1.MERGESORT()

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include<time.h>

#define MAX 500

int A[MAX] , B[MAX];


void merge(int,int,int);

void mergeSort(int,int);


int main(void)

{

   int n,i,j;

   int low , high , mid;

   clock_t s, e;

   double  cpu_exe_t;

   printf("\nPlease enter the size of the array: ");

   scanf("%d",&n);

   low=0;

   high=n-1;


   for(i=0;i<n;i++)

      A[i]=rand()%100;


   printf("\nThe array elements are: \n");

   for(i=0;i<n;i++)

      printf("%d\t",A[i]);
```

```
    s=clock();
    for(j=0;j<1000;j++)           //Delay loops
    for(i=0;i<1000;i++)
    mergeSort(low,high);
    e=clock();
    cpu_exe_t=(double)(e-s)/CLK_TCK;
    printf("\nThe sorted array is :\n");
    for(i=0;i<n;i++)
        printf("%d\t",B[i]);
    printf("\nCPU execution time is %lf\n",cpu_exe_t);
    return 0;
}
void mergeSort(int low ,int high)
{
    if(low<high)
    {
        //mid = (low+high)/2;
        int mid = low + (high-low)/2;   //To avoid integer overflow
        mergeSort(low,mid);
        mergeSort(mid+1,high);
        merge(low,mid,high);
    }
}
void merge(int low ,int mid ,int high)
{
    int i=low , j=mid+1 , k=low;
    while(i<=mid && j<=high)
    {
        if(A[i]<=A[j]){
            B[k++]=A[i++];
        }
        else{
            B[k++]=A[j++];
        }
    }
    while(i<=mid){
        B[k++]=A[i++];
    }
    while(j<=high){
        B[k++]=A[j++];
    }
}
```

2.QUICKSORT()

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


#define MAX 20000

int A[MAX];


void Quicksort(int low, int high);

int Partition(int low,int high);

void swap(int *p,int *q);


int main()

{

   int n,i,j;

   int low,high;


   clock_t s,e;

   double cpu_exe_t;


   printf("\n please enter the size of the array:");


   scanf("%d",&n);


   for(i=0;i<n;i++)

   {

      A[i]=rand()%100;
```

```
    }

    printf("\n The array elements are: \n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",A[i]);
    }
    s = clock();
    for(j=0;j<1000;j++)
    for(i=0;i<1000;i++)
    {
        low=0;
        high=n-1;
        Quicksort(low,high);
    }
    e= clock();
    cpu_exe_t = (double)(e-s)/CLK_TCK;

    printf("\n The sorted array is: \n");
    for(i=0;i<n;i++)
    {
        printf("%d \t", A[i]);
    }
    printf("\nCPU execution time is %lf",cpu_exe_t);

    return 0;
}
int partition(int low, int high)
{
    int i,j;
    int pivot=A[low];

    i=low;
    j=high+1;

    while(i<j)
    {
        do
        {
            ++i;
        }while(A[i]<=pivot);
        do
        {
            --j;
        }while(A[j]>pivot);

        if(i<j)
        {
            swap(&A[i],&A[j]);
        }
    }
    swap(&A[low],&A[j]);

    return j;
}
void Quicksort(int low, int high)
{
    int j;
    if(low<high)
    {
        j=partition(low,high);
        Quicksort(low,j-1);
        Quicksort(j+1,high);
    }
}
void swap(int *a,int *b)
{
    int t;
    t = *a;
    *a =*b;
    *b=t;
}
```

3.INSERTIONSORT()

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


int a[1000];

void linear_sort(int n)

{

    int i,j,key;

    for(i=1;i<n;i++)

    {

        key = a[i];

        j = i-1;

        while(j>=0 && key<a[j])

        {

            a[j+1] = a[j];

            j = j-1;

        }

        a[j+1] = key;

    }

}


int main()

{

    int i,j,n,key;

    time_t s,e;
    double cpu_exe_t;
    printf("Enter the no of elements: ");
    scanf("%d",&n);
    printf("\nEnter the %d elements: ", n);
    for(i=0;i<n;i++)
        a[i] = rand()%100;
    printf("\nThe entered nos are: \n");
    for(i=0;i<n;i++)
        printf("%d\t",a[i]);
    s = clock();
    for(i=0;i<1000;i++)
    {
        for(j=0;j<1000;j++)
        {
            linear_sort(n);
        }
    }
    e = clock();
    cpu_exe_t = (double)(e-s)/CLK_TCK;

    printf("\nSorted elements are: \n");
    for(i=0;i<n;i++)
        printf("%d\t",a[i]);
    printf("\nThe CPU execution time: %lf",cpu_exe_t);
    return 0;
}
```

4.HEAPSORT ()

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include<time.h>


#define MAX 10000


void exchange (int *p, int *q)

{

   int t;

   t = *p;

   *p = *q;

   *q = t;

}

void HeapSort(int *A, int n)

{

   int i;

   for(i=n/2; i>=1; i--)

       Heapify(A,n,i);

   for(i=n; i>=2; i--)

   {

        exchange(&A[i], &A[1]);

        Heapify(A,i-1,1);

   }

}
```

```
void Heapify(int *A, int n, int i)
{
    int largest, l, r;
    largest = i;
    l = 2*i; r = 2*i+1;
    if(l<=n && A[l]>A[largest])
        largest = l;
    if(r<=n && A[r] > A[largest])
        largest = r;
    if(largest != i)
    {
        exchange(&A[largest] ,&A[i]);
        Heapify(A, n, largest);
    }
}
int main()
{
    int i,n,A[MAX],k,j;
    srand(1);
    time_t start,end;
    double cpu_exec_time;



    printf("Heap Sort..\n");
    printf("Enter the value of n: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
        start = clock();
    for(i=0;i<100;i++)
```

```
        for(j=0;j<100;j++)
        {
            for(k=0;k<n;k++)
                A[k] = rand() % 100+1;
            HeapSort(A,n);
        }
        end = clock();
        cpu_exec_time = (double) (end - start) / CLOCKS_PER_SEC;

        printf("Sorted array: ");
        for(i=1;i<=n;i++)
            printf("%d\t",A[i]);
        printf("\n Execution time = %lf\n", cpu_exec_time);
    return 0;
}
```

5.DIJKSTRA ()

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#define infinity 999


void dij(int n,int v,int cost[10][10],int dist[100])

{

   int i,u,count,w,flag[10],min;

   for(i=1;i<=n;i++)

      flag[i]=0,dist[i]=cost[v][i];

   count=2;

   while(count<=n)

   {

      min=99;

      for(w=1;w<=n;w++)

         if(dist[w]<min&& !flag[w])

            min=dist[w],u=w;

      flag[u]=1;

      count++;

      for(w=1;w<=n;w++)

         if((dist[u]+cost[u][w]<dist[w])&& !flag[w])

            dist[w]=dist[u]+cost[u][w];

   }

}


void main()
```

```
{
    int v,n,i,j,cost[10][10],dist[10];
    printf("\nEnter the number of nodes :\n");
    scanf("%d",&n);
    printf("\nEnter the cost matrix :\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=infinity;
        }
    }
    printf("\nEnter the source matrix :\n");
    scanf("%d",&v);
    dij(n,v,cost,dist);
    printf("\nShortest path :\n");
    for(i=1;i<=n;i++)
        if(i!=v)
            printf("%d->%d,cost=%d\n",v,i,dist[i]);

}
```

6.PRIMS ()

CODE:

```
#include <stdio.h>

#include <stdlib.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void main()

{
   printf("Enter the number of nodes: ");

   scanf("%d",&n);

   printf("\n Enter the adjacency matrix: ");

   for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
   {
      scanf("%d",&cost[i][j]);

      if(cost[i][j]==0)
         cost[i][j]=999;
   }
```

```
visited[1]=1;

printf("\n");

while(ne < n)
{
   for(i=1,min=999;i<=n;i++)
      for(j=1;j<=n;j++)

      if(cost[i][j]<min)

      if(visited[i]!=0)
   {
      min=cost[i][j];

      a=u=i;

      b=v=j++;
   }
   if(visited[u]==0 || visited[v]==0)
   {
      printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

      mincost+=min;

      visited[b]=1;
   }
```

```
        cost[a][b]=cost[b][a]=999;

    }

    printf("\n Minimun cost=%d\n",mincost);

}
```

7.FLOYDS ()

CODE:

```
#include <stdio.h>

int min(int,int);

void printM(int D[10][10],int n)
{
   int i,j;

   for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
         printf("%d",D[i][j]);
      printf("\n");
   }
}

void floyds(int D[10][10], int n)
{
   int i,j,k;

   for(k=1;k<=n;k++)
```

```
    {
        printf("With %d as Intermediate Vertex",k);
        printf("Cost Matrix now: \n");
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(i==j)
                    D[i][j]=0;
                else
                    D[i][j]=min(D[i][j],D[i][k]+D[k][j]);
                printM(D,n);
    }
}
int min(int a,int b)
{
    return(a<b)?a:b;
}

void main()
{
    int D[10][10],w,n,e,u,v,i,j;
    printf("Enter the number of vertices: \n");
    scanf("%d",&n);

    printf("\n Enter the Cost Matrix: ");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
             scanf("%d",&D[i][j]);
    }
    printf("The initial cost of matrix: \n");

    printM(D,n);


    floyds(D,n);

    printf("\n The Final Cost Matrix: ");

    printM(D,n);

    printf("The Shortest Paths are: \n");

    for(i=1;i<=n;i++)

       for(j=1;j<=n;j++)

    {

       if(i!=j)

           printf("\n <%d,%d> ====> %d",i,j,D[i][j]);

    }

    return 0;

}
```

8.KNAPSACK()

CODE:

```
#include<stdio.h>

#define MAX 200

int V[MAX][MAX] = {0};

int res[200]={0};

int count=0;


int max(int a, int b) {

return (a > b)? a : b;

}


int knapSack(int W, int wt[], int val[], int n)

{

int i, j;



for (i = 0; i <= n; i++)

{

for (j = 0; j <= W; j++)

{

if (i==0 || j==0)

{

                                V[i][j] = 0;
}
else if (wt[i-1] <= j)
{                          V[i][j] = max(val[i-1] + V[i-1][j-wt[i-1]],  V[i-1][j]);
}
else {
                                        V[i][j] = V[i-1][j];
}
}

int k,m;
//Print matrix after every iteration
for(k=0;k<=n;k++) {
for(m=0;m<=W;m++) {
printf("%d ", V[k][m]);
}
printf("\n");
}
printf("\n");
}

i=n;
j=W;
while(i>0 && j>0)
{
if(V[i][j] != V[i-1][j])
{
res[count++]=i;
j=j-wt[i-1];
i--;
```

```
}
else
i--;
}

return V[n][W];
}

int main()
{
int i, n, W, optsoln;
int val[20],wt[20];
        printf("\nEnter number of items:\n");
        scanf("%d", &n);

printf("\nEnter the weights:\n");
for(i=0;i<n;i++)
scanf("%d",&wt[i]);

printf("\nEnter the values:\n");
for(i=0;i<n;i++)
scanf("%d",&val[i]);
printf("\nEnter the knapsack capacity:");
scanf("%d",&W);
optsoln=knapSack(W, wt, val, n);

printf("\nThe optimal soluntion is:%d",optsoln);

printf("\nThe optimal subset\n");
printf("Items included in knapsack are:");

for(i=count-1;i>=0;i--)

printf("\t%d",res[i]);

printf("\n");


return 0;

}
```

9.SUMSUBSET ()

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 50

#define TRUE 1

#define FALSE 0

int inc[MAX], w[MAX], sum, n;


int promising(int i, int wt, int total)

{

   return (((wt + total) >= sum) && ((wt == sum) || (wt + w[i + 1] <= sum)));

}


void sumset(int i, int wt, int total)

{

   int j;

   if (promising(i, wt, total))

   {


      if (wt == sum)

      {


         printf("\n{\t");


         for (j = 0; j <= i; j++)

            if (inc[j])
```

```
                printf("%d\t", w[j]);

            printf("}\n");
        }
        else
        {
            inc[i + 1] = TRUE;
            sumset(i + 1, wt + w[i + 1], total - w[i + 1]);
            inc[i + 1] = FALSE;
            sumset(i + 1, wt, total - w[i + 1]);
        }
    }
}

int main()
{
    int i, j, n, temp, total = 0;
    printf("\n Enter how many numbers:\n");
    scanf("%d", &n);
    printf("\n Enter %d numbers to the set:\n", n);

    for (i = 0; i < n; i++)
    {
        scanf("%d", &w[i]);
        total += w[i];
    }
    printf("\n Input the sum value to create sub set:\n");
    scanf("%d", &sum);
    for (i = 0; i <= n; i++)
        for (j = 0; j < n - 1; j++)
            if (w[j] > w[j + 1])
            {
                temp = w[j];
                w[j] = w[j + 1];
                w[j + 1] = temp;
            }

    printf("\n The given %d numbers in ascending order:\n", n);

    for (i = 0; i < n; i++)
        printf("%d \t", w[i]);

    if ((total < sum))
        printf("\n Subset construction is not possible");
    else
    {
        for (i = 0; i < n; i++)
            inc[i] = 0;

        printf("\n The solution using backtracking is:\n");
        sumset(-1, 0, total);
    }

    system("PAUSE");
    return 0;
}
```

10.NQUEEN ()

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include<math.h>


int a[30],count=0;


int place(int pos)

{

   int i;

   for(i=1;i<pos;i++)

   {

      if((a[i]==a[pos])|| ((abs(a[i]-a[pos])==abs(i-pos))))

         return 0;

   }

   return 1;

}


void printsol(int n)

{

   int i,j;

   count++;

   printf("\n\n solution #%d \n\n",count);

   for(i=1;i<=n;i++)

   {

      for(j=1;j<=n;j++)

      {
            if(a[i]==j)
                printf(" Q\t");
            else
                printf(" 0\t");
        }
        printf("\n");

    }
}

void queen(int n)
{
    int k=1;
    a[k]=0;
    while(k!=0)
    {
        a[k]=a[k]+1;
        while(a[k]<=n && !place(k))
        a[k]++;
        if(a[k]<=n)
        {
            if(k==n)
                printsol(n);
            else
                {
                k++;
                a[k]=0;
                 }
        }
        else k--;

    }
}

void main()
{
   int n;
   printf("\n enter no. of queens: ");
   scanf("\n %d",&n);
   queen(n);
   printf("\n total no. of solutions: %d" ,count);

}
```