Expt. No. ..... Page No. ..... Date : .....

1.

Aim: Write a program to implement quicksort and compute the time required to sort randomly generated element for n=1000, n=2000, n=3000

Description:

Quicksort is a divide and conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The time complexity is O(nlogn)

Algorithm:-

```
Algorithm Quicksort (P,q){
    if (P<q) then
        j:= Patition (a,P,q+1);
    Quicksort(P,j-1);
    Quicksort(j+1,q);
    }
}
```

Roll No. : ..... Lecturer's Signature : .....

```
Algorithm partition(a,m,p){
    v:=a[m]; i:=m; j:=p;
    repeat{
        repeat
            i:=i+1;
        until(a[i]≥v);
        repeat
            j:=j-1;
        until(a[j]≤v);
        if(i<j) then Interchange(a,i,j);
    }until(i≥j);
    a[m]:=a[j]; a[j]:=v; return j;
}
Algorithm change(a,i,j){
    p:=a[i];
    a[i]=a[j]; a[j]:=p;
}
```

Expt. No. ..... Page No. ..... Date : .....

Quicksort :-

```
public class Quicksort {
  public int partition(int A[], int low, int high) {
    int pivot = A[low];
    int i = low + 1;
    int j = high;
    do {
      while (i <= j && A[i] <= pivot)
        i = i + 1;
      while (i <= j && A[j] > pivot)
        j = j - 1;
      if (i < j)
        swap(A, i, j);
    } while (i < j);
    swap(A, low, j);
    return j;
  }
public void quicksort(int A[], int low, int high) {
  if (low < high) {
    int pi = partition(A, low, high);
    quicksort(A, low, pi - 1);
    quicksort(A, pi + 1, high);
  }
}
```

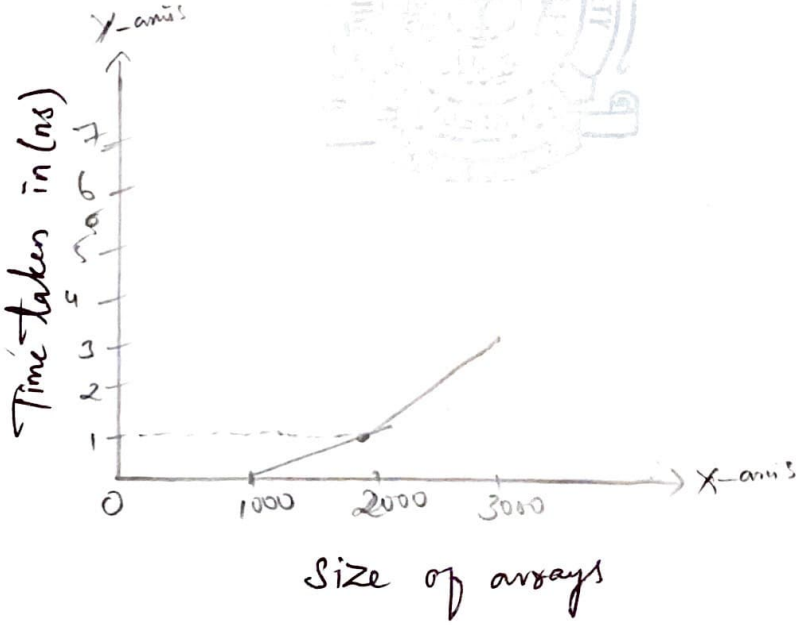
Roll No. : ..... Lecturer's Signature : .....

```
public void swap(int A[], int i, int j){
    int temp=A[i];
    A[i] =A[j];
    A[j]= temp;
}
public void display(int A[], int n){
    for(int i=0;i<n;i++)
        System.out.print(A[i]+" ");
    System.out.println();
}
public static void main(String args[]){
    Quicksort s = new Quicksort();
    int n=1000;
    int A[] = new int[n];
    for(int i=0; i<n;i++)
        A[i] =(int) (Math.random()*1000);
    System.out.println("Original Array:");
    s.display(A,n);
    long start=System.currentTimeMillis();
    s.quicksort(A,0, A.length-1);
    long end = System.currentTimeMillis();
```

```
        System.out.print("Sorted Array: ");
        s.display(A, n);
        System.out.println("Elapse Time :"+(end-start)+"ms");
    }
}
```

OUTPUT :-

Size of Array(n)	Time in nano seconds
1000	0
2000	1
3000	3



Expt. No. ..... Page No. ..... Date : .....

2.

Aim: Write a program to implement Mergesort and compute the time required to sort randomly generated element for $n=1000$, $n=2000$, $n=3000$

Description:-

Mergesort is a sorting technique based on divide and conquer technique. with worst-case time complexity being $O(n\log n)$, it is one of the most respected algorithms. Mergesort first divides the array into equal halves and then combines them in a sorted manner. The time complexity of Mergesort is $O(n\log n)$.

Algorithm:

```
Algorithm Mergesort (low, high) {
    if (low < high) then {
        mid := ⌊(low+high)/2⌋;
        Mergesort (low, mid);
        Mergesort (mid+1, high);
        Mergesort (low, mid, high);
    } }
```

Roll No. : ..... Lecturer's Signature : .....

Expt. No. ..... Page No. ..... Date : .....

Algorithm Merge (low, mid, high) {

h := low; i := low; j := mid+1;

while ((h ≤ mid) and (j ≤ high)) do {

if (a[h] ≤ a[j]) then {

b[i] := a[h]; h := h+1;

} else {

b[i] := a[j]; j := j+1;

}

i := i+1;

}

if (h > mid) then

For k := j to high do {

b[i] := a[k]; i := i+1;

}

else

for k := h to mid do {

b[i] := a[k]; i := i+1;

}

for k := low to high do a[k] := b[k];

}

Roll No. : ..... Lecturer's Signature : .....

Expt. No. ..... Page No. ..... Date : .....

Mergesort:

```
Public class MergeSort{
  public void mergesort(int A[], int left, int right){
    if(left<right){
      int mid=(left+right)/2;
      mergesort(A,left,mid);
      mergesort(A,mid+1,right);
      Merge(A,left,mid,right);
    }
  }
public void merge(int A[],int left, int mid, int right){
    int i=left;
    int j=mid+1;
    int k=left;
    int B[]=new int[right+1];
    while(i<=mid && j<=right){
      if(A[i]<A[j]){
          B[k]=A[i];
          i=i+1;
      }
      else{
        B[k]=A[j];
        j=j+1;
      }
```

Roll No. : ..... Lecturer's Signature : .....

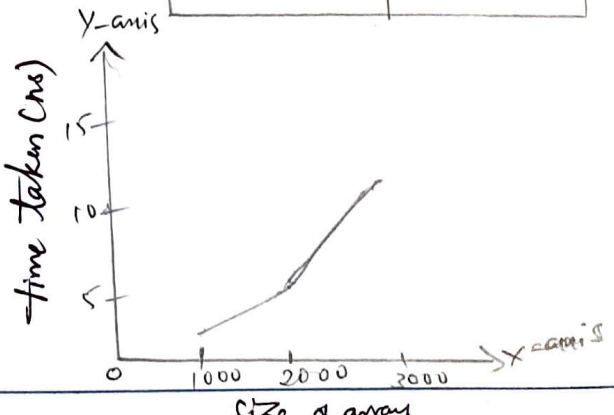
```
    k=k+1;
  }
while(i<=mid){
    B[k]=A[i];
    i=i+1;
    k=k+1;
  }
while(j<=right){
    B[k]=A[i];
    i=i+1;
    k=k+1;
  }
for(int x=left;x<=right+1;x++)
    A[x]=B[x];
  }
public void display(int A[], int n){
    for(int i=0;i<n;i++)
        System.out.print(A[i]+" ");
    System.out.println();
  }
public static void main(String args[]){
  Mergesort s=new Mergesort();
  int n=1000;
  int A[]=new int[n];
```


Roll No. : ..... Lecturer's Signature : .....

```
for(int i=0; i<n; i++)
    A[i] =(int) (Math.random( )*1000);
System.out.print("Original Array:");
S.display(A, n);
long start = System.currentTimeMillis();
S.mergesort(A, 0, A.length-1);
long end = System.currentTimeMillis( );
System.out.print("Sorted Array:");
S.display(A,n);
System.out.println("Elapse Time:"+(end -start)+"ms");
}
```

Output:-

Size of Array	time taken(ms)
1000	2
2000	5
3000	12



Expt. No. ..... Page No. ..... Date : .....

3.

Aim:-

write a program to implement 0/1 knapsack problem.

Description:-

In 0/1 knapsack, items cannot be broken which means the thief should take the item as a whole or should leave it. This is the reason behind calling it as 0/1 knapsack. The time complexity of knapsack is O(nw).

Algorithm:-

Algorithm DKP(P, w, n, m){

$S^0 := \{(0,0)\}$;

for := 1 to n-1 do{

$S_1^{i-1} := \{(P,w) \mid (P-P_i, w-w_i) \in S^{i-1}$ and

$w \leq m\}$;

$S^i :=$ Merge purge (S^{i-1}, S_1^{i-1});

}

Roll No. : ..... Lecturer's Signature : .....

$(px, wx) :=$ last pair in S^{n-1};

$(py, wy) := (p' + p_n, w' + w_n)$ where w' is the largest w in any pair in S^{n-1} such that $w + w_n \leq m$;

if $(px \geq py)$ then $x_n := 0$;

else $x_n := 1$;

TraceBack for $(x_{n-1}, \ldots x_1)$;

}

Algorithm Dknap (p, w, x, n, m) {

b[0] := 1; pair[1].p := pair[1].w := 0.0;

t := 1; h := 1;

b[1] := next := 2

for i := 1 to n-1 do {

k := t;

u := Largest (pair, w, t, h, i, m);

for j := t to u do {

pp := pair[j].p + p[i];

ww := pair[j].w + w[i];

$(px, wx) := $ last pair in S^{n-1};

$(py, wy) := (p' + p_n, w' + w_n)$ where w' is the largest w in any pair in S^{n-1} such that $w + w_n \leq m$;

if $(px > py)$ then $x_n := 0$;

else $x_n := 1$;

TraceBack for $(x_{n-1}, \ldots x_1)$;

}

Algorithm Dknap (p, w, a, n, m) {

b[0] := 1; pair[1].p := pair[1].w := 0.0;

t := 1; h := 1;

b[1] := next := 2;

for i := 1 to n-1 do {

k := t;

u := Largest (pair, w, t, h, i, m);

for j := t to u do {

pp := pair[j].p + p[i];

ww := pair[j].w + w[i];

```
while ((k≤h) and (pair[k].w ≤ ww)) do {
    pair[next].p := pair[k].p;
    pair[next].w := pair[k].w;
    next := next+1; k := k+1;
}
if((k≤h) and (pair[k].w = ww)) then {
    if pp < pair[k].p then pp := pair[k].p;
    k := k+1;
}
if pp > pair[next-1].p then {
    pair[next].p := pp; pair[next].w = ww;
    next := next+1;
}
while ((k≤h) and (pair[k].p ≤ pair[next].p)
    do k := k+1;
}
while (k≤h) do {
    pair[next].p := pair[k].p; pair[next].w := pair[k].w
    next := next+1; k := k+1;
}
t := n+1; h := next-1; b[i+1] := next;
} Traceback(p, w, pair, x, m, n);
}
```


Roll No. : ..... Lecturer's Signature : .....

Knapsack : code :

```
public class Knapsack{
    public void Knapsack(int p[], int w[], int c, int f[][])
{
  int n=p.length-1;
  int yMax = Math.min(w[n]-1, c);
  for(int y=0; y<=yMax; y++)
   f[n][y]=0;
  for(int y=w[n]; y<=c; y++)
   f[n][y] =p[n];
  for(int i=n-1; i>1; i--)
  {
    yMax =Math.min(w[i]-1, c);
    for(int y=0; y<=yMax; y++)
      f[i][y] =f[i+1][y];
    for(int y=w[i]; y<=c; y++)
      f[i][y] = Math.max(f[i+1][y], f[i+1][y-w[i]]+p[i]);
  }
  f[1][c] =f[2][c];
  if(c>=w[1])
    f[1][c] = Math.max(f[1][c], f[2][c-w[1]]+p[1]);
}
```


Roll No. : ..... Lecturer's Signature : .....

Knapsack : code :

```
public class Knapsack {
    public void Knapsack(int p[], int w[], int c, int f[][])
{
    int n=p.length-1;
    int yMax = Math.min(w[n]-1, c);
    for (int y=0; y<=yMax; y++)
    f[n][y]=0;
    for (int y=w[n]; y<=c; y++)
     f[n][y] =p[n];
    for (int i=n-1 ; i>1; i--)
    {
      yMax =Math. min (w[i]-1, c);
      for (int y=0; y<=yMax; y++)
        f[i][y] =f[i+1][y];
      for(int y=w[i]; y<=c;y++)
        f[i][y] = Math.max(f[i+1][y], f[i+1][y-w[i]]+p[i]);
    }
    f[1][c] =f[2][c];
    if (c>=w[1])
      f[1][c] = Math. max (f[1][c], f[2][c-w[1]] +p[1]);
    }
```


Roll No. : ..... Lecturer's Signature : .....

Expt. No. ..... Page No. ..... Date : .....

```
public void traceback (int[][] f, int[] w, int C, int [] x)
{ int n=w.length-1;
  for (int i=1; i<n; i++)
    if (f[i][C]==f[i+1][C])
      x[i]=0;
    else
    { x[i]=1;
      C-=w[i];
    }
  x[n]=(f[n][C]>0)?1:0;
}
public static void main (String [] args)
{ Knapsack k=new Knapsack();
  int n=3;
  int C=6;
  int []p={0,1,2,5};
  int []w={0,2,3,4};
  int[] x=new int[n+1];
  int[][]f=new int[n+1][C+1];
  k.knapsack(p,w,C,f);
  System.out.println("Optimal Profit:"+f[1][C]);
  k.traceback(f,w,C,x);
```


Roll No. : ..... Lecturer's Signature : .....

Roll No. : ..... Lecturer's Signature : .....

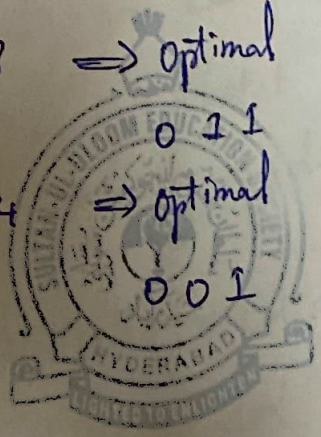
```
for(int i=1; i<=n; i++)
    System.out.println(x[i]+" ");
System.out.println();
}
}
```

OUTPUT :-

If input c=6 ⇒ optimal profit : 6
1 0 1

If input c=8 ⇒ optimal profit : 7
0 1 1

If input c=4 ⇒ optimal profit : 5
0 0 1



Roll No. : ..... Lecturer's Signature : .....

Expt. No. ..... Page No. ..... Date : .....

4.

Aim:-

Write a program to implement Dijkstra's Algorithm.

Description:

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph which may represent, for example, road networks it was conceived by scientist Edeger W. Dijstra in 1956. The time complexity of dijkstra's is $O(V^2)$

Algorithm:

```
Algorithm shortestpaths (v, cost, dist, n) {
    for i:=1 to n do{
        s[i]:=false; dist[i]:=cost[v, i];
    }
    s[v]:=true; dist[v]:=0.0;
    For n:=2 to n do{
     s[u]:=true
    for(each w adjacent to u with s[w]=false)do{
      if(dist[w]>dist[u]+cost[u,w])) then
      dist[w]:=dist[u] + cost[u,w];
}}
```

Roll No. : ..... Lecturer's Signature : .....

Roll No. : ..... Lecturer's Signature : .....

Dijkstra

```
Public class Dijkstra{
    int dist[];
    int S[];
    int n;
    int cost[][];
public Dijkstra()
{
  n=6;
  cost=new int[][]{
        {0,50,45,10,Integer.MAX_VALUE,Integer.MAX_VALUE},
      {Integer.MAX_VALUE,0,10,15,Integer.MAX_VALUE,
                  Integer.MAX_VALUE},
      {20,Integer.MAX_VALUE,Integer.MAX_VALUE,0,15,
                        Integer.MAX_VALUE},
      {Integer.MAX_VALUE,20,35,Integer.MAX_VALUE,0,
                        Integer.MAX_VALUE},
       {Integer.MAX_VALUE,Integer.MAX_VALUE,Integer.
                        MAX_VALUE,Integer.
                        MAX_VALUE,3,0},
        };
  dist=new int[n];
  S=new int[n];
}
```

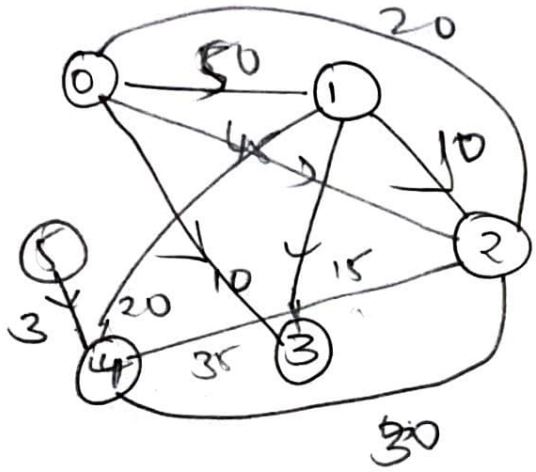

Roll No. : ..... Lecturer's Signature : .....

```
public void shortestPaths(int v)
{ for(int i=0; i<n; i++)
  {
    s[i]=0;
    dist[i]=cost[v][i];
  }
  s[v]=1;
  dist[v]=0;
 for(int i=1; i<n-1; i++)
 {
   int u=minDist();
   s[u]=1;
   for(int w=0; w<n; w++)
   {
     if(cost[u][w]!=0 && cost[u][w]!=Integer.MAX_VALUE
                                   && s[w]==0)
     { if(dist[w]>dist[u]+cost[u][w])
         dist[w]=dist[u]+cost[u][w];
     }
   }
 }
}
public int minDist()
{
 int min=Integer.MAX_VALUE;
 int u=-1;
```

```
for(int i=0; i<n; i++)
    if(dist[i]<min && S[i]!=1){
        min=dist[i];
        u=i;
    }
    return u;
}
public void displayDist()
{
for(int i=0; i<n; i++)
    System.out.println(dist[i]+"\t");
System.out.println();
}
public static void main(String args[])
{ Dijkstra d=new Dijkstra();
  d.shortestPaths(0);
  d.displayDist();
}
}
```

Expt. No. ..... Page No. ..... Date : .....

OUTPUT :-



0 45 45 10 25 2147483647

3

35 0 10 15 30 2147483647

85 50 0 65 30 2147483647

Expt. No. ..... Page No. ..... Date : .....

Expt. No. ..... Page No. ..... Date : .....

5

Aim:-

Write a program to implement prims Algorithm to find minimum cost spanning tree.

Description:

In computer science, Prims algorithm (also known as Jarnik's algorithm) is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph.

This means it finds a subset of the edges that forms a tree that includes every vertex. where the total weight of all the edges in the tree is minimized.

The time complexity of prims algorithm is $O(E \log V)$

Expt. No. ..... Page No. ..... Date : .....

Algorithm:-

```
Algorithm Prim (E, cost, n, t){
    Let (k,l) be an edge of the minimum cost
in E
    mincost := cost[k,l];
    t[1,1] := k; t[1,2] := l;
    For i := 1 to n do{
        IF (cost[i,l] < cost[i,k]) then near[i] := l;
        else near[i] := k;
        near[k] := near[l] := 0;
    For i := 2 to n-1 do{
        cost[j, near[j]] is minimum;
        t[i,1] := j; t[i,2] := near[j];
        mincost := mincost + cost[j, near[j]];
        near[j] := 0;
    For k := 1 to n do
        IF[(near[k] ≠ 0) and (cost[k, near[k]] > cost[k,j])
            then near[k] := j;
    }
    return mincost;
}
```