}$$

Let $n^{\frac{1}{2^i}} = 2$ $\left(T(2) = 2 \text{ Given}\right)$

$\Rightarrow \boxed{i = \log\log n}$

$$\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}+\dots+\frac{1}{2^{\log\log n}} = \sum_{i=1}^{\log\log n}\left(\frac{1}{2}\right)^i$$

$$= \sum_{i=0}^{\log\log n}\left(\frac{1}{2}\right)^i - 1$$

GP Series.

$$\frac{1-\frac{1}{2^{\log\log n}}}{1-\frac{1}{2}} - 1 \Leftarrow$$

$$= 2\left(1-\frac{1}{2^{\log\log n}}\right) - 1 \simeq 1$$

$$\frac{1}{2^0}+\frac{1}{2^1}+\frac{1}{2^2}+\dots+\frac{1}{2^{\log\log n-1}}$$

$$= \sum_{i=0}^{\log\log n-1}\left(\frac{1}{2}\right)^i = \frac{1-\frac{1}{2^{\log\log n}}}{1-\frac{1}{2}} = \frac{2\left(2^{\log\log n}-1\right)}{2^{\log\log n}}$$

$$\simeq 1$$

So,

$$= n^1\, T(2) + n\log n \{1\}$$

$$= n \times 2 + n\log n$$

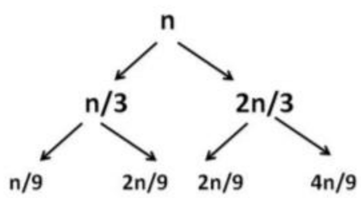
$$= O(n\log n)$$

(iii) T(n) = T(2n/3) + T(n/3) + n, if n > 1 and T(1) = 1 (Use Recurrence Tree)

Solution to the recurrence => T(n)= Θ(n log n)

<u>Explanation</u>

In the recurrence tree, the leaves are between the levels $\log_3 n$ and $\log_{3/2} n$. From the computation tree, it is clear that the maximum height is $\log_{3/2} n$. Therefore, the cost is at most $\log_{3/2} n \cdot cn = O(n \log n)$. Similarly, the minimum height is $\log_3 n$. Therefore, the cost is at least $\log_3 n \cdot cn = \Omega(n \log n)$. Thus, $T(n) = \Theta(n \log n)$



(Recurrence Tree)

Q. No : 9	a)	**Given a sorted array with n distinct elements, we are required to find three such distinct elements whose sum is equal to x. Design an $O(n^2)$ algorithm that implements the above mentioned requirements and analyze its time complexity.**

Evaluation Scheme

- Correct Algorithm with $O(n^2)$ Time Complexity: 12 Marks
- Other Correct Algorithm more than $O(n^2)$: 8 Marks
- Incorrect algorithm, but some valid steps: step marks (1-5 Marks)

Answer/Solution

```
void findTripletsSumX(int a[], int n)
{
   int i, j, k;
   for(i = 0; i < n-2; i++)
   {
       // index of the first element in the remaining elements.
       j = i + 1;
      // index of the last element.
       k = n - 1;
      while(j < k)
      {
              if ((a[i] + a[j]) + a[k] = x)
              {
                   printf("\n %d + %d + %d = %d", a[i], a[j], a[k], x);
                   return;
              }
              else if ((a[i] + a[j]) + a[k] > x)
                      k--;
              else
                      j++;
        }
    }
    printf("\nNo three elements are found, whose sum is equal to %d.", x);
 }
```

Time Complexity = $O(n^2)$

b)

a) Write a pseudocode for a divide-and-conquer algorithm for the exponentiation problem of computing a^n where a > 0 and n is a positive integer.

b) Set up and solve a recurrence relation for the number of multiplications made by this algorithm.

c) How does this algorithm compare with the brute-force algorithm for this problem?

Evaluation Scheme

- Part-a: Writing correct psedocode for computing a^n by divide-and-conquer algorithm : 4 Marks
- Part-b: Set up and solve a recurrence relation for a^n: 4 Marks
- Part-c: Comparision of divide-conquer approach with brute-force approach for solving exponentiation problem of computing a^n : 4 Marks

Answer/Solution

Part-a

```
//Computes a^n by a divide-and-conquer algorithm
//Input: A positive number a and a positive integer n
//Output: The value of a^n
Algorithm DivConqPower(a, n)
{
    if (n = 1)
        return a;
    else return DivConqPower(a, n/2) * DivConqPower(a, n/2);
}_x0007_
```

Part-b

The recurrence for the number of multiplications is

$T(n) = 2T(n/2) + 1$ for $n > 1$, $T(1) = 0$

By solving the master theorem,

a=2, b=2, k=0, p=0

$b^k = 2^0 = 1$ => Comparing a with b^k, a is found greater than b^k. So it is Case-1 of master theorem. The solution is

$\mathbf{T(n) = \Theta(n^{\log_b a}) = \Theta(n)}$

Part-c

Comparision

- In brutefore approach the recurrence relation for a^n is $T(n) = T(n-1) + 1$ as an is thought be $a^n = a*a^{n-1}$, The solution is $T(n)=\Theta(n)$.
- By comparing in terms of time complexity, both exibits the same means the same number of multiplications.
- Though the algorithm makes the same number of multiplications as the brute-force method, it has to be considered inferior to the latter because of the recursion overhead.

c) A and B are playing a guessing game where B first thinks up an integer X (positive, negative or zero, and could be of arbitrarily large magnitude) and A tries to guess it. In response to A's guess, B gives exactly one of the following three replies:

i) Try a bigger number

ii) Try a smaller number or

iii) You got it.

Write a program by designing an efficient algorithm to minimize the number of guesses A has to make. An example (not necessarily an efficient one) below:

Let B thinks up the number 35

A's guess	B's response
10	Try a bigger number
20	Try a bigger number
30	Try a bigger number
40	Try a smaller number
35	You got it

Evaluation Scheme

- Solving the problem by Binary Search Algorithm : 12 Marks
- Solving other than binary search : 0-6 Marks

Answer/Solution

/*You think a number x in between low & high, where x, low & high are integers. Computer will gues it with minimum number of attempt with your three types of feedback suck as Try a bigger number or Try a smaller number or You got it after each attempt.*/

```
void BINARY-SEARCH-GUESS(int low, int high, int x)
{
    int ag;
    while(1)
    {
        printf("\nYou: Guess a number:");
        //Function to generate random number within a range
        ag=RANDOM-NUM(low, high);
        printf("\nComputer: The number is %d.", ag);
        if (ag==x)
        {
          printf("\nYou: You Got it");
          break;
        }
        else if(ag<x)
        {
          printf("\nYou: Try a bigger number ");
          low=ag;
        }
       }
       else
       {
         printf("\nTry a smaller number ");
         high=ag;
       }
    }
}
```

Q. No: 10 **a)** **Given two halls and 10 activities, A=<A1, A2,....,A10> along with their start time (si) and finish time (fi) as si=<2, 3, 2, 8, 5, 7, 6, 4, 7, 2 > and fi=<3, 4, 5, 9, 6, 8, 7, 7, 9, 5>. Determine an efficient algorithm & apply to the above data where largest number of activities can be scheduled in these two halls.**

Evaluation Scheme

- Greedy Activity Selector Algorithm: 6 Marks
- Solution to the problem with proper explanation: 6 Marks

Answer/Solution

GREEDY-ACTIVITY-SELECTOR-MODIFIED (s, f, status)	TWO-STAGE-GREEDY-ACTIVITY-SELECTOR(s, f)
{ n ← length[s] //Selection of first activity of an optimal set of a stage for k ← 1 to n { if (status[k] ==0) A ← {ak} i ← k status[k]=1; break; } //Selection of rest activities of an optimal set of a stage for m ← i+1 to n	{ Sort the activities as per their finishing times //Initially no activity is selected for k ← 1 to n { status[k] =0; } for i ← 1 to 2 { A ← GREEDY-ACTIVITY-SELECTOR-MODIFIED (s, f, status) print Stage-i Optimal Activity Set: A } }

{ if (status[m]==0 and sm ≥ fi) { A ← A U {am} i ← m status[m]=1; } } return A }	

Given,

ai =<A1, A2, A3, A4, A5, A6, A7, A8, A9, A10>
si =<2, 3, 2, 8, 5, 7, 6, 4, 7, 2 >
fi =<3, 4, 5, 9, 6, 8, 7, 7, 9, 5>

Arranging the activities in increasing order with their finishing time.

a_i	s_i	f_i	First Stage Selection (Yes/No)	Second Stage Selection (Yes/No)
A1	2	3	√	
A2	3	4	√	
A3	2	5	x	√
A10	2	5	x	x
A5	5	6	√	
A7	6	7	√	
A8	4	7	x	x
A6	7	8	√	
A4	8	9	√	
A9	7	9	x	√

Optimal Schedule: Stage-1 = <A1, A2, A5, A7, A6, A4>
Stage-II = <A3, A9>

b) **PRINT-LCS() function is used to print LCS of two given sequences X & Y with lenths m & n respectively. Write an algorithm by using PRINT-LCS() function to print all possible LCS & apply to the following data.**
X=<0,1,0,0,1,1,0>
Y=<1,1,0,1,0,1>

<u>Evaluation Scheme</u>

- Correct Algorithm to print all possible LCS with PRINT-LCS function : 6 Marks
- Finding all possible LCS : 6 Marks
- Finding only one LCS: 3 Marks

<u>Answer/Solution</u>

Algorithm	PRINT-LCS-ALL(b, X, i, j)
1.Run LCS-LENGTH(X, Y); 2.Call PRINT-LCS-ALL(b, X, i, j); 3.Run LCS-LENGTH(Y, X); 4.Call PRINT-LCS-ALL(b, X, i, j);	{ length = b[i][j]; while (j>0 and b[i][j]==length) { PRINT-LCS(b, X, i, j); j--; } }

<u>Given</u>
X=<0,1,0,0,1,1,0>
Y=<1,1,0,1,0,1>

Index	**j**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
i		y_j	**1**	**1**	**0**	**1**	**0**	**1**
0	x_i	0	0	0	0	0	0	0
1	**0**	0	0^	0^	1\	1<	1\	1\
2	**1**	0	1\	1\	1^	2\	2<	2\
3	**0**	0	1^	1^	2\	2^	3\	3<
4	**0**	0	1^	1^	2\	2^	3\	3^
5	**1**	0	1\	2\	2^	3\	3^	4\
6	**1**	0	1\	2\	2^	3\	3^	4\
7	**0**	0	1^	2^	3\	3^	4\	4^

By calling PRINT-LCS(b, X, 7, 6) => **LCS-1:** 0101
PRINT-LCS(b, X, 7, 5) => **LCS-2:**1010

<u>Given</u>
X=<0,1,0,0,1,1,0>
Y=<1,1,0,1,0,1>

Index	**j**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
i		x_j	**0**	**1**	**0**	**0**	**1**	**1**	**0**
0	y_i	0	0	0	0	0	0	0	0
1	**1**	0	0^	1\	1<	1<	1\	1\	1<
2	**1**	0	0^	1\	1^	1^	2\	2\	2<
3	**0**	0	1\	1^	2\	2\	2^	2^	3\
4	**1**	0	1^	2\	2^	2^	3\	3\	3^
5	**0**	0	1\	2^	3\	3\	3^	3^	4\
6	**1**	0	2^	2\	3^	3^	4\	4\	4^

By calling PRINT-LCS(b, X, 6, 7) => **LCS-3:** 1110
PRINT-LCS(b, X, 6, 6) => **LCS-4:**1011
PRINT-LCS(b, X, 6, 5) => **LCS-5:**1001

Algorithm **For Computing a c table for getting the length of an LCS & b table for printing the LCS**	**Algorithm** **For Constructing an LCS**
LCS-LENGTH (X, Y) { m ← length[X] n ← length[Y] for i ← 0 to m c[i][0] ← 0 for j ← 1 to n c[0][j] ← 0 for i ← 1 to m { for j ← 1 to n { if (xi == yj) { c[i][j] ← c[i - 1][j - 1] + 1 b[i, j] ← "\\" } else if c[i - 1][j] ≥ c[i][j - 1] { c[i][j] ← c[i - 1][j] b[i, j] ← "^" } else { c[i][j] ← c[i][j-1] b[i][j] ← "<" } } //end for inner for } //end of outer for return c and b }	PRINT-LCS(b, X, i, j) { if i = 0 or j = 0 then return if b[i][j] = "\\" { PRINT-LCS(b, X, i - 1, j - 1) print xi } elseif b[i][j] = "^" PRINT-LCS(b, X, i - 1, j) else PRINT-LCS(b, X, i, j - 1) }

c) **Suppose a file contains 1 lakh characters & the characters in the data occur with following frequencies.**

character	**frequency**
a	**38,000**
b	**14,000**
c	**11,000**
d	**15,000**
e	**12,000**
f	**10,000**

Write Huffman code algorithm & apply on the above data to encode each character. A variable-length code can do considerably better than a fixed-length code. Justify your answer by taking the above data as an example.

Evaluation Scheme
- Huffman code algorithm : 4 Marks
- Huffman tree construction with the given data: 5 Marks
- Comparing huffman with fixed code: 3 Marks

Answer/Solution

Huffman Code Algorithm

```
HUFFMAN(C)
{
    n ← |C|
    Q ← C
    for I ← 1 to n - 1
      do allocate a new node z
           left[z] ← x ← EXTRACT-MIN (Q)
           right[z] ← y ← EXTRACT-MIN (Q)
           f [z] ← f [x] + f [y]
           INSERT(Q, z)
    return EXTRACT-MIN(Q) //Return the root of the tree.
}
```

Given Data

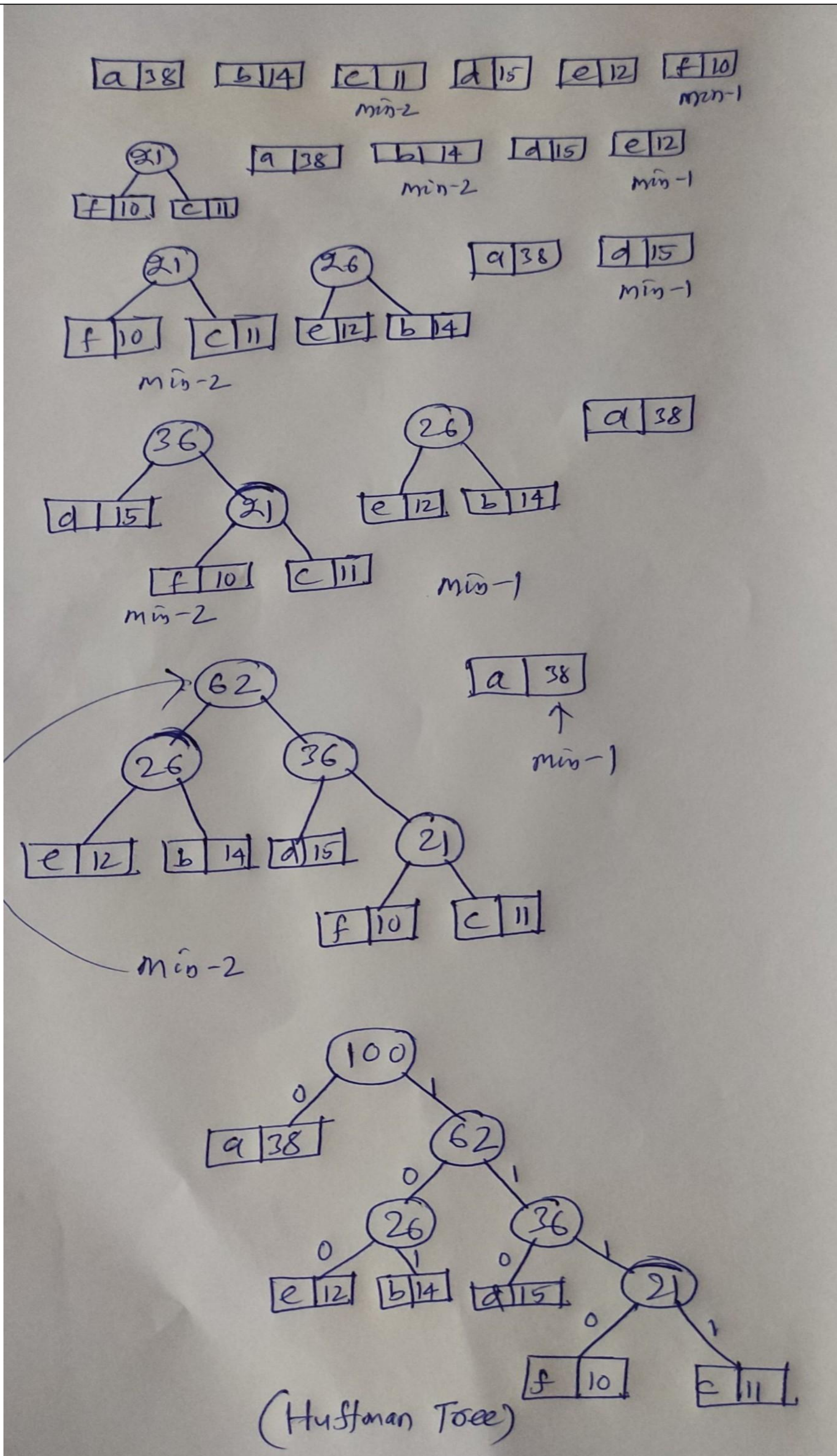
character	**frequency**	**Frequency/1000**
a	38,000	38
b	14,000	14
c	11,000	11
d	15,000	15
e	12,000	12
f	10,000	10

- Initially Construct a min heap by taking value as each character's frequency
- Each step extract two chacters with min. values (let min-1 & ymin-2 & add the values min-1 & min-2, let the addition is z. Construct a binary tree with z as the root with left child as min-1 & right child as min-2. After that insert z back to min priority queue.

Drawing of Huffman Tree &Codes

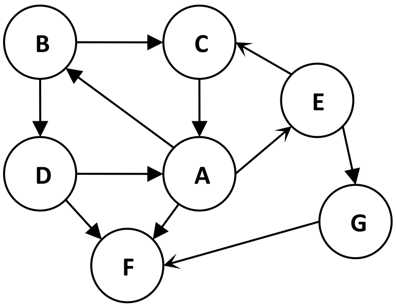
Sl. No.	Character	Frequency	Huffman Code
1	**a**	**38, 000**	**0**
2	**b**	**14, 000**	**101**
3	**c**	**11, 000**	**1111**
4	**d**	**15, 000**	**110**
5	**e**	**12, 000**	**100**
6	**f**	**10, 000**	**1110**

Number of bits required in Huffman Code (Variable code) = 38000*1 + 14000*3 +11000*4 + 15000*3 +12000*3 +10000*4 = 245000 bits
With Fixed code=100000 *3 = 300000 bits.
Comaring both Variable code is better.


a 38
b 14
c 11
d 15
e 12
f 10
min-2
min-1
21
26
36
62
100
0
1
(Huffman Tree)

Q. No: 11

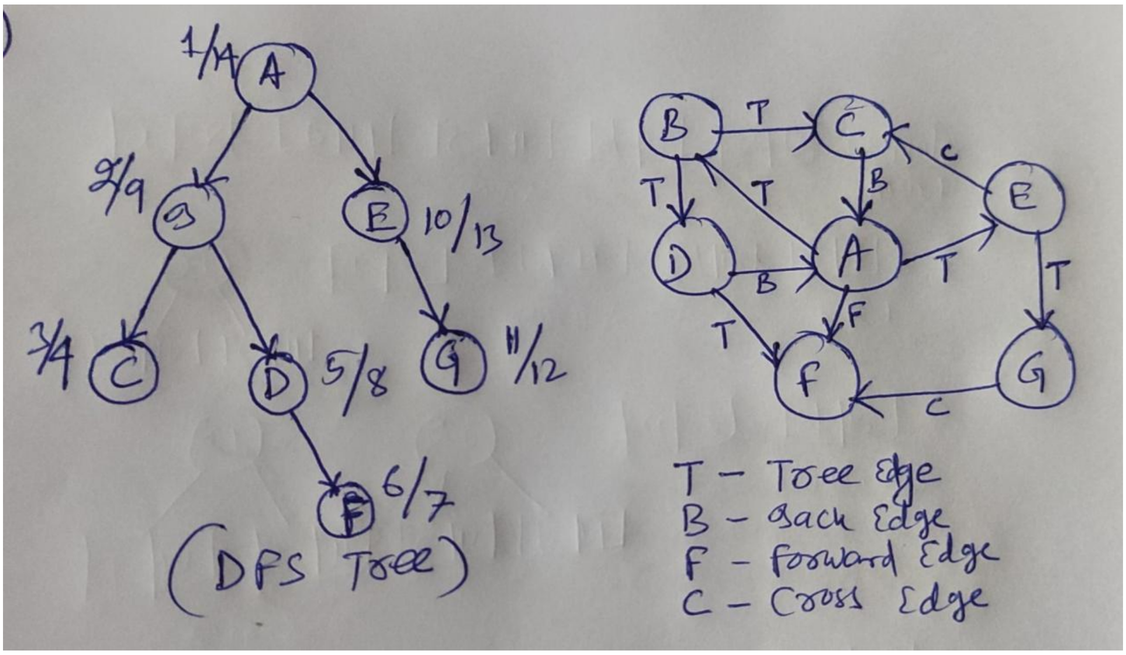
a) **Consider the adjacency list of the directed graph given below.**
A→B, E, F; B→C, D; C→A; D→A, F; E→G; F→NIL; G→F
Demonstrate the DFS (depth-first search) algorithm on it. Write the initialization and explain how the relevant parameters and data structures are updated during the execution. In the final step, you should write the DFS tree(s), and also the forward edges, cross edges, and back edges, if any. Use node 'A' as source node while answering the question.



Evaluation Scheme

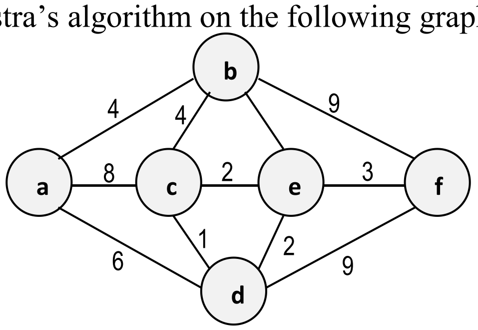
- Explanation of DFS algorithm through DFS tree/forest: 8 Marks
- Explanation/marking of tree edges, forward edges, cross edges, and back edges on final step of the DFS tree/forest or on the given graph: 4 Marks

Answer/Solution



DFS Sequencs: A, B, C, D, F, E, G

b) Execute Dijkstra's algorithm on the following graph where a is the source vertex.



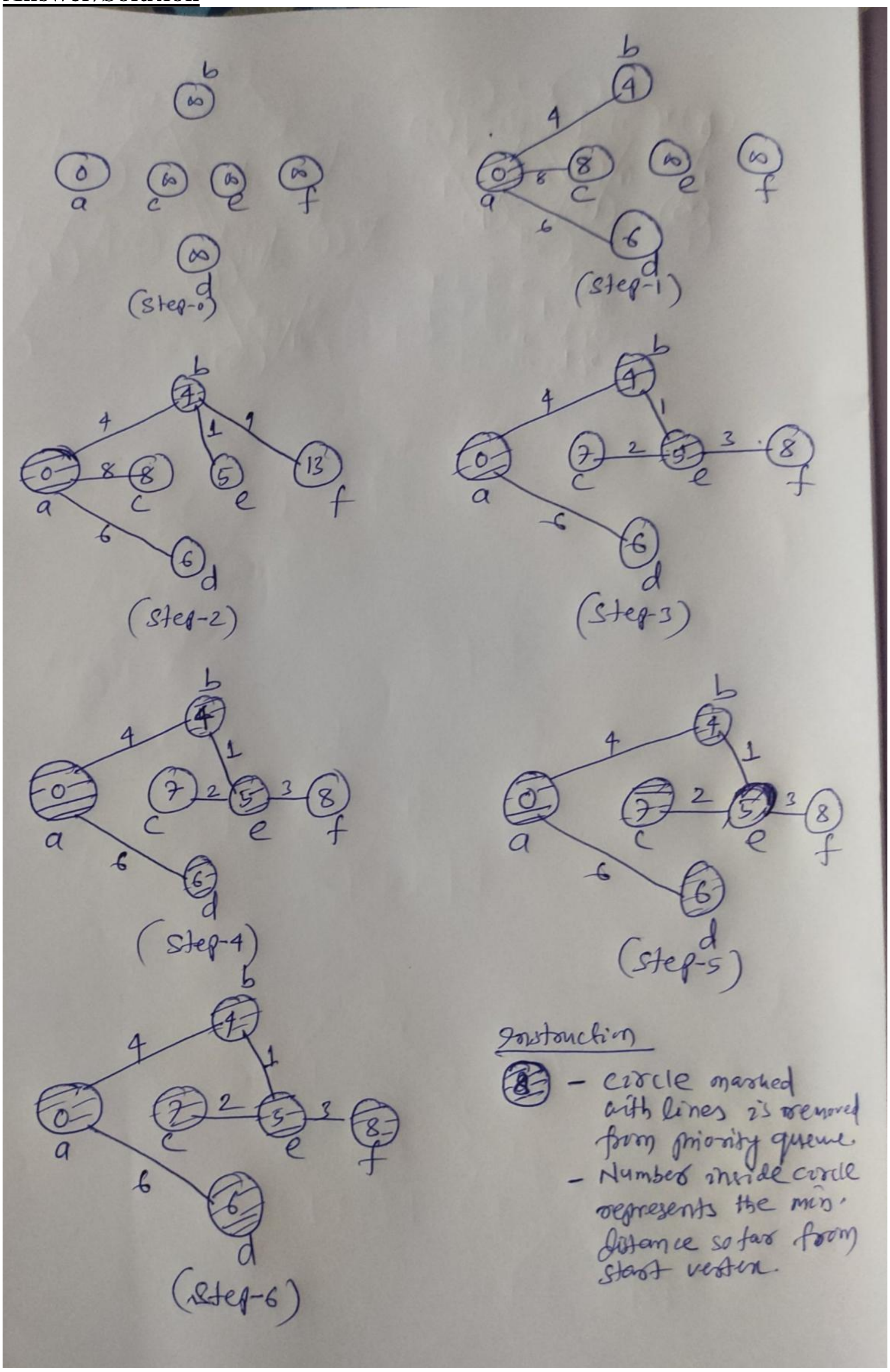
Show the intermediate steps through diagrams that will indicate the following.

a) The order in which the vertices are removed from the priority queue.
b) The final distance values d[] for each vertex.

Evaluation Scheme

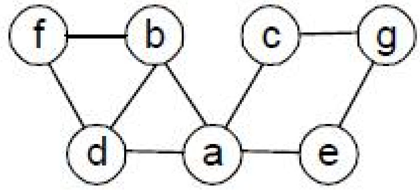
- Missing weight (b, e) will be assumed 1 (default). If any other value is assumed by the students, also will be accepted.Accordingly answer will vary.
- Execute Dijkstra's algorithm through step by step diagram: 12 Marks

Answer/Solution



.

c) **Consider the following graph:**



Consider the adjacency list of this graph given below.

a→b, c, d, e; b→a, d, f; c→a, g; d→a, b, f; e→a, g; f→b, d

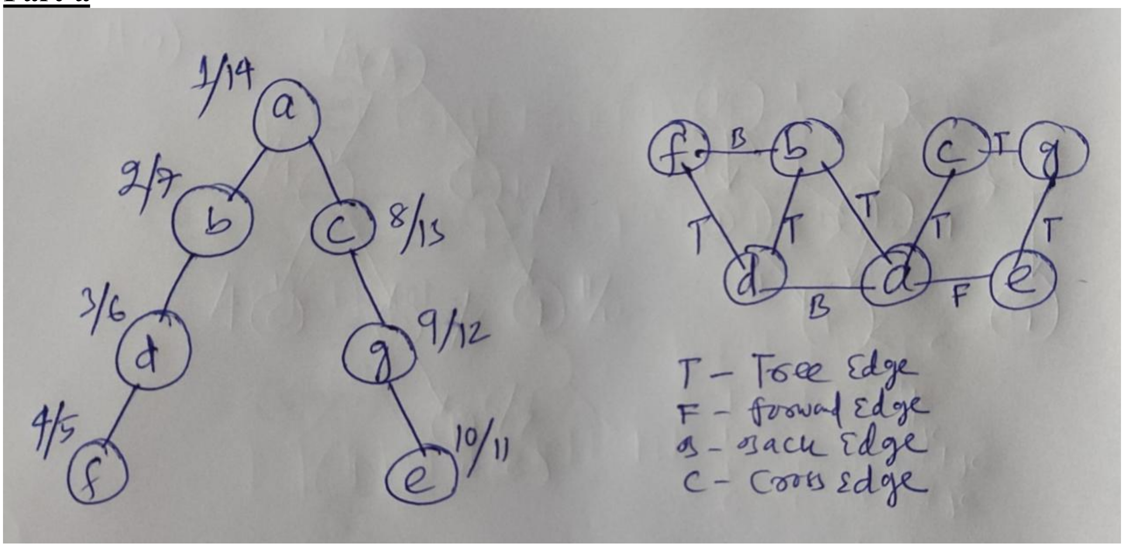
a) **Compute the DFS tree starting with vertex 'a' and draw the tree edges, forward edges, back edges and cross edges.**
b) **Write the order in which the vertices were reached for the first (i.e.pushed into the stack)**
c) **Write the order in which the vertices became dead ends (i.e. popped from the stack)**

<u>**Evaluation Scheme**</u>

- Compuing the DFS tree starting with vertex 'a' and draw the tree edges, forward edges, back edges and cross edges: 8 Marks
- Writing the correct order in which the vertices were reached for the first: 2 Marks
- Writing the correct order in which the vertices became dead ends: 2 Marks

<u>**Answer/Solution**</u>

<u>**Part-a**</u>



<u>**Part-b:**</u>

The correct order in which the vertices were reached for the first is as follows:

a, b, d, f, c, g, e

<u>**Part-c:**</u>

The correct order in which the vertices became dead ends is as follows:

f, d, b, e, g, c, a