RANK Improvement WORKBOOK

GATE 2023

Computer Science & Information Technology

Algorithms

MADE EASY Publications

www.madeeasypublications.org

Contents

RANK Improvement WORKBOOK

Computer Science & Information Technology

RIB Workbook 2022 : Algorithms

Answer Key

Chapter-1 : Fundamental of Algorithm

1. (a, b, c)	2. (3)	3. (b)	4. (c)	5. (a, b, d)	6. (c)	7. (c)
8. (a)	9. (d)	10. (a)	11. (b)	12. (c)	13. (a)	14. (c)
15. (b)	16. (b)	17. (a, b, c, d)	18. (a)	19. (c)	20. (a)	21. (c)
22. (b)	23. (c)	24. (a)	25. (a)	26. (c)	27. (c)	28. (#)
29. (5.7)	30. (b)	31. (c)	32. (b)	33. (d)	34. (d)	35. (a, c)
36. (b, d)	37. (c)	38. (d)	39. (48)	40. (c)	41. (a)	42. (b)
43. (b)	44. (a, b, c)	45. (d)	46. (b)	47. (2.70)	48. (b)	49. (a)
50. (a)	51. (d)	52. (b)				

Chapter-2 : Divide and Conquer Method

1. (c)	2. (c)	3. (a)	4. (a)	5. (2)	6. (c)	7. (d)
8. (256)	9. (16)	10. (b)	11. (b)	12. (c)	13. (c)	14. (b)
15. (c)	16. (d)	17. (c)	18. (a)	19. (a)	20. (d)	21. (d)
22. (d)	23. (c)	24. (b)	25. (b)	26. (c)	27. (b)	

Chapter-3 : Trees and Binary Heap

1. (d)	2. (11)	3. (b)	4. (a)	5. (b)	6. (a)	7. (a)
8. (c)	9. (a)	10. (c)	11. (9)	12. (1014)	13. (b)	14. (b)
15. (a)	16. (9)	17. (a)	18. (896)	19. (a)	20. (a)	21. (b)
22. (a)	23. (b)	24. (c)	25. (c)	26. (#)	27. (a)	28. (a)

Chapter-4 : Greedy Method

1. (c)	2. (d)	3. (28)	4. (a)	5. (51)	6. (b)	7. (d)
8. (b)	9. (c)	10. (2)	11. (2)	12. (b)	13. (a)	14. (b)
15. (a, b, c, d)	16. (b)	17. (c)	18. (a)	19. (b)	20. (446)	21. (52)
22. (2.56)	23. (2.21)	24. (b, c)	25. (3)	26. (40)	27. (b)	28. (12)
29. (d)	30. (82)	31. (19)	32. (c)	33. (24)	34. (a)	35. (9)
36. (b)	37. (14)	38. (b)	39. (5)	40. (a)	41. (a, c, d)	42. (a, c)
43. (c)	44. (c)	45. (d)	46. (b)	47. (b)	48. (c)	49. (c)

Chapter-5 : Dynamic Programming

1. (d)	2. (6500)	3. (158)	4. (c)	5. (a)	6. (a)	7. (71)
8. (122)	9. (a)	10. (c)	11. (2)	12. (1)	13. (c)	

Chapter-6 : Graph Traversal and Sorting Algorithms

1. (c, d)	2. (9)	3. (b)	4. (c)	5. (c)	6. (c)	7. (c)
8. (c)	9. (d)	10. (c)	11. (2)	12. (7)	13. (c)	14. (4)
15. (120)	16. (a)	17. (b)	18. (a)	19. (a)	20. (a)	21. (a, b, c)
22. (b)	23. (24)	24. (d)	25. (b)	26. (d)	27. (#)	28. (b)
29. (b)	30. (c)	31. (c)	32. (#)	33. (a)	34. (16)	35. (c)
36. (3900)	37. (b)	38. (c)	39. (c)	40. (d)	41. (d)	42. (b)
43. (d)						

Chapter-7 : Miscellaneous

1. (8)	2. (3.09)	3. (327)	4. (b)	5. (b)	6. (a)	7. (65536)
8. (c)	9. (25)	10. (42)	11. (c)	12. (a)	13. (8000)	14. (c)

01 Fundamental of Algorithm

2. (3)

(i), (iii), (iv) correct.

5. (a, b, d)

$f(n) = O(g(n)), d(n) = O(h(n))$

$f(n) + d(n)$ asymptotically smaller it equal to $g(n) + h(n)$

$\therefore\ f(n) + d(n) = O(g(n) + h(n))$

If $p(n)$ polynomial then $\log p(n)$ asymptotically smaller or equal to $\log n$.

$\therefore\quad \log p(n) = O(\log n)$

If $2^{f(n)}$ asymptotically bigger than $2^{g(n)}$

Then $f(n)$ asymptotically bigger than $g(n)$.

6. (c)

$$g(n) \geq c_1 n,\ h(n) \leq c_2 n$$
$$\therefore\quad g(n) \cdot h(n) \geq c_3 n$$
$$\Rightarrow\quad g(n) \cdot h(n) = \Omega(n)$$
$$f(n) + [g(n) \cdot h(n)] = \max(\Omega(n), \Theta(n))$$
$$f(n) + [g(n) \cdot h(n)] = \Omega(n)$$

7. (c)

$$f(n) = O(n^2),\quad f(n) \leq C_1 \cdot n^2$$
$$g(n) = \Omega(n^2),\quad g(n) \geq C_2 \cdot n^2$$
$$h(n) = \Omega(n),\quad h(n) \geq C_3 \cdot n$$
$$\text{So,}\quad [f(n) + (g(n) \cdot h(n))] = \Omega[n^2 + n^2 \cdot n]$$
$$= \Omega(n^3)$$

8. (a)

$2^{(\log_2 \log_2 n)^2}$ growth rate is slow,

$2^{\sqrt{\log_2 n}}$ growth rate is fast.

Since, for bigger values in above table sequence:

$$(2^{\log_2 \log_2 n})^2 < 2^{(\log_2 \log_2 n)^2} < 2^{\sqrt{\log_2 n}} < (\log_2 n)^{\log_2 n}$$

9. (d)

S_1: $f(n) = O(f(n)^2)$

If $f(n) = 1/n$

then $f(n) \neq O(f(n)^2)$

since $1/n \geq C_1(1/n)^2$

So, false, i.e., not always **true**.

S_2: $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$

Since $f(n) = n$ and $g(n) = n^2$

then $f(n) = O(g(n))$ but not $f(n) = \Omega(g(n))$

So, false i.e., not always **true**.

S_3: $f(n) \neq O(g(n))$ and $g(n) \neq O(f(n))$

If $f(n) = n^2$ and $g(n) = n$

then $f(n) \neq O(g(n))$ and $g(n) = O(f(n))$

i.e., not always true.

10. (a)

- $f(n) = \Omega(n^3)$
  $f(n) \geq C_1(n^3)$
- $g(n) = O(n^2)$
  $g(n) \leq C_1(n^2)$
- $h(n) = \Theta(n^2)$
  $h(n) = C_1(n^2)$

So, $f(n) + [g(n) \times h(n)]$

$= \Omega(n^3 + [1 \times n^2])$

$= \Omega(n^3 + n^2)$

$= \Omega(n^3)$

11. (b)

Compute all functions in power of e :

$$f(n) = (\log n)^{n-1}$$
$$= (e^{\log\log n})^{n-1} = e^{n\log\log n - \log\log n}$$
$$g(n) = 2^n = (e^{(\log 2)})^n = e^{n\log 2}$$
$$h(n) = e^n / n = e^n / e^{\log n} = e^{n-\log n}$$

So, $f(n) \neq O(h(n))$ and $h(n) = O(f(n))$

$g(n) \neq \Omega(f(n))$

12. (c)

$$f(n) = 2^{\log_2 n} = n^{\log_2 2} = n$$
$$g(n) = n^{\log n}$$
$$h(n) = n^{1/\log n}$$
$$= \sqrt[\log n]{n}\left[n > \sqrt[\log n]{n} \text{ for all large value of } n\right]$$

[It is less than n since max power of n is always less than 1 for large value of n]

So, $g(n) \geq f(n \geq h(n)$

So, $f(n) = O(g(n))$ and $g(n) = \Omega(h(n))$

13. (a)

$$f(n) = \Omega(n) \Rightarrow f(n) \geq c \cdot n$$
$$g(n) = O(n) \Rightarrow g(n) \leq c \cdot n$$
$$h(n) = \theta(n) \Rightarrow c_1 \cdot n \leq h(n) \leq c_2 \cdot n$$
$$f(n) \cdot g(n) = c \cdot n \; [\because \; f(n) \geq c \cdot n \; \& \; g(n) \leq c \cdot n]$$
$$\underbrace{f(n).g(n)}_{\geq c.n} + \underbrace{h(n)}_{\theta(n)} = \Omega(n)$$

So, option (a) is correct.

14. (c)

1. $\dfrac{e^{n\log n}}{n} = e^{n\log n - \log n}$

2. $2^{n\log n} = e^{(\log_2)^{n\log n}}$
$= (e^{n\log n})$

3. $n^{\sqrt{n}} = (e^{\log n})^{\sqrt{n}}$
$= (e^{\sqrt{n}\log n})$

15. (b)

$f_1 = n^4$ (polynomial time)

$f_2 = 4^n$ (ex-polynomial time)

$f_3 = n^{110/37} = n^{3.79}$ (polynomial)

$f_4 = (3.25)^n$ (exponential)

$f_3 < f_1 < f_4 < f_2$

16. (b)

$\log(\log n)! < (\log\log n)! < \log n! < (\log n)! < (\log n)^{\log n}$

Hence option (b) is the correct answer.

17. (a, b, c, d)

True. O is transitive, and $h(n) = \Omega(f(n))$ is the same as $f(n) = O(h(n))$.

True.

True. For every node with 1-based index $i > 1$, the node with index floor $(i / 2)$ is larger.

True. Insert takes $O(\log n)$ time per operation, and gets called $O(n)$ times.

18. (a)

$f(n) = \Omega(g(n))$ i.e. $f(n) \geq g(n)$

19. (c)

$f(n) = O(g(n))$ that means $f(n) \leq c\, g(n)$

20. (a)

Let's take an e.g.

$$\theta(n^2 \log n + n) \approx \theta(n^2 \log n)$$

21. (c)

$$f(n) = \sum_{i=0}^{99} \sum_{j=0}^{n-1} \left(\sum_{k=0}^{j-1} 1 + \sum_{m=0}^{i-1} 1 \right)$$
$$= \sum_{i=0}^{99} \sum_{j=0}^{n-1} \sum_{k=0}^{j-1} 1 + \sum_{i=0}^{99} \sum_{j=0}^{n-1} \sum_{m=0}^{i-1} 1$$
$$= O(n^2) + O(n) = O(n^2)$$

22. (b)

$$\sum_{i=1}^{2n} \sum_{j=1}^{n} 1 - \sum_{j=1}^{n} j = \sum_{i=1}^{2n} n - \frac{n(n+1)}{2}$$
$$= 2n^2 - \frac{n^2}{2} - \frac{n}{2} = \frac{3}{2}n^2 - \frac{n}{2} = \frac{3n^2 - n}{2}$$

23. (c)

1st loop will execute (log n) times

(i) For which 2nd for loop executes n times

(ii) For which 3rd for loop executes log n items.

So, time complexity is log $n[n + \log n] = \Theta(n \log n)$

24. (a)

for $(i = 1; i \le m; i++) \Rightarrow 0(m)$

for $(J = 1; J \times J \le m; J++) \Rightarrow 0(\sqrt{m})$

T.C. $= 0(m \times m^{1/2}) = 0\ (m^{3/2})$

25. (a)

For n, inner loop execute for $\frac{n}{2}$ times.

For $\frac{n}{2}$, inner loop execute for $\frac{n}{4}$ times.

For $\frac{n}{4}$, inner loop execute for $\frac{n}{8}$ times.

So, $\text{T.C.}(n) = \frac{n}{2}+\frac{n}{4}+\frac{n}{8}+.....+1$

$= n\left(\frac{1}{2}+\frac{1}{4}+.....+1\right) = 0(n)$

26. (c)

$\text{T.C} = \log 1 + \log 2 + \log 3 + ... + \log n$

$= \log (1.2..3 ..... n)$

$= \log(n!)$

$= \log(n^n)$

$= n \log n$

Option (c) $T(n \log n)$ is correct.

27. (c)

For reverse sorted order:

$i = 1$	2	.....	$n-1$
$J = 1$ time run	2 time run		$n-1$ time run

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

$= O(n^2)$

29. (5.7)

$T(n) = 0$; if $n = 1$

$= 1$; if $n = 2$

$= 3T\left(\frac{2n}{3}\right) + 2;\ n \ge 3$

$= \theta\,(n^{2.7})$

$x = 2.7$

$x + 3 = 2.7 + 3 = 5.7$

30. (b)

GCD algorithm has worst case time complexity when both P and Q are prime numbers.

The worst case time complexity is $O(\log_2 \max(Q, P))$.

31. (c)

Given function is,

```
Function (int n)
{
    if (n <= 1) return;          // constant time
    for (int i = 1; i < n^2; i++)
    {
        printf ("GATE");         // O(n^2) time)
    }
    function (0.6n);  // recursive call with 0.6n
}
```

Hence, the recurrence relation will be,

$T(n) = T(0.6n) + n^2$

$\Rightarrow \quad T(n) = T\left(\frac{3}{5}n\right) + O(n^2)$

32. (b)

$S_1 = \sum_{r=0}^{\log n - 1} \frac{nr}{2^r}$ and $S_2 = \sum_{r=0}^{\log n - 1} r2^r$

$S_1 = 0 + \frac{n}{2} + \frac{2n}{2^2} + ...+ \frac{n \times (\log n - 2)}{2^{(\log n - 2)}} + \frac{n \times (\log n - 1)}{2^{(\log n - 1)}}$

$\therefore \quad S_1 \cong \Theta(n)$

$S_2 = 0 + 2^1 + 2 \cdot 2^2 + ... + (\log n - 2)\ 2^{(\log n - 2)} + (\log n - 1)\ 2^{(\log n - 1)}$

$\therefore \quad S_2 \cong \Theta(n \log n)$

33. (d)

$$\begin{aligned}T(n) &= \sqrt{n}T(\sqrt{n})+n\\ &= n^{1/2}\,T(n^{1/2})+n\\ &= n^{1/2}[n^{1/4}T(n^{1/4})+n^{1/2}]+n\\ &= n^{3/4}\,T(n^{1/4})+n+n\\ &= n^{3/4}[n^{1/8}T(n^{1/8})+n^{1/4}]+n+n\\ &= n^{7/8}\,T(n^{1/8})+n+n+n\\ &\vdots\\ &= n^{2^k-1/2^k}T(n^{1/2^k})+k\cdot n\end{aligned}$$

$$n^{1/2^k} = 2$$

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

$$\log n = 2^k$$

$$k = [\log\log n]$$

$$= 2\cdot\left[n^{1/2}\cdot n^{1/2^2}\cdot n^{1/2^3}\ldots\ldots n^{1/2^{\log\log n}}\right] + n(\log\log n)$$

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

$$= 2\frac{n}{2}+n\log\log n$$

$$= \Theta(n\log\log n)$$

34. (d)

Use Back-substitution method:

$$\begin{aligned}T(n) &= T(n-1)+\frac{1}{n}\\ &= T(n-2)+\frac{1}{n-1}+\frac{1}{n}\\ &\vdots\\ &= T(n-k)+\frac{1}{n-(k-1)}+\ldots..\frac{1}{n-1}+\frac{1}{n}\\ &= n-k=\theta=n=k\\ &= 1+\frac{1}{2}+\frac{1}{3}\ldots..+\frac{1}{n}\approx\theta(\log n)\end{aligned}$$

38. (d)

$$T(n) = 2T\left(\frac{n}{\sqrt{2}}\right)+n$$

Using Master's theorem, $a = 2$, $b = \sqrt{2}$, $f(n) = n$

$$n^{\log_b a} = n^{\log_{\sqrt{2}} 2} = n^{\left(\frac{\log 2}{\log\sqrt{2}}\right)}$$

$$= \frac{1}{n^{(1/2)}} = n^2$$

$\Rightarrow \quad n^{\log_b a} = n^2$

$\Rightarrow \quad n^2 > f(n)$

$\Rightarrow\ T(n)$ is $O(n^2)$

39. (48)

Strassen's algorithm time complexity using divide and conquer method is $\Theta(n^{\log_2 7}) = S(n)$.

Arjun's algorithm time complexity will be = $\Theta(n^{\log_4 P}) = T(n)$.

Compare both the complexity to get value of 'P'. We know $T(n)$ should be smaller than (s) since Arjun's algorithm asymptotically faster.

$$n^{\log_2 7} > n^{\log_4 P}$$

$$\log_2 7 > \log_4 P$$

$$\log_2 7 > \log_2 \sqrt{P}$$

$$7^2 > \left(\sqrt{P}\right)^2$$

$$49 > P$$

So maximum value of P is 48.

40. (c)

Here Master's theorem is not applicable directly.

$$T(n) = 2.T\left(\sqrt{n}\right)+\log\left(\sqrt{n}\right)$$

Put $\quad n = 2^k$

$$T(2^k) = 2.T\left(2^{\frac{k}{2}}\right)+\log_2\left(2^{\frac{k}{2}}\right)$$

$$T(2^k) = 2.T\left(2^{\frac{k}{2}}\right)+\frac{k}{2}$$

Put $\quad S(k) = T(2^k)$

$$S(k) = 2.S\left(\frac{k}{2}\right)+\frac{k}{2}$$

Now apply Master's theorem:

$$S(k) = \Theta(k \log k)$$
$$T(2^k) = \Theta(\log n \,.\, \log \log_2 n)$$

41. (a)

Applying Master theorem

Here $a = 2,\ b = 4,\ d = \frac{1}{2}$

$$n^{\log_b a} = n^{\log_4 2} = n^{\log_4 (4)^{1/2}} = n^{1/2}$$

Also $f(n) = n^{1/2} = n^{\log_b a}$

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

(According to case 2 of Master theorem)

$$= \Theta(\sqrt{n} \log n)$$

42. (b)

$$T(n) = \log n + \log \sqrt{n} + \log \sqrt{\sqrt{n}} + \ldots + \log \sqrt[n]{n}$$

$$= \log n + \frac{1}{2}\log n + \frac{1}{2}\log \sqrt{n} + \ldots + \log \sqrt[n]{n}$$

$$= \log n + \frac{1}{2}\log n + \frac{1}{4}\log n + \frac{1}{8}\log n + \ldots + \log \sqrt[n]{n}$$

$$= \log_2 n \left(1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots \frac{1}{2^n}\right)$$

$$= O(\log_2 n) \text{ since } \left(1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots \cong 1\right)$$

43. (b)

Apply Master Theorem:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$f(n) = n^{1/2}$$

and here $a = 2,\ b = 4$

So, $(n^{\log_b a}) = (n^{\log_4 2})$

Will gives $(n^{1/2})$

So, $f(n) = \theta(n^{1/2})$

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

44. (a, b, c)

Recurrence relation for Bar (n):

$$T(n) = 4T\left(\frac{n}{2}\right) + \theta(n^2)$$

$\Rightarrow$ $T(n) = \theta(n^2 \log n)$

(a) is correct.

Now, $n^2 \log(n^2) = n^2.2 \log n = \Omega(n^2 \log n)$

Hence, $T(n)$ can also be written as

$$T(n) = \Omega(n^2 \log n)$$

$\therefore$ (b) is correct.

Similarly, (c) is also correct.

45. (d)

A. $T(n) = T(n-1) + n \Rightarrow T(n) = O(n^2)$

B. $T(n) = T\left(\frac{n}{2}\right) + 1 \Rightarrow T(n) = O(\log n)$

C. $T(n) = 2T\left(\frac{n}{2}\right) + n \Rightarrow T(n) = O(n \log n)$

D. $T(n) = 2T\left(\frac{n}{2} + 8\right) + n \Rightarrow T(n) = O(n \log n)$

Therefore, option (d) is correct.

46. (b)

We can write recurrence relation for this function:

$$T(n) = 8T\left(\frac{n}{2}\right) + O(n^3)$$

Which, by Master's theorem, is $O(n^3 \log n)$.

47. (2.70)

The recurrence relation for the given function is

$$T(n) = 3T\left(\frac{2n}{3}\right) + n$$

Using Master method,

$$a = 3;\ b = \frac{3}{2}$$

Applying Master method is

$$f(n) = n,\ n^{\log_b a} = n^{\log_{3/2} 3} = n^{2.70}$$

48. (b)

Apply Master theorem:

$$n^{\log_b a} = n^{\log_4 16} = n^2$$

and $f(n) = n^2$

Asy equal so case (ii) apply

$$T(n) = \theta(n^2 \log_2 n)$$

49. (a)

Apply Master's theorem

$$n^{\log_{10/9} 1} = n^0 = 1$$

and $f(n) = n$

So, $T(n) = \theta(n)$

50. (a)

Apply Master's theorem

$$n^{\log_8 7} < f(n) = n^2$$

$$T(n) = \theta(n^2)$$

51. (d)

Apply Master theorem:

$$n^{\log_b a} = n^{\log_4 2} = n^{\frac{\log_2 2}{\log_2 4}} = n^{1/2} = \sqrt{n}$$

and $f(n^2) = \sqrt{3}$

Thus, $T(n) = \theta\left(\sqrt{n}\right)$

52. (b)

$$n^{\log_b a} = n^{\log_4 3} < f(n) = n\log n$$

$f(n)$ is asymptotically larger

$$T(n) = \theta(n \log_2 n)$$

■■■■

02 Divide and Conquer Method

3. (a)

Since sorted array is given, we can apply binary search to find element 'x' in array. After finding location of 'x' element left x is our required answer if atleast 1 element present left to element 'x' which will take $O(\log n)$ time only.

4. (a)

Since two arrays are already in sorted order. We first select median of X and Y array in $O(1)$ time lets say 'a' and 'b' respectively. Compare X[a] and Y[b] if X[a] < = Y(b) then we apply above procedure to X[a,...n], Y[b,...,n] till their are ≤ 2 elements left in each array.

So, total time = $\log(n) + O(1) = \log(n)$.

5. (2)

- Recurrence relation for ternary search
  $T(n) = T(n/3) + 4,\ T(1) = 1$
- Recurrence relation for binary search
  $T(n) = T(n/2) + 2,\ T(1) = 1$
  Number of comparison in binary search : $2\log_2 n + 1$
  Number of comparisons in ternary search $= 4\log_3 n + 1$
  Since $(2\log_2 n + 1) < (4\log_3 n + 1)$
  Hence binary search is preferred.
  So, S_1 is incorrect.
- In AVL tree, worst case deletion and insertion. Both have 0(log$n$) complexity.
- Statement S_3 is also correct.

$$T(n) = T(\sqrt{n}) + \log n$$
$$n = 2^m$$
$$T(2^m) = T(2^{m/2}) + m$$

Put $T(2^m) = S(m)$

So, $S(m) = S\left(\frac{m}{2}\right) + m$

By applying Master theorem, we get

$S(m) = \theta(m)$

$m = \log_2 n\ (2^m = n,\ m = \log_2 n)$

So, $T(n) = \theta(\log n)$

6. (c)

The time complexity is $O(\log n)$ using binary search.

The idea is to go to the middle element at index $\frac{n}{2}$ calculate $a_{n/2} = a + \left[\frac{n}{2} - 1\right] \times d$ and check

$a\left[\frac{n}{2}\right] = a_{n/2}$ or not if equal check on RHS only otherwise LHS.

7. (d)

Let's take a number: 1 2 3 4 5 6 7 8 9 10

For 5 : 1 comparison

For 2, 8 : on second comparison i.e. 2

For 1, 3, 6, 7 : 3 comparison

For 4, 7, 10 : 7 comparison

$$\text{Average} = \frac{1\times1 + 2\times2 + 4\times3 + 3\times4}{10} = 2.9$$

8. (256)

Best case of quick sort is $O(n \log n)$ and it takes 2048 msec.

$$2048 = c_1 \cdot n \log n \quad \ldots(1)$$

Worst case of quick sort is $O(n^2)$ and it takes 324 msec.

$$324 = c_1 \cdot n^2 \quad [n = 18]$$
$$324 = c_1 \cdot 18^2$$
$$c_1 = 1$$

Substitute $c_1 = 1$ in equation (1)

$c_1 \cdot n \log n = 2048 \quad [c_1 = 1]$

$n \log n = 2048$

Put $n = 2^K$

$2^K \times K = 2048 \quad ...(2)$

The value of $K = 8$ which satisfies equation (2)

$\because$ Vivek's file size $= 2^K = 2^8 = 256$

9. (16)

The best case of quicksort takes 72 msec in $O(n \log n)$

$$72 = C \cdot n \log n$$
$$= C \cdot 8 \log_2 8 = C \times 24$$
$$C = 3$$

To calculate the file of Rahul,

$$3 \times n \times \log n = 192$$
$$\Rightarrow \quad n \log n = 64$$
$$\Rightarrow \quad n = 16$$

Hence Rahul's file size is 16.

10. (b)

This Join algorithm is similar to merge algorithm in merge sort

It takes $O(n + n) = O(2n) = O(n)$.

11. (b)

Since both arrays are sorted, so applying merge procedure will take $(n + n - 1) = O(n)$ time.

12. (c)

So, recurrence relation should be:

$$T(n) = 2T(n/2) + O(\log n) + O(n)$$

by using extended Master theorem:

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

13. (c)

Worst case for finding K^{th} smallest element in array of size 'n' using partition function is when every time partition function split array into two part one with $n - 1$ elements and other with 1 element i.e., $T(n - 1)$ and we have to do atmost n comparison for one partition i.e.

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

14. (b)

Before merging blindly, we have to sort both array individually which will take $O(m \log m)$ and $O(n \log n)$ time respectively. Then merging will take $O(m + n)$ in worst case.

Total number of comparisons

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

15. (c)

Time complexity to sort n elements using merge sort $= \Theta(n \log n)$

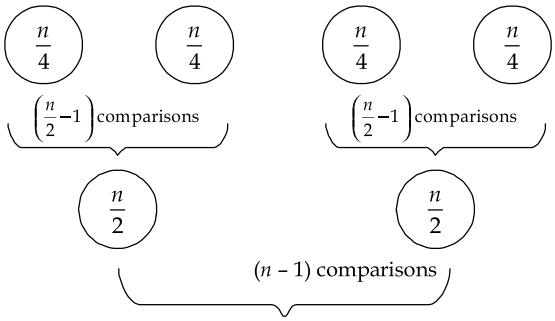
$$\Theta(n) = \Theta\left(\frac{n}{\log n} \log \frac{n}{\log n}\right)$$

$$\Theta(n) = \Theta\left(\frac{n}{\log n}[\log n - \log\log n]\right)$$

$$\Theta(n) = \Theta\left(\frac{n}{\log n} \log n\right) \quad [\log n - \log\log n$$
$$= O(\log n)]$$
$$\Theta(n) = \Theta(n)$$

16. (d)

Let us consider n-elements where each of the 4 list are having $\frac{n}{4}$ elements.



Total comparison $= 2\left(\frac{n}{2} - 1\right) + n - 1 = 2n - 3$

Here, $n = 4 \times 8 = 32$

So, Total comparison $= 2 \times 32 - 3 = 61$

17. (c)

Time complexity to sort n elements using merge sort = $\theta(n \log n)$

$$\theta(n) = \theta\left(\frac{n}{\log n}\log\frac{n}{\log n}\right)$$

$$\theta(n) = \theta\left(\frac{n}{\log n}[\log n - \log\log n]\right)$$

$$\theta(n) = \theta\left(\frac{n}{\log n}\log n\right)[\log n - \log\log n = O(\log n)]$$

$$\theta(n) = \theta(n)$$

18. (a)

First let's find the height of the tree (say h).

$$\frac{(\log m)}{2^h} = 1$$

$$h = O(\log \log m)$$

The time to merge from level i to level $i + 1$ = $O(\log n)$.

So, the total time to merge log m sorted lists into a single list of log n elements

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

19. (a)

Quick sort takes $\theta(n \log n)$ in average case and it is not stable sort.

20. (d)

In merge sort, merging procedure take auxiliary space i.e. $\theta(n)$. It is not in-place sort.

21. (d)

I/P is already sorted

(i) If first element select as a pivot

$$T(n) = T(n-1) + O(n) \rightarrow O(n^2)$$

(ii) Median of three elements

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

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

22. (d)

Median of first, last and middle elements can be anything. If first and last elements then worst case $O(n^2)$.

Arithmetic mean of elements need not be median of elements and if first element is pivot then again worst case can be $O(n^2)$.

23. (c)

In the given array, 3 elements (4, 5, 10) are in correct position.

So there are 3 possible pivots.

24. (b)

Upper bound for merging two lists with m and n elements is $m + n - 1$ comparison.

Thus, here $n + n - 1 = 2n - 1$ comparisons required.

25. (b)

Insertion sort is special case of merge sort in which one of sub arrays is sorted elements and other one is single elements.

Selection sort is also special case of quick sort (choose min/max element as pivot).

26. (c)

Pick any nut and test with all the bolts. Segregate the set of bolts into two-sets, one that contains bolts, which are greater in size than the chosen nut and the other that has bolts which are smaller than the nut. Also, find the matching bolts for the nut. Take this bolt and test with all the nuts and perform a similar segregation. Thus, we have succeed in dividing the problem into smaller sub problems, each of which will be solved recursively using the same procedure. Therefore the recurrence relation for number of comparison:

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

This is the same recurrence as that of quick-sort.

27. (b)

Using divide and conquer approach, closest pair can be found in $O(n \log n)$ time.

Algorithm :

Step 1 : Divide the set into two equal sized parts by the line l, and recursively compute the distance in each part. [d_1 = closest pair (left half); d_2 = closest pair (right half)] and returning the points in each set in order that is sorted by y-coordinate].

Step 2 : Let 'd' be the minimal of two minimal distances

$$d = \min(d_1, d_2) \ldots O(1)$$

Step 3 : Eliminate points that lie farther than 'd' apart from l, ... $O(n)$.

Step 4 : Merge the two sorted lists into one sorted list .... $O(n)$.

Step 5 : Scan the remaining points in the y-order and compute the distances of each point to its 5 neighbour ... $O(n)$.

Step 6 : If any of these distances is less than 'd' the update 'd' ... $O(1)$.

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

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

■■■■

03 Trees and Binary Heap

1. (d)

Number of leaf nodes = n
Let internal nodes be K
$\therefore$ Total nodes = $K + n$
For m-ary tree, number of leaf nodes with K internal nodes = $(m-1)K + 1$
$\therefore (m-1)K + 1 = n$

$$\therefore \quad K = \frac{n-1}{m-1}$$

$\therefore$ Total number of nodes

$$= \frac{n-1}{m-1} + n = \frac{n-1+n(m-1)}{m-1}$$

$$= \frac{n-1+nm-n}{m-1} = \frac{nm-1}{m-1}$$

2. (11)

Let I be the number of internal nodes, L be number of leaf nods and T be total number of nodes

$$I + L = T \quad \ldots(i)$$
$$n \cdot I + 1 = T \quad \ldots(ii)$$

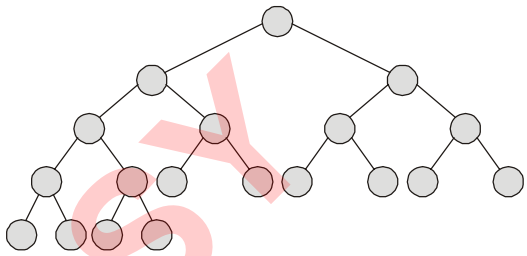
From (i) and (ii);

$$I + L = nI + 1$$
$$40 + 401 = 40 \cdot n + 1$$
$$440 = 40 \cdot n$$
$$n = 11$$

3. (b)

At level 0, $2^0 = 1$ nodes
At level 1, $2^1 = 2$ nodes
⋮
At level i, 2^i nodes

4. (a)



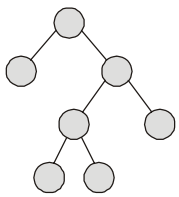
10 leaves and 19 nodes

5. (b)

Total number of sons pointer = nm
Total number of non-null pointer = $n-1$
Number of null pointer = $n(m-1) + 1$

6. (a)

Let's take an e.g.:



Here, n_0 = Leaf nodes = 4
$n_2 = 3$
Thus, $n_0 = n_2 + 1$

7. (a)

At height 0, maximum number of nodes = 1
At height 1, maximum number of nodes = 3
At height 2, maximum number of nodes = 7
⋮
At height k, maximum number of nodes = $2^{k+1} - 1$

8. (c)

N_1 = Number of lead nodes
N_2 = Number of nodes of degree 2
$N_1 = N_2 + 1$
$N_2 = N_1 - 1 = 10 - 1$
$N_2 = 9$

9. (a)

In n-ary tree, total number of vertices are

$$m = xn + 1 \quad \text{...(i)}$$

where, x = Number of internal nodes
l = Number of leaf nodes

From (i)

$$x + l = nx + 1$$
$$l = (n-1)x + 1$$

10. (c)

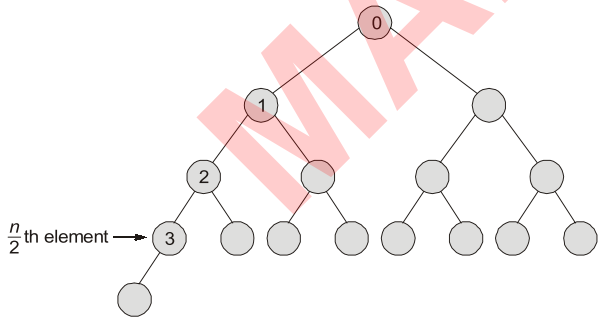
At level 8, maximum number of nodes

$$= 2^8 = 256$$

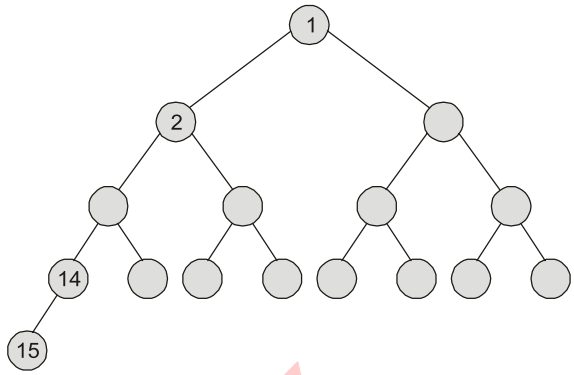
12. (1014)

Since there are total 1024 elements hence there will be total 10 levels of the heap. $\left(\frac{n}{2}\right)^{th}$ element will be the 1st element of 9th level.

Considering a heap of 16 elements for e.g.:



Minimum element '3' is possible it can't be less than that because its three ancestors should be smaller and we have only 3 elements left i.e. '0', '1' and '2'.



Since the element at $\left(\frac{n}{2}\right)^{th}$ index should always be smaller than the element at $(n-1)^{th}$ index, hence maximum 14 is possible.

Considering the same analogy for 1024 elements minimum possible element has key '8' and the maximum possible has key '1022'.

Difference = 1022 – 8 = 1014

13. (b)

Visit each node one time by comparing parent node with its children, will take $O(n)$ time in worst case.

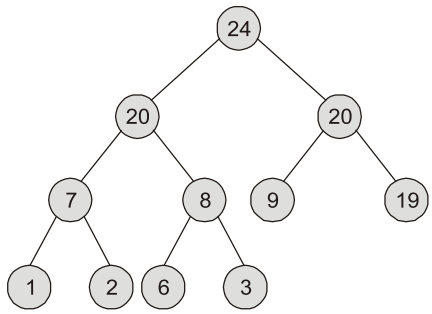
14. (b)

Unsorted double linked list:

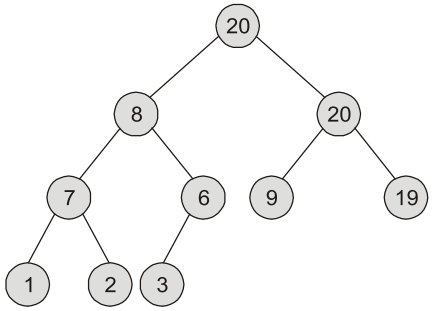
Find	Insert	Decrease key	Overall
$O(n^2)$	$O(\sqrt{n})$	$O(n \log n)$	$O(n^2)$

15. (a)

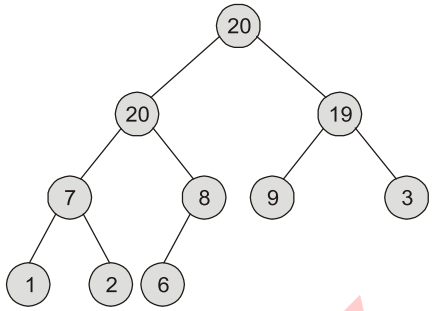
Given max heap:



[After 1 deletion 2 possibilities]

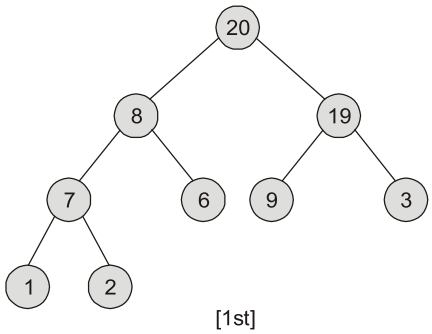


[1st]

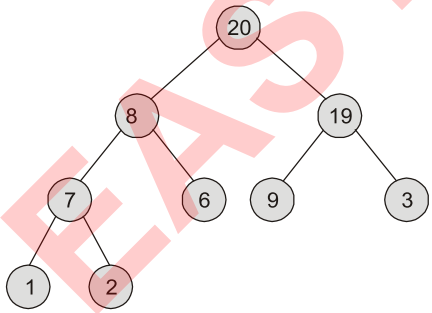


[2nd]

[After 2nd deletion]



[1st]



i.e. only 1 heap tree possible.

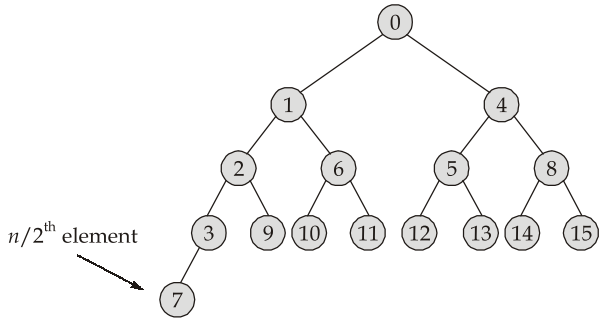
16. (9)

Since there are total 1024 elements hence there will be total 11 levels of the heap. $(n/2)^{th}$ element can be present at last level in worst case i.e. 11^{th} level.

$(n/2)^{th}$ element can also be present in best case level i.e. 2nd level

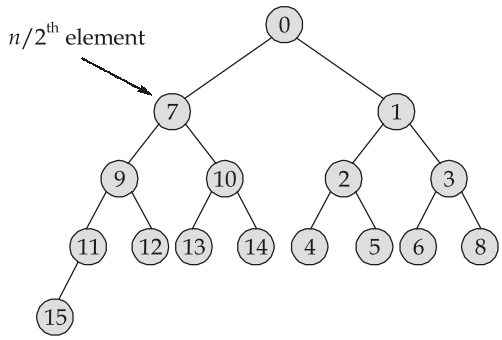
Assume for $n = 16$

In worst case:



$(n/2)^{th}$ element is at last level.

In best case:



$(n/2)^{th}$ element is at second level.

So, difference = [11 – 2] = 9

17. (a)

deletemin( ): Root element is the smallest element. Remove it and swap it with the rightmost leaf node to maintain heap property. Then search for the minimum element out of the remaining elements by looking in the lowermost min (even) level. Let it be at index

'*m*'. Swap this element at index m with the root. Do this recursively for the subtree rooted at index m to maintain the min-max property.

deletemax( ): The max element is one out of the two children of root. Find it and then follow the same steps as deletemin( ) operation, this time choosing the max element out of the remaining to be swapped at every stage.

18. (896)

Maximum element will be the first element in all such orderings. Remaining 8 elements will be placed such that 3 elements will be in right subtree and 5 elements will be in left subtree. So, for right subtree, select 3 elements from 8 elements and then we can place those 3 elements in the right subtree in 2 ways. Similarly remaining 5 elements can be put in ${}^4C_1 * 2$ ways in the left subtree. So, total orderings possible = ${}^8C_3 * 2 * {}^4C_1 * 2 = 896$.

19. (a)

Max heap property: The value of each node is less than or equal to the value of its parent, with the maximum value element at the root.

20. (a)

Using bottom up design, means adjust all internal nodes into max heap. Then time complexity will be $O(n)$.

21. (b)

Perform min heapify at rooted i that takes $O(\log n)$ time.

22. (a)

(ii) is false, because if array index starts from 0,

leaves range from $\left\lfloor \frac{n}{2} \right\rfloor + 2$ to $n - 1$.

23. (b)

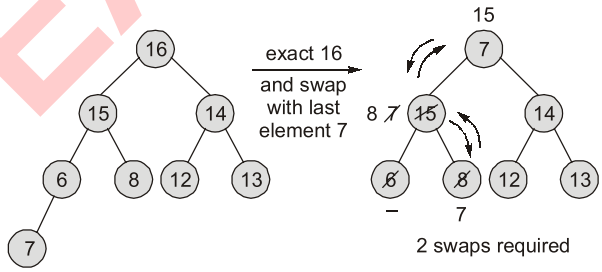
Heap sort makes $O(n)$ calls to the heapify/adjust procedure and uses a max heap for sorting elements in ascending order.

24. (c)

It is a max heap, so smallest can be in leaf nodes, but here mention $A[i] < A[i + 1]$.
So $A[15]$ contain smallest element of A.

25. (c)

Let's first make max heap tree:



27. (a)

Insertion of new element in sorted array takes $O(n)$ time. Binary search can use to find the position of insertion that takes $O(\log n)$ but in worst case. For shift the element takes $O(n)$, overall it takes $O(n)$ time.

28. (a)

For constructing priority queue, extract-min takes $O(\log n)$ time.

■■■■

04 Greedy Method

2. (d)

In worst case, i.e., dense graph, $E = O(V^2)$

$\therefore$ Time complexities of

1. 'V' invocations of Dijkstra algo = $O(V^3 \log V)$
2. 'V' invocations of Bellman-ford algo = $O(V^4)$
3. '1' invocation of Floyd-warshall algo = $O(V^3)$

$\therefore$ Floyd-Warshall is best.

3. (28)

Consider the following execution of Dijkstra's algorithm:

	P	Q	R	S	T	U	V	W
P	0	1	3	∞	∞	∞	∞	∞
Q	0	1	3	9	4	18	∞	∞
R	0	1	3	9	4	18	∞	∞
T	0	1	3	9	4	8	∞	∞
U	0	1	3	9	4	8	∞	13
S	0	1	3	9	4	8	11	13
V	0	1	3	9	4	8	11	13
W	0	1	3	9	4	8	11	13

The path calculated using Dijkstra's algorithm from S to V is 11.

But as per the given figure path 'S' to 'V' has '2' as the minimum weight,

$$P \xrightarrow{1} Q \xrightarrow{3} T \xrightarrow{4} U \xrightarrow{5} W \xrightarrow{-11} V.$$

This path will be correctly calculated if edge [W-V] is relaxed before edge [Q-S]. For that, the path from S to P should be greater than 13. Hence [13 to 20] all weights are possible, giving '8' possible values.

The second way which is possible is, we reach vertex 'V' from 'P' using path [Q-S], but for that, the path [P → Q → S → V] should be less than or equal to 2. Let the path of Q to S be 'z'.

It can be written as, $1 + z + 2 \le 2$

$\Rightarrow \quad z + 3 \le 2$

$\Rightarrow \quad z \le -1$

Hence the values ranging from [–1 to –20] are also possible where both (–1) and (–20) are inclusive giving 20 different values

So, Total possible values = 8+ 20 = 28

4. (a)

Starting vertex is D

	A	B	C	D	E	F	G	H
D	∞	∞	∞	0 Nill	∞	∞	∞	∞
F	16	17 D	13 D	–	∞	3 D	∞	8 D
B	16	7 E	13 D	–	∞	–	∞	8 D
H	16	–	13 D	–	∞	–	12 B	8 D
E	16	–	13 D	–	10 H	–	12 B	
G	16	–	13 D	–	–	–	12 B	
C	16	–	–	–	–	–	–	–
A	–	–	–	–	–	–	–	–

So the order of relaxed the vertices by using Dijkstra's algorithm is DFBHEGCA.

5. (51)

Minimum spanning tree by using Dijkstra's is

$$E_{max} = 0 + 2 + 1 + 3 + 6 + 7 + 9 + 8 + 10 + 12 = 58$$

$$E_{sp} = 1 + 5 + 1 = 7$$

Value of $E_{max} - E_{sp} = 58 - 7 = 51$

6. (b)

I. If the graph contains a negative weight cycle then no shortest path exist.

II. If shortest path is well defined then it contains at most V-1 edges. Running V-1 iterations of Bellman-Ford will find the path.

7. (d)

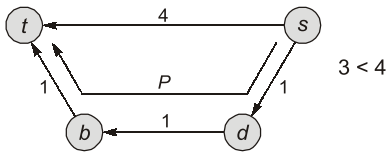
Initial distance from *A* to all other vertices is ∞.

Output order	A	S ∞	C ∞	D ∞	B ∞	E ∞	F ∞
A	–	3	2	1	6	∞	∞
A, D	–	3	2	–	6	5	∞
A, D, C	–	3	–	–	6	5	∞
A, D, C, S	–	–	–	–	6	5	9
A, D, C, S, E	–	–	–	–	6	–	9
A, D, C, S, E, B	–	–	–	–	–	–	9
A, D, C, S, E, B, F	–	–	–	–	–	–	–

So, the correct order is A, D, C, S, E, B, F.

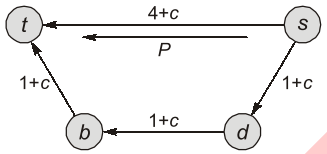
8. (b)

- S_1 is **false**, since counter example present



3 < 4

After adding constant 'c'



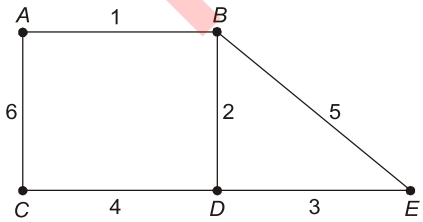
So, 4+*c* < 3 + 3*c*

- S_2 is **true**, since in Bellman-ford at P^{th} iteration we get shortest path in *G* using *P* or less than *P* edges.

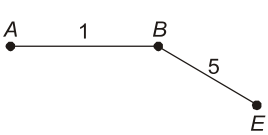
9. (c)

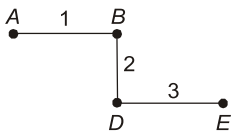
Considering each statement :

- Consider the graph,



Path between A to E :





Even though all edge weights are distinct, there are 2 shortest path between A and E vertices.

- Multiplying all edge weights by a positive number will always not change the cost of Minimum Spanning Tree i.e. positive integer 1 multiplying all edge weights will give same cost.
- Bellman Ford algorithm finds all negative edge weight cycle in the graph that are reachable from the source. If a graph contains a "negative cycle" reachable from the source, then there is no cheapest path. Any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle.

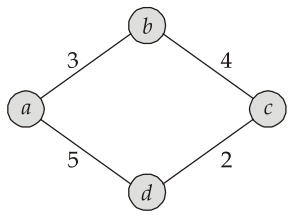
10. (2)

	Using Dijkstra algorithm		Actual Shortest path	
A → B	2	AB	2	AB
C	7	AC	7	AC
D	4	ABD	3	ACEGD
E	9	ACE	9	ACE
F	6	ABDF	5	ACEGDF
G	2	ACEG	2	ACEG

So, there are 2 wrong path calculated to D, F vertices.

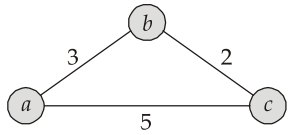
11. (2)

S_1 : A graph with distinct edge weights can have more than one shortest path.

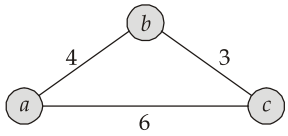


Two path (*a* – *d* – *c*) (*a* – *b* – *c*)

S_2 : Consider the graph as shown below:



There are two shortest paths between a and c i.e. $(a - b - c)$ and $(a - c)$.
Add 1 to every edge.



Now only $(a - c)$ is the shortest path thus changed.
Hence, both S_1 and S_2 are true.

12. (b)

Time complexity of Bellman-Ford algorithm is $O(V, E)$
Graph G is complete graph so $E = (V^2)$
Time complexity = $O(V \times V^2) = O(V^3)$

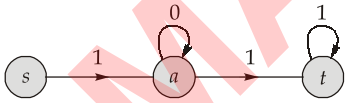
13. (a)

S_1: Bellman-Ford algorithm may not give shortest path if it contain a negative weight cycle. So S_1 is incorrect

S_2: Dijkstra's algorithm may give incorrect result if it contain negative weight cycle but it always terminates so S_1 is incorrect.

14. (b)

- The parent pointers may not lead back to the source node if a zero length cycle exists. Take an example [π means parent]



Relaxing the (S, a) edge will set $d[a] = 1$ and $\pi[a] = S$. Then relaxing the (a, a) edge will set $d[a] = 1$ and $\pi[a] = a$.
Following the π-pointers from t will no longer give a depth to S.
So, it is no correct algorithm.

- Option (b) correctly states about radix sort.

15. (a, b, c, d)

False. Only in unweighted graphs.
False, you'd prefer depth-first search, which can easily be used to produce a topological sort of the graph, which would correspond to a valid task order. BFS can produce incorrect results.
False. Dijkstra's algorithm always visits each node at most once; this is why it produces an incorrect result in the presence of negative-weight edges.
False. The negative-weight cycle has to be reachable from s.

16. (b)

In Bellman Ford algorithm, apply edge relaxation for all edges and repeat $|V| - 1$ times.

17. (c)

All edges have equal weights, we can consider it as unweighted graph.
BFS algo can used to find the shortest path for unweighted graph. BFS algo takes $O(V + E)$ time.

18. (a)

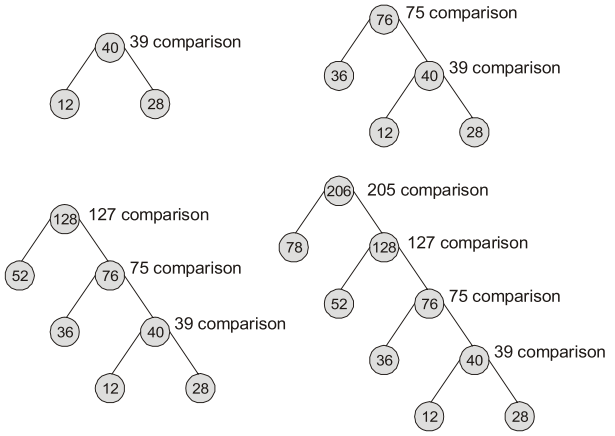
(a) can't be as intermediate node because there is a direct path s to r that costs 10.
(b) is the initiate state and (c) is the finite state and (d) can be as intermediate.

19. (b)

Main operation in Dijkstra's algorithm	Binary heap	Unsorted array	Sorted linked list
(i) n times delete min	$n * \log(n)$	$n * O(n)$	$n * \theta(1)$
(ii) Find adj of all vertices	$\frac{\theta(n^2)}{\theta(n + e)}$	$\frac{\theta(n^2)}{\theta(n + e)}$	$\frac{\theta(n^2)}{\theta(n + e)}$
(iii) $\theta(e)$ times key decrement	$\theta(e) * \log n$	$\theta(e) * O(1)$	$O(e) * n$
	$O(E \log n)$ $\approx O(E \log V)$	$O(n^2)$ $O(V^2)$	$O(n^3)$ $O(V^3)$

20. (446)

By using optimal merge sort algorithm:

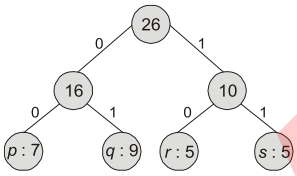


So, total number of comparisons = 205 + 127 + 75 + 39 = **446**.

21. (52)

of p's = 7
of q's = 9
of r's = 5
of s's = 5

Huffman tree will be:



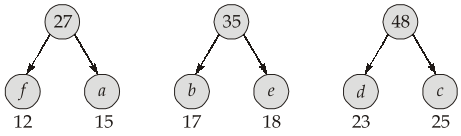
Encoded message:
p = 00
q = 01
r = 10
s = 11

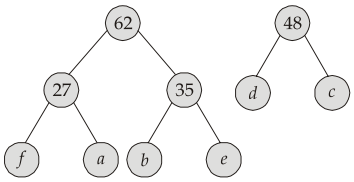
Number of bits for given message = (7 × 2) + (9 × 2) + (5 × 2) + (5 × 2)
= 14 + 18 + 10 + 10 = 52

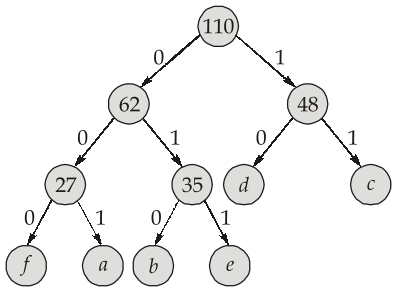
22. (2.56)

Total characters are 110 bits.

As Huffman tree:







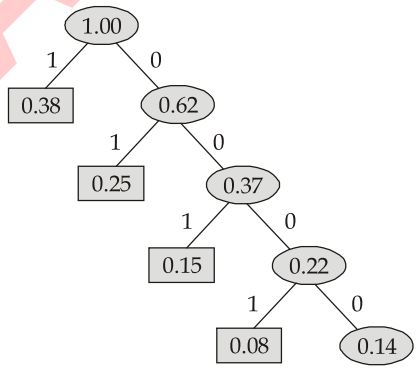
According to Huffman encoding
a = 001 ⇒ 3 bit
b = 010 ⇒ 3 bit
c = 11 ⇒ 2 bit
d = 10 ⇒ 2 bit
e = 011 ⇒ 3 bit
f = 000 ⇒ 3 bit

Number of bits needed = (3 × 12) + (3 × 15) + (3 × 17) + (3 × 18) + (2 × 23) + (2 × 25) = 282

Average bits = $\frac{282}{110}$ = 2.56 bits/char

23. (2.21)

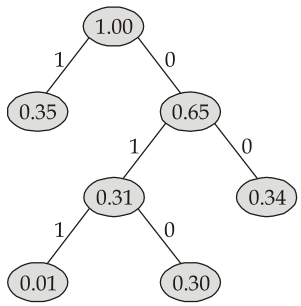
Huffman coding is used to find the optimal average length.



Average length = 0.38 × 1 + 0.25 × 2 + 0.15 × 3 + 0.08 × 4 + 0.14 × 4
= 0.38 + 0.5 + 0.45 + 0.32 + 0.56 = 2.21

24. (b, c)

Using min heap data structure:



W : 011 [3 bit]
X : 010 [3 bit]
Y : 00 [2 bit]
Z : 1 [1 bit]

Expected length: $[0.35 \times 1 + 0.34 \times 2 + 0.30 \times 3 + 0.01 \times 3] \times 200$

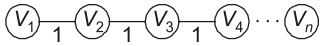
$= [0.35 + 0.68 + 0.90 + 0.03] \times 200$

$= [1.96] \times 200 = 392$

27. (b)

Prim's algorithm will pick up the edge with least weight for a particular node, [provided it does not form a cycle] weight of edge (V_{i-1}, V_i) or $(V_i, V_{i-1}) = 1$

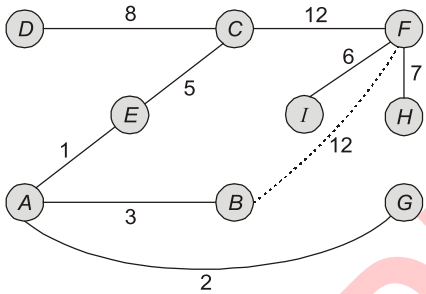
$\therefore$ MST will be



$\therefore$ Total edge weight $= 2 \times (n-1) = 2n-2$.

28. (12)

Minimum spanning tree:

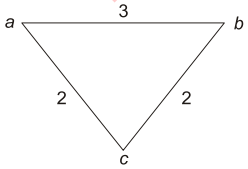


If BF edge represented by dotted line is added to the MST we can throw away largest edge in the loop ($AECFBA$).

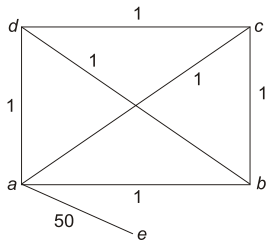
$BF_{max} = 12$

29. (d)

I. MST contain ac and bc but not contain ab, which is the shortest path between a and b



II. We may be forced to select the edges with weight much higher than average

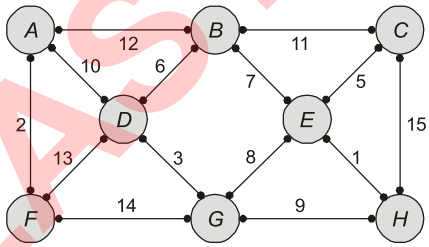


$$\text{Average weight} = \frac{50+6}{7} = 8$$

Expected MST weight $= 4 \times 8 = 32$

Actual MST weight $= 50 + 6 = 56$

30. (82)



The minimum value of sum (a, b, c, d, e, f, g) $= 8 + 9 + 11 + 12 + 13 + 14 + 15 =$ **82**

31. (19)

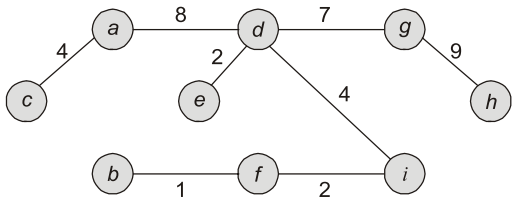
$x \le 11$

$z \le 8$

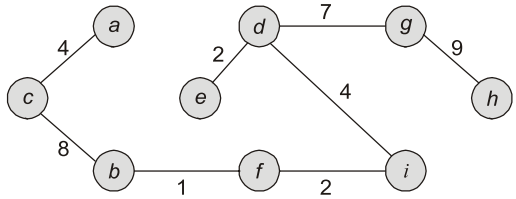
$x + z = 11 + 8 = 19$

32. (c)

Graph has 2 MST's:



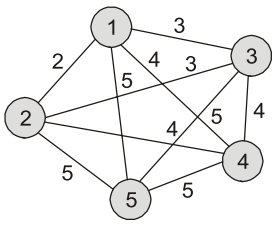
and



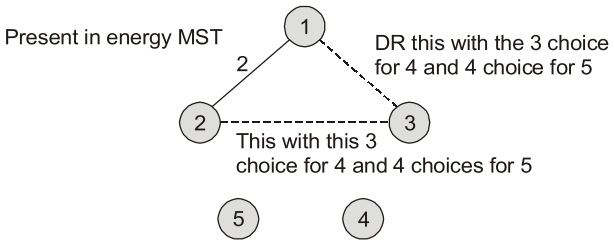
Since option (c) has forest edge (*b*, *f*) after (*a*, *d*) which is not allowed in Prim's.

33. (24)

Graph with edge weight looks like:



MST:



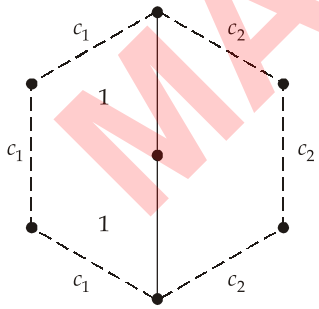
For every weight '4' edge we have 3 choices and for every weight '5' edge, we have 4 choices.

Therefore ${}^{3}C_{1}$ (for 4) × ${}^{4}C_{1}$ (for 5) = 3 × 4 = 12

Number of edges with weight '3' = 2

So, Total MST = 2 × 12 = 24

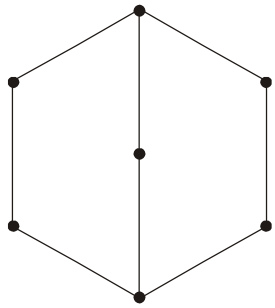
34. (a)



Three choices from c_1 and three choices from c_2

$\Rightarrow \quad 3c_2 \times 3c_2 = 9$

So, $x = 9$



We want second minimum spanning tree so for middle point we have 2 choices and rest there are 6 edges.

$\Rightarrow \quad 6c_5 \times 2c_1 = 12$

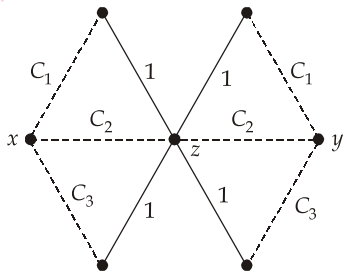
So, $y = 12$

$|2^9 - 2^{12}| = 2^9 \times (1 - 2^3)$

$= 2^9 \times 7 = 3584$

35. (9)

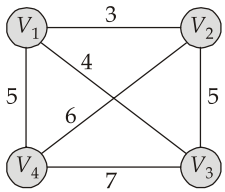
All minimum cost edge will be present in MST if not involve in cycle



Now, to connect 'x' and 'y' we have 3 choices each so, number of Minimum Spanning Tree (MST) are 3 × 3 = 9.

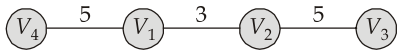
36. (b)

S_1 : Consider the following graph:

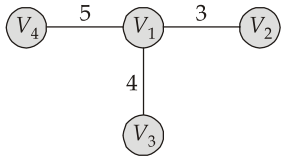


Minimum bottleneck spanning tree:

(A):



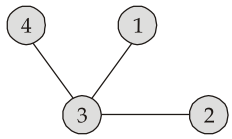
(B):



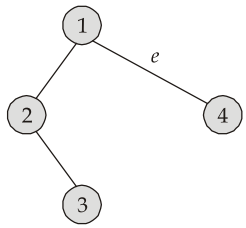
However only (b) is MST therefore S_1 is false.

S_2: Consider 2 MST's:

(A):



(B):



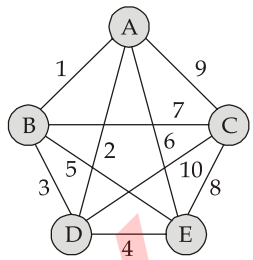
Where A has a lighter bottleneck edge. This means that B has an edge 'e' that is heavier than every edge of A. If we include this edge in A, it will form a cycle in which e would be heaviest. This contradicts the definition of MST.

Thus, 'e' cant be present in B, meaning that both A and B have same bottleneck edge.

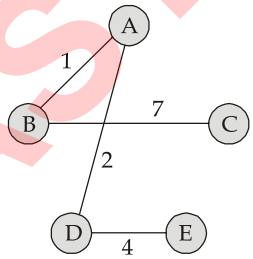
Thus, S_2 is true.

37. (14)

Consider edges weights are numbered as below:



So minimum cost spanning tree will be:



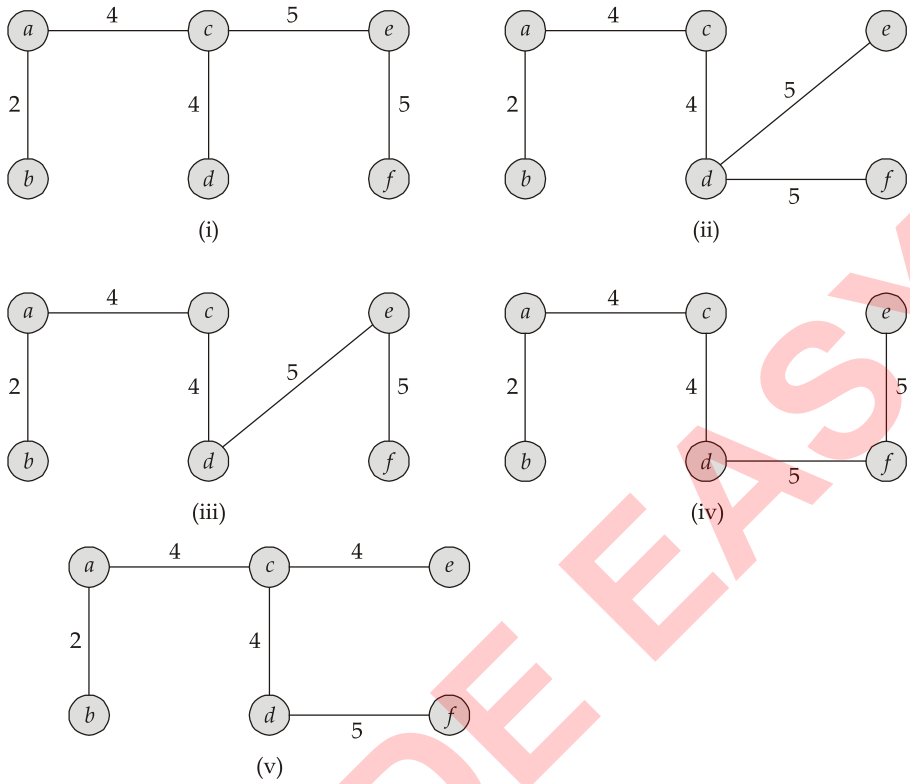
In worst case, cost of MST will be 1 + 2 + 4 + 7 = 14.

38. (b)

If all weight are unique than adding a constant to every edge weight in a directed graph can change the set of edges that belongs to minimum cost spanning tree.

39. (5)

We get same number of spanning tree using Prim's or Kruskal's algorithm.

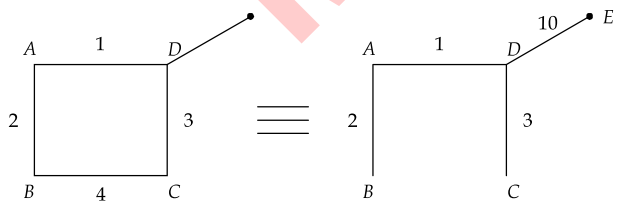


Total 5 minimum spanning tree exists.

40. (a)

If the weights are unique, then every minimum spanning tree V contains minimum edge weight.

But statement S_2 is false.



Clearly, maximum edge weight is in MST.

41. (a, c, d)

(a) True.
(b) False, can construct counter-example.
(c) True.
(d) True. It can be the first edge added by Kruskal's algorithm.

42. (a, c)

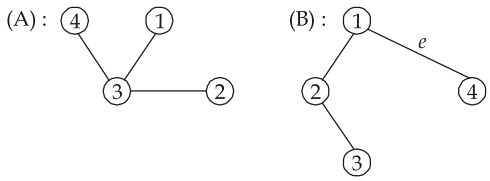
MST of G will have AD, CE, DF, AB, BE, GE as edges. Hence, sum will be

$(5 + 5 + 6 + 7 + 7 + 9) = 39$

43. (c)

S_1 is true.

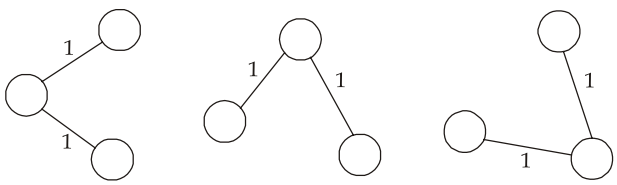
S_2 : Consider 2 MST's:



Where A has a lighter bottleneck edge. This means that B has an edge 'e' that is heavier than every edge of A. If we include this edge in A, it will form a cycle in which e would be heaviest. This contradicts the definition of MST. Thus, 'e' can't be present in B, meaning that both A and B have same bottleneck edge. Thus, S_2 is true.

44. (c)

Here all non-diagonal elements in the adjacency matrix are 1. So, every vertex is connected every other vertex of the graph. And, so graph M has 3 distinct minimum spanning trees.



45. (d)

A maximum/minimum spanning tree is a spanning tree of a weighted graph having maximum/minimum weight.

46. (b)

For Prim's algorithm, always use min heap not max heap.

47. (b)

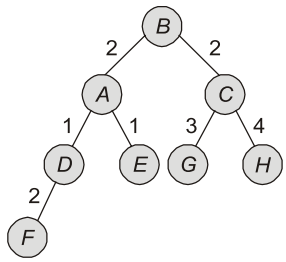
Maximum will be n, when $i = n$. Because suppose the newly added node can join all the vertex with minimum cost, then we will have to replace all older edges.

48. (c)

When applying Prim's algorithm, after vertex d, should go to vertex c, then should go to vertex n not from d to e. Because we should take minimum $(d \rightarrow e) = 6$ and $(c - e) = 4$.
So choose $c - e$ path.

49. (c)

Minimum spanning tree of the graph is



The above tree is almost complete tree.

■■■■

05 Dynamic Programming

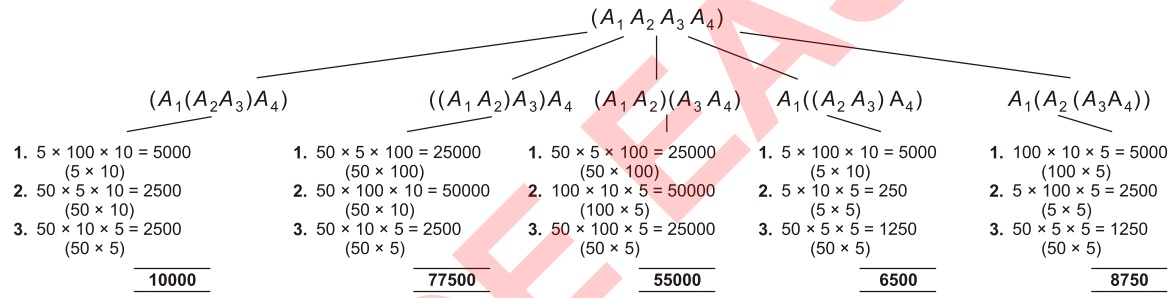
1. (d)

The given recurrence relation is denoted for matrix chain multiplication problem and the following are the conditions for recurrence relation.

If $i = j$; $m[i, j] = 0$

If $i < j$; $m[i, j] = \min_{i \le k < j} \{m[i,k] + m[k+1,j] + P_{i-1} P_k P_j\}$

2. (6500)

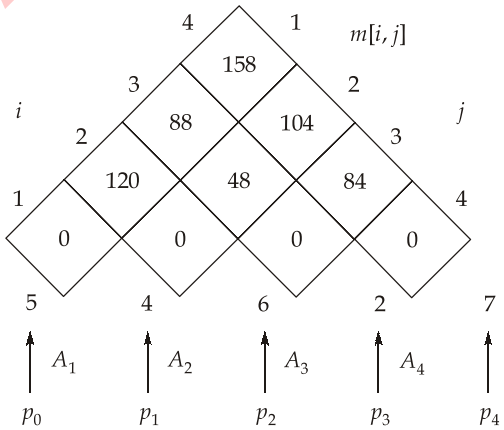


The minimum number of multiplication required using basic matrix multiplication method will be 6500.

3. (158)

$[A_1(A_2A_3)]\ A_4$ will give minimum answer.

4. (c)



The parenthesization that minimizes the number of scalar multiplications is $((A_1(A_2A_3))A_4)$:

$$(48 + 40) + 70 = 158$$

6. (a)

- Matrix chain multiplication using dynamic programming take $O(n^3)$ time while without dynamic programming (n^n).
- Fibonacci number take $O(n)$ using dynamic programming while $O(2^n)$ using without dynamic programming.
- LCS problem take $O(n^2)$ with dynamic programming and without dynamic programming $O(2^{2n})$.
- 0-1 Knapsack take $O(n^2)$ with dynamic programming i.e. one of no complete problem, because of very less repetition $O(m, n) \approx O(2^n)$ and without dynamic programming $O(2^n)$.

7. (71)

$$\left(\frac{p_1}{w_1}, \frac{p_2}{w_2}, \frac{p_3}{w_3}, \ldots, \frac{p_7}{w_7}\right) = (4, 2, 3.33, 1.142, 5, 6, 2.5)$$

Decreasing order of $\frac{p_i}{w_i}$ is $x_6, x_5, x_1, x_3, x_7, x_2, x_4$

Since capacity of bag is 17 we can only add the following items using the $\frac{p_i}{w_i}$ ratio.

$$x_6(4) + x_5(2) + x_1(3) + x_3(6) + x_7(2)$$

$$\text{Total profit} = 1 \times 24 + 1 \times 10 + 1 \times 12 + 1 \times 20 + 1 \times 5$$
$$= 24 + 10 + 12 + 20 + 5$$
$$= 71 \text{ units}$$

8. (122)

Maximum deadline is 6. Therefore maximum 6 jobs can be scheduled.

The following table shows the completion of tasks according to their deadlines to obtain maximum profit.

J_3	J_7	J_4	J_5	J_6	J_1
15	29	17	16	24	21
1	2	3	4	5	6

Profit = 122

9. (a)

Option (a) 19, as it is a 0/1 Knapsack we can include item 1, 2 and 4 to get maximum profit of 3 + 5 + 11 = 19.

10. (c)

$R^{(k)}[i, j] = R^{(k-1)}[i, j] \text{ || } (R^{(k-1)}[i, k] \text{ && } R^{(k-1)}[k, j])$

Transitive closure of directed graph computed using Floyd Warshall's algorithm. Which computes using all pairs shortest path algorithm.

$$R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array},$$

$$R^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$R^{(2)} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} (1, 2) \text{ \& } (2, 3) \Rightarrow (1, 3)$$

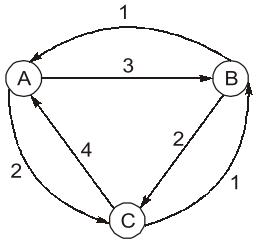
$$R^{(3)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, (1, 3) \text{ \& } (3, 4) \Rightarrow (1, 4)$$

$$R^{(4)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \text{Transitive closure of } R$$

11. (2)

$$D = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 3 & 2 \\ 1 & 0 & 2 \\ 4 & 1 & 0 \end{bmatrix} \end{array}$$

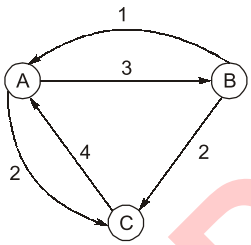
$\Rightarrow$



$$D^* = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 3 & 2 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{bmatrix} \end{array}$$

After removal of (C, B) edge, the graph is:

G_1 :



$$\Rightarrow \quad D_1 = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 3 & 2 \\ 1 & 0 & 2 \\ 4 & \infty & 0 \end{bmatrix} \end{array}$$

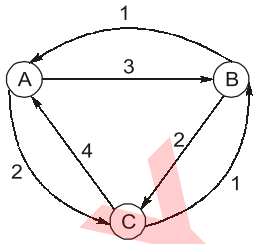
$$\Rightarrow \quad D_1^* = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 3 & 1 \\ 1 & 0 & 2 \\ 4 & 7 & 0 \end{bmatrix} \end{array}$$

$\Rightarrow$ "C to A" and "C to B" shortest path distances are changed.

12. (1)

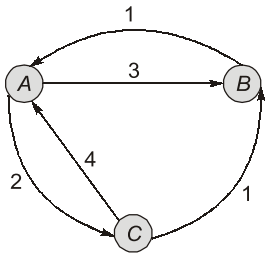
$$D = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 3 & 2 \\ 1 & 0 & 2 \\ 4 & 1 & 0 \end{bmatrix} \end{array}$$

$\Rightarrow$



$$D^* = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 3 & 2 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{bmatrix} \end{array}$$

After removal of (B, C) edge, the graph is:

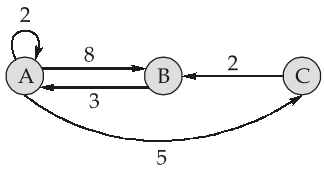


$$\Rightarrow \quad D_1 = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 3 & 2 \\ 1 & 0 & \infty \\ 2 & 1 & 0 \end{bmatrix} \end{array}$$

$$\Rightarrow \quad D_1^* = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 3 & 1 \\ 1 & 0 & 3 \\ 2 & 1 & 0 \end{bmatrix} \end{array}$$

$\Rightarrow$ B to C shortest path distances are changed.

13. (c)



$$\begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{bmatrix} \end{array}$$ is adjacency matrix for above graph.

$$A_0 = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{bmatrix} \end{array}$$

$$\Rightarrow \quad A_1 = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ \infty & 2 & 0 \end{bmatrix} \end{array}$$

$$A_2 = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix} \end{array}$$

$$\Rightarrow \quad A_3 = \begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 7 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix} \end{array}$$

■■■■

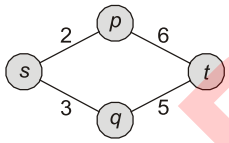
06 Graph Traversal and Sorting Algorithms

1. (c, d)

	DFS traversal	BFS traversal
Undirected graph	• Tree edges • Back edges [No forward edges No cross edges]	• Tree edges • Cross edges [No forward edges No back edges]
Directed graph	• Tree edges • Back edges • Cross edges • Forward edges	• Tree edges • Cross edges • Back edges [No forward edges]

3. (b)

Even if no two edges have the same weight, there could be two paths having the same weight.



To find the shortest path between s and t.

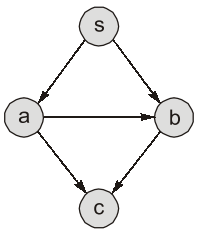
$s \to p \to t \Rightarrow 8$

$s \to q \to t \Rightarrow 8$

So, there are two distinct paths.

4. (c)

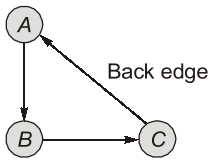
Consider the following graph:



- Vertices '*a*' and '*b*' both will be on the stack at the same time if at least one of them have an edge on the other vertex. Hence, If there is no directed path from '*a*' to '*b*' then there exist a directed path from '*b*' to '*a*'.
- Since, vertices '*a*' and '*b*' are traversed while performing DFS at vertex '*s*', so it can be said that there exist directed path from '*s*' to '*a*' and '*s*' to '*b*'. Hence, There exist directed path from '*s*' to 'a' and directed path from '*s*' to '*b*'.

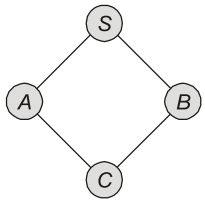
5. (c)

If directed graph contain cycle then it does not contain topological sorting. Since back edges in graph create cycles. So, if topological sorting exist for any graph, then it does not create back edge.

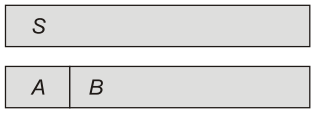


6. (c)

Consider a graph:



During the breadth first traversal of the graph. The status of the queue will be as follows:



$\therefore$ $S - A \to 1$ edge

$S - B \to 1$ edge

Difference = 0

B	C

$\therefore$ $S - B \rightarrow 1$ edge

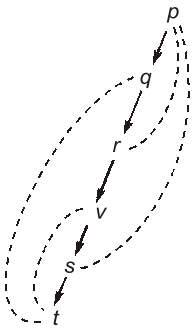
$S - C \rightarrow 2$ edge

Difference = 1

Hence, statements S_1 and S_3 are correct.

7. (c)

Apply the given DFS:

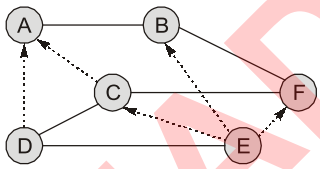


Tree edges are: {*p*, *q*}, {*q*, *r*}, {*r*, *v*}, {*v*, *s*}, {*s*, *t*}

Back edges are: {*t*, *q*}, {*t*, *v*}, {*s*, *p*}, {*r*, *p*}

8. (c)

Back edges are those edges connecting a vertex *u* to an ancestor *V* in a depth first tree.



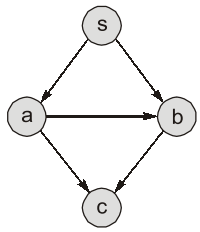
There are only 5 back edges.

9. (d)

1. If the graph has a Hamiltonian cycle, then it is possible depending on the graph ordering of the graph, that DFS will find that cycle and that the DFS tree will have depth V-1. However, DFS is not guaranteed to find that cycle. If DFS does not find that cycle then the depth of the DFS tree will be less than V-1.
2. False because, if the graph contains a negative-weight cycle, then no shortest path exist.

10. (c)

Consider the following graph:



- Vertices '*a*' and '*b*' both will be on the stack at the same time if at least one of them have an edge on the other vertex. Hence, If there is no directed path from '*a*' to '*b*' then there exist a directed path from '*b*' to '*a*'.
- Since, vertices '*a*' and '*b*' are traversed while performing DFS at vertex '*s*', so it can be said that there exist directed path from '*s*' to '*a*' and '*s*' to '*b*'. Hence, There exist directed path from '*s*' to '*a*' and directed path from '*s*' to '*b*'.

11. (2)

S_1: '*abedfc*' : DFS traversal.

S_2: '*acdebf*' : DFS traversal.

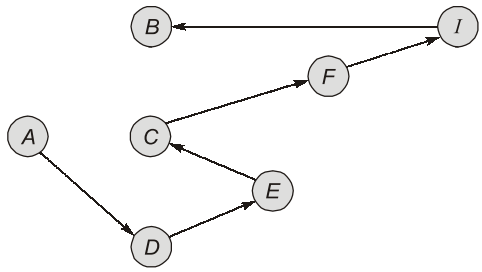
S_3: '*abcfde*' : It is neither BFS traversal nor DFS traversal.

S_4: '*bacedf*' : BFS traversal but not DFS traversal.

S_5: '*cdabef*' : BFS traversal but not DFS traversal.

12. (7)

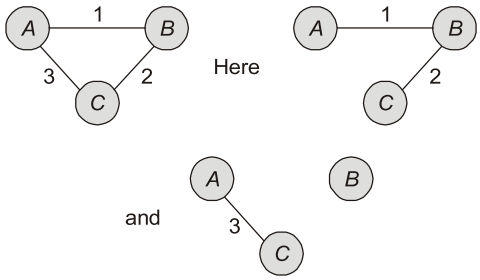
Maximum height of stack needed, when depth of the graph is maximum, so,



Which is 7, so maximum height of stack is 7.

13. (c)

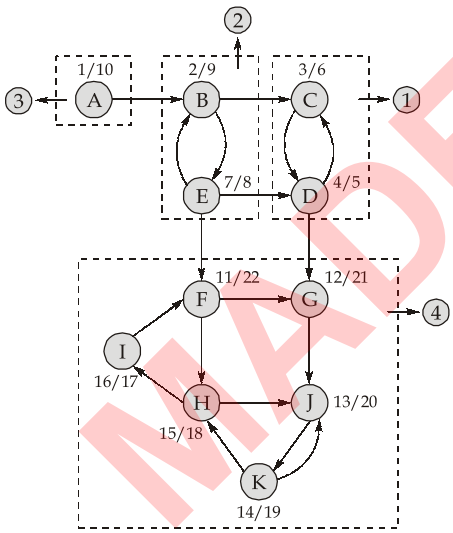
(i) For a directed graph, the absence of back edge in DFS tree means no cycle present.
So false.

(ii) 

Two paths are possible but cost is same.
So false.

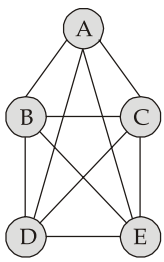
(iii) Depth of any vertex in BFS always less than equals to depth of same vertex in DFS.
So true.

14. (4)



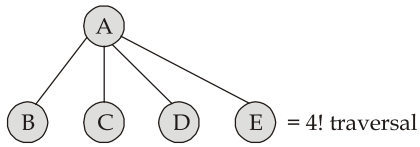
15. (120)

Complete graph on 5 vertices:



BFS traversal:

1. We have 5 choices to select initial vertex i.e. A, B, C, D and E.
2. With initial vertex as A



So total number of traversal are $5 \times 4! = 5! = 120$
Note: Number of BFS traversals on complete graph with n vertex are $n!$

16. (a)

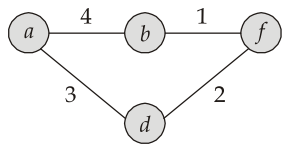
Take 2 new vertices a and b. Connect a to all vertices in v_1 and b to all vertices in v_2.
Perform BFS from a to b. The length of the path obtained minus 2 (one edge from a and one edge to b) will give the required result.

17. (b)

If $\forall i \forall j (P(i \to j) \wedge P(j \to i))$ then i and j belongs to same strongly connected components where i and j are vertices and $P(i \to j)$ means there is a path from i to j.
Solving by elimination
Option (d): P(U → T) does not exists thus false.
Option (c): From R we can reach P Q V S T and from P Q V S T we can reach R thus R should not be separated thus false.

18. (a)

(a) For a directed graph if DFS tree does not have back edges then there is no cycle.
(b) Shortest path between two vertices may not be unique.



Shortest path between $a - f$ is not unique.
(c) A complete graph can have maximum $n^{n-2} = 4^{4-2} = 16$ MST.
So only option (a) is true.

19. (a)

True. This graph only contains one cycle, which can be found by a DFS. Just remove the heaviest edge in that cycle.

False. If the heaviest edge in the graph is the only edge connecting some vertex to the rest of the graph, then it must be in every minimum spanning tree.

False. In the worst case we get unlucky and all the keys hash to the same slot, for $\Theta(n)$ time.

False. In a graph with three nodes, r u and v, with edges (r, u) and (r, v), and r is the starting point for the DFS, u and v are siblings in the DFS tree, neither as the ancestor of the other.

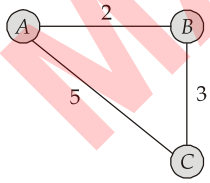
20. (a)

The condition can be simplified as, $n = m - 1$, i.e., the number of edges = number of vertices – 1. But this does not mean that G is a tree because even a disconnected graph can satisfy the given condition. So, option (a) is the best possible answer as we will have to use BFS.

21. (a, b, c)

- Only option (d) is false because distance 2 vertices can have different paths.

  Example:



  All edges having distinct weight. Here A to C have two different paths both having same cost of 5 unit.

22. (b)

Here are two possible orderings for BFS:

- A, G, F, B, C, D, E, H
- A, C, B, F, G, H, E, D

Here are two possible orderings for DFS:

- A, G, H, F, C, D, E, B
- A, B, C, E, D, F, H, G

So, I and III are BFS, II and IV are DFS.
Option (b) is correct.

23. (24)

$$b \underbrace{[a\ e\ h\ f]}_{4!}\ g = 24$$

24. (d)

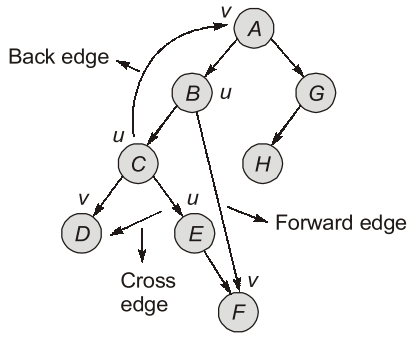
Given an adjacency-list representation Adj of a directed graph, the out degree of a vertex u is equal to the length of Adj[u] and the sum of the lengths of all the adjacency lists in Adj is $|E|$. Thus the time to compute the out-degree of one vertex v is $\Theta(|\text{Adj}(v)|)$ and for all vertices is $\Theta(V + E)$ with $\Theta(V)$ additional storage.

25. (b)

DFS of a graph whether its directed or undirected, iff there is a cycle, it must contain a back edge.

26. (d)

All are true.
Let's take an e.g.:



28. (b)

Whether the representation of graph is in adjacency list or adjacency matrix, time complexity of finding the set of vertices from a given vertex is 0.

29. (b)

(i) In a DFS of undirected graph, there is either tree or back edge.

30. (c)

Code:

```
for(i = 0, i < n; i++)
{
    for(J = 1 + i; J < n; J++)
    {
        t = a[i] a[J];
        a[i][J] = a[J][i];   } swap
        a[J][i] = t;
    }
}
```

$$n-1+n-2+\ldots..+1=\frac{n(n-1)}{2}$$

31. (c)

```
for(k = 0; k > n, k++)
{
    for(i = 0; 1 < n, i++)
    {
        for(J = 0; J > n, J++)
        {
            if (a[k][J] && a[J][i])
                b[k][i] = 1;
        }
    }
}
```

First initialize all cells of $b[n][n]$ as zero and then apply above code segment to get the square of graph whose adjustment matrix is given by $a[n][n] \cdot b[n][n]$, will give the matrix of the square of the graph.

33. (a)

By using counting sort we can find number of elements present in range of j to k.

Example:

Let given array

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
B	2	0	2	3	0	1

Index is max element in given array O(n) time.

Array B count number of time element present in array.

Now $B[a] = B[a] + B[a-1]$

	0	1	2	3	4	5
B	2	2	4	7	7	8

Now if we want number of element in range of j to k.

Use $= B[\ ] = B[k] - B[j-1]$ to find number of elements.

34. (16)

So, both algorithm take same steps:

$$8n^2 = 32\, n\log n$$
$$n^2 = 4\, n\log n$$
$$n = 4\log n$$

So, when $n = 16$

$$16 = 4\log_2 16$$
$$\Rightarrow \quad 16 = 4 \times 4$$
$$\Rightarrow \quad 16 = 16$$

So, when $n = 16$, both algorithms take same number of steps.

35. (c)

S_1 : There exists a implementation in which we make merge sort stable.

S_2 : Radix sort also works in linear time if the elements to sort are integers in the range $\{0 \ldots.. n^d\}$ for any constant d given range (0 to cn) is subset of (0 to n^d).

36. (3900)

We know the range of the element present in the array A, so can use counting sort also, takes $O(n)$ time.

$$C \cdot n = 650$$
$$C \cdot 50 = 650$$
$$C = 13$$

For $n = 300$

$$300 \times 13 = 3900$$

37. (b)

Selection sort takes $n-1$ swap maximum.

38. (c)

In bubble sort, compare adjacency elements.
28, 37, 14, 51, 57, 34, 88
After 1st iteration: 28, 14, 37, 51, 34, 57, 88
After 2nd iteration: 14, 28, 37, 34, 51, 57, 88

39. (c)

Insertion and quick sort have a $O(n^2)$ worst case performance while heap sort have $O(n \log n)$ worst case performance.

40. (d)

When input array is sorted:

(i) Quick sort takes $O(n^2)$ while average case $\theta(n \log n)$.

(ii) Bubble sort takes $O(n)$ while average case $O(n^2)$.

(iii) Selection sort takes $O(n^2)$ whatever the input.

41. (d)

(i) If we use linked list instead of array, no effect on performance, it remains $O(n^2)$.

(ii) If we use binary search, it takes log i comparison for pass i in worst case but still it takes i array shift operation for pass i. Thus, no effect on performance.

42. (b)

A binary tree of height h has at must 2^h leaves.
Number of leaves in the decision tree = Number of permutation on n elements = $n!$

$$n! \le 2^h$$
$$\log(n!) \le \log_2 2^h$$
$$\log(n!) \le h$$
$$h \ge \log(n!)$$

43. (d)

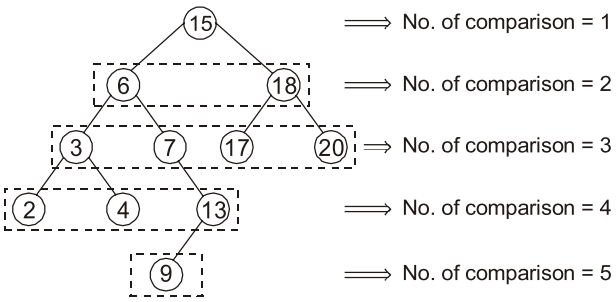
- Selection sort has least number of swaps. If record size large, swapping might be costly. Hence selection sort is better.
- Quick sort performs better when input size is large.

■■■■

07 Miscellaneous

2. (3.09)

The tree when constructed will result in



Let 'x' be the number of comparisons.
Let P(x) be the probability of elements having 'x' comparisons

x	1	2	3	4	5
P(x)	$\frac{1}{11}$	$\frac{2}{11}$	$\frac{4}{11}$	$\frac{3}{11}$	$\frac{1}{11}$

$$E(x) = \frac{1\times1+2\times2+3\times4+4\times3+5\times1}{11}$$

$$= \frac{1+4+12+12+5}{11} = 3.09$$

3. (327)

Step 1: Initially car tank is full. So from s to a_3 can be travelled without any cost.
Step 2: Since at city a_4 cost is minimum, full the tank at a_4, and buy gas for 20 miles (in 20).
Step 3: At city a_5, cost is more than city a_6, so at a_5, only buy gas for 5 miles to reach city a_6 (in 25).
Step 4: Drive to city a_6 and buy gas for 20 miles (80).
Step 5: Drive to city a_7 and buy gas for 15 miles (135).
Step 6: Drive to city 8 and buy gas for 5 miles (45).
Step 7: Drive to city 9 and by gas for 4 miles (16).

Total cost = 6 + 20 + 25 + 80 + 135 + 45 + 16
= 327

4. (b)

Number of balloons originally	Total balloons shot
1	1
2	3
3	5
n	$2 \times n - 1$

5. (b)

Since a degree sequence is given, Haval-Hakimi algorithm can be applied to it.
In this initially we have to sort the given degree sequence, and then decrement the degree sequence $a[i]^{th}$ time i starting from 0. The obtained output is sorted again.
In this way, the algorithm will take $O(n^2 \log n)$ time.

6. (a)

A vertex is super sink if and only if $M[i, j] = 0$ $\forall$ j and $M[j, i] = 1$ $\forall$ $j \neq i$.
Traverse over the matrix and check the above condition,
Using below program we can find super-sink in directed graph:

```
i = 0;
do
{
    j = i + 1;
    while ((j < n) && !A[i][j]) j++;
    if (j < n) i = j;
```

```
} while (i < n);
flag = 1;
for (j = 0; j < n; j++)
    if ((j != i) && (A[i][j] || !A[j][i])) flag = 0;
        if (flag) printf("Sink exists");
        else printf ("Sink does not exist");
```

7. (65536)

Suppose data size of A is n

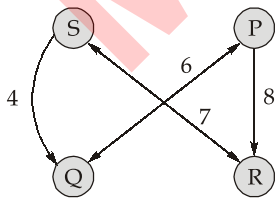
$$C \cdot \log\log n = 75$$
$$C \cdot \log\log 256 = 75$$
$$C = \frac{75}{3} = 25$$
$$25 \cdot \log\log n = 100$$
$$\log\log 65536 = 4$$
$$\log 16 = 4$$
$$n = 65536$$

8. (c)

Index	1	2	3	4	5	6	7
A (input)	7	1	2	5	3	6	4
B (output)	2	3	5	7	4	6	1

- Since B is not a sorted array, but it is reverse index of all elements of array 'A'. i.e., array B is permutation of array A (1 permutation but of $n!$ permutation).
- A sorted permutation of A always give sorted array 'B'.

9. (25)



Optimal cost = 4 + 6 + 7 + 8 = 25

10. (42)

The maximum sum of the contiguous subsequence will be obtained by using (10, –8, 30, 10) as a subsequence.
Maximum sum (10, –8, 30, 10) = 42

11. (c)

False. Counting sort is stable. It is not in-place, however, since we must make additional space to store the counts of the various elements. This space requirement grows as the size of the input increases. Additionally, we have to make a separate output array to produce the answer using counting sort.
False. The level of a vertex only provides the length of the shortest path from s.
True.
False:
Case 1: $f(n) = g(n) = n$. In this case, it $2^n = O(2^n)$ is certainly true.
Case 2: $f(n)=10n$, $g(n) = n$. In this case, we have $2^{(10n)} \neq O(2^n)$.

12. (a)

If we see the pattern carefully by taking small values, it always follows that $A(n)$ is always the sum of $A(n-1)$ and $A(n-2)$.
Hence, $A(n)$ is equivalent to the n^{th} Fibonacci number.
Thus, using dynamic programming, the time complexity will be $O(n)$.

13. (8000)

Algorithm (A_1):
For $n = 16$, A_1 takes 48 seconds
$$\Rightarrow \quad cn = 48$$
$$c(16) = 48 \Rightarrow c = 3$$
So, for $n = 64$, $T(A_1)$ will equal to,
$$T(A_1) = c.n = 3(64) = 192 \text{ seconds}$$
Algorithm (A_2):
For $n = 16$, 512 seconds
$$kn^2 = 512$$
or $$k(16)^2 = 512$$
$$\therefore \quad k = 2$$
Now, for $n = 64$
$$T(A_2) = 2 \times (64)^2$$
$$= 2 \times 4096 = 8192 \text{ seconds}$$
Therefore,
$$T(A_2) - T(A_1) = (8192 - 192) \text{ seconds}$$
$$= 8000 \text{ seconds}$$

14. (c)

Algorithm is a logical step not the implementation by any language. So only II is incorrect.

■■■■