ALGORITHM DESIGN 1

ASSIGNMENT: 1 → Introduction to AD; Heaps; Graph & Related Algorithms; Greedy Method

SECTION - A

Method of induction:-

① Prove :- for $n \geq 5$, $4n < 2^n$

Ans Proof (induction)

Base case :- For $n = 5$, we have $2^5 = 32 > 20 = 4(5)$

Inductive Hypothesis :-

Suppose that $2^K > 4K$ for some integer, $K \geq 5$

Induction step :- We wish to show that $2^{K+1} > 4(K+1)$

$$2^{K+1} = 2 \cdot 2^K = 2^K + 2^K > 4K + 4 > 4(K+1)$$

We have supposed true for K and shown true for K+1. Thus, by induction, $2^n > 4n$, if $n \geq 5$ (proved).

② Prove :- for all $n > 0$, $n^3 \leq n^2$

Ans Let represent the above fun^c, $f(n) = n^3 \leq n^2$

Base case, $n = 1$; $f(1) = 1^3 \leq 1^2$;

$1 \leq 1$ (true)

$f(n)$ is true for $n = 1$.

Inductive Hypothesis for $n = K$; so, $f(K) = K^3 \leq K^2$ (true)

Now for $(n = K+1)$

$$f(K+1) = (K+1)^3 = K^3 + 1 + 3K^2 + 3K$$
$$= K^2 + 1 + 2K + 2K^2 + K + K^3$$
$$= (K+1)^2 + 2K^2 + K + K^3 \leq (K+1)^2$$

So, $f(K+1) = (K+1)^3 \leq (K+1)^2$ is true ; i.e. inductive step

$n^3 \leq n^2$; for all $n > 0$ (thus proved)

Name → Sanjib Kumar Mohanty Sec → CSIT - A
Reg → 2041012276

COUNTER EXAMPLES

[redacted]

(1) P: a & b are even integers
Q: a + b is an even integer
P → Q (according to statement)
However, (Q → P) converse is not necessarily be true if we take a & b as odd integers.
Let's see an counter example:-
a = 3 & b = 9 (a & b are odd)
a + b = 3 + 9 = 12 (which is even)
Hence, converse is not true.

(2) If $(a+b)^2 = c^2$, such that $a+b=c$
P = a, b, c (real no.'s) — I

Q = $(a+b)^2 = c^2$; P → Q.
If $(a+b)^2 = c^2$, then a, b & c are real integers such that Q → P (converse of statement)

This converse is always true even if we take away real no.'s as counter example.

The statement can't be proved false.

(3) P = a, b, c ; Q = a/bc ; P → Q
converse ⇒ Q → P.
Let ⇒ a = 6 ; b = 8 ; c = 3. ; b × c = 24
⇒ $\frac{24}{a}$, $\frac{24}{b}$ = 4. But the converse is not true, because 8 is not divisible by 6.

(4) P :- a & b are rational
& Q :- ab is rational
$P \rightarrow Q$ ⇒ A/T statement
Converse is $Q \rightarrow P$, i.e. if product is rational, then a & b are rational.
However, the converse is not necessarily true if we take any two irrational no.s,
Counter Example such that product is rational.
Let $a = 3/\sqrt{2}$

PROOF OF CORRECTNESS

(1) Interchange (a, b)
Let a = 10 ; b = 5
a = 10 + 5 = 15
b = 15 - 5 = 10
a = 15 + 10 = 25
∴ ALGORITHM is NOT CORRECT

(2) To show :- $a + b < min(a, b)$; let a = 1, b = 2
$a + b = 1 + 2 = 3$
$min(1, 2) = 1$
$3 < 1$ (not possible)
∴ ALGORITHM is NOT CORRECT.

(3) Loop in-variant, $m \geq 0$
$a = bx + m$
(a) Before the 1st iteration ;
$m = a$, if $x = 0$; $a = 0 + m$
$\Rightarrow a = a$ (TRUE)

(b) $m' = m + b$ (l times) $(K < l < m)$ $m > 0$

Let $x' = x + l$

let $l = K$; $m = a$

$a = K'b + m'$

$a = Kb + m - b - b \ldots b$ (K times)

$a = m$; $a = a$; $m'' = m' - b$

$c = K + 1 = x''$; $u = (x+1)b + m' - b$

$a = m$

$a = a$ (true)

(c) Post cond^n - $m = a \bmod b$

$0 \le m < b$ &

$a = bx + m$

for n-1 cond^n ; $m = a \bmod (b(i-1)+1$

$= a \bmod b$

Hence,

m counts m no: after post cond^n of loops.

(4) (a) Since, 1st element is considered to be smallest element of array.

$i_{min} = 0 \Rightarrow K = 1$.

(b) Let's assume j iteration is true.

T.P. $\rightarrow$ $(j+1)$ iteration is true

$\Rightarrow$ upto $A[j-1]$ is sorted.

Now, it finds $A[j]$ which is smaller than $j-1$, it means element are sorted in correct place.

∴ invariant is at 1st iteration of inner loop ; $i_{min} = 0$, $K = 1$, $i < n$, true if $[A[i] < A[0])$, $A[0]$ & $A[1]$ is smaller.

④ for jth iteration, K = j+1 & comparison made b/w inserted & sorted array.
Now element is placed in context with A[min] if A[K] < A[K'] before A[min] is sorted.

⑤ j = n+1
→ A[1] ≤ A[i] ≤ A[3] ... A[A; min]
is already sorted.

⑤ Input m & n
Output product of m & n
x = m; y = n; b = 0.
Before 1st iteration;
z = mn - xy; z - m - mn = 0.
Term^n x = 0 prod. will be 0.
z = mn - xy
z = mn

⑥ Let y = 2; z = 3
x = 1
z = 3 (True) → x = 2, z = 2
z = 2 (n) → x = 4, z = 1
z = 1 (n) → x = 8; z = 0.
z = 0 false x = 8 = 2^3
∴ Algo. is correct

⑦ y = 5; z = 4
Part 1 (z = 4) while (2 > 0)
Part 2 (z = 2) while (2 > 0)
if (z mod = = 1) → 2 % 2 = 0 → false
y = 2 × 10 → y = 20
z = 2/2 → z = 1.

Part 3 (x=1) while (x>0)

if (x%2==1) → 1%2=0 → false.

y = 2×20 = 40

x = 1/2 = 0.

(8) Let

A = [1, 2, 4] B = [3, 10, 12, 14]

for n = 1 to 7

1 < 3 ⇒ c[1] = 1

i++;

2 < 3 ⇒ c[2] = 2

i++;

4 | < 3 ⇒ else ⇒ c[3] = 3 j++;

4 < 10 ⇒ c[4] = 4 i++;

now, i=5, but A[5] is out of index.

∴ This program isn't working correctly.

(9) $(\log n)^2 \times \sqrt{n} < 17n < n \log n < n^2 \log n < n^2 - 20n < n^2 - n^3 + n^4 < 2^n - 20n$.

(10) (a) $2^{2\log n + \lceil \log n \rceil} = (A)\ n^2$

$2^{n \log n + \lceil \log n \rceil} < n^2$

⇒ Let n=2

⇒ 2 = 1

$2^{0.602} = 1.5$? 10.

9. $(\log n)^2 < \sqrt{n} < 17n < n \log n < n^2 \log n < n^2 - 20n < n^2 - n^3 + n^4 < 2^n - 20n$

10.

(a) $2^{\lceil \log n \rceil + \lfloor \log n \rfloor}$

$= 2^{\log n + \log n} = 2^{\log n} \cdot 2^{\log n}$

$= 2^{\log n^2} = n^2$

$f(n) = n^2 \quad g(n)^2$

$f(n) = O(g(n))$

$n^2 = cn^2 \quad c = \forall n \geq 0$

Hence proved.

(b) $2^{\lfloor \log n \rfloor} = O(2^{\lceil \log n \rceil})$

$f(n) = 2^{\log n} = n \quad g(n) = 2^{\log n} = n$

$f(n) = g(n)$

$n = cn \quad$ put $c = 1 \quad \forall n \geq 0$

$n = n$

Hence proved.

(c) $2^{2\lfloor \log \log n \rfloor} = = O_2^{2\lceil \log \log n \rceil}$

$f(n) = 2^{2\lfloor \log \log n \rfloor} = 2^{\log \log n} = (\log n)^2$

$g(n) = 2^{2\lceil \log \log n \rceil} = 2^{\log \log n} \cdot (\log n)^2$

$f(n) = g(n)$

$(\log n)^2 = c \cdot (\log n)^2$ for $c = 1 \quad \forall n > 0$

Hence $2^{2\lfloor \log \log n \rfloor} = O(2^{2\lceil \log \log n \rceil})$

proved.

(d) Given; $f(n) = O(g(n))$

such that $n > n_0$ and $c > 0$

$f(n) \leq c(g(n))$

Taking log both sides;

$\log f(n) \leq \log c + \log \log (n) \quad \forall n \geq n_0$

since n_0 and c are constants there be constant d such that $d \geq \frac{\log c}{\log (g(n_0))} + 1$

(d-1) $\log(g(n_0)) \geq \log c$

$d \log(g(n_0)) \geq \log c + \log(g(n_0))$

$\log f(n) \leq \log c + \log(g(n)) \leq d \log(g(n))$

$\log(f(n)) = O(\log(g(n)))$ proved.

(e) Given $f(n) = O(g(n))$

$f(n) \leq c\, g(n)$

Multiply 2 on both side

$2f(n) \leq c\, 2g(n)$

$2f(n) = O(2g(n))$

proved.

(f) We have to prove $\log^k n = O(n^{1/k})$ for any positive integer k

put $k=2$ we get

$\log_2 n < (n)^{1/2}$

put $k = e \Rightarrow \log e^n = n^{1/e}$ which is true.

put $n = e+1 \Rightarrow \log_{e+1} n < n^{1/e+1}$

Hence for any $k > 0$

$\log^k n = O(n^{1/k})$

11. (a) $A(n) = A(n/3 + 5 + \lfloor \log n \rfloor) + n \log\log n$

$a = 1 \quad b = 3 \quad k = 1$

$a < b^k$

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

(b) $B(n) = \min_{0<k<n} (3 + B(k) + B(n-k))$

$B(n) + B(n-k) = k\,\frac{(k+1)(n-k)(n-k+1)}{2}$

$= \frac{n(n+1)}{2} - k(n-k) = O(n^2)$

(c) $D(n) = n/(n-5)\, D(n-1)+1$

$D(n) = n/n-5 \cdot D(n-1)+1$

$D(n-1) = \frac{n-1}{n-6} \quad D(n-2)+1$

$$D(n) = \frac{n}{n-5}\left[\frac{n-1}{n-6} D(n-2)+1\right]+1$$

$$\vdots$$

$$\frac{n(n-1)(n-2)\cdots D(0)^{+n}}{(n-5)(n-6)(n-7)\cdots} = \frac{n!}{(n-5)!}$$

$$= O(n!)$$

(d) $E(n) = E(\lfloor 3n/4 \rfloor) + 1 \;/\sqrt{n}$

$a = 1,\ b = 4/3,\ K = -1/2$

$b^K = (4/3)^{-1/2} = 0.866,\quad 1 > 0.866$

$E(n) = \Theta(n^{\log_{4/3} 1})$

$= \Theta(n^{\log 1/\log 4/3}) = O(n^0) = O(1)$

(e) $F(n) = F(\lfloor \log_2 n \rfloor) + \log n$

$= F(\log\log n) + \log n + \log\log n$

$= F(\log\log\log n) + \log n + \log\log n + \log\log\log n$

$= \Theta(\log n)$

(f) $G(n) = n + 7\sqrt{n} \cdot G(\lfloor \sqrt{n} \rfloor)$

let $G(2) = 1$

so,

$G(n) = n + 7n^{1/2} \times G(n^{1/2})$

$= n + 7n^{1/2}(n^{1/2} + 7n^{1/4} G(n^{1/4}))$

$= n + 7n + 7^2 n^{3/4} \cdot G(n^{1/4})$

$= n + 7n + 7^2 n + 7^3 n^{7/8} \cdot G(n^{1/8})$

$\vdots$

$= n + 7n + 7^2 n + \cdots 7^{k-1} n + 7^k n^{\left(\frac{2^k-1}{2^k}\right)} + G(n^{1/2^k})$

$= n\left(\frac{7^k - 1}{6}\right) + 7^{15} \cdot n^{\frac{2k-1}{2k}} \cdot G(n^{1/2k})$

$= n\left(\frac{7\log\log n - 1}{6}\right) + 7^{\log\log n}\, n^{\left(\frac{\log n - 1}{\log n}\right)}$

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

(g) $H(n) = \log 2\,(H(n-1)) + 1$

$H(0) = 1$

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

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

$H(n-2) = H(n-3) + 1$

⋮

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

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

⋮

$H(n-k) + k \quad [n-k = 0,\ n > k]$

$= H(0) + n$

$= 1 + n$

$H(n) = Q(n)$

(h) $I(n) = I(\lfloor n^{1/4} \rfloor) + 1$

⇒ Base case → $I(4) = 1$

$I(n) = I(n^{1/4}) + 1$

$= I(n^{1/16}) + 1 + 1$

$= I(n^{1/64}) + 1 + 1 + 1$

⋮

$= I(n^{1/4^k}) + k$

$= I(4) + \frac{1}{4} \log \log k$

$= 1 + \frac{1}{4} \log \log k$

$I(n) = Q(\log \log n)$

(i) $J(n) = J(n - \lfloor n/\log n \rfloor) + 1$

Here, $n >> \lfloor n/\log n \rfloor$

So, $n^{\log_b a} = n^{\log 1} = 1$

So, $f(n) = O(1)$

By Masters theorem,

$J(n) = Q(\log n)$

(j) $C(n) = 3C(\lceil n/2 \rceil - 5) + n/\log n$

$= 3C(n/2) + n/\log n \quad [n/2 >> 5]$

$\therefore n/2 - 5 \approx n/2$

⇒ $a = 3$ $b = 2$ and $f(n) = n/\log n$

Now, $n^{\log_2 3} = n^{1.58}$

So,

$C(n) = Q(n^{1.58})$

12. for i = 1 to n do
    for j = i to 2 * i do
        output "ALGORITHMS"

(a) $T(n) = \sum_{i=1}^{n} \sum_{j=1}^{2i} 1$

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

$= \sum_{i=1}^{n} (i+1) = \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1$

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

13. Sum (n)
    s ← 0
    for i ← 1 to n do
        s ← s + i
    return s

(a) Sum of n numbers
(b) Basic operation is addition
(c) n+1 times
(d) O(n)
(f) s(n)
    s ← 0 $s = \frac{n(n+1)}{2}$ returned efficiency O(1).

14. function multiply (y, z)
    if z = 0 then
        return 0
    else
        if z is odd then
            return (multiply (2y, z/2) + y)
        else return (multiply 2y, z/2))

Ans If z = 0 multiply (y, 0) = 0 irrespective of what y is
if z ≠ 0 multiply (y, z) which depend on whether z is even or odd.
If z+1 is odd then product will be
multiply ((2y, z/2) + y)
If z+1 is even, product will be multiply (2y, z/2). Hence proved.

15. Initialization:

let $y=1$, $z=2$

$x=0$ $\quad 2 \bmod 2 = 0 \neq 1$

$y=2$, $z=1$ $\quad x = x+y = 0+2 = 2$

and $1 \times 2 = 2$

Hence this initialization is correct in 1st iteration of the loop.

Maintenance:

let there be j natural numbers product of j natural nos. $= \prod_{i=1}^{j} i$ which is true for any number i.

This statement is also true for '$j+1$' natural numbers.

Termination:

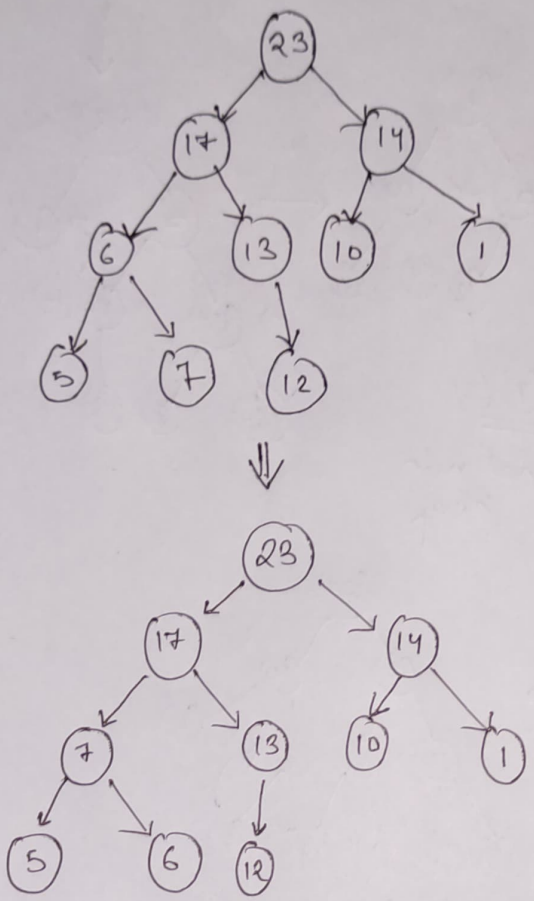
Let $y = n-1$, $z = n$ process continues untill $z=1$ irrespective of whether n is even or odd. Thus statement is true.

Hence this statement is proved to be correct.

SECTION-B

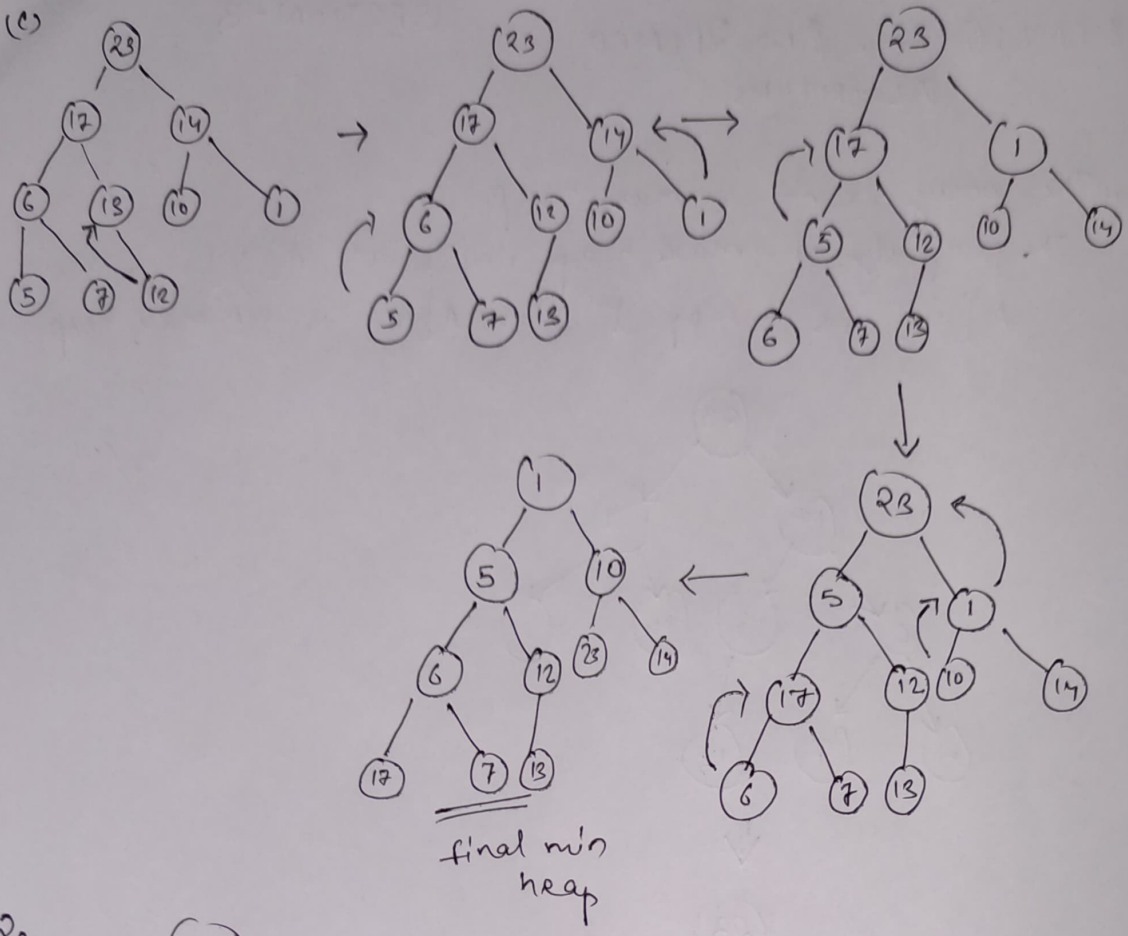
* HEAPS; GRAPHS AND RELATED ALGORITHMS

1. (a) The array is not a max heap.
The property of max heap fails at A[4]=6, so, A[4] is swapped with A[9] to get max heap.

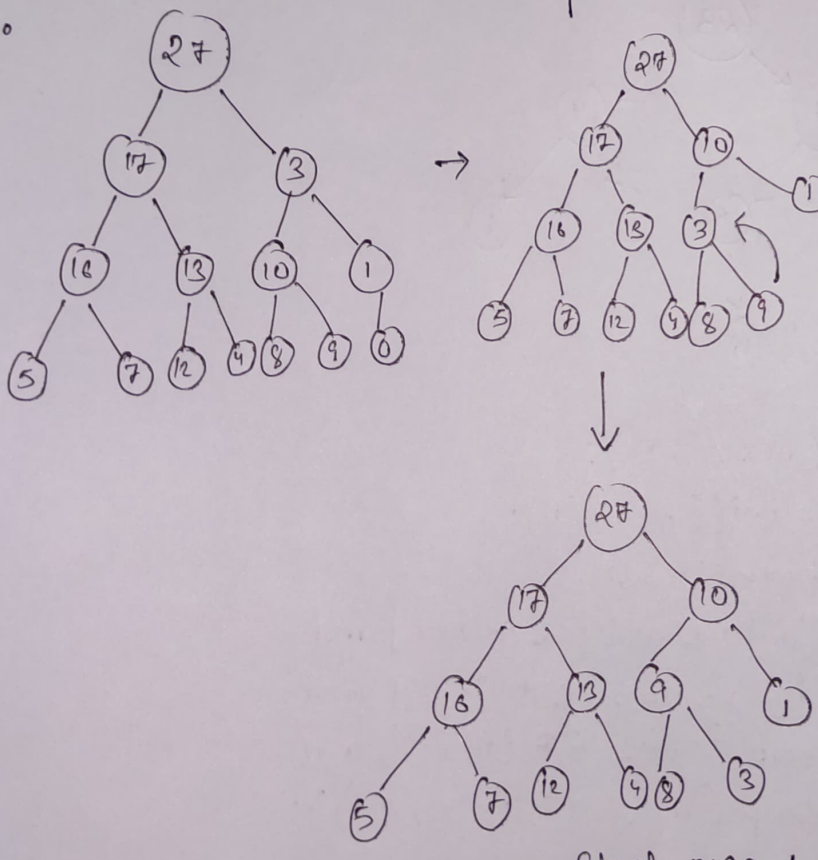


(b)
```
def isHeap(arr, i, n):
    if i >= int((n-2)/2):
        return True
    if (arr[i] >= arr[2*i+1] and
        arr[i] >= arr[2*i+2] and
        isHeap(arr, 2*i+1, n) and
        isHeap(arr, 2*i+2, n)):
        return True
    return False
# arr is the array
# i is the starting index
# n is the length of the array.
```

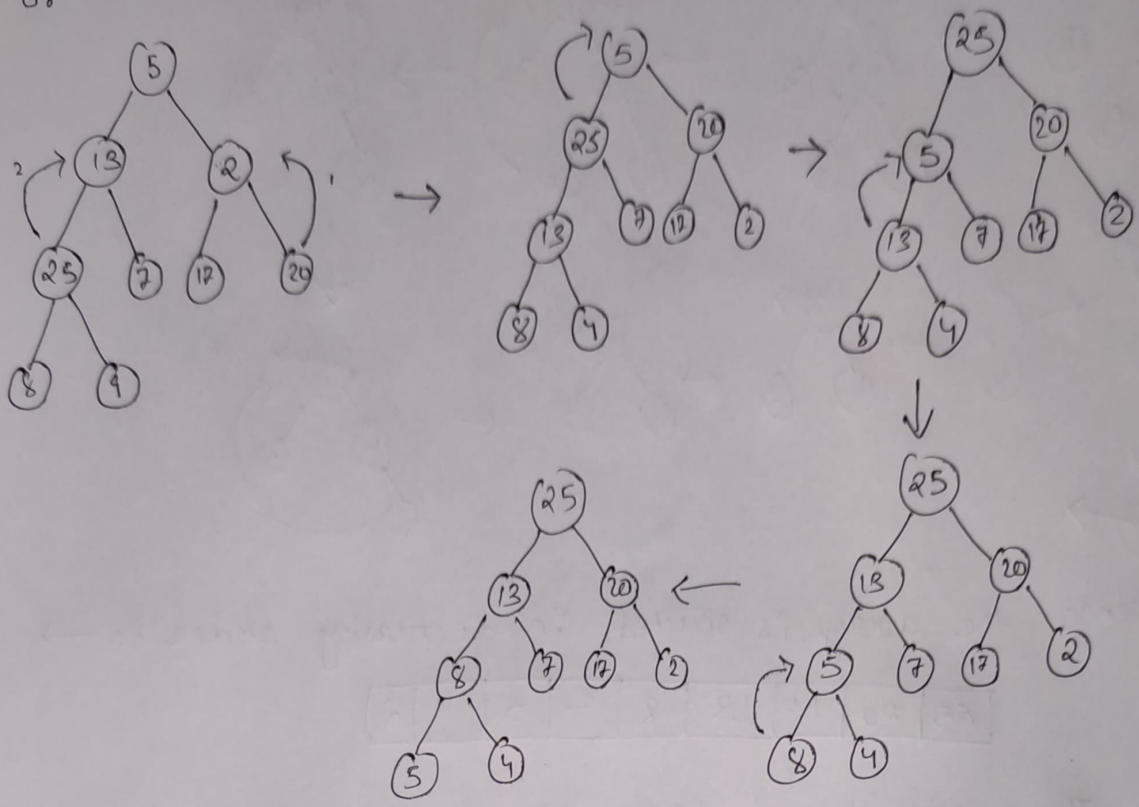
(c)



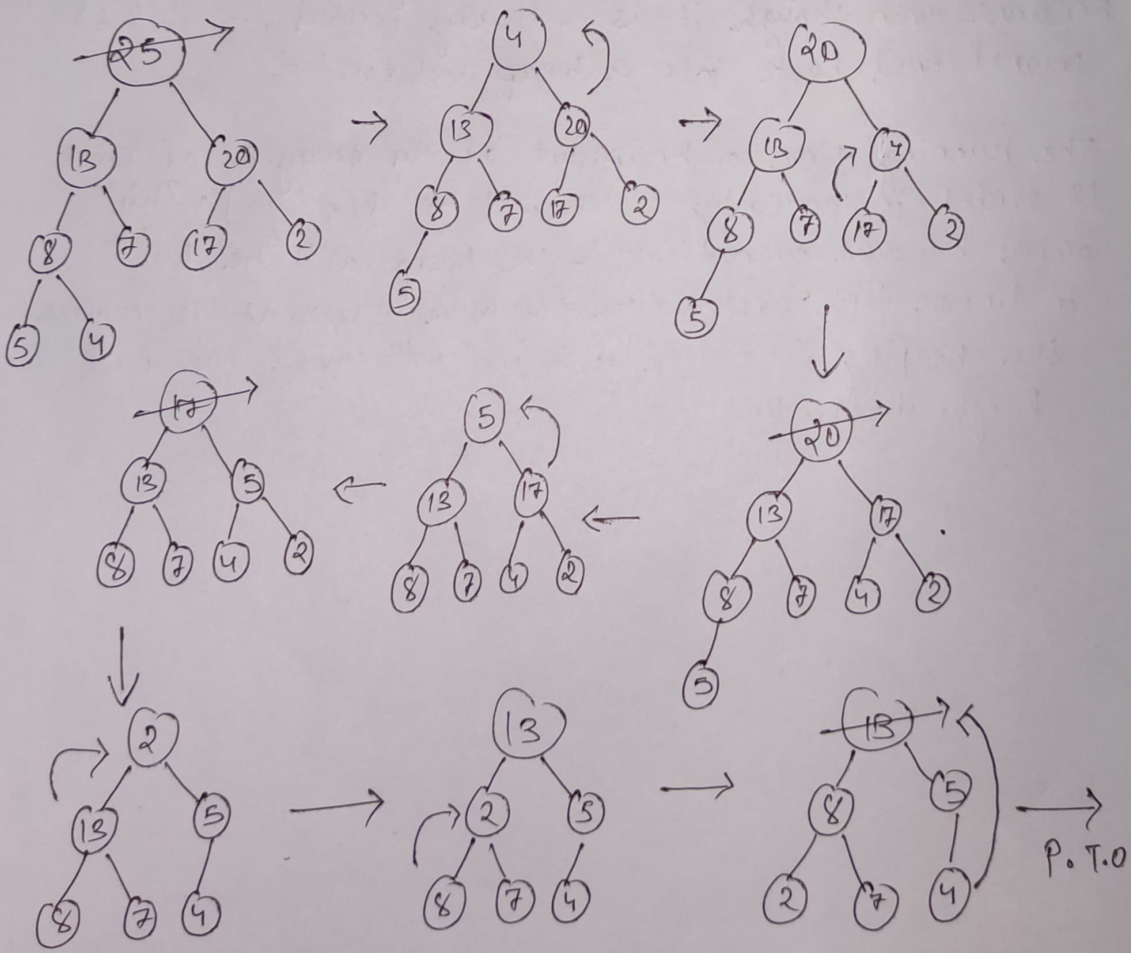
2.



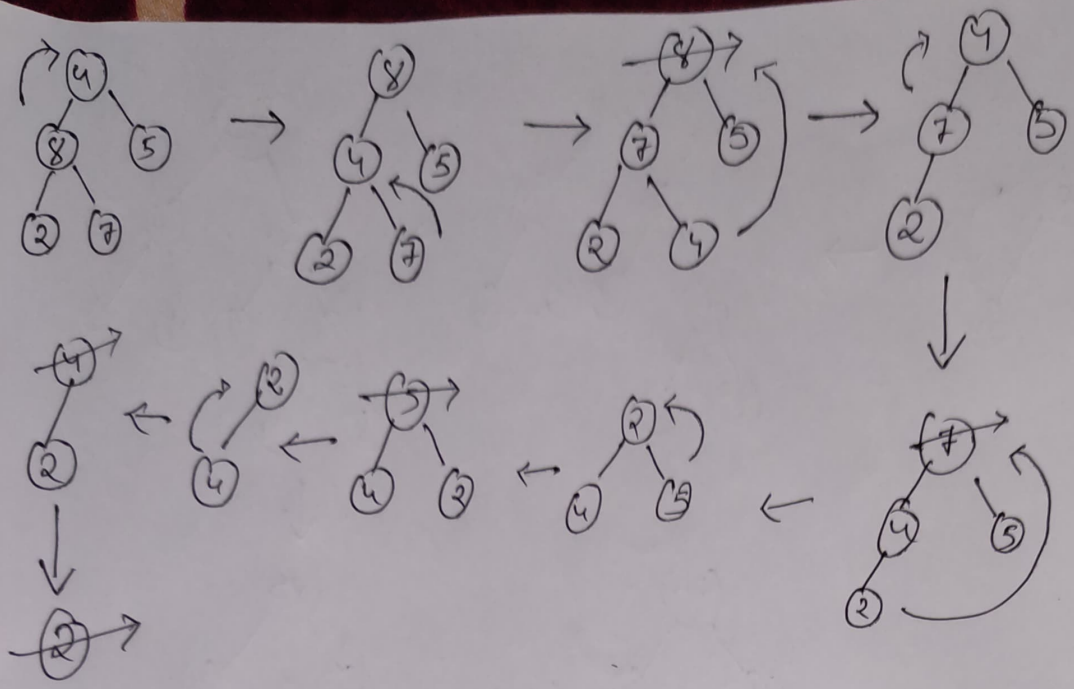
3.



Performing Heap Sort



P.T.O



Hence the array is sorted in decreasing order in →

25	20	17	13	8	7	5	4	2

4. The running time of heap sort of an array A of length n that is already sorted in increasing order is $\Omega(n \log n)$ because even though it is already sorted, it will be transformed back into a heap and sorted.

The running time of heapsort of an Array A of length is sorted in decreasing order will be $\Theta(n \log n)$. This occurs because even though the heap will be built in linear time every time the max element is removed the heapify is called and it will cover the full height of the tree.

BFSTraversal.java

```java
package assignment1;
    import java.io.*;
    import java.util.*;
    public class BFSTraversal
    {
        private int node;       /* total number number of nodes in the graph */
        private LinkedList<Integer> adj[];      /* adjacency list */
        private Queue<Integer> que;           /* maintaining a queue */
        BFSTraversal(int v)
        {
            node = v;
            adj = new LinkedList[node];
            for (int i=0; i<v; i++)
            {
                adj[i] = new LinkedList<>();
            }
            que = new LinkedList<Integer>();
        }
        void insertEdge(int v,int w)
        {
            adj[v].add(w);      /* adding an edge to the adjacency list (edges are bidirectional in this example) */
        }
        void BFS(int n)
        {
            boolean nodes[] = new boolean[node];       /* initialize boolean array for holding the data */
            int a = 0;
            nodes[n]=true;
            que.add(n);       /* root node is added to the top of the queue */
            while (que.size() != 0)
            {
                n = que.poll();        /* remove the top element of the queue */
                System.out.print(n+" ");    /* print the top element of the queue */
                for (int i = 0; i < adj[n].size(); i++)  /* iterate through the linked list and push all neighbors into queue */
                {
                    a = adj[n].get(i);
                    if (!nodes[a])      /* only insert nodes into queue if they have not been explored already */
                    {
                        nodes[a] = true;
                        que.add(a);
                    }}}}
        public static void main(String args[])
        {
            BFSTraversal graph = new BFSTraversal(6);
            graph.insertEdge(0, 1);
            graph.insertEdge(0, 3);
            graph.insertEdge(0, 4);
            graph.insertEdge(4, 5);
            graph.insertEdge(3, 5);
            graph.insertEdge(1, 2);
            graph.insertEdge(1, 0);
            graph.insertEdge(2, 1);
            graph.insertEdge(4, 1);
            graph.insertEdge(3, 1);
            graph.insertEdge(5, 4);
            graph.insertEdge(5, 3);
            System.out.println("Breadth First Traversal for the graph is:");
            graph.BFS(0);
        }}
```

Problems | Javadoc | Declaration | Search | Console

<terminated> BFSTraversal [Java Application] C:\Users\LENOVO\Dow

```
Breadth First Traversal for the graph is:
0 1 3 4 2 5
```

DepthFirstSearch.java

```java
package assignment1;
import java.io.*;
class DepthFirstSearch {
    private int V; // No. of vertices
    private LinkedList<Integer> adj[];
    // Constructor
    @SuppressWarnings("unchecked") DepthFirstSearch(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }
    // Function to add an edge into the graph
    void addEdge(int v, int w)
    {
        adj[v].add(w); // Add w to v's list.
    }

    // A function used by DFS
    void DFSUtil(int v, boolean visited[])
    {
        // Mark the current node as visited and print it
        visited[v] = true;
        System.out.print(v + " ");

        // Recur for all the vertices adjacent to this
        // vertex
        Iterator<Integer> i = adj[v].listIterator();
        while (i.hasNext()) {
            int n = i.next();
            if (!visited[n])
                DFSUtil(n, visited);
        }
    }
    void DFS(int v)
    {
        // Mark all the vertices as not visited(set as false by default in java)
        boolean visited[] = new boolean[V];
        // Call the recursive helper function to print DFS traversal
        DFSUtil(v, visited);
    }

    // Driver Code
    public static void main(String args[])
    {
        DepthFirstSearch g = new DepthFirstSearch(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);
        System.out.println(
            "Following is Depth First Traversal "
            + "(starting from vertex 2)");
        g.DFS(2);
    }
}
```

Problems | Javadoc | Declaration | Search | Console

<terminated> DepthFirstSearch [Java Application] C:\Users\LENOVO\

```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
```

graph.java

```java
class KruskalMSTalgorithm {
    // A class to represent a graph edge
    class Edge implements Comparable<Edge>
    {
        int src, dest, weight;

        // Comparator function used for
        // sorting edgesbased on their weight
        public int compareTo(Edge compareEdge)
        {
            return this.weight - compareEdge.weight;
        }
    };

    // A class to represent a subset for
    // union-find
    class subset
    {
        int parent, rank;
    };

    int V, E; // V-> no. of vertices & E->no.of edges
    Edge edge[]; // collection of all edges

    // Creates a graph with V vertices and E edges
    KruskalMSTalgorithm(int v, int e)
    {
        V = v;
        E = e;
        edge = new Edge[E];
        for (int i = 0; i < e; ++i)
            edge[i] = new Edge();
    }

    // A utility function to find set of an
    // element i (uses path compression technique)
    int find(subset subsets[], int i)
    {
        // find root and make root as parent of i
        // (path compression)
        if (subsets[i].parent != i)
            subsets[i].parent
                = find(subsets, subsets[i].parent);

        return subsets[i].parent;
    }

    // A function that does union of two sets
    // of x and y (uses union by rank)
    void Union(subset subsets[], int x, int y)
    {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        // Attach smaller rank tree under root
        // of high rank tree (Union by Rank)
        if (subsets[xroot].rank
            < subsets[yroot].rank)
            subsets[xroot].parent = yroot;
        else if (subsets[xroot].rank
                 > subsets[yroot].rank)
            subsets[yroot].parent = xroot;

        // If ranks are same, then make one as
        // root and increment its rank by one
        else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
    }

    // The main function to construct MST using Kruskal's
    // algorithm
    void KruskalMST()
    {
        // Tnis will store the resultant MST
        Edge result[] = new Edge[V];

        // An index variable, used for result[]
        int e = 0;

        // An index variable, used for sorted edges
        int i = 0;
        for (i = 0; i < V; ++i)
            result[i] = new Edge();

        // Step 1: Sort all the edges in non-decreasing
        // order of their weight. If we are not allowed to
        // change the given graph, we can create a copy of
        // array of edges
        Arrays.sort(edge);

        // Allocate memory for creating V subsets
        subset subsets[] = new subset[V];
        for (i = 0; i < V; ++i)
            subsets[i] = new subset();

        // Create V subsets with single elements
        for (int v = 0; v < V; ++v)
        {
            subsets[v].parent = v;
            subsets[v].rank = 0;
        }

        i = 0; // Index used to pick next edge

        // Number of edges to be taken is equal to V-1
        while (e < V - 1)
        {
            // Step 2: Pick the smallest edge. And increment
            // the index for next iteration
            Edge next_edge = edge[i++];

            int x = find(subsets, next_edge.src);
            int y = find(subsets, next_edge.dest);

            // If including this edge does't cause c
            // include it in result and increment th
            // of result for next edge
            if (x != y) {
                result[e++] = next_edge;
                Union(subsets, x, y);
            }
            // Else discard the next_edge
        }

        // print the contents of result[] to display
        // the built MST
        System.out.println("Following are the edges
                           + "the constructed MST");
        int minimumCost = 0;
        for (i = 0; i < e; ++i)
        {
            System.out.println(result[i].src + " -- "
                               + result[i].dest
                               + " == " + result[i].weight);
            minimumCost += result[i].weight;
        }
        System.out.println("Minimum Cost Spanning Tree "
                           + minimumCost);
    }

    // Driver Code
    public static void main(String[] args)
    {

        /* Let us create following weighted graph
                10
            0--------1
            | \  |
        6| 5\ |15
            |    \ |
            2--------3
                4    */
        int V = 4; // Number of vertices in graph
        int E = 5; // Number of edges in graph
        KruskalMSTalgorithm graph = new KruskalMSTalgorithm(V, E);

        // add edge 0-1
        graph.edge[0].src = 0;
        graph.edge[0].dest = 1;
        graph.edge[0].weight = 10;

        // add edge 0-2
        graph.edge[1].src = 0;
        graph.edge[1].dest = 2;
        graph.edge[1].weight = 6;

        // add edge 0-3
        graph.edge[2].src = 0;
        graph.edge[2].dest = 3;
        graph.edge[2].weight = 5;

        // add edge 1-3
        graph.edge[3].src = 1;
        graph.edge[3].dest = 3;
        graph.edge[3].weight = 15;

        // add edge 2-3
        graph.edge[4].src = 2;
        graph.edge[4].dest = 3;
        graph.edge[4].weight = 4;

        // Function call
        graph.KruskalMST();
    }
}
```

Problems Javadoc Declaration Search Console

```
<terminated> DepthFirstSearch [Java Application] C:\Users\LENOVO\Dc
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
```

Prism_MST.java

```java
package assignment1;

import java.util.*;

class Prism_MST {
    private static final int V = 5;

    int minKey(int key[], Boolean mstSet[])
    {
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++)
            if (mstSet[v] == false && key[v] < min) {
                min = key[v];
                min_index = v;
            }

        return min_index;
    }

    void printMST(int parent[], int graph[][])
    {
        System.out.println("Edge \tWeight");
        for (int i = 1; i < V; i++)
            System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
    }

    void primMST(int graph[][])
    {
        // Array to store constructed MST
        int parent[] = new int[V];

        // Key values used to pick minimum weight edge in cut
        int key[] = new int[V];

        // To represent set of vertices included in MST
        Boolean mstSet[] = new Boolean[V];

        // Initialize all keys as INFINITE
        for (int i = 0; i < V; i++) {
            key[i] = Integer.MAX_VALUE;
            mstSet[i] = false;
        }

        key[0] = 0;
        parent[0] = -1;

        for (int count = 0; count < V - 1; count++) {
            int u = minKey(key, mstSet);

            mstSet[u] = true;

            for (int v = 0; v < V; v++)

                if (graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
        }

        printMST(parent, graph);
    }

    public static void main(String[] args)
    {
        /* Let us create the following graph
        2 3
        (0)--(1)--(2)
        | / \ |
        6| 8/ \5 |7
        | /     \ |
        (3)-------(4)
            9          */
        Prism_MST t = new Prism_MST();
        int graph[][] = new int[][] { { 0, 2, 0, 6, 0 },
                { 2, 0, 3, 8, 5 },
                { 0, 3, 0, 0, 7 },
                { 6, 8, 0, 0, 9 },
                { 0, 5, 7, 9, 0 } };

        // Print the solution
        t.primMST(graph);
    }
}
```

Problems | Javadoc | Declaration | Search | Console

<terminated> Prism_MST [Java Application] C:\Users\LENOVO\Downloads\eclipse\plugins\org.e

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

DijkstraShortestPath.java

```java
package assignment1;
import java.lang.*;
import java.io.*;

class Dijistras_ALG {
    static final int V = 9;
    int minDistance(int dist[], Boolean sptSet[])
    {
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++)
            if (sptSet[v] == false && dist[v] <= min) {
                min = dist[v];
                min_index = v;
            }

        return min_index;
    }

    void printSolution(int dist[], int n)
    {
        System.out.println("Vertex Distance from Source");
        for (int i = 0; i < V; i++)
            System.out.println(i + " tt " + dist[i]);
    }

    void dijkstra(int graph[][], int src)
    {
        int dist[] = new int[V];

        Boolean sptSet[] = new Boolean[V];

        for (int i = 0; i < V; i++) {
            dist[i] = Integer.MAX_VALUE;
            sptSet[i] = false;
        }

        dist[src] = 0;

        for (int count = 0; count < V - 1; count++) {
            int u = minDistance(dist, sptSet);
            sptSet[u] = true;
            for (int v = 0; v < V; v++)

                if (!sptSet[v] && graph[u][v] != 0 &&
                        dist[u] != Integer.MAX_VALUE && dist[u] + graph[u][v] < dist[v])
                    dist[v] = dist[u] + graph[u][v];
        }

        printSolution(dist, V);
    }

    public static void main(String[] args)
    {
        /* Let us create the example graph discussed above */
        int graph[][] = new int[][] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
        Dijistras_ALG t = new Dijistras_ALG();
        t.dijkstra(graph, 0);
    }
}
```

Problems | Javadoc | Declaration | Search | Console

<terminated> Prism_MST [Java Application] C:\Users\LENOVO\Down

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

CheckBiparitnessByBFS.java

```java
package assignment1;
    import java.util.*;
    import java.lang.*;
    import java.io.*;

    class Bipartite
    {
        final static int V = 4; // No. of Vertices

        // This function returns true if
        // graph G[V][V] is Bipartite, else false
        boolean isBipartite(int G[][],int src)
        {
            // Create a color array to store
            // colors assigned to all vertices.
            // Vertex number is used as index
            // in this array. The value '-1'
            // of colorArr[i] is used to indicate
            // that no color is assigned
            // to vertex 'i'. The value 1 is
            // used to indicate first color
            // is assigned and value 0 indicates
            // second color is assigned.
            int colorArr[] = new int[V];
            for (int i=0; i<V; ++i)
                colorArr[i] = -1;

            // Assign first color to source
            colorArr[src] = 1;

            // Create a queue (FIFO) of vertex numbers
            // and enqueue source vertex for BFS traversal
            LinkedList<Integer>q = new LinkedList<Integer>();
            q.add(src);

            // Run while there are vertices in queue (Similar to BFS)
            while (q.size() != 0)
            {
                // Dequeue a vertex from queue
                int u = q.poll();

                // Return false if there is a self-loop
                if (G[u][u] == 1)
```

CheckBiparitnessByBFS.java

```java
                    return false;

                // Find all non-colored adjacent vertices
                for (int v=0; v<V; ++v)
                {
                    if (G[u][v]==1 && colorArr[v]==-1)
                    {
                        // Assign alternate color to this adjacent v of u
                        colorArr[v] = 1-colorArr[u];
                        q.add(v);
                    }
                    // An edge from u to v exists and destination
                    //  v is colored with same color as u
                    else if (G[u][v]==1 && colorArr[v]==colorArr[u])
                        return false;
                }
            }
            // If we reach here, then all adjacent vertices can
            // be colored with alternate color
            return true;
        }
        // Driver program to test above function
        public static void main (String[] args)
        {
            int G[][] = {{0, 1, 0, 1},
                {1, 0, 1, 0},
                {0, 1, 0, 1},
                {1, 0, 1, 0}
            };
            Bipartite b = new Bipartite();
            if (b.isBipartite(G, 0))
            System.out.println("Yes");
            else
            System.out.println("No");
        }
```

Problems | Javadoc | Declaration | Search | Console

<terminated> Prism_MST [Java Application] C:\Users\LENOVO\Downloads\eclipse\plugins\org.eclipse.justj.

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

TopologicalOrdering.java

```java
package assignment1;
import java.io.*;
import java.util.*;

// This class represents a directed graph
// using adjacency list representation
class Graph {
    // No. of vertices
    private int V;

    // Adjacency List as ArrayList of ArrayList's
    private ArrayList<ArrayList<Integer> > adj;

    // Constructor
    Graph(int v)
    {
        V = v;
        adj = new ArrayList<ArrayList<Integer> >(v);
        for (int i = 0; i < v; ++i)
            adj.add(new ArrayList<Integer>());
    }

    // Function to add an edge into the graph
    void addEdge(int v, int w) { adj.get(v).add(w); }

    // A recursive function used by topologicalSort
    void topologicalSortUtil(int v, boolean visited[],
                             Stack<Integer> stack)
    {
        // Mark the current node as visited.
        visited[v] = true;
        Integer i;
        Iterator<Integer> it = adj.get(v).iterator();
        while (it.hasNext()) {
            i = it.next();
            if (!visited[i])
                topologicalSortUtil(i, visited, stack);
        }
        stack.push(new Integer(v));
    }

    void topologicalSort()
    {
```

TopologicalOrdering.java

```java
        stack.push(new Integer(v));
    }

    void topologicalSort()
    {
        Stack<Integer> stack = new Stack<Integer>();

        // Mark all the vertices as not visited
        boolean visited[] = new boolean[V];
        for (int i = 0; i < V; i++)
            visited[i] = false;

        // Call the recursive helper
        // function to store
        // Topological Sort starting
        // from all vertices one by one
        for (int i = 0; i < V; i++)
            if (visited[i] == false)
                topologicalSortUtil(i, visited, stack);

        // Print contents of stack
        while (stack.empty() == false)
            System.out.print(stack.pop() + " ");
    }

    // Driver code
    public static void main(String args[])
    {
        // Create a graph given in the above diagram
        Graph g = new Graph(6);
        g.addEdge(5, 2);
        g.addEdge(5, 0);
        g.addEdge(4, 0);
        g.addEdge(4, 1);
        g.addEdge(2, 3);
        g.addEdge(3, 1);

        System.out.println("Following is a Topological "
                           + "sort of the given graph");
        // Function Call
        g.topologicalSort();
    }
}
```

Problems | Javadoc | Declaration | Search | Co

<terminated> Prism_MST [Java Application] C:\Users\LENO

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

ConnectedComponents.java

```java
package assignment1;
    // Java program for the above approach
    import java.util.*;
    class ConnectedComponents{

    // Stores the parent of each vertex
    static int []parent = new int[1000000];

    // Function to find the topmost
    // parent of vertex a
    static int root(int a)
    {

        // If current vertex is
        // the topmost vertex
        if (a == parent[a])
        {
            return a;
        }

        // Otherwise, set topmost vertex of
        // its parent as its topmost vertex
        return parent[a] = root(parent[a]);
    }

    // Function to connect the component
    // having vertex a with the component
    // having vertex b
    static void connect(int a, int b)
    {

        // Connect edges
        a = root(a);
        b = root(b);

        if (a != b) {
            parent[b] = a;
        }
    }

    // Function to find unique top most parents
    static void connectedComponents(int n)
    {
```

ConnectedComponents.java

```java
    // Function to find unique top most parents
    static void connectedComponents(int n)
    {
        HashSet<Integer> s = new HashSet<Integer>();

        // Traverse all vertices
        for (int i = 0; i < n; i++)
        {
        s.add(parent[i]);
        }
        System.out.println(s.size());
    }
    static void printAnswer(int N,int [][] edges)
    {
        for (int i = 0; i <= N; i++)
        {
            parent[i] = i;
        }
        for (int i = 0; i < edges.length; i++)
        {
            connect(edges[i][0], edges[i][1]);
        }

        // Print answer
        connectedComponents(N);
    }

    // Driver Code
    public static void main(String[] args)
    {

        // Given N
        int N = 8;

        // Given edges
    int [][]edges = {{ 1, 0 }, { 0, 2 },
                     { 5, 3 }, { 3, 4 },
                     { 6, 7 }};

        // Function call
        printAnswer(N, edges);
    }
    }
```

Problems | Javadoc | Declaration | Search | Console

<terminated> ConnectedComponents [Java Application] C:\Users\LEN

```
3
```

* GREEDY METHOD

SECTION-C

1. No. of objects = n
   max capacity of knapsack = m
   weight of object 'i' = w_i
   fraction value : x_i , where $0 <= x_i <= 1$
   Profit earned after placing $x_i = p_i x_i$
   Goal : To maximize profit subject to capacity constraint.

(a) Maximize

$$\sum_{i=1}^{n} p_i x_i$$

Subject to

$$\sum_{i=1}^{n} w_i x_i \leq m$$

let

$n = 3, m = 20$

$(P_1\ P_2\ P_3) = (25, 4, 15)$

$(w_1\ w_2\ w_3) = (17, 15, 10)$

if

	x_1	x_2	x_3	$\sum_{i=1}^{n} w_i x_i$	$\sum_{i=1}^{n} p_i x_i$
→	1/2	1/3	1/4	16.5	24.85
→	1	2/15	0	20	28.2
→	0	1	1/2	20	31.5

Strategy-1

maximize objective function

Put the object with the highest profit in the knapsack. Then use a fraction of the last to fill the sack.

$$\sum_{i=1}^{n} p_i x_i = 25 \times 1 + 24 \times 2/15 + 0 = 28.5$$

$$\sum_{i=1}^{n} w_i x_i = 18 \times 1 + 15 \times 2/15 + 0 = 20$$

strategy doesn't yield an optimal, so as the capacity used quickly exhausted which constrained the profit attained.

Strategy-2

maximize capacity

Choose objects according to least weight.

The approach is that the more no. of objects in knapsack, more is the profit.

$$\sum_{i=1}^{3} p_i x_i = 0 + 24 \times 2/3 + 15 \times 1 = 31$$

$$\sum_{i=1}^{3} w_i x_i = 0 + 15 \times 2/3 + 10 \times 1 = 20$$

∴ solution is not optimal as rate of % change in profit not high enough.

Strategy-3

maximizing profit per weight/unit (p_i / w_i)

$$\sum_{i=1}^{2} p_i x_i = 0 + 24 \times 1 + 15 \times 1/2 = 31.5$$

$$\sum_{i=1}^{2} w_i x_i = 0 + 15 \times 1 + 10 \times 1/2 = 20$$

∴ solution is optimal.

(b) Fractional Knapsack Greedy Algorithm

```
Algorithm Knapsack (P, w, x : arrayvals ; m, n : int) {
  // Input : the objects are in increasing order
  // so that P[i]/w[i] > P[i+1]/w[i+1];
  // Output : optimal solution vector x
  for (i=0; i<=n; i++)
      x[i] = 0;
  capacity = m;
  for ( i=1 ; i<=n ; i++) {
    if w[i] > capacity then exit () // cause → overflow.
    else
        x[i] = 1 ; // greedy choice.
        capacity = capacity - w[i] ;
  }
  if i <= n then x[i] = capacity / w[i]; // choose a fraction.
}
```

(c) $n = 7$

$m = 15$

$(P_1\ P_2\ P_3\ P_4\ P_5\ P_6\ P_7) = (10, 5, 15, 7, 6, 18, 3)$

$(w_1\ w_2\ w_3\ w_4\ w_5\ w_6\ w_7) = (2, 3, 3, 5, 7, 14, 1)$

$$\frac{p[i]}{w[i]} = [5, 1.3, 3, 1, 6, 4.5, 3)$$

$$x_i = (1\ \underline{2/3}\ 1\ 0\ 1\ 1\ 1)$$

$$\sum x_i w_i = 1\times 2 + \frac{2}{3}\times 3 + 1\times 5 + 0\times 7 + 1\times 1 + 1\times 4 + 1\times 1$$

$$= 15$$

$$\sum x_i p_i = 1\times 10 + 2/3 \times 5 + 1\times 15 + 0\times 7 + 1\times 6 + 1\times 18 + 1\times 3$$

$$= 54.6$$

2. Algorithm Cover (V, E)

```
{
  U := ϕ
  repeat
  {
    q be a vertex from V of maximum degree;
    Add q to U; Eliminate q from V;
    E := E - {(x,y) such that x = q or y = q};
  } untill (E = ϕ); // U is the node cover
}
```

Ans

The algorithm considers an empty set U and the algorithm operates on the graph untill edge set is ϕ.

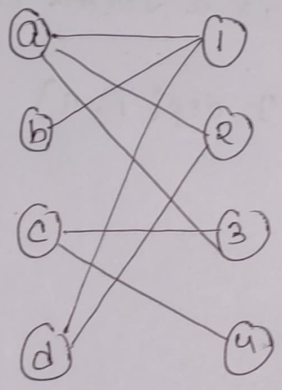
The algorithm takes the vertex with highest degree and adds into set U and eliminates the same from V.

By this greedy technique at each step the algorithm is trying to remove as much as edges as possible from set E.

At the end no edges are left and set U is ensured to be the edge cover.

Eg.

G:



Initially

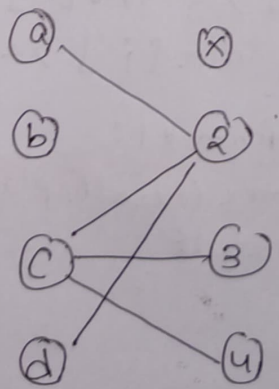
U = ϕ

Step 1

U = {ϕ}

Delete v({1}), E({a,1}, {b,1}, {c,1})

G:



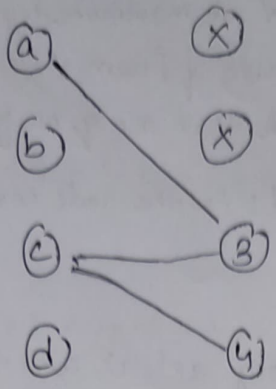
U = {1}

Step-2

U = {1, 2}

delete [v{2}, B({2,a}, {2,c}, {2,d})]

G:



Step-3

U = {1, 2}

delete (c)

Step-4

U = {1, 2, c, 3}

so, u = {1, 2, c, 3} is a vertex cover.

3.

(a) function find_platform (arr[], dep[], n)

```
platform = 1
result = 1
i = 1
j = 0
for (i = 1 to n) do
    platform = 1
```

(b) checking for overlap

```
for (j = i+1 to n)
    if (arr[i] >= arr[i] && arr[i] <= dep[j]
        || arr[j] >= arr[i] && arr[j] >= dep[i]
        platform + 1;
result = max (result, platform);
return result.
```

(c) Time complexity = $O(n^2)$.