VISVESVARAYATECHNOLOGICAL UNIVERSITY

"JnanaSangama",Belgaum-590014,Karnataka.

LABREPORT
on

Analysis andDesign of Algorithms

Submittedby

Rachana N (1BM23CS416)

inpartialfulfillmentfortheawardofthedegreeof
BACHELOROFENGINEERING
in
COMPUTERSCIENCEANDENGINEERING

B.M.S.COLLEGEOFENGINEERING
(AutonomousInstitutionunderVTU)
BENGALURU-560019
April-2024toAugust-2024

B.M.S.CollegeofEngineering,

BullTempleRoad,Bangalore560019

(AffiliatedtoVisvesvarayaTechnologicalUniversity,Belgaum)

DepartmentofComputerScienceandEngineering

CERTIFICATE

Thisistocertifythatthe Lab workentitled"**AnalysisandDesign of Algorithms**"carriedoutby **Rachana N (1BM23CS416),** who is bonafide student of **B.M.S. College of Engineering.**It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester April-2024 to August-2024.The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (23CS4PCADA)** work prescribed for the said degree.

MLakshmiNeelima
Assistant Professor
DepartmentofCSE
BMSCE, Bengaluru

Dr.Jyothi S Nayak
ProfessorandHead
DepartmentofCSE
BMSCE,Bengaluru

IndexSheet

12	Implement FractionalKnapsackusingGreedytechnique.	38-40
13	From a given vertex in a weighted connected graph, find shortest pathsto other vertices using Dijkstra's algorithm.	41-43
14	Implement "N-QueensProblem"using Backtracking.	44-45

CourseOutcome

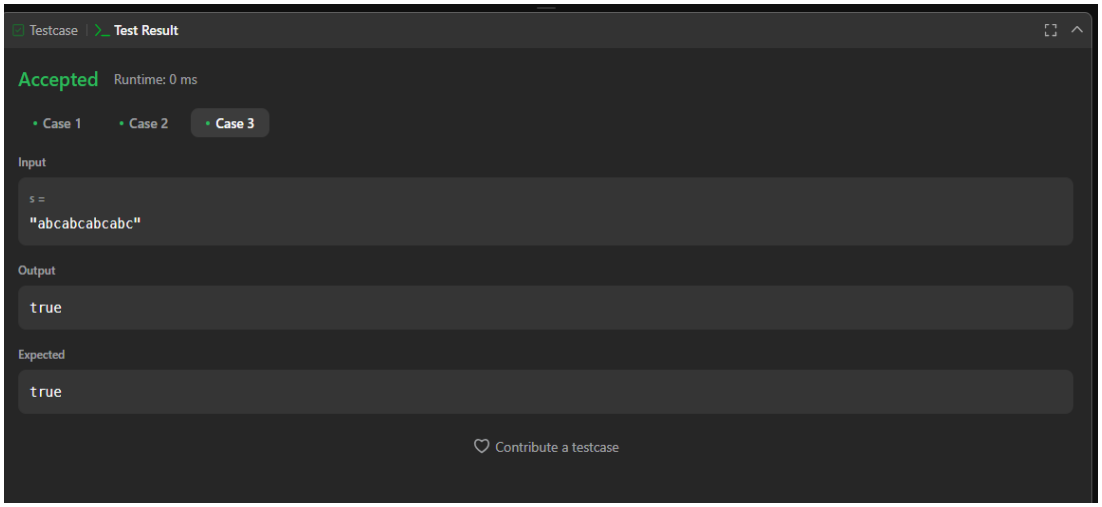
CO1	AnalyzetimecomplexityofRecursiveandNon-recursivealgorithms usingasymptotic notations.
CO2	Applyvarious designtechniques forthegiven problem.
CO3	Applytheknowledgeofcomplexityclasses P,NP,andNP-Completeand prove certain problemsareNP-Complete
CO4	Designefficientalgorithmsandconductpracticalexperimentstosolve problems.

1. **Leetcode exercises on Stacks, Queues, Circular Queues, Priority Queues.**

RepeatedSubstring Matching

```
boolrepeatedSubstringPattern(char*s){ int
    n = strlen(s);
    for(intlen=1;len<=n/2;len++){ if (n
        % len == 0) {
            boolisRepeated=true;
            for(inti=len;i<n;i++){ if
                (s[i] != s[i % len]) {
                    isRepeated=false;
                    break;
                }
            }
            if(isRepeated)returntrue;
        }
    }
    returnfalse;}
```

OUTPUT:



2. Leetcode exercises on DFS, BFS

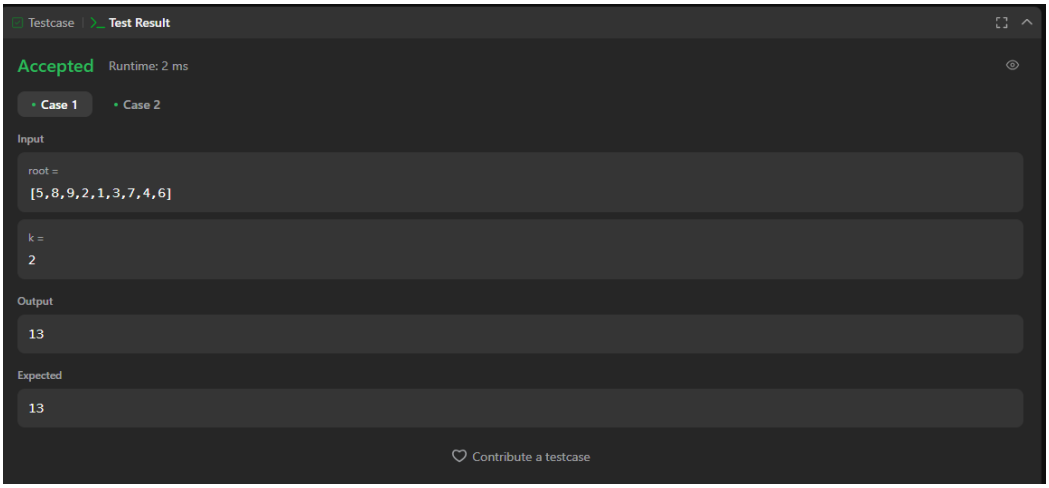
KthLargestSuminaBinaryTree

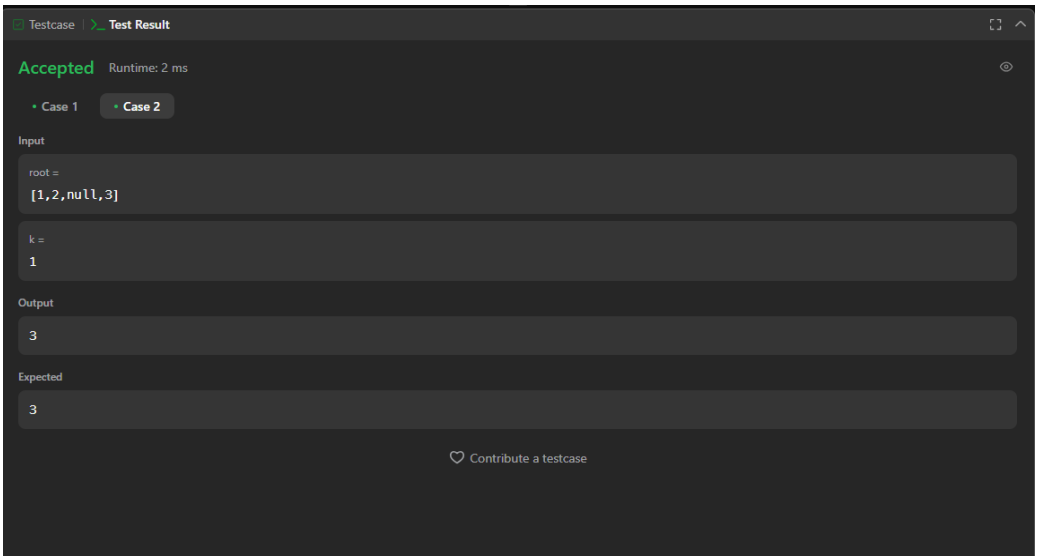
```
voiddfs(structTreeNode*root,intlevel,longlonglevelSum[]){ if
   (root == NULL)
      return;
   levelSum[level] += root->val;
   dfs(root->left, level + 1, levelSum);
   dfs(root->right,level+1,levelSum);
}
longlongkthLargestLevelSum(structTreeNode*root,intk){ if
   (root == NULL)
      return-1;// Ifthetreeisempty
   longlong*levelSum=(longlong*)calloc(1000,sizeof(longlong));
   dfs(root, 0, levelSum);


   //Findthekthlargestlevelsum
   long long* levelSums = (long long*)malloc(1000 * sizeof(long long)); // Dynamically
allocate array
   intnumLevels=0;
   for(inti=0;i<1000&&levelSum[i]!=0;++i){
      levelSums[numLevels++] = levelSum[i];
   }
   for (int i = 0; i < numLevels - 1; ++i) {
      for(intj=i+1;j<numLevels;++j){
         if(levelSums[i]<levelSums[j]){
            longlongtemp=levelSums[i];
            levelSums[i] = levelSums[j];
            levelSums[j] = temp;
         }
      }
```

```
    }
    if (k <= numLevels) {
        returnlevelSums[k-1];
    }
    else{
        return-1;// Ifthere arefewerthanklevelsinthetree
    }
}
```

OUTPUT:



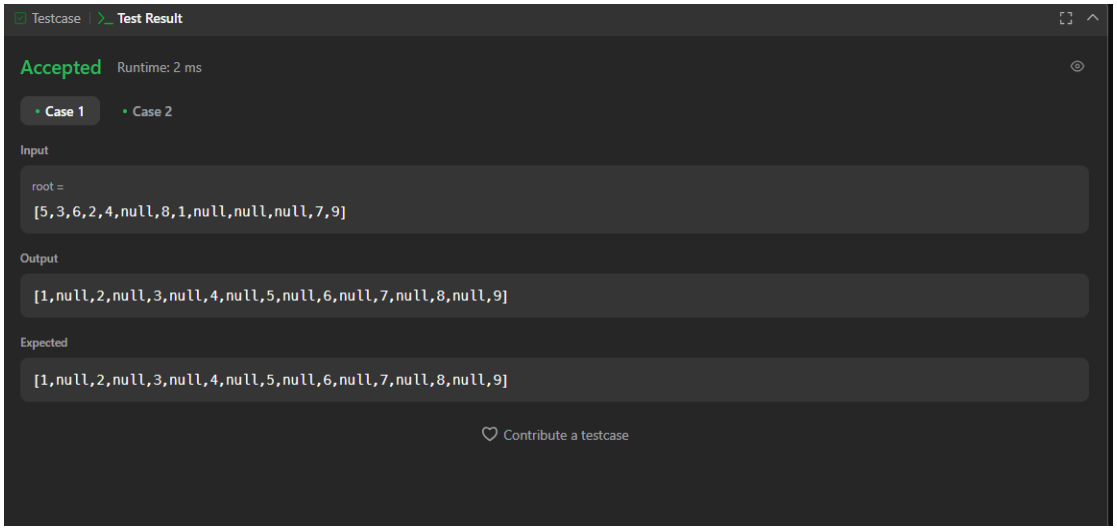


3. LeetcodeexercisesonTreesandGraphs.

IncreasingOrderSearch Tree

```
structTreeNode*increasingBST(structTreeNode*root){ if
    (!root)
        return root;
    structTreeNode*lft=increasingBST(root->left); if
    (lft){
        structTreeNode*temp=lft;
        while (temp->right)
            temp=temp->right;
        root->left = NULL;
        temp->right = root;
        root->right=increasingBST(root->right);
        root = lft;
    }
    else
        root->right=increasingBST(root->right); return
    root;
}
```

OUTPUT:



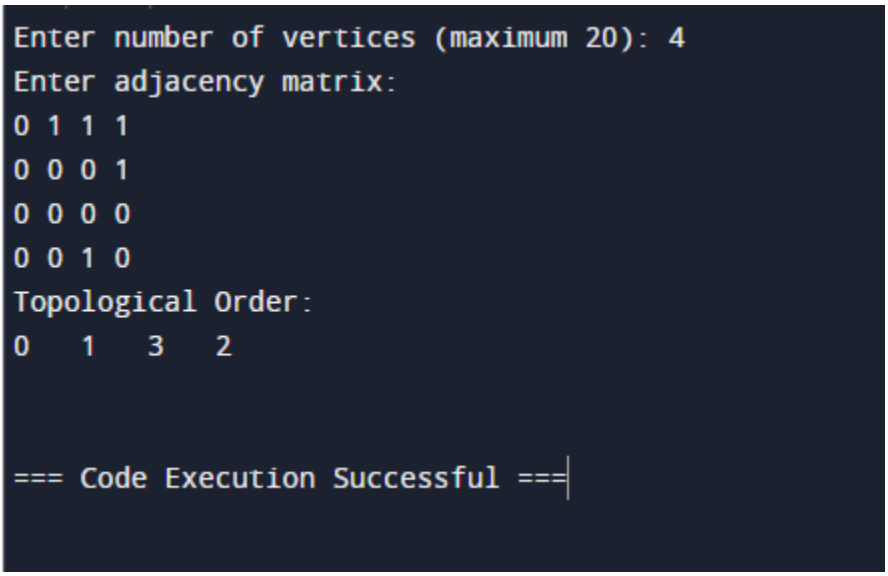
4. **Write program to obtain the Topological ordering of vertices in a given digraph.**

//TopologicalSorting–SourceRemovalTechnique

```
#include<stdio.h>
voidtopologicalSort(inta[20][20],intn){ int
   i, j, k;
   intsum,top=-1,indegree[20],s[20],u,v,T[20]; for (j
   = 0; j < n; j++) {
      sum = 0;
      for(i=0;i<n;i++){ sum
         += a[i][j];
      }
      indegree[j]=sum;
   }
   for (i = 0; i < n; i++)
      {if(indegree[i]==0){
         top++;
         s[top]=i;
      }
   }
   k =0;
   while(top!=-1){ u
      = s[top];
      top--;
      T[k++] =u;
      for(i = 0; i < n; i++){
         if(a[u][i]!=0){//Edgeexists v =
            i;
            indegree[v]--;
            if(indegree[v]==0){
```

```
                top++;
                s[top]=v;
            }
        }
    }
    printf("TopologicalOrder:\n");
    for (i = 0; i < n; i++) {
        printf("%d\t",T[i]);
    }
    printf("\n");
}
intmain() {
    inta[20][20],n;
    printf("Enternumberofvertices(maximum20):"); scanf("%d",
    &n);
    printf("Enteradjacencymatrix:\n");
    for (int i = 0; i < n; i++) {
        for(intj=0;j<n;j++){ scanf("%d",
            &a[i][j]);
        }
    }
    topologicalSort(a,n);
    return 0;
}
```

OUTPUT:



//DFSTopologicalSorting

```
#include<stdio.h>
inta[20][20],n,res[20],visited[20],j=0;
void DFS(int v){
   visited[v]=1;
   for(int i=0;i<n;i++){
      if(a[i][v]==1&&visited[i]==0)
         DFS(i);
   }
   res[j++]=v;
}
void main()
{
   int i,j,v;
   printf("Enterno.ofvertices:");
   scanf("%d",&n);
   printf("Enteradjacencymatrix:\n");
```

```
    for(inti=0;i<n;i++){
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    }
    for(inti=0;i<n;i++)
        visited[i]=0;
    for(v=0;v<n-1;v++){
        if(visited[v]==0)
            DFS(v);
    }
   printf("TopologicalOrder:\n");
    for(i=0;i<n;i++)
        printf("%d\t",res[i]);
}
```

OUTPUT:

```
Enter no. of vertices:5
Enter adjacency matrix:
0 1 0 1 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 0
1 0 0 1 0
Topological Order:
4   0   2   1   3

=== Code Exited With Errors ===
```

5. ImplementJohnsonTrotteralgorithmtogeneratepermutations.

```
#include <stdio.h>
#include<stdbool.h>
#define MAXN 10
//Directionarray,dir[i]storesthedirectionofithelementinpermutation int
dir[MAXN];
int n; // Numberofelements in the permutation
//Functiontoprintthecurrentpermutation
void printPermutation(int perm[]) {
    for (int i = 0; i < n; i++)
        printf("%d",perm[i]);
    printf("\n");
}
//Functiontoswaptwointegers void
swap(int *a, int *b) {
    int temp =*a;
    *a=*b;
    *b =temp;
}
//FunctiontogenerateallpermutationsusingJohnson-Trotteralgorithm void
generatePermutations() {
    intperm[MAXN];//Currentpermutation
    for (int i = 0; i < n; i++) {
        perm[i]=i+1;//Initializepermutationto1,2,...,n dir[i] = -
        1; // All directions initially set to -1 (left)
    }
    printPermutation(perm);//Print theinitial permutation

    int k, mobile, pos;
```

```
    bool found;
    while(true){
        //Step1:Findthelargestmobileinteger mobile
        = -1;
        for(int i =0; i <n; i++) {
            if ((dir[i] == -1 && i != 0 && perm[i] > perm[i - 1]) ||
                (dir[i]==1&&i!=n -1&&perm[i]>perm[i+1])){ if
                (mobile == -1 || perm[i] > perm[mobile]) {
                    mobile =i;
                }
            }
        }
        if(mobile==-1)//Nomoremobileintegers,algorithmterminates break;
        //Step2:Swapthemobileintegerwiththeadjacentintegeritispointingto k =
        mobile + dir[mobile];
        swap(&perm[mobile],&perm[k]);
        swap(&dir[mobile], &dir[k]);
        //Step3:Reversethedirectionofallintegersgreaterthanthemobileinteger for (int i
        = 0; i < n; i++) {
            if(perm[i] >perm[k]) {
                dir[i]=-dir[i];
            }
        }
        //Printthecurrentpermutation
        printPermutation(perm);
    }
}
intmain() {
    printf("Enterthenumber ofelements(maximum %d):", MAXN);
```

```
    scanf("%d",&n);
    if(n <=0||n>MAXN){
        printf("Invalidinput.Numberofelementsshouldbebetween1and%d.\n", MAXN);
        return1;
    }
      printf("Permutations:\n");
    generatePermutations();//GeneratepermutationsusingJohnson-Trotteralgorithm return
    0;
}
```

OUTPUT:

```
Enter the number of elements (maximum 10): 3
Permutations:
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3


=== Code Execution Successful ===
```

6. Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>voi
dsplit(int[],int,int);
voidcombine(int[],int,int,int);
void main()
{
  inta[15000],n,i,j,ch,temp;
  clock_t start,end;
   printf("1:Formanual entryofNvalue and arrayelements");
   printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");
   printf("\n3:Toexit\n");
  while(1){
    printf("\nEnteryourchoice:");
    scanf("%d", &ch);switch(ch){
      case1:printf("Enterthenumberofelements:"); scanf("%d",&n);
                printf("Enterarrayelements:");
                for(i=0;i<n;i++) {
                scanf("%d",&a[i]);
                 }
                start=clock();
                split(a,0,n-1);
                end=clock();
                printf("Sortedarrayis:");
```

```
                    for(i=0;i<n;i++)
                    printf("%d\t",a[i]);
      printf("\nTimetakentosort%dnumbersis%fSecs\n",n,(((double)(end-
start))/CLOCKS_PER_SEC));
                    break;
    case 2:
                n=500;
                while(n<=14500){
                for(i=0;i<n;i++)
                      {
                       //a[i]=random(1000);
                       a[i]=n-i;
                      }
                start=clock();
                split(a,0,n-1);
          //Dummy loop to create delay
              for(j=0;j<500000;j++){temp=38/600;}
                end=clock();
printf("Time taken to sort %d numbers is %f Secs\n",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                     n=n+1000;
                      }
               break;
  case 3: exit(0);
  }
getchar();
   }
}
voidsplit(inta[],intlow,inthigh){
 int mid;
```

```
 if(low<high){
  mid=(low+high)/2;
  split(a,low,mid);
  split(a,mid+1,high);
  combine(a,low,mid,high);
 }
}
voidcombine(inta[],intlow,intmid,inthigh){ int
 c[15000],i,j,k;
 i=k=low;
 j=mid+1;
 while(i<=mid&&j<=high){
  if(a[i]<a[j]){
   c[k]=a[i];
   ++k;
   ++i;
  }
  else{
   c[k]=a[j];
   ++k;
   ++j;
  }
 }
 if(i>mid){
  while(j<=high){
  c[k]=a[j];
   ++k;
   ++j;
  }
 }
```

```
if(j>high){
 while(i<=mid){
 c[k]=a[i];
  ++k;
  ++i;
 }
}
for(i=low;i<=high;i++){
 a[i]=c[i];
}
}
```

OUTPUT:

```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500
    to 14500
3:To exit

Enter your choice:2
Time taken to sort 500 numbers is 0.000550 Secs
Time taken to sort 1500 numbers is 0.000605 Secs
Time taken to sort 2500 numbers is 0.000643 Secs
Time taken to sort 3500 numbers is 0.000727 Secs
Time taken to sort 4500 numbers is 0.000792 Secs
Time taken to sort 5500 numbers is 0.000879 Secs
Time taken to sort 6500 numbers is 0.000978 Secs
Time taken to sort 7500 numbers is 0.001026 Secs
Time taken to sort 8500 numbers is 0.001104 Secs
Time taken to sort 9500 numbers is 0.001184 Secs
Time taken to sort 10500 numbers is 0.001271 Secs
Time taken to sort 11500 numbers is 0.001362 Secs
Time taken to sort 12500 numbers is 0.001440 Secs
Time taken to sort 13500 numbers is 0.001542 Secs
Time taken to sort 14500 numbers is 0.001622 Secs
```

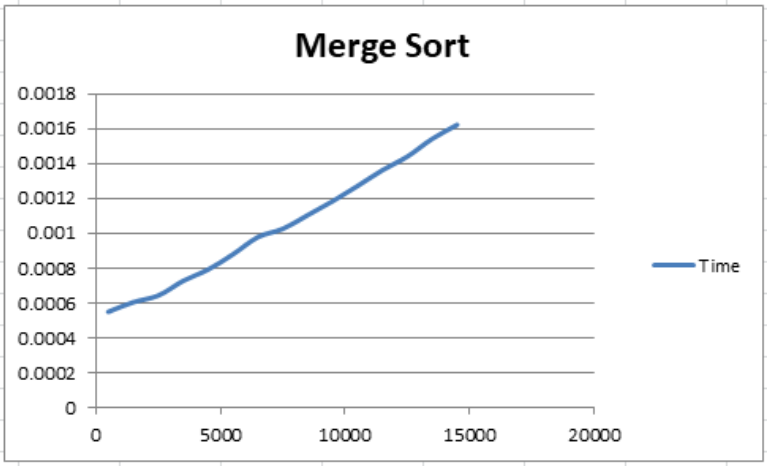
```
Enter your choice:1
Enter the number of elements: 6
Enter array elements: 8 3 4 1 6 7
Sorted array is: 1  3  4  6  7  8
Time taken to sort 6 numbers is 0.000003 Secs

Enter your choice:3


=== Code Execution Successful ===
```

Graph:

N	Time
500	0.00055
1500	0.000605
2500	0.000643
3500	0.000727
4500	0.000792
5500	0.000879
6500	0.000978
7500	0.001026
8500	0.001104
9500	0.001184
10500	0.001271
11500	0.001362
12500	0.00144
13500	0.001542
14500	0.001622



7. Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

```
#include  <stdio.h>
#include   <time.h>
#include<stdlib.h>
voidquicksort(int[],int,int);
int partition(int [], int, int);
void swap(int *, int *);
intmain() {
   inta[15000],n,i,j,temp;
   clock_t start, end;
   printf("1:FormanualentryofNvalueand arrayelements\n");
   printf("2: To display time taken for sorting number of elements N in the range 500 to
14500\n");
   printf("3:Toexit\n");
   int ch;
   while(1){
      printf("\nEnteryourchoice:");
      scanf("%d", &ch);
      switch(ch){
         case 1:
            printf("Enterthenumberofelements:");
            scanf("%d", &n);
            printf("Enterarrayelements:");
            for (i = 0; i < n; i++) {
               scanf("%d",&a[i]);
            }
            start = clock();
            quicksort(a,0,n-1); end
            = clock();
```

```
            printf("Sortedarray:");
            for (i = 0; i < n; i++) {
               printf("%d ", a[i]);
            }
            printf("\nTime taken to sort %d numbers is %f seconds\n", n, ((double)(end -
start)) / CLOCKS_PER_SEC);
            break;
         case 2:
            n =500;
            while(n <=14500){
               for(i=0;i<n;i++){ a[i]
                  = n - i;
               }
               start = clock();
               quicksort(a,0,n-1);
               end = clock();
               printf("Time taken to sort %d numbers is %f seconds\n", n, ((double)(end -
start)) / CLOCKS_PER_SEC);
               n=n +1000; //Incrementn aftereach iteration
            }
            break;
         case 3:
            exit(0);
         default:
            printf("Invalidchoice!Pleaseenteragain.\n");
            break;
      }
      getchar();
   }
```

```
    return0;
}

voidquicksort(inta[],intlow,inthigh){ if
    (low < high) {
        intsplit=partition(a,low,high);
        quicksort(a, low, split - 1);
        quicksort(a, split + 1, high);
    }
}

intpartition(inta[],intlow,inthigh){ int
    pivot = a[low];
    inti = low, j=high +1;

    do {
        do {
            i++;
        }while (a[i]<pivot&&i <= high);

        do {
            j--;
        }while (a[j]>pivot&&j >= low);

        if (i < j)
            {swap(&a[i],&a[j])
            ;
        }
    } while (i < j);
    swap(&a[low],&a[j]);
```

```
return j;
```

```
}
voidswap(int*a,int*b){ int
   temp = *a;
   *a=*b;
   *b =temp;
}
```

OUTPUT:

```
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500
    to 14500
3: To exit

Enter your choice: 2
Time taken to sort 500 numbers is 0.000163 seconds
Time taken to sort 1500 numbers is 0.001387 seconds
Time taken to sort 2500 numbers is 0.003564 seconds
Time taken to sort 3500 numbers is 0.008917 seconds
Time taken to sort 4500 numbers is 0.015828 seconds
Time taken to sort 5500 numbers is 0.019442 seconds
Time taken to sort 6500 numbers is 0.037293 seconds
Time taken to sort 7500 numbers is 0.047849 seconds
Time taken to sort 8500 numbers is 0.052973 seconds
Time taken to sort 9500 numbers is 0.072288 seconds
Time taken to sort 10500 numbers is 0.086499 seconds
Time taken to sort 11500 numbers is 0.109287 seconds
Time taken to sort 12500 numbers is 0.128227 seconds
Time taken to sort 13500 numbers is 0.147480 seconds
Time taken to sort 14500 numbers is 0.174636 seconds
```

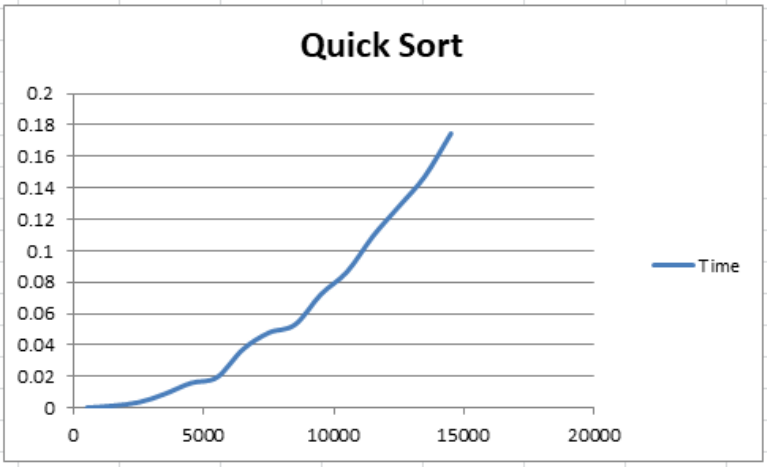
```
Enter your choice: 1
Enter the number of elements: 7
Enter array elements: 10 4 3 7 5 1 9
Sorted array: 1 3 4 5 7 9 10
Time taken to sort 7 numbers is 0.000003 seconds

Enter your choice: 3


=== Code Execution Successful ===
```

Graph:

N	Time
500	0.000163
1500	0.001387
2500	0.003564
3500	0.008917
4500	0.015828
5500	0.019442
6500	0.037293
7500	0.047849
8500	0.052973
9500	0.072288
10500	0.086499
11500	0.109287
12500	0.128227
13500	0.14748
14500	0.174636



8. Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
voidheapsort(intn,inta[]);
void heapify(int n, int a[]);
void swap(int* a, int* b);
void main(){
   inta[15000],n,i,j,ch,temp;
   clock_t start, end;
   printf("\n1:Forsortingof arrayelements");
   printf("\n 2:To displaytimetaken forsortingnumberof elements Nin therange500 to 14500");
   printf("\n3:Toexit");
   while (1)
   {
      printf("\nEnteryourchoice:");
      scanf("%d", &ch);
      switch(ch){
      case 1:
         printf("Enterthenumberofelements:"); scanf("%d",
         &n);
         printf("Enterarrayelements:");
         for (i = 0; i < n; i++)
         {
            scanf("%d",&a[i]);
         }
         start=clock();
         heapsort(n,a);
```

```
            end=clock();
             printf("Sortedarrayelementsare\n"); for
    (i = 0; i < n; i++){
        printf("%d",a[i]);
    }
            printf("\n");
            printf("Timetakentosort%dnumbersis%fSecs\n",n,(((double)(end-start))/
CLOCKS_PER_SEC));
            break;
        case 2:
                n=500;
      while(n<=14500){
      for(i=0;i<n;i++){
   //a[i]=random(1000);
   a[i]=n-i;
 }
      start=clock();
      heapsort(n,a);
          //Dummy loop to create delay
    for(j=0;j<500000;j++){temp=38/600;}
             end=clock();
            printf("Timetakentosort%dnumbersis%fSecs\n",n,(((double)(end-
start))/CLOCKS_PER_SEC));
            n=n+1000;
 }
            break;
        case 3:
            exit(0);
        }
        getchar();//Consumenewlinecharacterleftininput buffer
```

```
    }
}
voidheapify(intn,int a[])
{
    int i, p, c, item;
    for(p=(n-1)/2;p>=0;p--){ item =
        a[p];
        c = 2 * p + 1;
        while(c<n){
            if(c+1<n&& a[c]<a[c+1])
            {
                c++;
            }
            if(item>=a[c])
                break;
            a[p]=a[c];
            p = c;
            c=2 * p +1;
        }
        a[p]=item;
    }
}
voidswap(int*a,int*b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b =temp;
}
voidheapsort(intn,int a[])
```

```
{
   int i;
   heapify(n,a);
   for(i = n-1; i >0; i--)
   {
      swap(&a[0],&a[i]);
      heapify(i, a);
   }
}
```

OUTPUT:

```
 1:For sorting of array elements
 2:To display time taken for sorting number of elements N in the range 500
     to 14500
 3:To exit
Enter your choice:2
Time taken to sort 500 numbers is 0.001497 Secs
Time taken to sort 1500 numbers is 0.006219 Secs
Time taken to sort 2500 numbers is 0.016747 Secs
Time taken to sort 3500 numbers is 0.034821 Secs
Time taken to sort 4500 numbers is 0.048089 Secs
Time taken to sort 5500 numbers is 0.072304 Secs
Time taken to sort 6500 numbers is 0.097209 Secs
Time taken to sort 7500 numbers is 0.136986 Secs
Time taken to sort 8500 numbers is 0.183444 Secs
Time taken to sort 9500 numbers is 0.281507 Secs
Time taken to sort 10500 numbers is 0.330495 Secs
Time taken to sort 11500 numbers is 0.355252 Secs
Time taken to sort 12500 numbers is 0.389277 Secs
Time taken to sort 13500 numbers is 0.429252 Secs
Time taken to sort 14500 numbers is 0.559296 Secs
```

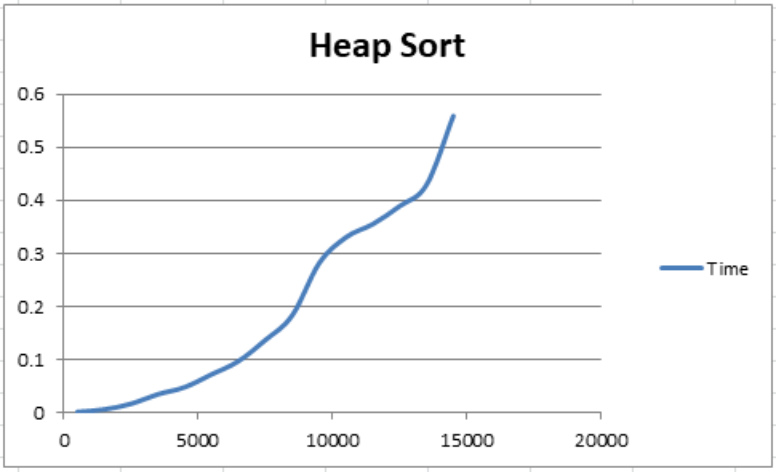
```
Enter your choice:1
Enter the number of elements: 6
Enter array elements: 1 7 2 1 6 4
Sorted array elements are
1 1 2 4 6 7
Time taken to sort 6 numbers is 0.000002 Secs

Enter your choice:3


=== Code Execution Successful ===
```

Graph:

N	Time
500	0.001497
1500	0.006219
2500	0.016747
3500	0.034821
4500	0.048089
5500	0.072304
6500	0.097209
7500	0.136986
8500	0.183444
9500	0.281507
10500	0.330495
11500	0.355252
12500	0.389277
13500	0.429252
14500	0.559296



9. Implement0/1Knapsackproblemusingdynamicprogramming.

```
#include<stdio.h>
#define N 4
int max(int a, int b) {
   return(a>b)?a:b;
}
voidknapsack(intW,intweights[],intprofits[]){ int
   dp[N + 1][W + 1];
   for(int i = 0; i <=N;i++) {
      for(intw=0;w<=W;w++){ if (i
         == 0 || w == 0)
            dp[i][w]=0;
         elseif(weights[i -1] <=w)
            dp[i][w]=max(profits[i -1]+dp[i-1][w-weights[i-1]],dp[i-1][w]); else
            dp[i][w] =dp[i-1][w];
      }
   }
   int maxProfit = dp[N][W];
   printf("MaximumProfit:%d\n",maxProfit);
   int w = W;
   printf("Objectsselectedintheknapsack:\n"); for
   (int i = N; i > 0 && maxProfit > 0; i--) {
      if(maxProfit==dp[i-1][w]) continue;
      else{
         printf("Object%d(Weight=%d,Profit=%d)\n",i,weights[i-1],profits[i-1]); maxProfit -=
         profits[i - 1];
         w-=weights[i-1];
```

```
        }
    }
}
intmain() {
    intweights[20],n;
    int profits[20],W;
    printf("Enternumberofweights:");
    scanf("%d", &n);
    printf("EnterMaximumwight:");
    scanf("%d",&W);
    printf("Entertheweights:\n"); for
    (int j = 0; j < n; j++) {
            scanf("%d",&weights[j]);
        }
    printf("Entertheprofits:\n"); for
    (int j = 0; j < n; j++) {
            scanf("%d",&profits[j]);
        }
    knapsack(W,weights,profits);
    return 0;
}
```

OUTPUT:

```
Enter number of weights: 5
Enter Maximum wight:8
Enter the weights:
1 4 3 2 1
Enter the profits:
10 20 15 13 11
Maximum Profit: 45
Objects selected in the knapsack:
Object 3 (Weight = 3, Profit = 15)
Object 2 (Weight = 4, Profit = 20)
Object 1 (Weight = 1, Profit = 10)


=== Code Execution Successful ===
```

10. ImplementAllPairShortestpathsproblemusingFloyd'salgorithm.

```
#include <stdio.h>
#include<limits.h>
voidfloyd(intn,intcost[][n],intD[][n]){ int
   i, j, k;
   for (i = 0; i < n; i++) {
      for(j = 0; j < n; j++){
         D[i][j] =cost[i][j];
      }
   }
   for (k = 0; k < n; k++) {
      for(i = 0; i < n; i++){
         for(j = 0; j < n; j++) {
            if(D[i][k]!=INT_MAX&&D[k][j]!=INT_MAX&&D[i][j]>D[i][k]+
D[k][j]){
               D[i][j]=D[i][k] +D[k][j];
            }
         }
      }
   }
}
voidprintShortestPaths(intn,intD[][n]){
   printf("Shortestpathsbetweeneverypairofvertices:\n"); for
   (int i = 0; i < n; i++) {
      for(int j =0; j <n; j++) {
         if(D[i][j]==INT_MAX){
            printf("INF\t");
```

```
            }else {
                printf("%d\t",D[i][j]);
            }
        }
        printf("\n");
    }
}
intmain(){ int
    n;
    printf("Enterthenumberofverticesinthegraph:"); scanf("%d",
    &n);
    int cost[n][n];
    printf("Enterthecostadjacencymatrix(use'-1'forinfinity):\n"); for
    (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d",&cost[i][j]);
            if (cost[i][j] == -1) {
                cost[i][j]=INT_MAX;
            }
        }
    }
    int D[n][n];
    floyd(n, cost, D);
    printShortestPaths(n,D);
    return 0;
}
```

OUTPUT:

```
Enter the number of vertices in the graph: 5
Enter the cost adjacency matrix (use '-1' for infinity):
0  2  -1  -1  3
4  0   3  -1  -1
7  -1  0  -1  -1
8  -1  1  0   -1
-1  9  2  -1  0
Shortest paths between every pair of vertices:
0   2   5   INF 3
4   0   3   INF 7
7   9   0   INF 10
8   10  1   0   11
9   9   2   INF 0


=== Code Execution Successful ===
```

11. Find Minimum Cost Spanning Tree of a given undirected graph usingPrim's algorithm.

```
#include <stdio.h>
#defineMAX9999
voidprims(intn,intcost[n][n]) {
   inti,j,u,min,sum=0,source,K=0; int S[n],
   d[n], P[n], T[n-1][2];
   min=MAX;
   source = 0;
   for (i = 0; i < n; i++) {
      for(j=0;j<n;j++){
         if(cost[i][j]!=0&&cost[i][j]<min){ min =
            cost[i][j];
            source=i;
         }
      }
   }
   for(i=0;i<n;i++){ S[i] =
      0;
      d[i]=cost[source][i];
      P[i] = source;
   }
   S[source]=1;
   for(i=1;i<n;i++){ min =
      MAX;
      u=-1;
      for(j = 0; j < n; j++){
         if(S[j]==0&&d[j]<=min){ min
            = d[j];
            u =j;
```

```
            }
        }
        T[K][0]=u;
        T[K][1]=P[u];
        K++;
        sum+=cost[u][P[u]];
        S[u] = 1;
        for(j = 0; j < n; j++){
            if(S[j]==0&&cost[u][j]<d[j]){ d[j]
                = cost[u][j];
                P[j] =u;
            }
        }
    }
    if(sum >=MAX) {
        printf("Spanningtreedoesnot exist.\n");
    }else {
        printf("SpanningtreeexistsandMSTis:\n"); for
        (i = 0; i < n-1; i++) {
            printf("%d-%d\n",T[i][0], T[i][1]);
        }
        printf("Thecost ofspanningtree(MST)is %d\n", sum);
    }
}
intmain(){ int
    n;
    printf("Enternumberofvertices:");
    scanf("%d", &n);
    int cost[n][n];
    printf("Enterthecost adjacencymatrix:\n");
```

```
    for (int i = 0; i < n; i++) {
        for(intj=0;j<n;j++){
            scanf("%d",&cost[i][j]);
        }
    }
    prims(n,cost);
    return 0;
}
```

OUTPUT:

```
Enter number of vertices: 4
Enter the cost adjacency matrix:
0 1 3 9999
1 0 1 9999
3 1 0 2
9999 9999 2 0
Spanning tree exists and MST is:
1 - 0
2 - 1
3 - 2
The cost of spanning tree (MST) is 4


=== Code Execution Successful ===
```

- **FindMinimumCostSpanningTreeofagivenundirectedgraphusingKruskal's algorithm.**

```
#include<stdio.h>
#defineMAX9999//Infinityvalueassumed void
kruskals(int c[][100], int n);
intmain(){ int
    n, i, j;
    intc[100][100];//Assumingamaximumsizeforthecostmatrix printf("Enter
    the number of nodes: ");
```

```
    scanf("%d",&n);
    printf("Enterthecostmatrix:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d",&c[i][j]);
            if(c[i][j]==0)//Assuming0representsnoedge,setittoalargevalue c[i][j] =
                MAX;
        }
    }
    kruskals(c,n);
    return 0;
}
voidkruskals(intc[][100],intn){ int
    ne = 0, mincost = 0;
    int parent[100];
    intmin,u,v,a,b,i,j; for(i
    = 1; i <= n; i++)
        parent[i] = 0;
    while(ne!=n-1){
        min =MAX;
        for (i = 1; i <= n; i++) {
            for(j=1;j<=n;j++){
                if(c[i][j] <min) {
                    min=c[i][j]; u
                    = a = i;
                    v =b =j;
                }
            }
        }
        while(parent[u]!=0)
```

```
        u = parent[u];
      while(parent[v]!=0)
        v=parent[v];
      if (u != v) {
        printf("Edge%d-%d:%d\n",a,b,min);
        parent[v] = u;
        mincost+=min;
        ne++;
      }
      c[a][b]=c[b][a]=MAX;
    }
    printf("Minimumcostofspanningtree:%d\n",mincost);
}
```

OUTPUT:

```
Enter the number of nodes: 4
Enter the cost matrix:
0 6 1 4
2 0 3 4
3 1 0 5
1 1 1 0
Edge 1-3: 1
Edge 3-2: 1
Edge 4-1: 1
Minimum cost of spanning tree: 3


=== Code Execution Successful ===
```

12. ImplementFractionalKnapsackusingGreedytechnique.

```
#include<stdio.h>
voidknapsack(intn,float weight[],floatprofit[],floatcapacity)
{
  floatx[20],tp=0; int
  i, j, u;
  u =capacity;
  for(i=0;i<n;i++) x[i]
    = 0.0;
  for(i=0;i<n;i++){ if
    (weight[i] > u)
      break;
    else {
      x[i] =1.0;
      tp=tp+profit[i]; u
      = u - weight[i];
    }
  }
  if(i <n)
    x[i] =u / weight[i];
  tp = tp + (x[i] * profit[i]);
  printf("\nTheresultvectoris:-"); for
  (i = 0; i < n; i++)
    printf("%f\t", x[i]);
  printf("\nMaximumprofitis:-%f",tp);
}
intmain() {
  floatweight[20],profit[20],capacity;
  int num, i, j;
```

```
floatratio[20],temp;
printf("Entertheno.ofobjects:-");
scanf("%d", &num);
printf("Enterthewtsofeachobject:-\n"); for
(i = 0; i < num; i++) {
  scanf("%f",&weight[i]);
}
printf("Entertheprofitsofeachobject:-\n"); for
(i = 0; i < num; i++) {
  scanf("%f",&profit[i]);
}
printf("Enterthecapacityofknapsack:-");
scanf("%f", &capacity);
for (i = 0; i < num; i++) {
  ratio[i]=profit[i]/weight[i];
}
for(i = 0; i < num; i++) {
  for(j=i+1;j<num;j++){ if
    (ratio[i] < ratio[j]) {
      temp = ratio[j];
      ratio[j]=ratio[i];
      ratio[i] = temp;
      temp=weight[j];
      weight[j]=weight[i];
      weight[i] = temp;
      temp = profit[j];
      profit[j] = profit[i];
      profit[i] = temp;
    }
  }
```

```
  }
  knapsack(num,weight,profit,capacity);
  return(0);
}
```

OUTPUT:

```
Enter the no. of objects:-4
Enter the wts of each object:-
3 1 2 4
Enter the profits of each object:-
20 26 22 21
Enter the capacity of knapsack:-8

The result vector is:- 1.000000 1.000000    1.000000    0.500000
Maximum profit is:- 78.500000

=== Code Execution Successful ===
```

13. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

```
#include<stdio.h>
#defineMAX9999//Infinityvalueassumed void
dijkstras(int c[][100], int n, int src);
intmain() {
   intn, src,i, j;
   intc[100][100];//Assumingamaximumsizeforthecostmatrix
   printf("Enter the number of nodes: ");
   scanf("%d",&n);
   printf("Enterthecostmatrix:\n");
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n; j++) {
         scanf("%d",&c[i][j]);
      }
   }
   printf("Enterthesourcenode(1to%d):",n); scanf("%d",
   &src);
   dijkstras(c,n,src);
   return 0;
}
voiddijkstras(intc[][100],intn,intsrc){ int
   dist[100], vis[100];
   intcount, min, u, i, j;
   //Initialization
   for(j=1;j<=n;j++){
      dist[j] = c[src][j];
      vis[j] = 0;
   }
   dist[src] =0;
```

```
    vis[src]=1;
    count= 1;
    // Main loop
    while(count!=n){ min
       = MAX;

       //Findtheminimumdistancevertexfromthesetofverticesnot yetprocessed for (j =
       1; j <= n; j++) {
          if(dist[j]<min&&vis[j]!=1){ min =
             dist[j];
             u =j;
          }
       }
       vis[u]=1;
       count++;
       //Updatedistvalueoftheadjacentverticesofthepickedvertex for (j
       = 1; j <= n; j++) {
          if(min+c[u][j]<dist[j]&&vis[j]!=1){ dist[j] =
             min + c[u][j];
          }
       }
    }

    //Outputshortest distances
    printf("Shortestdistancesfromnode%d:\n",src); for
    (j = 1; j <= n; j++) {
       printf("Distancetonode%dfromnode%d: %d\n",j,src, dist[j]);
    }
}
```

OUTPUT:

```
Enter the number of nodes: 4
Enter the cost matrix:
0 2 3 4
1 0 3 4
2 1 0 3
5 6 2 0
Enter the source node (1 to 4): 2
Shortest distances from node 2:
Distance to node 1 from node 2: 1
Distance to node 2 from node 2: 0
Distance to node 3 from node 2: 3
Distance to node 4 from node 2: 4


=== Code Execution Successful ===
```

14. Implement“N-QueensProblem”usingBacktracking.

```
#include   <stdio.h>
#include  <stdlib.h>
#defineMAX_N10
int x[MAX_N];
intPlace(intk,inti,intn) {
   for(int j = 1; j <=k-1; j++) {
      if(x[j]==i||abs(x[j]-i)==abs(j-k)){ return 0;
      }
   }
   return1;
}
void NQueens(int k, int n) {
   for(inti=1;i<=n;i++){
      if(Place(k,i,n)){ x[k]
         = i;
         if (k == n) {
            printf("Solution:");
            for(intj=1;j<=n;j++){
               printf("%d ", x[j]);
            }
            printf("\n");
         }else {
            NQueens(k +1, n);
         }
      }
   }
}
```

```
intmain(){ int
    n;
   printf("Enterthenumberofqueens(n):");
    scanf("%d", &n);
    for(inti=0;i<=n;i++){ x[i]
        = 0;
    }
    NQueens(1,n);
    return 0;
}
```

OUTPUT:

```
Enter the number of queens (n): 5
Solution: 1 3 5 2 4
Solution: 1 4 2 5 3
Solution: 2 4 1 3 5
Solution: 2 5 3 1 4
Solution: 3 1 4 2 5
Solution: 3 5 2 4 1
Solution: 4 1 3 5 2
Solution: 4 2 5 3 1
Solution: 5 2 4 1 3
Solution: 5 3 1 4 2


=== Code Execution Successful ===
```