# DAY 2 PROGRAMS

# CSA0672-DESIGN AND ANALYSIS OF ALGORTHM

NAME : M.VENKATA SAI

REGNO: 192110433

1) MERGE SORT

PROGRAM:

```c
#include <stdio.h>
#include <stdlib.h>

void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
```

# DAY 2 PROGRAMS

```
        i++;

    } else {

        arr[k] = R[j];

        j++;

    }

    k++;

    }


    while (i < n1) {

        arr[k] = L[i];

        i++;

        k++;

    }


    while (j < n2) {

        arr[k] = R[j];

        j++;

        k++;

    }
}


void mergeSort(int arr[], int l, int r) {
    if (l < r) {

        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);

        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);

    }
}
```
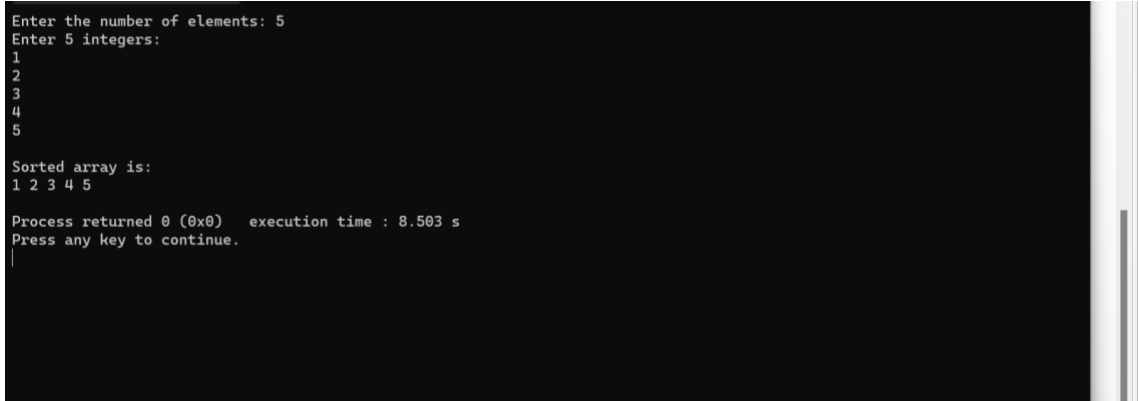
# DAY 2 PROGRAMS

```c
int main() {
    int n, i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d integers:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    mergeSort(arr, 0, n - 1);

    printf("\nSorted array is: \n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

## OUTPUT:

```
Enter the number of elements: 5
Enter 5 integers:
1
2
3
4
5

Sorted array is:
1 2 3 4 5

Process returned 0 (0x0)   execution time : 8.503 s
Press any key to continue.
```

# DAY 2 PROGRAMS

## 2) MINIMUM AND MAXIMUM IN ARRAY

PROGRAM:

```c
#include<stdio.h>
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j)
{
 int max1, min1, mid;
 if(i==j)
 {
  max = min = a[i];
 }
 else
 {
  if(i == j-1)
  {
   if(a[i] <a[j])
   {
    max = a[j];
    min = a[i];
   }
   else
   {
    max = a[i];
    min = a[j];
   }
  }
  else
```

# DAY 2 PROGRAMS

```c
  {
   mid = (i+j)/2;

   maxmin(i, mid);

   max1 = max; min1 = min;

   maxmin(mid+1, j);

   if(max <max1)

    max = max1;

   if(min > min1)

    min = min1;

  }

 }

}

int main ()

{

 int i, num;

 printf ("\nEnter the total number of numbers : ");

 scanf ("%d",&num);

 printf ("Enter the numbers : \n");

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



 max = a[0];

 min = a[0];

 maxmin(1, num);

 printf ("Maximum element in an array : %d\n", max);

 return 0;

}
```

# DAY 2 PROGRAMS

OUTPUT:

```
Enter the total number of numbers : 5
Enter the numbers :
1
3
67
45
9
Minimum element in an array : 1
Maximum element in an array : 67

Process returned 0 (0x0)   execution time : 10.764 s
Press any key to continue.
```

## 3) SUBSETS OF THE GIVEN SET

PROGRAM:

```c
#include <stdio.h>
char string[50], n;
void subset(int, int, int);
 int main()
{
    int i, len;
    printf("Enter the len of main set : ");
    scanf("%d", &len);
    printf("Enter the elements of main set : ");
    scanf("%s", string);
    n = len;
    printf("The subsets are :\n");
    for (i = 1;i <= n;i++)
        subset(0, 0, i);
}
void subset(int start, int index, int num_sub)
{
```

# DAY 2 PROGRAMS

```c
    int i, j;
    if (index - start + 1  ==  num_sub)
    {
        if (num_sub  ==  1)
        {
            for (i = 0;i < n;i++)
                printf("%c\n", string[i]);
        }
        else
        {
            for (j = index;j < n;j++)
            {
                for (i = start;i < index;i++)
                    printf("%c", string[i]);
                printf("%c\n", string[j]);
            }
            if (start != n - num_sub)
                subset(start + 1, start + 1, num_sub);
        }
    }
    else
    {
        subset(start, index + 1, num_sub);


    }

}
```

OUTPUT:

# DAY 2 PROGRAMS

```
Enter the len of main set : 3
Enter the elements of main set : 123
The subsets are :
1
2
3
12
13
23
123

Process returned 0 (0x0)   execution time : 6.922 s
Press any key to continue.
```

## 4) CONTAINER LOADING PROBLEM

PROGRAM:

```c
#include <stdio.h>

#include <stdlib.h>


#define MAX_ITEMS 100

#define MAX_WEIGHT 100


int weight[MAX_ITEMS];

int value[MAX_ITEMS];

int dp[MAX_ITEMS][MAX_WEIGHT];


int max(int a, int b) {

  return (a > b) ? a : b;

}


int knapsack(int n, int w) {

  int i, j;

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

    for (j = 0; j <= w; j++) {

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

        dp[i][j] = 0;
```

```c
    } else if (weight[i-1] <= j) {

      dp[i][j] = max(value[i-1] + dp[i-1][j-weight[i-1]], dp[i-1][j]);

    } else {

      dp[i][j] = dp[i-1][j];

    }

  }

}

  return dp[n][w];

}


int main() {
  int n = 4;
  int w = 10;
  weight[0] = 1;
  weight[1] = 2;
  weight[2] = 4;
  weight[3] = 5;
  value[0] = 5;
  value[1] = 4;
  value[2] = 6;
  value[3] = 8;
  int result = knapsack(n, w);
  printf("Result: %d\n", result);
  return 0;
}
```

OUTPUT:

# DAY 2 PROGRAMS

```
Result: 19

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
```

## 5) MINIMUM SPANNING TREE WITH PRIMS ALGORITHM

## PROGRAM:

```c
#include <stdio.h>

#include <limits.h>


#define V 5


int minKey(int key[], bool mstSet[]) {

    int min = INT_MAX, minIndex;

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

        if (mstSet[v] == false && key[v] < min)

            min = key[v], minIndex = v;

    return minIndex;

}


void printMST(int parent[], int graph[V][V]) {

    printf("Edge \tWeight\n");

    for (int i = 1; i < V; i++)

        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);

}


void primMST(int graph[V][V]) {
```

# DAY 2 PROGRAMS

```
    int parent[V];

    int key[V];

    bool mstSet[V];

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

        key[i] = INT_MAX, 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] && mstSet[v] == false && graph[u][v] < key[v])

                parent[v] = u, key[v] = graph[u][v];

    }

    printMST(parent, graph);

}


int main() {

    int graph[V][V] = {{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}};

    primMST(graph);


}
```

OUTPUT:

# DAY 2 PROGRAMS

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

-------------------------------
Process exited after 0.05377 seconds with return value 0
Press any key to continue . . .
```

## 6)N-QUEENS PROBLEM

### PROGRAM:

#include <stdio.h>

#include <stdbool.h>


#define N 8


int col[N];


bool check(int row) {

 int i;

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

  if (col[i] == col[row] ||

    row - i == col[row] - col[i] ||

    row - i == col[i] - col[row])

   return false;

 return true;

}


void backtrack(int row) {

 int i;

 if (row == N) {

# DAY 2 PROGRAMS

```
    for (i = 0; i < N; i++) printf("(%d, %d)\n", i, col[i]);

    printf("\n");

    return;

  }
  for (i = 0; i < N; i++) {

    col[row] = i;

    if (check(row)) backtrack(row + 1);

  }
}


int main() {

  backtrack(0);

  return 0;

}
```

OUTPUT:



## 7)TRAVELSALES MAN PROBLEM

PROGRAM:

#include <stdio.h>

# DAY 2 PROGRAMS

```c
#include <stdbool.h>

#define MAX 20
#define INF 99999

int n, d[MAX][MAX], x[MAX];
int best_tour_length = INF, tour_length[MAX];

void backtrack(int curr_pos) {
  int i;
  if (curr_pos == n) {
    tour_length[curr_pos] = d[x[n - 1]][x[0]];
    int tour = 0;
    for (i = 0; i < n; i++) tour += tour_length[i];
    if (tour < best_tour_length) best_tour_length = tour;
    return;
  }
  for (i = 0; i < n; i++) {
    if (x[i] == -1) {
      x[i] = curr_pos;
      tour_length[curr_pos] = d[x[curr_pos - 1]][i];
      backtrack(curr_pos + 1);
      x[i] = -1;
    }
  }
}

int main() {
  int i, j;
```
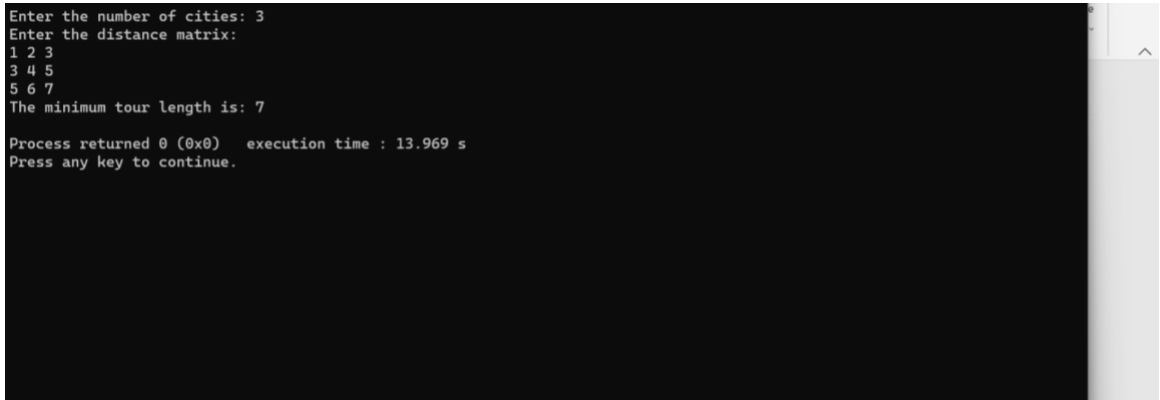
# DAY 2 PROGRAMS

```c
printf("Enter the number of cities: ");

scanf("%d", &n);

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

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

  for (j = 0; j < n; j++) {

    scanf("%d", &d[i][j]);

    x[i] = -1;

  }

x[0] = 0;

backtrack(1);

printf("The minimum tour length is: %d\n", best_tour_length);

return 0;

}
```

## OUTPUT:



```
Enter the number of cities: 3
Enter the distance matrix:
1 2 3
3 4 5
5 6 7
The minimum tour length is: 7

Process returned 0 (0x0)   execution time : 13.969 s
Press any key to continue.
```

## 8) KNAPSACK USING DYNAMIC PROGRAMMING

## PROGRAM:

```c
#include <stdio.h>

#include <stdlib.h>


#define MAX_WEIGHT 100
```

# DAY 2 PROGRAMS

```c
// Struct to represent an item
struct Item {
  int weight;
  int profit;
};


// Function to find the maximum profit for the knapsack
int knapsack(struct Item items[], int n, int weight) {
  int dp[n + 1][weight + 1];


  for (int i = 0; i <= n; i++) {
    for (int w = 0; w <= weight; w++) {
      if (i == 0 || w == 0) {
        dp[i][w] = 0;
      } else if (items[i - 1].weight <= w) {
        dp[i][w] =
            fmax(dp[i - 1][w], dp[i - 1][w - items[i - 1].weight] + items[i - 1].profit);
      } else {
        dp[i][w] = dp[i - 1][w];
      }
    }
  }


  return dp[n][weight];
}


int main() {
  struct Item items[] = {{40, 80}, {30, 70}, {20, 50}, {30, 80}};
  int n = sizeof(items) / sizeof(items[0]);
```
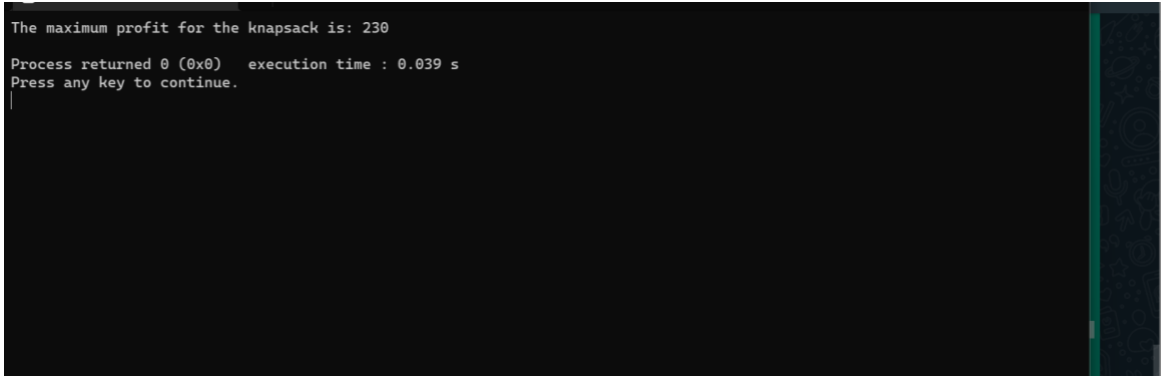
# DAY 2 PROGRAMS

```c
  int weight = MAX_WEIGHT;

  printf("The maximum profit for the knapsack is: %d\n", knapsack(items, n, weight));

  return 0;
}
```

## OUTPUT:

```
The maximum profit for the knapsack is: 230

Process returned 0 (0x0)   execution time : 0.039 s
Press any key to continue.
```

## 9)OPTIMAL BINARY SEARCH TREE

### PROGRAM:

```c
// A naive recursive implementation of optimal binary
// search tree problem
#include <stdio.h>
#include <limits.h>

// A utility function to get sum of array elements
// freq[i] to freq[j]
int sum(int freq[], int i, int j);

// A recursive function to calculate cost of optimal
```

# DAY 2 PROGRAMS

```
// binary search tree

int optCost(int freq[], int i, int j)

{

// Base cases

if (j < i)  // no elements in this subarray

        return 0;

if (j == i)         // one element in this subarray

        return freq[i];


// Get sum of freq[i], freq[i+1], ... freq[j]

int fsum = sum(freq, i, j);


// Initialize minimum value

int min = INT_MAX;


// One by one consider all elements as root and

// recursively find cost of the BST, compare the

// cost with min and update min if needed

for (int r = i; r <= j; ++r)

{

        int cost = optCost(freq, i, r-1) +

                                optCost(freq, r+1, j);

        if (cost < min)

                min = cost;

}


// Return minimum value

return min + fsum;

}
```

# DAY 2 PROGRAMS

```
// The main function that calculates minimum cost of

// a Binary Search Tree. It mainly uses optCost() to

// find the optimal cost.

int optimalSearchTree(int keys[], int freq[], int n)

{

        // Here array keys[] is assumed to be sorted in

        // increasing order. If keys[] is not sorted, then

        // add code to sort keys, and rearrange freq[]

        // accordingly.

        return optCost(freq, 0, n-1);

}


// A utility function to get sum of array elements

// freq[i] to freq[j]

int sum(int freq[], int i, int j)

{

        int s = 0;

        for (int k = i; k <=j; k++)

        s += freq[k];

        return s;

}


// Driver program to test above functions

int main()

{

        int keys[] = {10, 12, 20};

        int freq[] = {34, 8, 50};

        int n = sizeof(keys)/sizeof(keys[0]);
```
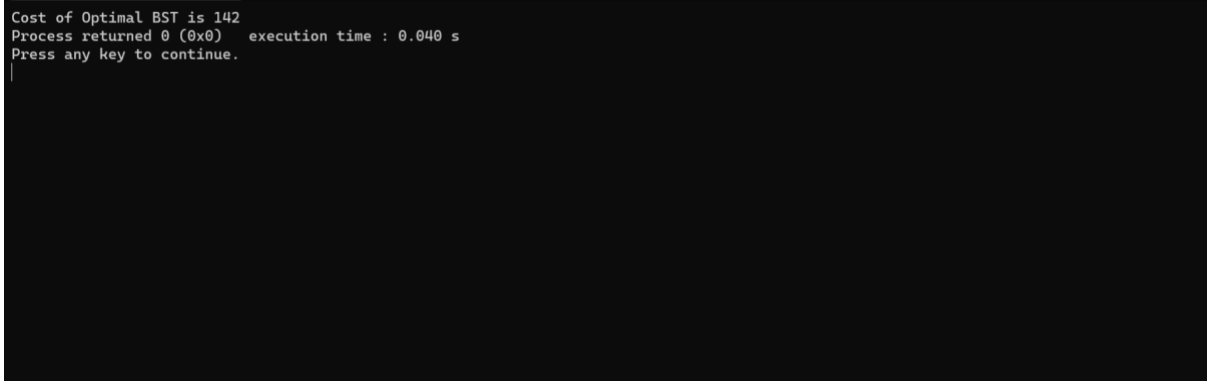
# DAY 2 PROGRAMS

```
        printf("Cost of Optimal BST is %d ",

                        optimalSearchTree(keys, freq, n));

        return 0;

}
```

## OUTPUT:

```
Cost of Optimal BST is 142
Process returned 0 (0x0)    execution time : 0.040 s
Press any key to continue.
```

## 10)SUM OF SUBSETS USING BACK TRACKING

## PROGRAM:

```c
#include <stdio.h>
#include <stdlib.h>
static int total_nodes;
void printValues(int A[], int size){
  for (int i = 0; i < size; i++) {
    printf("%*d", 5, A[i]);
  }
  printf("\n");
}
void subset_sum(int s[], int t[], int s_size, int t_size, int sum, int ite, int const target_sum){
  total_nodes++;
  if (target_sum == sum) {
    printValues(t, t_size);
```

# DAY 2 PROGRAMS

```c
        subset_sum(s, t, s_size, t_size - 1, sum - s[ite], ite + 1, target_sum);

        return;

    }

    else {

      for (int i = ite; i < s_size; i++) {

        t[t_size] = s[i];

        subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);

      }

    }

}

void generateSubsets(int s[], int size, int target_sum){

  int* tuplet_vector = (int*)malloc(size * sizeof(int));

  subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);

  free(tuplet_vector);

}

int main(){

  int set[] = { 5, 6, 12 , 54, 2 , 20 , 15 };

  int size = sizeof(set) / sizeof(set[0]);

  printf("The set is ");

  printValues(set , size);

  generateSubsets(set, size, 25);

  printf("Total Nodes generated %d\n", total_nodes);

  return 0;

}
```

## OUTPUT:

# DAY 2 PROGRAMS

```
The set is    5   6   12   54    2   20   15
        5    6   12    2
        5   20
Total Nodes generated 127

Process returned 0 (0x0)   execution time : 0.039 s
Press any key to continue.
```

## 11)MINIMUM SPANNING TREE USING GREEDY TECHNIQUES

## PROGRAM:

```c
#include <stdio.h>

#include <limits.h>


#define V 5


int minKey(int key[], int mstSet[]) {

    int min = INT_MAX, min_index;

    int v;

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

        if (mstSet[v] == 0 && key[v] < min)

            min = key[v], min_index = v;


    return min_index;

}


int printMST(int parent[], int n, int graph[V][V]) {

    int i;

    printf("Edge   Weight\n");

    for (i = 1; i < V; i++)
```

```c
        printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);
}


void primMST(int graph[V][V]) {
    int parent[V]; // Array to store constructed MST
    int key[V], i, v, count; // Key values used to pick minimum weight edge in cut
    int mstSet[V]; // To represent set of vertices not yet included in MST

    // Initialize all keys as INFINITE
    for (i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = 0;

    // Always include first 1st vertex in MST.
    key[0] = 0; // Make key 0 so that this vertex is picked as first vertex
    parent[0] = -1; // First node is always root of MST

    // The MST will have V vertices
    for (count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = 1;

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

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

    // print the constructed MST
    printMST(parent, V, graph);
```

# DAY 2 PROGRAMS

```
}

int main() {
    /* Let us create the following graph

      2   3
     (0)--(1)--(2)
     |  /\  |
    6| 8/  \5 |7
     |/    \|
     (3)-------(4)
      9       */
    int graph[V][V] = { { 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 }, };

    primMST(graph);

    return 0;
}
```

## OUTPUT:

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

Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

## 12) Mth MAXIMUM AND Nth MINIMUM

## PROGRAM:

```
#include <stdio.h>
```

# DAY 2 PROGRAMS

```c
int main ()

{
        int number[30],rev[30];
        int i, j, a, n,m,nt;
        printf("Enter the value of N\n");

    scanf("%d", &n);



    printf("Enter the numbers \n");

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

            scanf("%d", &number[i]);



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

    {

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

      {

        if (number[i] < number[j])
```

# DAY 2 PROGRAMS

```c
        {

            a = number[i];

            number[i] = number[j];

            number[j] = a;

        }

    }

}

for(i=(n-1),j=0; i>=0; i--,j++)
    {
            rev[j]=number[i];

    }
    printf("\nEnter the Mth maximum =");
    scanf("%d",&m);

    printf("\n%d th maximum number is %d",m,number[m-1]);
    printf("\nEnter the Nth minimum =");
    scanf("%d",&nt);

    printf("\n%d th minimum number is %d",nt,rev[nt-1]);
    return 0; }
```

# DAY 2 PROGRAMS

OUTPUT:

```
Enter the value of N
4
Enter the numbers
23
34
56
7

Enter the Mth maximum =2

2 th maximum number is 34
Enter the Nth minimum =1

1 th minimum number is 7
Process returned 0 (0x0)   execution time : 21.918 s
Press any key to continue.
```