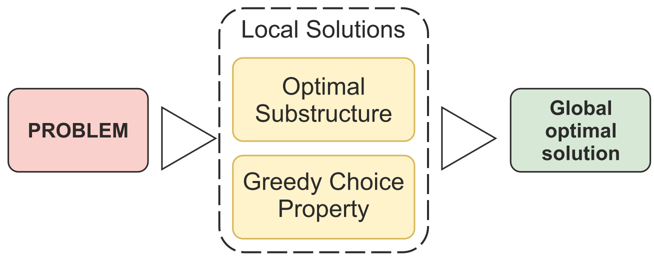
ACE
Engineering Publications
SINCE 2011

GATE | PSUs



Computer Science & Information Technology

ALGORITHMS

Text Book:
Theory with worked out Examples and Practice Questions

Algorithms

(Solutions for Text Book Practice Questions)

Chapter 1

Algorithm Analysis & Asymptotic Notations

01. Ans: (a)

Sol: $f_3 < f_2 < f_4 < f_1$

02. Ans: (b)

Sol: $\sqrt{\log n} > c.\log\log n$ for all n≥2 from the Big Oh definition $\sqrt{\log n} \neq O(\log\log n)$

(a) $100\ n\log n \le c.\ \frac{n\log n}{100}$ for $n \ge 1$

(c) $\underset{n-\infty}{\text{Lt}} \frac{n^x}{n^y} = 0$ Because $0 < x < y$

(d) $2n > c.\ n^k$ for $n \ge 1$ if $k = \frac{1}{c}$

03. Ans: (b)

Sol: $n(\log n)^{10} \le C.\ n^2 \log n$ for $n \ge 2^{52}$

04. Ans:(a)

Sol: I $\rightarrow$ $(n+k)^m \le C.n^m$ for $n \ge n_0$ and $(n+k)^m \ge C.n^m$ for $n \ge n_0$

$\Rightarrow (n+k)^m = \Theta(n^m)$

II $\rightarrow$ $2^{n+1} \le 3.2^n$ for $n \ge 1$

$\Rightarrow 2^{n+1} = O(2^n)$

III $\rightarrow$ $2^{2n} \nleq C.2^n$ for $n \ge 1$

05. Ans: (c)

Sol: When k = 6

$10n.\log_{10}^{n} = 60000000$

$0.0001n^2 = 1000000$

06. Ans: (a), (b) & (d)

Sol: $g_1(n) \le C.\ g_2(n)$ for $n \ge 101$

when $n > 100$, $g_1(n) = n^3$ upto $n \le 10{,}000$ and $g_1(n) = n^2$, $n \ge 10{,}000$

when $n > 100$, $g_2(n) = n^3$

$g_2(n)$ is always greater than or equal to $g_1(n)$.

07. Ans: (d)

Sol: $3n^{\sqrt{n}} \le c.n!$ for $n \ge 5$ from Big oh notation

$3n^{\sqrt{n}} = O(n!)$

$\rightarrow n! > 3n^{\sqrt{n}}$ for $n \ge 5$

h(n) is not O(f(n))

$\rightarrow n! > c.2^{\sqrt{n}\log_2 n}$ for $n \ge 4$

$\rightarrow 2^{\sqrt{n}\log_2 n} > c.3n^{\sqrt{n}}$ for $n \ge 4$

08. Ans: (d)

Sol: (a) $2^n \le n!\ \forall\ n \ge 1$ but

$n! \nleq C.\ n^{\log n}\ \forall\ n \ge n_o$ (false)

(b) $2^n \ngeq 2^{n!}\ \forall\ n \ge 1$ (false)

(c) $n! \nleq C.\ 2^n\ \forall\ n \ge 5$ (false)

(d) $n^{\log n} \le C.\ 2^n\ \forall\ n \ge n_o$ and

$n! \ge C.\ 2^n$ for $n \ge 1$ (true)

09. Ans: (b)

Sol: The upper bound cannot be more than $\sqrt{n}$ and lower bound will be Ω (1) when the loop terminates with the condition n % i = 0 for the first time.

Big O notation describes the tight upper bound and Big Omega notation describes the tight lower bound for a algorithm. The *for* loop in the question is run maximum $\sqrt{n}$ times and minimum 1 time. Therefore, $T(n)=O(\sqrt{n})$ and $T(n)=\Omega(1)$.

10. Ans: (c)

Sol: Average case running time always less than (or) equal to worst case Time.

$\therefore$ A(n) = O(W(n))

By definition of asymptotic notations. Average lies always between the best and the worst case inclusive.

11. Ans: (c)

Sol: $j=\frac{n}{1}+\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+..+1$

$\Rightarrow n\left(\frac{1}{1}+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+...+\frac{1}{n}\right)=n[O(1)]$

$\Rightarrow \Theta(n)$ [$\because$ Polynomial of degree 1]

12. Ans: (d)

Sol: j is multiplied by 2 for every iteration

it runs k times + 1 for condition fail.

$n = 2^k \Rightarrow \log n = k$

It uses $\lfloor \log_2 n \rfloor + 1$ comparisons

13. Ans: (b)

14. Ans: (c)

Sol: T(n) = 1, n ≤ 2

$T(n)=T(\sqrt{n})+K, n>2$

$n^{2^{-K}}=2$

$2^{-K}=\log_n 2$

$2^K=\log_2 n \Rightarrow K=\log\log_2 n$

15. Ans: (c)

Sol: In all cases, when A[i] = 1 for all i = 1,n

A[i] = 0; for all i = 1,n

A[i] =1; for i =1,n/2

$=0$; for $i=\frac{n}{2}+1,...n$

The order of magnitude is Θ(n).

16. Ans: (d)

Sol: $T(n)=2T(n-1)+1$

Because recursive calls (n –1) two times.

Now,

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

$\Rightarrow T(n)=2[2T(n-2)+1]+1$

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

Continue this we get

$=2^{n-1}+2^{n-2}+...+2+1=2^n-1$

$\Rightarrow 2^n-1\le 1.2^n$ for $n\ge 1=O(2^n)$

17. Ans: (a)

Sol: $T(n) = 2T(n-1) + n$

Let $a_n = T(n)$

$a_n = 2a_{n-1} + n$ ………(1)

$a_n - 2a_{n-1} = n$

replace 'n' by n + 1

$a_{n+1} - 2a_n = n$

$E(a_n) - 2a_n = n$

$(E-2)\,a_n = n$

Characteristic equation $\phi(E) = 0$

$\Rightarrow E - 2 = 0$

$\therefore$ characteristic root = 2

$\therefore$ Complementary function = $C_1\,2^n$

Let $a_n = An + B$

Substitute 'a_n' in equation (1)

$An + B = 2\{A(n-1) + B\} + n$

$\Rightarrow An = 2An + n \quad B = -2A + 2B$

$\Rightarrow -An = n$ By substituting 'A'

$\Rightarrow A = -1 \quad B = 2$

$\therefore$ The solution is $T(n) = C_1.2^n - n - 2$

By applying initial condition we get $C_1 = 2$

$\therefore\ T(n) = 2^{n+1} - n - 2$

18. Ans: (d)

Sol: $\Theta(\log_2 \log_2 n)$

19.

Sol: $T(n) = O(n^2)$

$T(n) = n+(n-1) + (n-2) + \ldots\ldots\ldots\ldots + T(1)$

$= \dfrac{n(n+1)}{2}$

(or)

$T(n) - T(n-1) = n,\ T(1) = 1$

$T(n) - T(1) = 2+3+\ldots.n$

$T(n) = 1+2+3+\ldots.n = \dfrac{n(n+1)}{2}$

20. Ans: (b)

Sol: $\rightarrow f1(n)$ requires $T(n) = 2T(n-1) + 3T(n-2) + 1$

$\Rightarrow \Theta(2^n)$

$\rightarrow f2(n)$ requires n because i runs n times

$\Rightarrow \Theta$ (n)

21. Ans: (c)

Sol: $f_1(8) = 2 * f_1(7) + 3 * f(6) = 1640$

$f_1(7) = 2 * f_1(6) + 3 * f_1(5) = 547$

$f_1(6) = 2 * f_1(5) + 3 * f_1(4) = 182$

$f_1(5) = 2 * f_1(4) + 3 * f_1(3) = 61$

$f_1(4) = 2 * f_1(3) + 3 * f_1(2) = 20$

$f_1(3) = 2 * f_1(2) + 3 * f_1(1) = 7$

$f_1(2) = 2 * f_1(1) + 3 * f_1(0) = 2$

$f_2(8) = 1640$

$f_1(n)$ and $f_2(n)$ are same

$\downarrow \qquad \downarrow$

Recursive Iteration

Chapter 2 Divide & Conquer

01. Ans: (a)

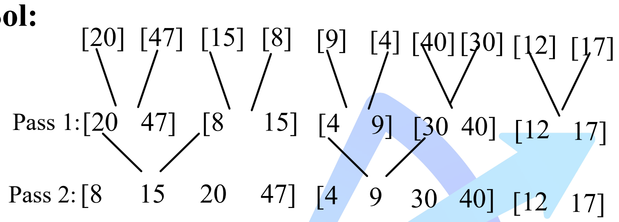
Sol: Divide and conquer strategy.

02. Ans: (c)

Sol: To make two sorted arrays as single sorted array it requires m + n comparisons.

03. Ans: (b)

Sol:



04. Ans: (b)

Sol: In merge-sort algorithm number of splits are proportional to height and in each level work done is n^2

$\therefore$ Total Time Complexity $O(n^2 \log n)$

The Complexity of Merge Sort for n elements is $O(n \log n)$.

05. Ans: (a)

Sol: In the worst case the selected pivot element will be placed in either first (or) last position

Then the required recurrence equation is

$$T(n) = T(n-1) + \Theta(n)$$

By solving using substitution method we get

$$\therefore T(n) = O(n^2)$$

06. Ans: (c)

Sol: Quick sort takes worst time when elements are in sorted order.

[1 2 3 4] takes $O(n^2)$ where n = 4

[5 4 3 2 1] takes $O(n^2)$ where n = 5

clearly number of elements are different in both cases.

So, $t_1 < t_2$ as $t_1 = O(n^2)$ for n = 4

and $t_2 = O(n^2)$ for n = 5

07. Ans: (b)

Sol: By applying divide and conquer concept one list would have 1/5 elements and the other would have 4/5 of the no. of elements 'n' comparisons are required for fixing up the pivot.

08. Ans: (c)

Sol: If we choose pivot randomly, Randomized quick sort still may have worst case time of $O(n^2)$.

It can be $O(n \log n)$ if pivot always divides the array into two equal sub parts.

09. Ans: (a)

Sol: Binary search takes time of $O(\log n)$ for a set of n-elements. Total time for n-elements $= O(n \log n)$

10. Ans: (d)

Sol: If we subtract each number by 1 then we get the range $[0, n^3 - 1]$. Considering all number as 3-digit base n: each digit ranges from 0 to $n^3 - 1$. Sort this using radix sort. This uses only three calls to counting sort. Finally, add 1 to all the numbers. Since there are 3 calls, the complexity is $O(3n) \approx O(n)$.

11. Ans: (b) & (d)

Sol: Using Divide and Conquer method not more than $\left(\frac{3n}{2} - 2\right)$ comparisons are required in all cases of input.

12. Ans: 148

Sol: Minimum number of comparisons required to find the minimum and maximum of 100 numbers = 1.5 (100) – 2 = 148

13. Ans: (a)

Sol: After comparing the key with the middle element, the search is made either in the left or right sublist with n/2 elements.

$\therefore$ T(n) = T(n/2)+k, where 'k' is constant.

14. Ans: (b)

Sol: $T(2^k) = 3\,T(2^{k-1}) + 1$

Let $n = 2^k$

$T(n) = 3\,T(n/2) + 1$

Solve it using back substitution.

(or)

$T(2^k) = 3T(2^{k-1})+1$

$a_k = 3a_{k-1}+1$

$(E-3)a_k = 1$

$a_k = C_1 3^k$

$a_k = c\cdot 3^k \quad a_k = \frac{1}{1-3} = \frac{-1}{2}$

$a_k = c\cdot 3^k - \frac{1}{2} \quad a_o = c - \frac{1}{2} = 1 \Rightarrow c = \frac{3}{2}$

$a_k = \frac{3}{2}\cdot 3^k - \frac{1}{2} = \left(\frac{3^{k+1} - 1}{2}\right)$

15.

Sol: T(n) = T(n/2) + n, T(1) = 1

Master Theorem

a = 1, b = 2, k = 1, p = 0

Since $a < b^K$, so it is case (3) of master theorem.

$\therefore$ T(n) = O(n)

Case 3: $1 < 2^1$

16. Ans: (a)

Sol: T(n) = 2T (n/2) + log n

By using Master Theorem, a = 2, b = 2, k = 0, p =1

As $a > b^k$, so it is case(i) of Master Theorem

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

$= \Theta(n)$

17. Ans: (a)

Sol: Applying Master Theorem; case '3' holds and hence T(n) is O(n);

$a = 3, b = 4, K = 1$

Since $a<b^K$ so it is Case 3 of master theorem.

$\therefore\ T(n) = O(n)$

Master Theorem:

$a = 3, b = 4, k = 0, p = 0$

$3 > 4°.\ O(n\log_4 3)$

18. Ans: (b)

Sol: $T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n}, n \geq 2$

$T(1) = 1$

Master Theorem:

$a = 2, b = 2, k = 1/2, p = 0$

$2 > 2^{1/2} O(n\log_2 2) = O(n)$

19. Ans: (c)

Sol: It is Θ (n log n) is also O (n^2) &

O(n log n) is not Ω (n^2).

If we use binary search then there will be $\log_2 n!$ comparisons in the worst case, which is (n log n). But the algorithm as a whole will still have a running time of Θ(n^2) on average because of the series of swaps required for each insertion.

(or)

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

$T(2^k) = 2T(2^{k-1})+n$

$ak - 2^{k-1} = 2k$

$(\Sigma-2)ak = 2\cdot 2^k$

$ak = c\cdot 2^k$

$ak = \frac{2.2^k}{\Sigma-2} = 2\cdot c(k,1)\ 2^{k-1}$

$= 2k\cdot 2^{k-1}$

$a1 = c\pm 1$

$ak = 2^k+k\cdot 2^k$

$T(2^k) = 9k = n+n\log_2 n \Rightarrow O(n \log n)$

20. Ans: (a)

Sol: Applying master-theorem Case-III holds.

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

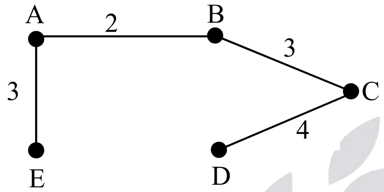
$\Rightarrow T(n) = \Theta(n)$ (Master Theorem Case 3.a)

Chapter 3

Greedy Method

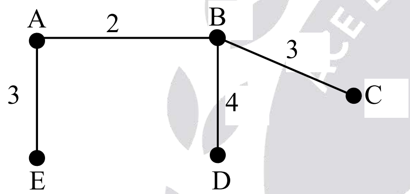
01.

Sol: Apply Kruskal's Algorithm



Cost=12

Another possibility

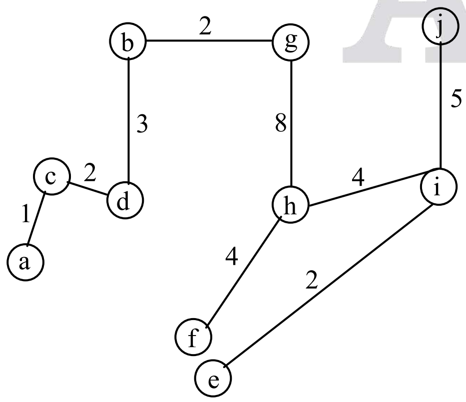


02. Ans: (a), (b) & (d)

Sol: There can be multiple minimum cost spanning trees under the presence of non-distinct edge costs.

03. Ans: (b)

Sol:



Weight =1+2+3+2+8+4+5+4+2=31

04. Ans: (b)

Sol: When v_i is connected to v_{i+1}, the edge cost is 2 there will be (n-1) such edges with a cost of '2'.

Therefore total cost = 2(n–1)

= 2n–2

(or)

$$\sum_{i=1}^{n} 2|\vartheta_i - \vartheta_i - 1| = 2\sum_{i=1}^{n}|1| = 2|n-1| = 2n-2$$

05. Ans: (a), (d)

Sol: If there are multiple edges in the graph with the minimum weight 'w', then it is not necessary that the specific edge with cost 'w' must be present in all spanning trees. Rest all options are correct.

There may be many edges of weight w in the graph and e.

06. Ans: (b)

Sol: Cost of MST with '4' vertices is 3+4+6

Cost of MST with '5' vertices is 3+4+6+8

In general cost for 'n' vertices we have

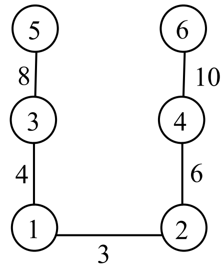
= 3+4+6+8+…………+2n–2

= n^2–n+1

07. Ans: (c)

Sol:

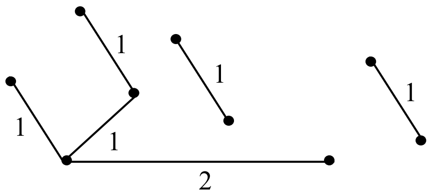
10 + 6 + 3 + 4 + 8 = 31



08. Ans: 8

Sol: In the given graph |V| = 9, |E| = 13

MST, must contains |V| – 1 number of edges = 8 edges.



09. Ans: (d)

Sol: Kruskal's Algorithm uses min Heap to keep the list of edges $\Theta(m \log m)$

With the Union-Find data structure implemented this way, Kruskal's algorithm can be analyzed. The sorting of the edges can be done in O(mlogn) which is O(mlogn) for any graph (why?). For each edge (u,v) we check whether u and v are in the same tree, this is done with two calls to Find which is O(logn), and we union the two if necessary which is O(1). Therefore the loop is O(mlogn). Hence the total time complexity is O(mlogn).

10. Ans: (d)

Sol: (a) $(a^1–b)$, $(d^1–f)$, $(b^2–f)$, $(d^2–c)$, $(d^3–e)$– valid

(b) $(a^1–b)$, $(d^1–f)$, $(d^2–f)$, $(b^2–f)$, $(d^3–e)$– valid

(c) $(d^1–f)$, $(a^1–b)$, $(d^2–c)$., $(b^2–f)$,$(d^3–e)$– valid

(d) $(d^1–f)$,$(a^1–b)$,$(b–f^2)$,$(d–e^3)$, $(d^2–c)$– invalid

11. Ans: (a)

Sol: Number of edges in the shortest path can be determined as a result of Dijkstra's single source shortest path algorithm.

12. Ans: (b)

Sol:

(a) $P \xrightarrow{1} Q \xrightarrow{2} R \xrightarrow{4} S \xrightarrow{7} T \xrightarrow{3} U$ **-Invalid** (not in increasing order)

(b) $P \xrightarrow{1} Q \xrightarrow{2} R \xrightarrow{3} U \xrightarrow{4} S \xrightarrow{4} T$ **-Valid**

(c) $P \xrightarrow{1} Q \xrightarrow{2} R \xrightarrow{3} U \xrightarrow{7} T \xrightarrow{4} S$ **-Invalid**

(d) $P \xrightarrow{1} Q \xrightarrow{7} T \xrightarrow{2} R \xrightarrow{3} U \xrightarrow{4} S$ **-Invalid**

13. Ans: (c)

Sol: Edges and vertices of the graph can be maintained in the form of a heap data structure to have a linear time complexity algorithm.

14. Ans: (d)

Sol: In Dijkstra's Shortest Path Algorithm, we always consider the vertex which are reachable from the source with last cost, and update cost label of the vertex, if the present cost is minimum than the previous cost.

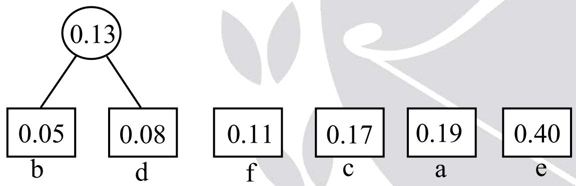
Apply Greedy based Dijkstra's Algorithm.

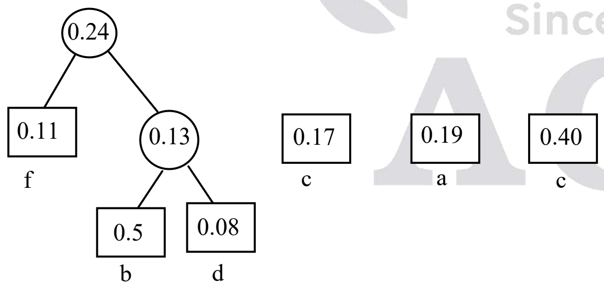
	A	B	C	D	E	F	G	T	
S	4	3	∞	7	∞	∞	∞	∞	
B	4		∞	7	∞	∞	∞	∞	
A			5	7	∞	∞	∞	∞	
C				7	6	∞	∞	∞	
E				7		∞	8	10	⇒ SACET

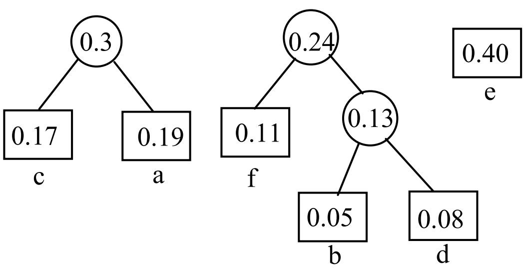
∴ The shortest path from S → T is SACET.

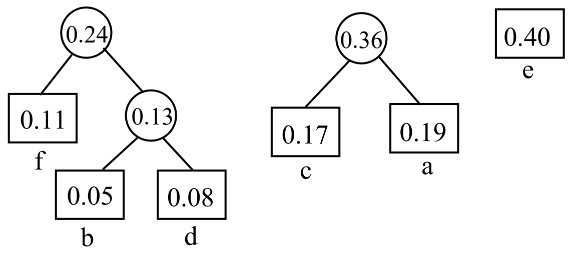
15.

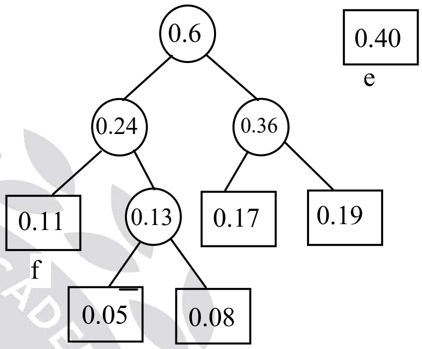
Sol:

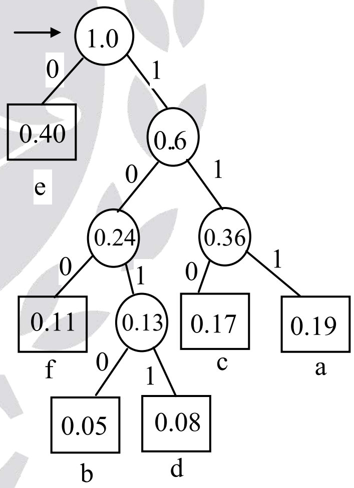












Prefix codes:

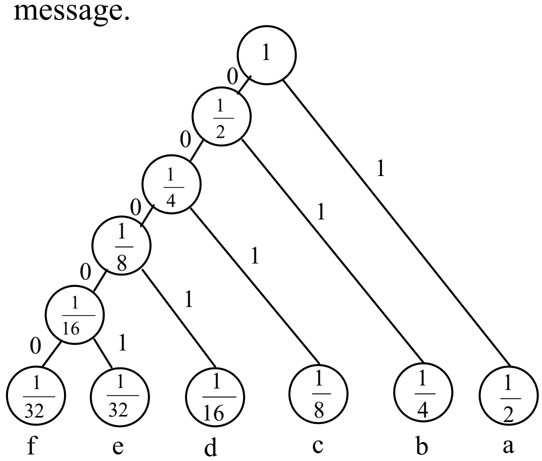
a → 111 ; b → 1010 ;

c → 110 ; d → 1011 ; e → 0

16. Ans: (a)

Sol: Obtain the Huffman Encode tree by applying optimal merge pattern algorithm.

Assign '0' to the left branch and '1' to the right branch in the encode tree. Collect the stream of binary bits to get the codes of the message.



17. Ans: (d)

Sol: The average Number of bits /message is obtained by using the formula

$$\sum_{i=1}^{n} d_i * q_i$$

d_i= distance from root to message i

q_i = probability of message i

$$\frac{1}{2}\to 1\,;\ \frac{1}{4}\to 2\,;\ \frac{1}{8}\to 3\,;\frac{1}{16}\to 4\,;\frac{1}{32}\to 5$$

$$=\frac{1}{2}\times 1+\frac{1}{4}\times 2+\frac{3}{8}+\frac{4}{16}+\frac{5}{32}+\frac{5}{32}$$

$$=\frac{16+16+12+8+10}{32}$$

$$=\frac{32+30}{32}$$

$$=\frac{62}{32}=1.9375$$

Chapter 4 Graph Techniques, Components, Heaps

01. Ans: (b)

02. Ans: (b)

Sol: Binary search – O(log n)

Insertion sort – O(n)

Merge sort – O(n log n)

Selection sort – $O(n^2)$

03. Ans: (d)

Sol: Nodes 2, 3, 5 are the Articulation points.

04. Ans: (b)

Sol: In the array representation of Binary Tree, the parent is at location $\lfloor i/2 \rfloor$, where as left child is at 2i and right child at 2i + 1.

05. Ans: (a), (c) & (d)

Sol: a, b, f, e, h, g is not possible as one cannot visit 'e' after 'f'.

06. Ans: (c)

Sol: In order to find 7^{th} smallest element, we have to perform '7' deletion operations so it takes O(7logn) = Θ(logn).

07. Ans: (c)

08. Ans: (a)

Sol: Smallest element lie at the leaf level. Which has roughly n/2 elements. Which would require number of comparisons and number of elements.

09. Ans: (c)

Sol: In selection sort, each iteration takes one swap in the worst case. Hence it requires O(n) swaps in the worst case.

10. Ans: (b)

Sol: Algorithm to determine leaders in an array:

```
int max = a[n]
for i ← n – 1 to 1 by –1
{
  if (a[i] > max)
  {
        print a[i];
        max = a[i];
  }
}
```

While scanning the array from right to left remember the greatest element seen so far and compare it with the current element to test for leadership.

11. Ans: (a)

Sol: Merge sort takes a time of O(n log n) in all cases of input. Whereas other sorting techniques have complexity of $O(n^2)$ in worst case.

12. Ans: (c)

Sol: This is the case when the graph is represented by cost Adjacency matrix.

13. Ans: (a), (b) & (d)

Sol: These choices violates the FIFO discipline of the queue and hence are not valid BFS traversals.

14. Ans: (a)

Sol: To sort 'n' elements using selection sort it requires O(n) swaps.

15. Ans: (c)

Sol: In **option (a)** (13) cannot be the child of (12) in max-Heap.

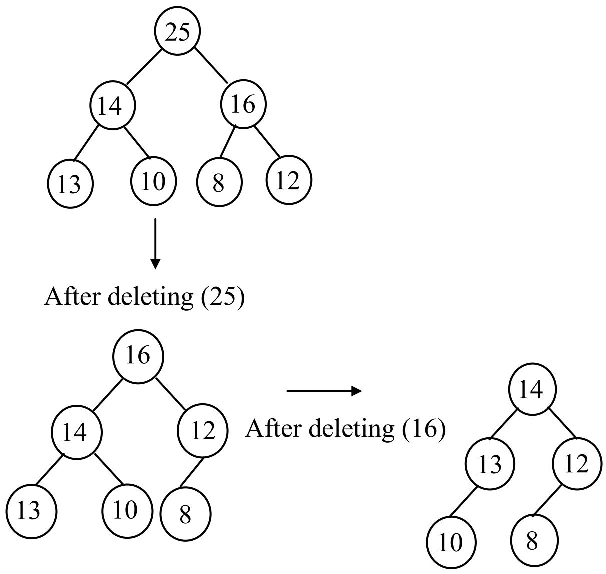
In **option (b)** (16) cannot be the child of (14).

In **option (d)** (16) cannot be the child of smaller value node.

In (a), s[3] which is the left child of a[1] is greater than the parent (13>12). In (b), also a[3]>a[1] (16>14). In (d), a[6] which is right child of a[2] is greater than a[2] (16>12).

16. Ans: (d)

Sol:



The height of a Max Heap is $\Theta(\log n)$. While insertion, we need to traverse from leaf element to root (in worst). If we perform binary search for finding the correct position then we need to do $\Theta(\log\log n)$ comparisons. In reality, it is not possible to perform binary search on elements from leaf to root as they are not in sequence.

17. Ans: (b)

Sol: Trees in option (d) violate the property of Max-Heap. Tree (a) satisfies the property of Max-Heap, but it is not a complete Binary Tree. In (c) and (d), heap property is not satisfied between 5 and 8.

$\therefore$ Tree (b) is the correct answer.

18. Ans: a-r, b-p, c-s, d-q

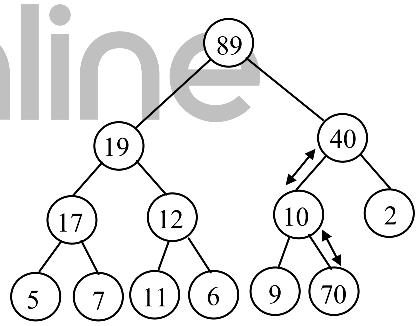
19. Ans: a-q, b-r, c-s, d-p

20. Ans: (d)

Sol: If we subtract each number by 1 then we get the range $[0, n^3 - 1]$. Considering all number as 3-digit base n: each digit ranges from 0 to $n^3 - 1$. Sort this using radix sort. This uses only three calls to counting sort. Finally, add 1 to all the numbers. Since there are 3 calls, the complexity is $O(3n) \approx O(n)$.

21. Ans: (c)

Sol: Tree Representation of the array is



22. Ans: (d)

Sol: If x is found at loc '1'

$\rightarrow$ 1 comparison

x is found at loc '2'

$\rightarrow$ 2 comparisons

x is found at loc '3'

$\rightarrow$ 3 comparisons

x is found at loc 'n'

→ 'n' comparisons

Total comparisons = n(n+1)/2

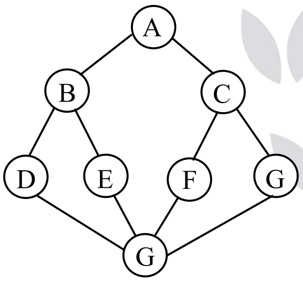
Therefore average = $\dfrac{\frac{n(n+1)}{2}}{n}$

23. Ans: (c)

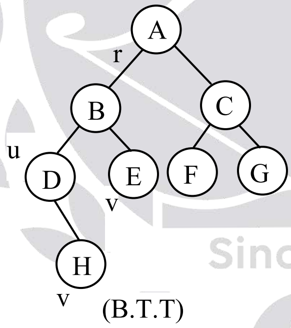
Sol: (c) is always true but (b) & (d) are true in only few cases. (a) is never true.

24. Ans: (c)

Sol: If 'u' is visited before 'v', then the distance from 'r' to 'u' will be less than or equal to the distance from 'r' to 'v' as demonstrated in the example given below



(G)



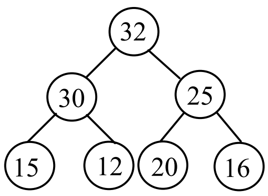
(B.T.T)

25. Ans: (d)

Sol: Based on Kirchoffs Theorem

26. Ans: (a)

Sol:

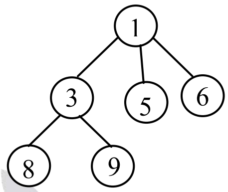


27. Ans: (d)

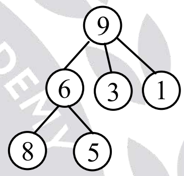
Sol: Since the degree of u & v in 'G' is at least 2.

28. Ans: (d)

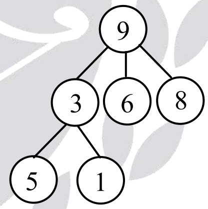
Sol:



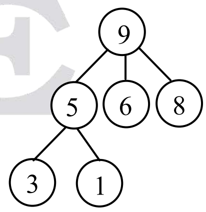
Tree for option (a) '1' in at top but not possible



Tree for option (b) '8' is greater than '6' not possible



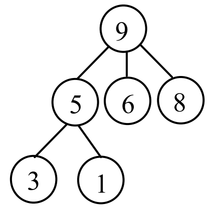
Tree for option (c) '5' is greater than '3' and hence not Possible



Tree for option (d) is max Heap

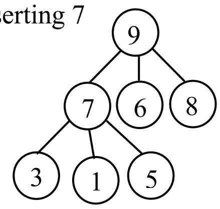
29. Ans: (a)

Sol:

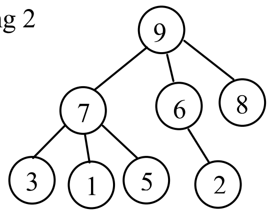


(a) Original tree

→ Inserting 7

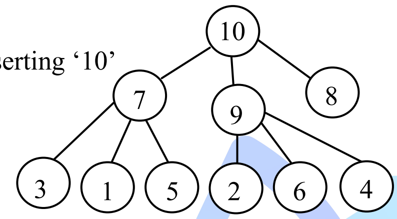


Inserting 2



In the array it is :10,7,9,8,3,1,5,2,6,4

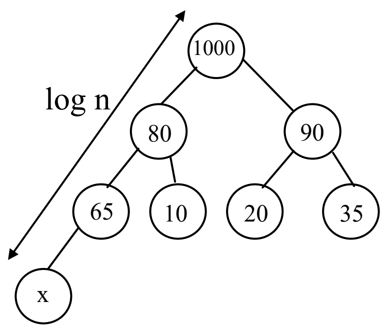
Inserting '10'



30. Ans: (b)

Sol: Consider the sample tree given below.

Binary search to be applied along the path which has log n elements.



The height of a Max Heap is Θ(logn). While insertion, we need to traverse from leaf element to root (in worst). If we perform binary search for finding the correct position then we need to do Θ(loglogn) comparisons.

In reality, it is not possible to perform binary search on elements from leaf to root as they are not in sequence.

31. Ans: (b) & (c)

Sol: Procedure Heapify with adjust would require time of O(n) for n-elements, now with addition n-elements, total being '2n' would still be order of O(n).

We can reduce the problem to building Heap for 2n elements. Time complexity for building heap is O(n).

Chapter 5

Dynamic Programming

01. Ans: (a) & (b)

02.

Sol: for i ← 1 to n

for j ← 1 to n

if (A[i ,j] = **0**) then P [i, j] = **0;**

else P[i ,j] =1;

for i ← 1 to n

for j ← 1 to n

for k ← 1 to n

P[**i ,j**] = min{P**[i,k]+P[k,j], P[i,j]**}

Time complexity is **O (n^3).**

03. Ans: (c)

Sol: Using LCS algorithm, in Dynamic programming we can write

expr1 = 1+l(i–1,j–1); and

expr2 = max (l(i–1, j), l(i,j–1))

04. Ans: (b)

Sol: We can compute using either CMO (or) RMO of L(M,N).

05. Ans: 34

Sol:

i↓ \ i→ A	0	1 q	2 p	3 q	4 r	5 r	
	0	0	0	0	0	0	
p	0	0	1	1	1	1	→ p
q	0	1	1	2	2	2	→ q
p	0	1	2	2	2	2	
r	0	1	2	2	3	3	
q	0	1	2	3	3	3	→ r
r	0	1	2	3	4	4	→ r
p	0	1	2	3	4	4	

LCS = pqrr

and length of longest common subsequence

= x = 4

Similarly remaining LCS are qsqr, qprr

∴ y = 3

∴ x + 10y = 4 + 10(3) = 34.

06. Ans: (b)

Sol: Apply Principle of optimality

07. Ans: (c)

08. Ans: (c)

Sol: It is based on Dynamic programming.

Use the recurrence that arises in this problem and multiply as

$(M_1\times (M_2 \times M_3)) \times M_4 = 19{,}000$

See the recurrence of Matrix Chain Product.