Q.2 a. Solve the recurrence relation: T (n)=2T(n/2) + n; T(1) = 0 (6)

Answer:

Refer Complexity analysis of merge soft.

b. Write the psuedocode for the Selection sort algorithm. What loop variant does it maintain? Why does it need to run for only the first n-1 elements rather than for all n elements? Give the best and Worst case time complexity of Selection sort. (4+1+1+4)

Answer:

Selection sort(A)

```
n ← 1 to length[A]
    for j ← 1 to n-1
      smallest ← j
        for i ← j + 1 to n
          if A[ i ] < A[ smallest ]
              then  smallest ← i
        Exchange A[ j ] ↔ A[ smallest ]
```

The algorithm maintains the loop in variants that at the start of each iteration of the for loop, the sub array A [1....*j-1*] consists of the *j-1* smallest elements in the array A[1.......*n*] , and this sub array is in the sorted order.

After the first n-1 elements, according to the previous invariants, the sub array A [1........*n-1*] contains the smallest n-1 elements, sorted. Hence element A*[n]* must be the largest element.

Best and Worst case Analysis:

When there is n value is the array, then during the first time, the function executes n-1 times, at the second time, it executes n-2 times…

so the time need to sort is n-1+n-2+…+1= (n(n-1))/2,

when n equal to infinite,

$= n^2$

So , the best, average and worst case time complexities of the selection sort are all **$\Theta(n2)$**

Or

$$S(n) = \sum_{J=1}^{n-1} \sum_{i=i+1}^{n} = \sum_{j=1}^{n-1}(n-j) = n(n-1) - \sum_{j=1}(n-1)$$

$= n(n-1) - 1/2\,(n-1)n$

$= 1/2\,(n-1)n$

$= \Theta(n^2)$

Q.3 a. Prove that for any two function f (n) and g (n),
***f*(n)= Θ (g(n)) if and only if *f*(n) =O(g(n)) and f(n) = Ω(g(n)) (5)**

Answer:

We have to prove that f(n)=O(g(n)) and f(n)=Ω(g(n)) f(n)= Θ (g(n))

To prove one proposition:

$f(n)=O(g(n))$ *i.e.* $f(n) \le c_0 g(n),\ n \ge n_0$

$f(n)=\Omega(g(n))$ *i.e.* $f(n) \ge c_1 g(n),\ n \ge n_1$

That means $c_1(g(n)) \le f(n) \le c_0 g(n),\ n \ge \max(n_0,n_1)$

Now to prove the other side of proposition:

$f(n)=\Theta(g(n))$; $c_0(g(n)) \le f(n) \le c_1 g(n),\ n \ge n_0$

Hence $f(n) \le c_1(g(n))$; $n \ge n_0$ *i.e.* $f(n)=O(g(n))$

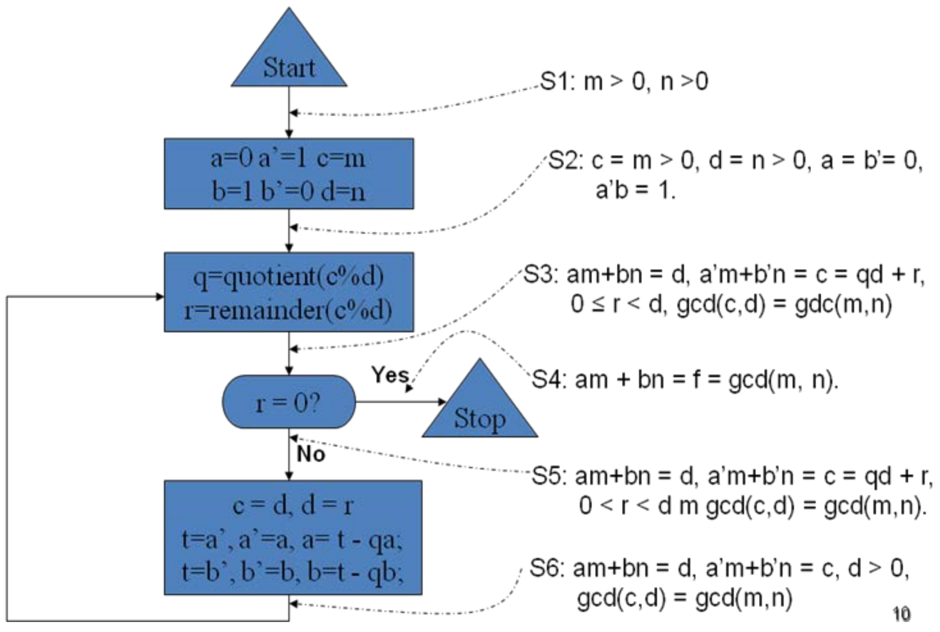
$f(n) \ge c_2(g(n))$; $n \ge n_0$ *i.e.* $f(n)=\Omega(g(n))$

Both side propositions are proved true, hence:

$f(n)=O(g(n))$ and $f(n)=\Omega(g(n))$ $\quad$ ~~f(n)~~ $\rightarrow \Theta(g(n))$

b. Draw the flow chart of extended Euclidian algorithm. (5)

Answer:


Start
S1: m > 0, n >0
a=0 a'=1 c=m
b=1 b'=0 d=n
S2: c = m > 0, d = n > 0, a = b'= 0, a'b = 1.
q=quotient(c%d)
r=remainder(c%d)
S3: am+bn = d, a'm+b'n = c = qd + r, 0 ≤ r < d, gcd(c,d) = gdc(m,n)
r = 0?
Yes
Stop
S4: am + bn = f = gcd(m, n).
No
S5: am+bn = d, a'm+b'n = c = qd + r, 0 < r < d m gcd(c,d) = gcd(m,n).
c = d, d = r
t=a', a'=a, a= t - qa;
t=b', b'=b, b=t - qb;
S6: am+bn = d, a'm+b'n = c, d > 0, gcd(c,d) = gcd(m,n)
10

c. Prove that travelling salesman problem is NP-Complete. (6)

Answer:

First, we have to prove that TSP belongs to NP

First, we have to prove that TSP belongs to NP. If we want to check a tour for credibility, we check that the tour contains each vertex once. Then we sum the total cost of the edges and finally we check if the cost is minimum. This can be completed in polynomial time thus TSP belongs to NP.

Secondly we prove that TSP is NP-hard. One way to prove this is to show that Hamiltonian cycle TSP (Hamiltonian cycle problem is NP-complete). Assume G = (V, E) to be an instance of Hamiltonian cycle. An instance of TSP is then constructed. We create the complete graph $G' = (V', E')$ where E' $\{(i, j){:}i, j \in V$ and $i \ne j$ Thus, the cost function is defined as:

$$t(i\ ,,j\ ) = \{\ \begin{array}{l} 0 \text{ if } (i,j) \in E \\ 1 \text{ if } (i,j) \notin E \end{array} \}$$

Now suppose that a Hamiltonian cycle h exists in G. It is clear that the cost of each edge in h is 0 in G as each edge belongs to E . Therefore, h has a cost of 0 in G′. Thus, if graph G has a Hamiltonian cycle then graph G has a tour of 0 cost.

Conversely, we assume that G′ has a tour h′ of cost at most 0. The costs of edges in E′ are 0 and 1 by definition. So each edge must have a cost of 0 as the cost of h′ is 0. We conclude that h′ contains only edges in E. So we have proven that G has a Hamiltonian cycle if and only if G has a tour of cost at most 0. **Thus TSP is NP-complete.**

Q.4 a. Write Quick sort algorithm and compute its worst case and best case time complexity. Illustrate the working on the array
A = <5, 3, 1, 9, 8, 2, 4, 7> (3+3+4)

Answer:
Algorithm:

```
Input: an array A[p, r]

 Quicksort (A, p, r) {
 if (p < r) {
 q = Partition (A, p, r)   //q is the position of the pivot element
Quicksort (A, p, q-1)
Quicksort (A, q+1, r)
}
}
```

Time complexity:
Best Case:
At every step, *partition*() splits the array as equally as possible ($k = (n+1)/2$; the left and right subarrays each have size $(n-1)/2$)). This is possible at every step only if $n = 2^k - 1$ for some k. However, it is always possible to split nearly equally.
The recurrence becomes
$C(n) = n + 2C((n-1)/2)$, $C(0) = C(1) = 0$,
which we approximate by
$C(n) = n + 2C(n/2)$, $C(1) = 0$
Except that the right side has n in place of n-1. The solution is **$C(n) = n\lg(n)$.**
Worst Case:
At every step, *partition*() splits the array as unequally as possible (k= 1 or $k = n$).
Then our recurrence becomes $C(n) = n + C(n-1)$, $C(0) = C(1) = 0$
$C(n) = n + C(n-1)$
$= n + n-1 + C(n-2)$
$= n + n-1 + n-2 + C(n-3)$
$= n + n-1 + n-2 + ... + 3 + 2 + C(1)$
$= (n + n-1 + n-2 + ... + 3 + 2 + 1) - 1$
$= n(n+1)/2 - 1$
$= n^2/2$
$= n^2$
Illustration of operation:
A= <5,3,1,9,8,2,4,7>

Step1: i= 5 *i.e.* pivot element ; 5,3,1,9,8,2,4,7 , j=7
Step2: 5,3,1,9, 8,2,4,7 i= 9, j=4
Step 3: 5,3,1,4,8,2,9,7 i= 4,j=9
Step4: 5,3,1,4,8,2,9,7 i=8,j=2
Step5: 5,3,1,4,2,8,9,7 i=2,j=8
Step 6: 2,3,1,4,5,8,9,7
Step7: <2,3,1,4> 5 <8,9,7>

Similarly Algorithm will create partition
< 1,2,3,4,>5,<7,8,9>

b. Write an algorithm to test whether a graph is bipartite. (6)

Answer:

To check if the given graph is bipartite, the algorithm traverse the graph labeling the vertices 0, 1, or 2 corresponding to unvisited, partition 1 and partition 2 nodes. If an edge is detected between two vertices in the same partition, the algorithm returns.

ALGORITHM: BIPARTITE (G, S)

```
For each vertex u in V[G] − {s}
  do color[u] ← WHITE
     d[u] ← ∞
     partition[u] ← 0
color[s] ← GRAY
partition[s] ← 1
d[s] ← 0
Q ← [s]
while Queue 'Q' is non-empty
    do u ← head [Q]
      for each v in Adj[u] do
          if partition [u] ← partition [v]
            then   return 0
           else
               if color[v] ← WHITE then
                   then color[v] ← gray
                        d[v] = d[u] + 1
                        partition[v] ← 3 − partition[u]
                        ENQUEUE (Q, v)
        DEQUEUE (Q)
Color[u] ← BLACK
Return 1
```

Q.5 a. Design an algorithm for Matrix multiplication with size N×N such that time complexity of the algorithm should not be greater than $2^{\log_2 N}$ (8)

Answer:

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$C_{11} = P_1 + P_4 - P_5 + P_7$
$C_{12} = P_3 + P_5$
$C_{21} = P_2 + P_4$
$C_{22} = P_1 + P_3 - P_2 + P_6$

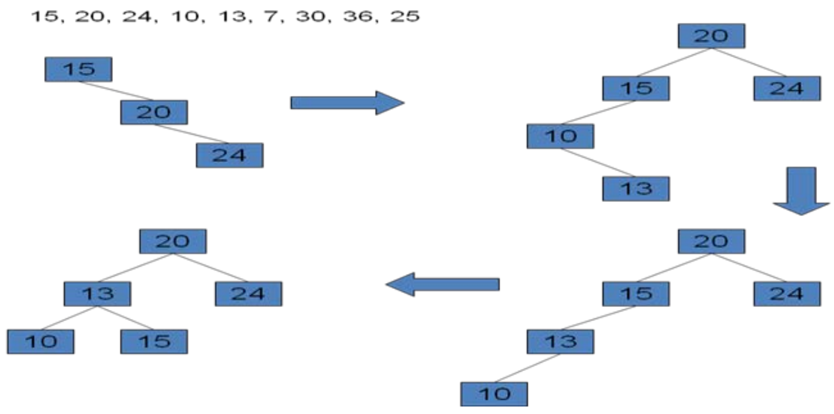
$P_1 = (A_{11}+ A_{22})(B_{11}+B_{22})$
$P_2 = (A_{21} + A_{22}) * B_{11}$
$P_3 = A_{11} * (B_{12} - B_{22})$
$P_4 = A_{22} * (B_{21} - B_{11})$

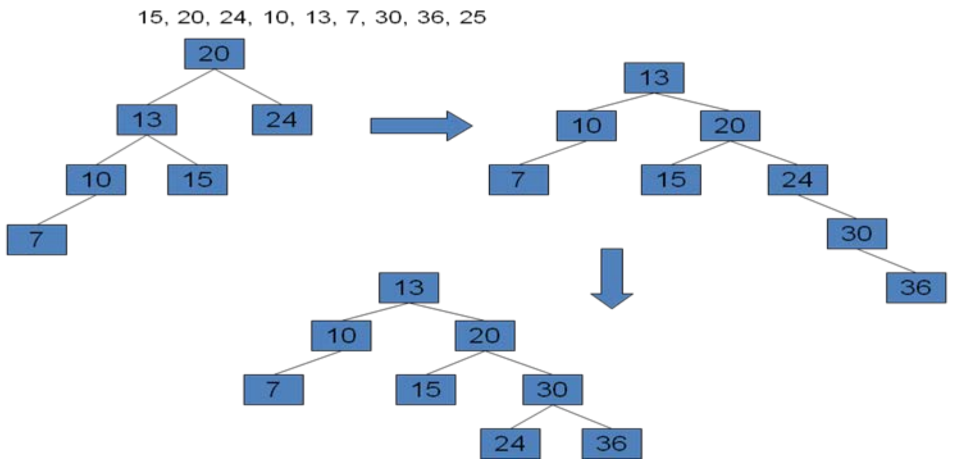
$P_5 = (A_{11} + A_{12}) * B_{22}$
$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$
$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

b. Construct an AVL search tree for the following given operation and values.
Insert 15, 20, 24, 10, 13, 7, 30, 36, 25
Remove 24 and 20 from the AVL tree **(2x4)**

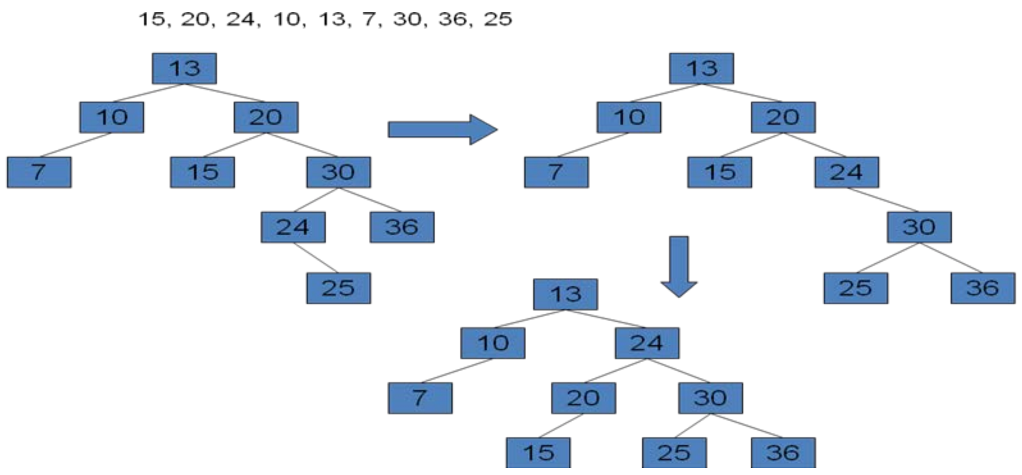
Answer:

(i)

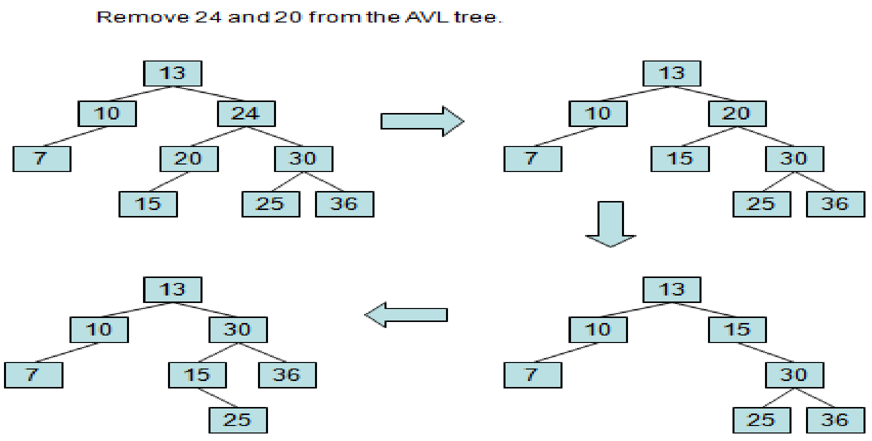




Final AVL tree after Insertion:



(ii) After Deletion



Q.6 a. Write heap sort algorithm and illustrate the working of the algorithm on the array A <4, 1, 3, 2, 16, 9, 10, 14, 8, 7> (5+5)

Answer:

HEAPSORT*(A)*

```
BUILD-MAX-HEAP(A)
 for i ← length[A] downto 2
do exchange A[1] ↔ A[i]
MAX-HEAPIFY(A, 1, i - 1)
```

<u>BUILD-MAX-HEAP(A)</u>

```
    n = length[A]
     for i ← ⌊n/2⌋ downto 1
    do MAX-HEAPIFY(A, i, n)
```

<u>MAX-HEAPIFY(A, i, n)</u>

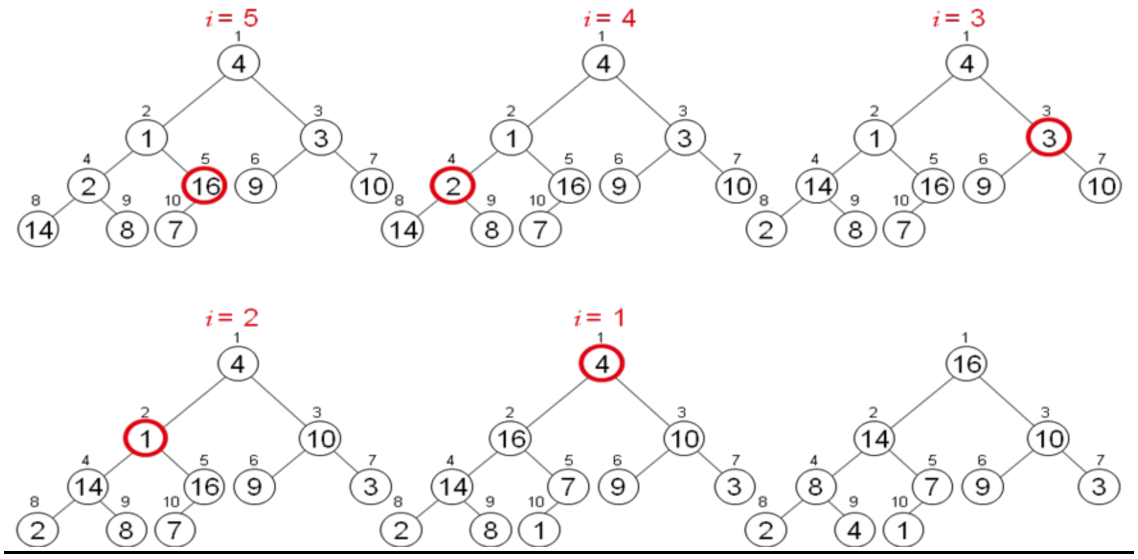
```
        l ← LEFT(i)
        r ← RIGHT(i)
        if l ≤ n and A[l] > A[i]
```

```
    then largest ←l
    else largest ←i
  if r ≤ n and A[r] > A[largest]
    then largest ←r
  if largest ≠ i
    then exchange A[i] ↔ A[largest]
         MAX-HEAPIFY(A, largest, n)
```

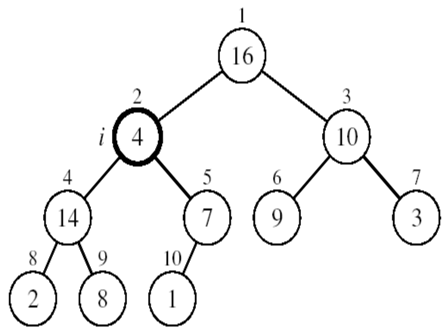
Illustration of operation

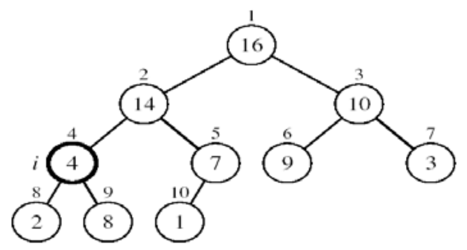
Heap Sort:

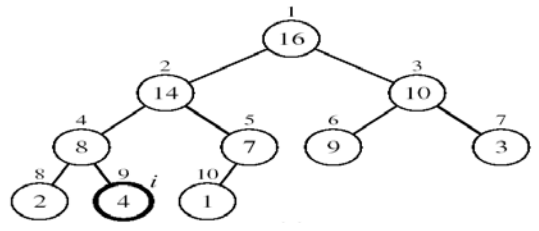
4	1	3	2	16	9	10	14	8	7



Max heapify:







b. What is Horner rule's? Explain with example. (6)

Answer:

A polynomial is expressed as:

$f(x) = a_0 + a_1x + a_2x^2 + ... + a_kx^k$

Where a_k are real numbers representing the polynomial coefficients and x^k are the polynomial variables. Horner's rule for polynomial division is an algorithm used to simplify the process of evaluating a polynomial *f(x)* at a certain value $x = x_0$ by dividing the polynomial into monomials (polynomials of the 1st degree). Each monomial involves a maximum of one multiplication and one addition processes. The result obtained from one monomial is added to the result obtained from the next monomial and so forth in an accumulative addition fashion. This division process is also known as synthetic division.

Example: Evaluate the polynomial $f(x) = x^4 + 3x^3 + 5x^2 + 7x + 9$ at x = 2

Since the polynomial is of the 4th degree, then n = 4

K	4	3	2	1	0
Step	$b_4 = 1$	$b_3 = 3 + 2 * 1$	$b_2 = 5 + 2 * 5$	$b_1 = 7 + 2 * 15$	$b_0 = 9 + 2 * 37$
Result	1	5	15	37	83

Therefore, f(2) = 83.

Q.7 a. Design an algorithm for topololical sorting using DFS. (2+4)

Answer:

Algorithm *topologicalDFS*(*G*)

```
 Input dag G
 Output topological ordering of G
 n ← G.numVertices()
for all  u ∈ G.vertices()
setLabel(u, UNEXPLORED)
for all  e ∈ G.edges()
  setLabel(e, UNEXPLORED)
```

```
    for all  v ∈ G.vertices()
   if  getLabel(v) = UNEXPLORED
    topological DFS(G, v)
```

Algorithm topological DFS(G, v)

```
    Input graph G and a start vertex v of G
    Output labeling of the vertices of G
        in the connected component of v
    setLabel(v, VISITED)
    for all  e ∈ G.incidentEdges(v)
    if  getLabel(e) = UNEXPLORED
        w ← opposite(v,e)
        if getLabel(w) = UNEXPLORED
            setLabel(e, DISCOVERY)
            topologicalDFS(G, w)
        else
            {e is a forward or cross edge}
    Label v with topological number n
    n ← n - 1
```

b. Write counting sort algorithm and illustrate it

A <4, 1, 3, 4, 3> (4+6)

Answer:

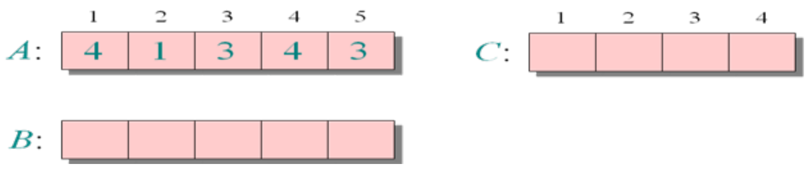
<u>Counting sort Algorithm</u>

Input: A[1 . . n], where A[j]€ {1, 2, …, k} .
Output: B[1 . . n], sorted.
Auxiliary storage: C [1 . . k] .

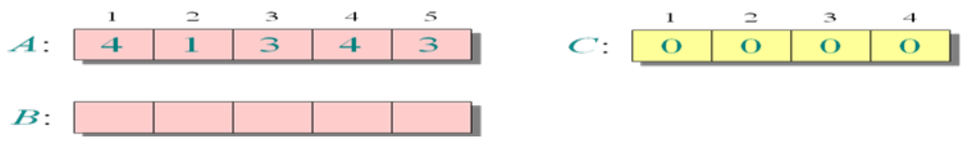
```
for i ← 1 to k
do C[i] ← 0
for j ← 1 to n
do C[A[ j]] ←C[A[ j]] + 1        ⊳ C[i] = |{key = i}|
for i ←2 to k
do C[i] ← C[i] + C[i–1]          ⊳ C[i] = |{key £ i}|
for j ←n down to 1
do      B[C[A[ j]]] ← A[ j]
        C[A[ j]] ←C[A[ j]] – 1
```

Illustration:

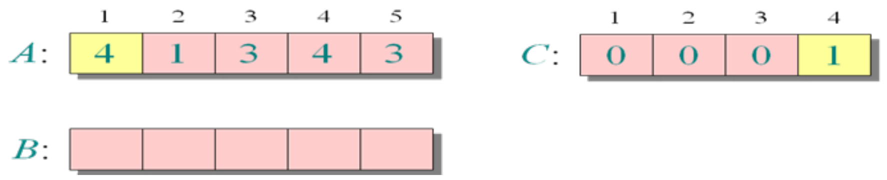
Step 1:



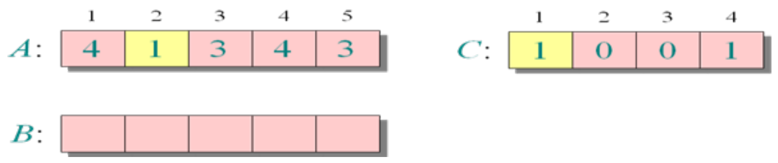
Loop 1:



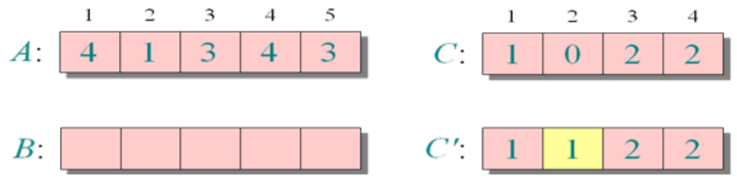
Result of Loop 1:



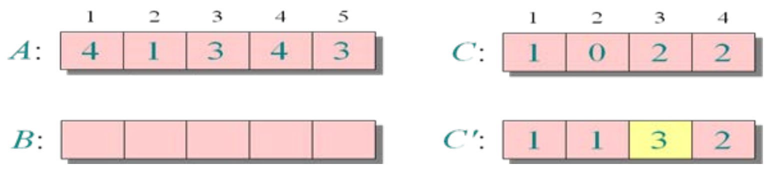
Loop 2:



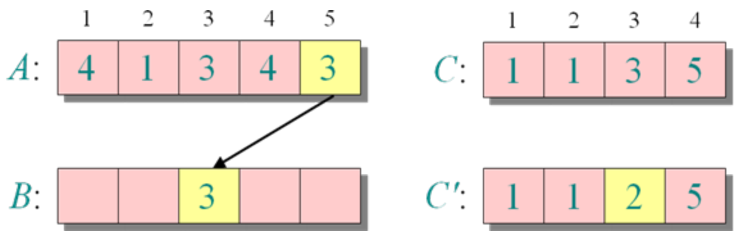
Result of Loop 2:



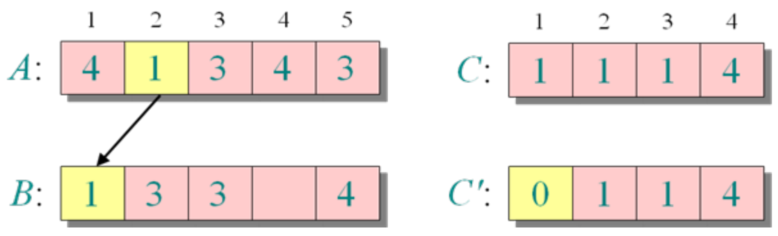
Loop 3:



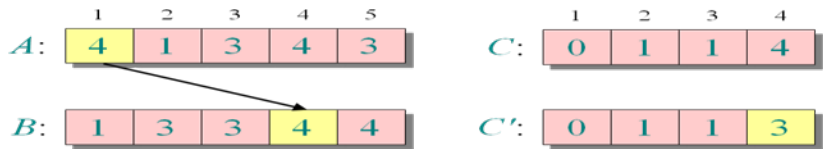
Loop 3:



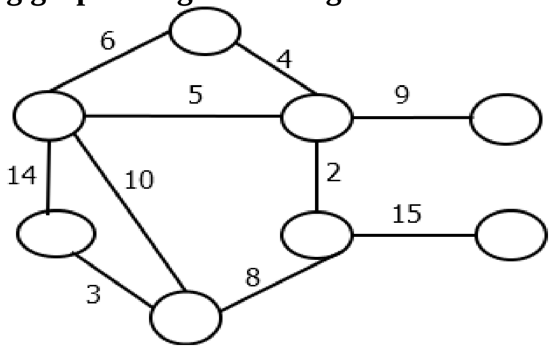
Loop 4:



Result of Loop 4:



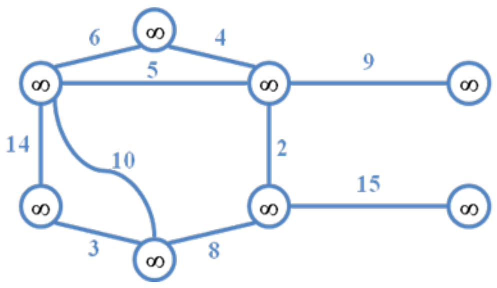
Q.8 a. What is minimum spinning tree? Generate the Minimum spinning tree for the following graph using Prim's algorithm. (3+7)



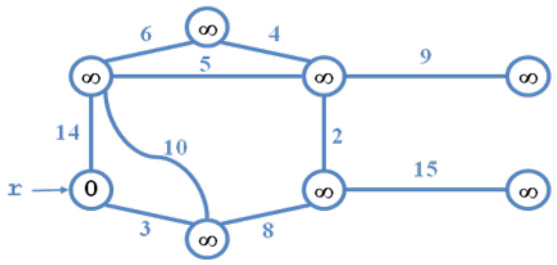
Answer:

A spinning tree of a graph is sub graph that contains all the vertices and is a tree. A graph may have many spinning tree. A minimum spinning tree is a spinning tree with weight less than or equal to the weight of other spinning tree

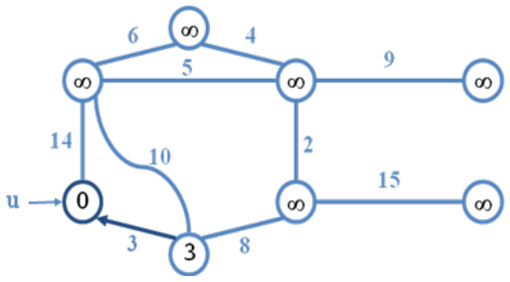
Step 1:



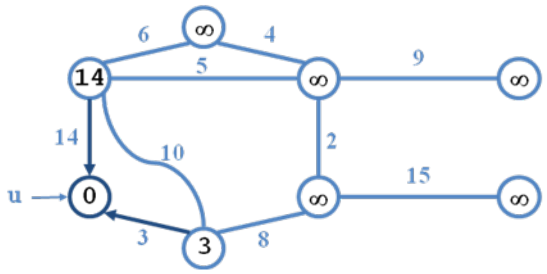
Step 2:



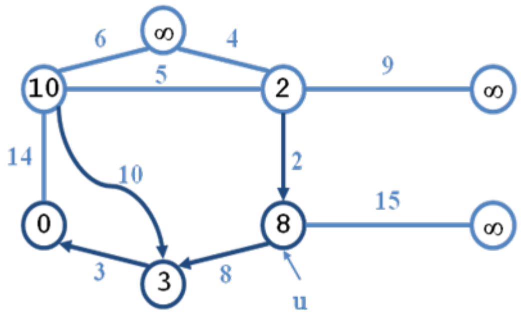
Step 3:



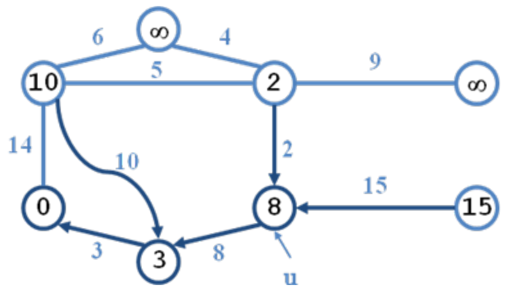
Step 4:



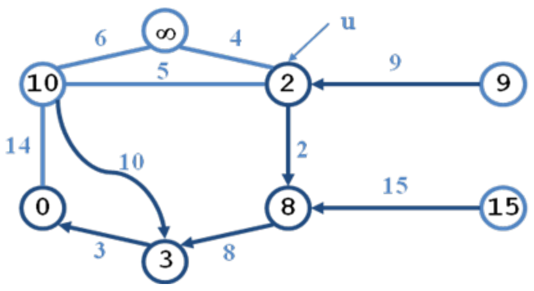
Step 5:



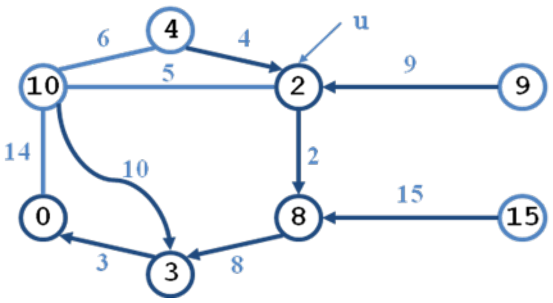
Step 6:



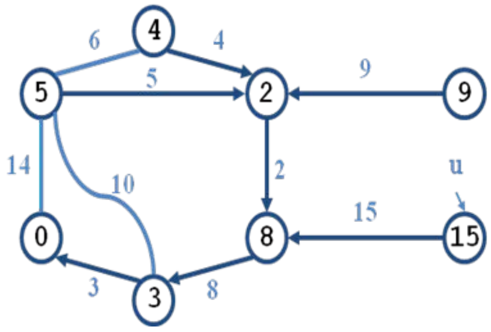
Step 7:



Step 8:



Final result:



b. Write the Horspool's string matching algorithm. (6)

Answer:

```
HorspoolMatching(P[O .. m -1], T[O .. n -1])
//Implements Horspool's algorithm for string matching
//Input: Pattern P[O .. m -1] and text T[O .. n -1]
//Output: The index of the left end of the first matching substring
// or -1 if there are no matches
ShiftTable(P[O .. m - 1]) //generate Table of shifts
i ←m - 1 //position of the pattern's right end

while i ≤ n - 1 do
k ←0 //number of matched characters

while k ≤ m- 1 and P[m- 1- k] = T[i- k]
do k← k+1
if k =m
retnrn i-m+l
else i ← i+Table[T[i]]
```

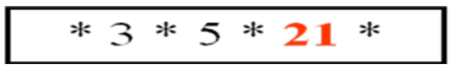
return -1

Q.9 a. Create B- tree of order 4 for the following operation with data
Insert: 5, 3, 21, 9, 1, 13, 2, 7, 10, 12, 4, 8
Delete: 2, 21, 10, 3, 4 **(5+5)**

Answer:

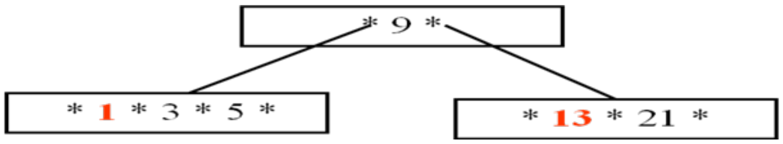
Step1:

Insert 5, 3, 21

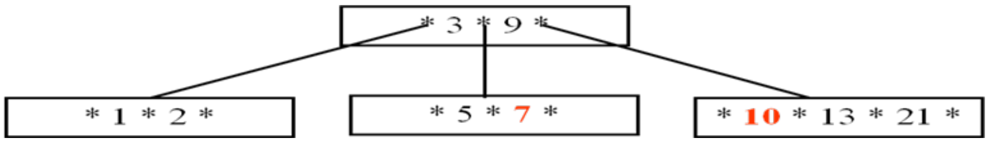


Step 2:

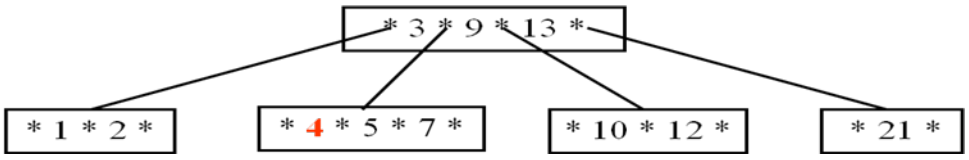
Insert 9,1, 13



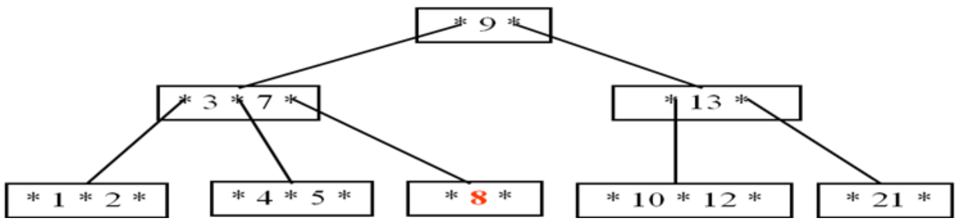
Step3: **Insert 2,7,10**



Step4: Insert 12,4

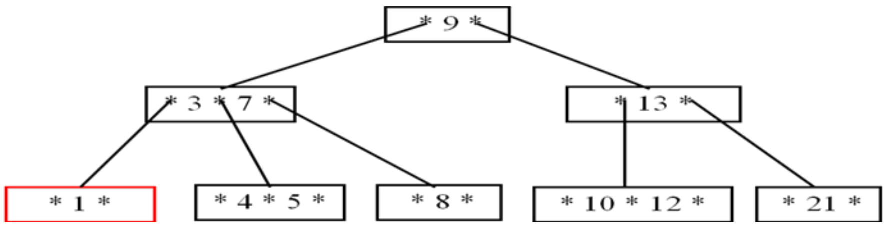


Step 5: Insert 8

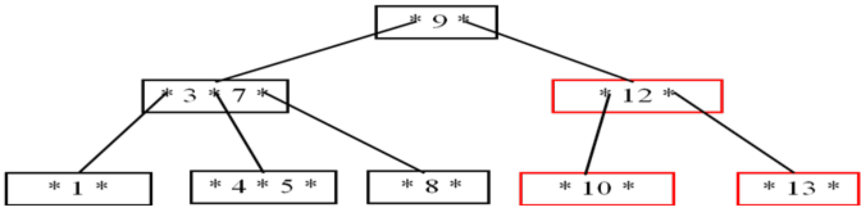


<u>**Delete:** 2, 21, 10, 3, 4</u>

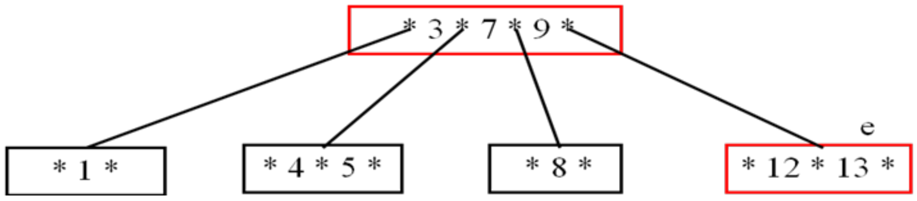
Step 1: Delete 2



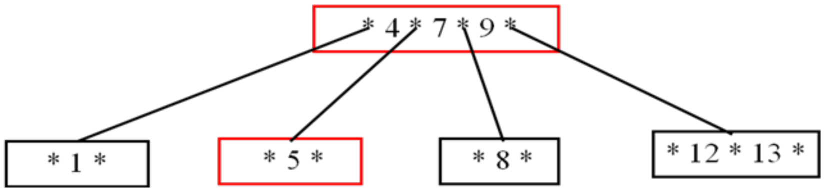
Step 2: Delete 21



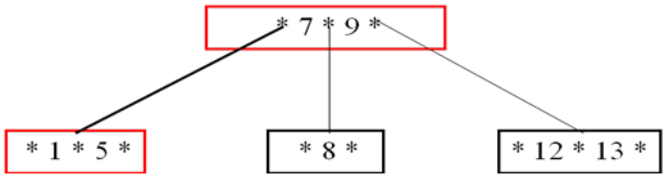
Step3: Delete 10



Step 4: Delete 3



Step 5: Delete 4



b. What is decision problem? Differentiate between Optimization and Decision Problems. (2+4)

Answer:

Decision problems are the computational problems for which the intended output is either “yes” or “no”. hence in decision problem, we equivalently talk of the language associated with the decision problem, namely, the set of inputs for which the answer is yes.

<u>Optimization problem vs. Decision problem:</u> Decision problem have Yes or No answer. Optimization problems require answers that are optimal configurations. Decision problems are easier than optimization; if we can show that a decision problem is hard that will imply that its corresponding optimization problem is also hard.

TEXT BOOK

I. Introduction to The Design & Analysis of Algorithms, Anany Levitin, Second Edition, Pearson Education, 2007 (TB-I)