1. **Implementation of Selection Sort Algorithm.**

→

#include<stdio.h>

int i, j, temp, a[50], size, min;

void selection\_sort()

{

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

{

min=i;

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

{

if(a[i]>a[j])

{

min=j;

}

}

if(min!=i)

{

temp=a[j];

a[j]=a[i];

a[i]=temp;

}

}

}

void main()

{

printf("Enter size of array : ");

scanf("%d",&size);

printf("Enter array elements : \n");

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

{

printf("a[%d]:",i);

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

}

selection\_sort();

printf("Array elements after selection sort : \n");

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

{

printf("a[%d]:%d\n",i,a[i]);

}

}

**o/p :**

Enter size of array : 4

Enter array elements :

a[0]:34

a[1]:55

a[2]:12

a[3]:35

Array elements after selection sort :

a[0]:12

a[1]:34

a[2]:35

a[3]:55

1. **Implementation of Insertion Sort Algorithm.**

→

#include<stdio.h>

int i, j, temp, a[50], size, x;

void insertion\_sort()

{

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

{

j=i-1;

x=a[i];

while(j>-1&&a[j]>x)

{

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

j--;

}

a[j+1]=x;

}

}

void main()

{

printf("Enter size of array : ");

scanf("%d",&size);

printf("Enter array elements : \n");

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

{

printf("a[%d]:",i);

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

}

insertion\_sort();

printf("Array elements after insertion sort : \n");

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

{

printf("a[%d]:%d\n",i,a[i]);

}

}

**o/p :**

Enter size of array : 5

Enter array elements :

a[0]:12

a[1]:6

a[2]:8

a[3]:3

a[4]:20

Array elements after insertion sort :

a[0]:3

a[1]:6

a[2]:8

a[3]:12

a[4]:20

**3. Program to implement heapsort.**

→

#include <stdio.h>

void Heapify (int a[], int n)

{

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

{

Adjust(a,i,n);

}

}

void Adjust (int a[], int i, int n)

{

int j, item;

j = 2\*i;

item = a[i];

while(j<=n)

{

if(j<n && a[j]<a[j+1])

{

j = j+1;

}

if(item>a[j])

{

break;

}

a[j/2] = a[j];

j = 2\*j;

}

a[j/2] = item;

}

void HeapSort (int a[], int n)

{

Heapify(a,n);

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

{

int temp = a[i];

a[i]=a[1];

a[1]=temp;

Adjust(a,1,i-1);

}

}

void main()

{

int a[10],i,n;

printf("Enter size of array : ");

scanf("%d",&n);

printf("Enter array elements : \n");

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

{

printf("a[%d]:",i);

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

}

HeapSort(a,n);

printf("Array elements after Heap Sort : \n");

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

{

printf("a[%d]:%d\n",i,a[i]);

}

}

**o/p :**

Enter size of array : 4

Enter array elements :

a[1]:5

a[2]:2

a[3]:12

a[4]:4

Array elements after Heap Sort :

a[1]:2

a[2]:4

a[3]:5

a[4]:12

**4. . Implementation of Binary search algorithm using Divide & Conquer method.**

→

**Iterative :**

#include <stdio.h>

int main()

{

int i, low, high, mid, n, search, a[10];

printf("Enter number of elements : ");

scanf("%d",&n);

printf("Enter %d elements : \n", n);

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

{

printf("a[%d] : ",i);

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

}

printf("Enter value to find : ");

{

scanf("%d", &search);

}

low = 1;

high = n;

mid = (low+high)/2;

while (low <= high)

{

if (a[mid] < search)

{

low = mid + 1;

}

else if (a[mid] == search)

{

printf("%d found at location %d.\n", search, mid);

break;

}

else

{

high = mid - 1;

}

mid = (low + high)/2;

}

if (low > high)

{

printf("Not found! %d isn't present in the list.\n", search);

}

return 0;

}

**o/p :**

Enter number of elements : 5

Enter 5 elements :

a[1] : 1

a[2] : 2

a[3] : 3

a[4] : 4

a[5] : 5

Enter value to find : 3

3 found at location 3.

**Recursive :**

#include <stdio.h>

int Binary\_Search (int a[], int i, int low, int x)

{

int mid;

if (low == i)

{

if (x = a[i])

{

return 1;

}

else

{

return 0;

}

}

else

{

mid = (i + low)/2;

if (x = a[mid])

{

return mid+1;

}

else if (x < a[mid])

{

Binary\_Search(a, i, mid - 1, x);

}

else

{

Binary\_Search(a, mid + 1, low, x);

}

}

}

void main()

{

int i, low, high, mid, n, x, a[10];

printf("Enter number of elements : ");

scanf("%d",&n);

printf("Enter %d elements : \n", n);

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

{

printf("a[%d] : ",i);

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

}

printf("Enter value to find : ");

{

scanf("%d", &x);

}

int found\_index = Binary\_Search(a, i, low, x);

if(found\_index == -1 )

{

printf("%d not found in the array",x);

}

else

{

printf("%d is found at index : %d",x,found\_index);

}

}

**o/p :**

Enter number of elements : 5

Enter 5 elements :

a[0] : 1

a[1] : 2

a[2] : 3

a[3] : 4

a[4] : 5

Enter value to find : 4

4 is found at index : 3

**5. Program to Find Max and Min.**

→

#include <stdio.h>

int min, max, a[10];

void MinMax(int i, int j)

{

int max1, min1, mid;

if(i==j)

{

min=max=a[i];

}

else if(i=j-1)

{

if(a[i]<a[j])

{

min=a[i];

max=a[j];

}

else

{

min=a[j];

max=a[i];

}

}

else

{

mid=(i+j)/2;

MinMax(1,mid);

max1=max;

min1=min;

MinMax(mid+1,j);

if(max<max1)

{

max=max1;

}

if(min>min1)

{

min=min1;

}

}

}

void main()

{

int i, n;

printf("Enter array's size : ");

scanf("%d",&n);

printf("Enter %d array elements :\n",n);

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

{

printf("a[%d]:",i);

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

}

min = a[1];

max = a[1];

MinMax(1,n);

printf("Minimum element of array : %d\n",min);

printf("Maximum element of array : %d",max);

}

**o/p :**

Enter array's size : 3

Enter 3 array elements :

a[1]:2

a[2]:3

a[3]:1

Minimum element of array : 1

Maximum element of array : 3

**6. Implementation of Merge Sort algorithm using Divide & Conquer method.**

→

#include<stdio.h>

void Merge\_Sort(int a[], int low, int high)

{

int mid;

if(low < high)

{

mid = (low + high) / 2;

Merge\_Sort(a, low, mid);

Merge\_Sort(a, mid+1, high);

Merge(a, low, mid, high);

}

}

void Merge(int a[], int low, int mid, int high)

{

int h = low, i = low, j = mid + 1, b[10], k;

while(h <= mid && j <= high)

{

if(a[h] <= a[j])

{

b[i] = a[h];

h = h + 1;

}

else

{

b[i] = a[j];

j = j + 1;

}

i = i + 1;

}

if(h > mid)

{

for(k = j; k <= high; k++)

{

b[i] = a[k];

i = i + 1;

}

}

else

{

for(k = h; k <= mid; k++)

{

b[i] = a[k];

i = i + 1;

}

}

for(k = low; k <= high; k++)

{

a[k] = b[k];

}

}

void main()

{

int a[10],i,n;

printf("Enter size of array : ");

scanf("%d",&n);

int low = 1, high = n;

printf("Enter array elements : \n");

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

{

printf("a[%d]:",i);

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

}

Merge\_Sort(a, low, high);

printf("Array elements after Merge Sort : \n");

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

{

printf("a[%d]:%d\n",i,a[i]);

}

}

**o/p :**

Enter size of array : 5

Enter array elements :

a[1]:2

a[2]:4

a[3]:1

a[4]:5

a[5]:3

Array elements after Merge Sort :

a[1]:1

a[2]:2

a[3]:3

a[4]:4

a[5]:5

**7. Implementation of Quick Sort algorithm using Divide & Conquer method.**

→

#include <stdio.h>

int partition(int a[],int low,int high)

{

int pivot,i,j,t,y;

pivot=a[low];

i=low;

j=high;

while(i<j)

{

while(a[i]<pivot && i<=high)

{

i++;

}

while(a[j]>pivot)

{

j--;

}

if(i<j)

{

t=a[i];

a[i]=a[j];

a[j]=t;

}

}

y=a[j];

a[j]=pivot;

pivot=y;

return j;

}

int quicksort(int a[],int low,int high)

{

if(low<high)

{

int z;

z=partition(a,low,high);

quicksort(a,low,z-1);

quicksort(a,z+1,high);

}

}

int main()

{

int low,high,arr[30],k,size;

printf("Enter size of the array: ");

scanf("%d",&size);

printf("Enter array elements: \n");

for(k=0;k<size;k++)

{

printf("arr[%d]: ",k);

scanf("%d",&arr[k]);

}

low=0;

high=size-1;

quicksort(arr,low,high);

printf("Array elements after Quick Sort: \n");

for(k=0;k<size;k++)

{

printf("arr[%d]: %d\n",k,arr[k]);

}

return 0;

}

**o/p :**

Enter size of the array: 4

Enter array elements:

arr[0]: 9

arr[1]: 4

arr[2]: 1

arr[3]: 7

Array elements after Quick Sort:

arr[0]: 1

arr[1]: 4

arr[2]: 7

arr[3]: 9

**8. Program to implement knapsack problem using greedy method.**

→

#include <stdio.h>

void Knapsack (int n, int m, float w[], float p[])

{

int i,j,u;

float x[20],sum=0;

u=m;

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

{

x[i]=0.0;

}

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

{

if(w[i]>u)

{

break;

}

else

{

x[i]=1;

sum=sum+p[i];

u=u-w[i];

}

}

if(i<n)

{

x[i]=u/w[i];

sum=sum+(x[i]\*p[i]);

}

printf("Result : ");

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

{

printf("\tx[%d] : %f\t",i,x[i]);

}

printf("\nMaximum profit is : %f",sum);

}

void main()

{

int i,j,n,m;

float w[20],p[20],ratio[20],temp;

printf("Enter no. of objects : ");

scanf("%d",&n);

printf("Enter the weights and profits of all the objects :");

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

{

printf("\nObject %d : ",i);

printf("\n\tProfit : ");

scanf("%f",&p[i]);

printf("\n\tWeight : ");

scanf("%f",&w[i]);

ratio[i]=p[i]/w[i];

printf("Ratio : %f",i,ratio[i]);

}

printf("\nEnter capacity of Knapsack : ");

scanf("%d",&m);

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

{

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

{

if(ratio[i]<ratio[j])

{

temp=ratio[j];

ratio[j]=ratio[i];

ratio[i]=temp;

temp=w[j];

w[j]=w[i];

w[i]=temp;

temp=p[j];

p[j]=p[i];

p[i]=temp;

}

}

}

Knapsack(n,m,w,p);

}

**o/p :**

Enter no. of objects : 3

Enter the weights and profits of all the objects :

Object 1 :

Profit : 25

Weight : 18

Ratio : 1.388889

Object 2 :

Profit : 24

Weight : 15

Ratio : 1.600000

Object 3 :

Profit : 15

Weight : 10

Ratio : 1.500000

Enter capacity of Knapsack : 20

Result : x[1] : 1.000000 x[2] : 0.500000 x[3] : 0.000000

Maximum profit is : 31.500000

**9. Program to implement job sequencing with deadlines using greedy methods.**

→

#include <iostream>

#include <algorithm>

using namespace std;

typedef struct

{

char id;

int deadline;

int profit;

}Job;

bool compareJob(Job a,Job b)

{

return a.profit>b.profit;

}

void bestJob(Job jobs[],int sizeOfJobs)

{

char jobsToDo[5]{'\0'};

int k;

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

{

k=jobs[i].deadline-1;

while(jobsToDo[k]!='\0'&&k>=0)

{

k--;

}

if(k!=-1)

{

jobsToDo[k]=jobs[i].id;

}

}

cout<<"Best order and jobs to do is \n";

k=0;

while(jobsToDo[k]!='\0')

{

cout<<jobsToDo[k]<<" ";

k++;

}

}

void display(Job jobs[],int n)

{

int i;

cout<<"Job Id: \t";

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

{

cout<<jobs[i].id<<"\t";

}

cout<<endl;

cout<<"Job Deadline: \t";

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

{

cout<<jobs[i].deadline<<"\t";

}

cout<<endl;

cout<<"Job Profit: \t";

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

{

cout<<jobs[i].profit<<"\t";

}

cout<<endl;

}

int main()

{

Job jobs[]= {

{'w',1,19},{'v',2,100},{'x',2,27},{'y',1,25},{'z',3,15}

};

display(jobs,5);

sort(jobs,jobs+5,compareJob);

bestJob(jobs,5);

return 0;

}

**o/p :**

Job Id: w v x y z

Job Deadline: 1 2 2 1 3

Job Profit: 19 100 27 25 15

Best order and jobs to do is

x v z

#include <stdio.h>

#include <stdlib.h>

typedef struct

{

char id;

int deadline;

int profit;

} Job;

void bestJob(Job jobs[], int sizeOfJobs)

{

char jobsToDo[5] = {'\0'};

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

{

int k = jobs[i].deadline - 1;

while(jobsToDo[k] != '\0' && k >= 0) k--;

if(k != -1) jobsToDo[k] = jobs[i].id;

}

printf("Best order and jobs to do is:\n");

for(int i = 0; jobsToDo[i] != '\0'; i++) printf("%c ", jobsToDo[i]);

}

int main() {

Job jobs[] = {

{'w', 1, 19},

{'v', 2, 100},

{'x', 2, 27},

{'y', 1, 25},

{'z', 3, 15}

};

int sizeOfJobs = sizeof(jobs) / sizeof(Job);

printf("Job Id:\t\tJob Deadline:\tJob Profit:\n");

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

printf("%c\t\t%d\t\t%d\n", jobs[i].id, jobs[i].deadline, jobs[i].profit);

bestJob(jobs, sizeOfJobs);

return 0;

}

**o/p:**

Job Id: Job Deadline: Job Profit:

w 1 19

v 2 100

x 2 27

y 1 25

z 3 15

Best order and jobs to do is:

x v z

#include <stdio.h>

#include <stdlib.h>

typedef struct

{

char id;

int deadline;

int profit;

} Job;

void bestJob(char ids[], int deadlines[], int profits[], int sizeOfJobs)

{

char jobsToDo[5] = {'\0'};

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

{

int k = deadlines[i] - 1;

while(jobsToDo[k] != '\0' && k >= 0) k--;

if(k != -1) jobsToDo[k] = ids[i];

}

printf("Best order and jobs to do is:\n");

for(int i = 0; jobsToDo[i] != '\0'; i++) printf("%c ", jobsToDo[i]);

}

int main()

{

char ids[] = {'w', 'v', 'x', 'y', 'z'};

int deadlines[] = {1, 2, 2, 1, 3};

int profits[] = {19, 100, 27, 25, 15};

int sizeOfJobs = sizeof(ids) / sizeof(char);

printf("Job Id:\t\tJob Deadline:\tJob Profit:\n");

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

printf("%c\t\t%d\t\t%d\n", ids[i], deadlines[i], profits[i]);

bestJob(ids, deadlines, profits, sizeOfJobs);

return 0;

}

**o/p :**

Job Id: Job Deadline: Job Profit:

w 1 19

v 2 100

x 2 27

y 1 25

z 3 15

Best order and jobs to do is:

x v z

#include<stdio.h>

#define MAX\_JOBS 100

// Job structure to store the job details

struct job

{

int id; // job ID

int deadline; // job deadline

int profit; // job profit

};

// Function to sort jobs based on profit

void sort\_jobs(struct job arr[], int n)

{

int i, j;

struct job temp;

for(i=0; i<n-1; i++)

{

for(j=0; j<n-i-1; j++)

{

if(arr[j].profit < arr[j+1].profit)

{

temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

// Function to find the maximum deadline in the jobs array

int find\_max\_deadline(struct job arr[], int n)

{

int i, max = arr[0].deadline;

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

{

if(arr[i].deadline > max)

{

max = arr[i].deadline;

}

}

return max;

}

// Function to perform job sequencing and find the maximum profit

void job\_sequencing(struct job arr[], int n)

{

// Sort jobs based on profit in descending order

sort\_jobs(arr, n);

// Find the maximum deadline

int max\_deadline = find\_max\_deadline(arr, n);

// Array to store the final sequence of jobs

int job\_seq[MAX\_JOBS] = {0};

// Iterate over each job in the array

int i, j, k;

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

{

// Find the last empty slot in the array before the job deadline

j = arr[i].deadline - 1;

while(j>=0 && job\_seq[j] != 0)

{

j--;

}

// If an empty slot is found, add the job to that slot

if(j>=0) {

job\_seq[j] = arr[i].id;

}

}

// Calculate the maximum profit and print the job sequence

int max\_profit = 0;

printf("Job sequence for maximum profit: ");

for(k=0; k<max\_deadline; k++)

{

if(job\_seq[k] != 0)

{

max\_profit += arr[job\_seq[k]-1].profit;

printf("%d ", job\_seq[k]);

}

}

printf("\nMaximum profit: %d\n", max\_profit);

}

int main()

{

// Input the number of jobs

int n;

printf("Enter the number of jobs: ");

scanf("%d", &n);

// Input the job details

struct job arr[MAX\_JOBS];

int i;

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

{

printf("Enter the details for job %d:\n", i+1);

printf("ID: ");

scanf("%d", &arr[i].id);

printf("Profit: ");

scanf("%d", &arr[i].profit);

printf("Deadline: ");

scanf("%d", &arr[i].deadline);

}

// Perform job sequencing and find the maximum profit

job\_sequencing(arr, n);

return 0;

}

**o/p:**

Enter the number of jobs: 5

Enter the details for job 1:

ID: J1

Profit: 20

Deadline: 2

Enter the details for job 2:

ID: J2

Profit: 15

Deadline: 2

Enter the details for job 3:

ID: J3

Profit: 10

Deadline: 1

Enter the details for job 4:

ID: J4

Profit: 5

Deadline: 3

Enter the details for job 5:

ID: J5

Profit: 1

Deadline: 3

Job sequence for maximum profit: J2 J1 J4

Maximum profit: 40

**10. Program for finding shortest path for multistage graph using dynamic programming.**

→

#include <stdio.h>

#define MAX\_N 100

#define MAX\_M 100

int graph[MAX\_N][MAX\_M];

int dp[MAX\_N];

int min(int a, int b) {

return a < b ? a : b;

}

int shortestPath(int n, int m) {

dp[n - 1] = 0; // Initialize the last stage

for(int i = n - 2; i >= 0; i--) { // Traverse the stages in reverse order

dp[i] = -1; // Initialize the minimum distance to -1 (indicating unreachable)

for(int j = i; j < m; j++) { // Traverse the nodes in the current stage

if(graph[i][j] != 0 && dp[j] != -1) {

int distance = graph[i][j] + dp[j];

dp[i] = (dp[i] == -1) ? distance : min(dp[i], distance); // Update the minimum distance

}

}

}

return dp[0];

}

int main() {

int n, m;

printf("Enter the number of stages: ");

scanf("%d", &n);

printf("Enter the number of nodes in each stage: ");

scanf("%d", &m);

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

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

for(int j = 0; j < m; j++) {

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

}

}

int shortest = shortestPath(n, m);

printf("The length of the shortest path is %d\n", shortest);

return 0;

}

**o/p:**

Enter the number of stages: 4

Enter the number of nodes in each stage: 5

Enter the graph matrix:

0 1 2 0 0

0 0 0 2 3

0 0 0 6 0

0 0 0 0 1

The length of the shortest path is 6

#include <stdio.h>

void multistage(int G[][100], int k, int n, int p[], float \*cost) {

int d[100], r;

cost[n] = 0.0;

for (int j = n - 1; j >= 1; j--) {

// Compute cost[j]

r = -1;

float min\_cost = \_\_FLT\_MAX\_\_;

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

if (G[j][i] != 0 && G[j][i] + cost[i] < min\_cost) {

min\_cost = G[j][i] + cost[i];

r = i;

}

}

cost[j] = min\_cost;

d[j] = r;

}

// Find a minimum-cost path

p[0] = 1;

p[k] = n;

for (int j = 1; j < k; j++) {

p[j] = d[p[j - 1]];

}

}

int main() {

int G[100][100], n, k, p[100];

float cost[100];

printf("Enter the number of vertices: ");

scanf("%d", &n);

printf("Enter the number of stages: ");

scanf("%d", &k);

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

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

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

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

}

}

multistage(G, k, n, p, cost);

printf("The shortest path is:\n");

for (int i = 0; i < k; i++) {

printf("%d ", p[i]);

}

printf("\n");

printf("The cost of the shortest path is: %.2f\n", cost[1]);

return 0;

}

**o/p:**

Enter the number of vertices: 4

Enter the number of stages: 3

Enter the adjacency matrix:

0 2 1 0

0 0 0 3

0 0 0 1

0 0 0 0

The shortest path is:

1 3 4

The cost of the shortest path is: 2.00

#include <stdio.h>

#include <limits.h>

#define MAX\_VERTICES 4

int G[MAX\_VERTICES][MAX\_VERTICES];

int ver, stage;

int path[MAX\_VERTICES];

int cost[MAX\_VERTICES];

void multistage() {

cost[ver] = 0;

for (int j = ver - 1; j >= 1; j--) {

int min\_cost = INT\_MAX;

int r = -1;

for (int i = j + 1; i <= ver; i++) {

if (G[j][i] != 0 && G[j][i] + cost[i] < min\_cost) {

min\_cost = G[j][i] + cost[i];

r = i;

}

}

cost[j] = min\_cost;

path[j] = r;

}

path[0] = 1;

path[stage] = ver;

}

int main() {

ver = 4; // Set the number of vertices to 4

stage = 3; // Example: Set the number of stages

int adjMatrix[4][4] = {

{0, 1, 2, 0},

{0, 0, 0, 3},

{0, 0, 0, 1},

{0, 0, 0, 0}

};

for (int i = 0; i < ver; i++) {

for (int j = 0; j < ver; j++) {

G[i][j] = adjMatrix[i][j];

}

}

multistage();

printf("The shortest path is:\n");

for (int i = 0; i < stage; i++) {

printf("%d ", path[i]);

}

printf("\n");

printf("The cost of the shortest path is: %d\n", cost[1]);

return 0;

}

**o/p:**

The shortest path is:

1 3 4

The cost of the shortest path is: 3

**11. Program to implement 8-queens problem using backtrack method.**

→

**Extra :**

1. **Implementation of Bubble Sort Algorithm.**

→

#include<stdio.h>

int i, j, temp, a[50], size;

void bubble\_sort()

{

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

{

for(j=0;j<size-1;j++)

{

if(a[j]>a[j+1])

{

temp=a[j];

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

a[j+1]=temp;

}

}

}

}

void main()

{

printf("Enter size of array : ");

scanf("%d",&size);

printf("Enter array elements : \n");

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

{

printf("a[%d]:",i);

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

}

bubble\_sort();

printf("Array elements after bubble sort : \n");

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

{

printf("a[%d]:%d\n",i,a[i]);

}

}

**o/p :**

Enter size of array : 4

Enter array elements :

a[0]:34

a[1]:55

a[2]:12

a[3]:35

Array elements after bubble sort :

a[0]:12

a[1]:34

a[2]:35

a[3]:55

1. **Find and delete the duplicate elements from an array.**

→

#include<stdio.h>

int i, j, a[50], size;

void remove\_duplicate()

{

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

{

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

{

if(a[i]==a[j])

{

a[i]=a[i+1];

size--;

}

}

}

}

void main()

{

printf("Enter size of array : ");

scanf("%d",&size);

printf("Enter array elements : \n");

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

{

printf("a[%d]:",i);

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

}

remove\_duplicate();

printf("Array after removing duplicate elements : \n");

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

{

printf("a[%d]:%d\n",i,a[i]);

}

}

**o/p :**

Enter size of array : 4

Enter array elements :

a[0]:5

a[1]:1

a[2]:5

a[3]:7

Array after removing duplicate elements :

a[0]:5

a[1]:1

a[2]:7

1. **Find and print the smallest and largest element of an array.**

→

#include<stdio.h>

int i, a[50], size, min, max;

void min\_max()

{

min=a[0];

max=a[0];

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

{

if(min>a[i])

{

min=a[i];

}

if(max<a[i])

{

max=a[i];

}

}

printf("\nMinimum element of array is : %d",min);

printf("\nMaximum element of array is : %d",max);

}

void main()

{

printf("Enter size of array : ");

scanf("%d",&size);

printf("Enter array elements : \n");

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

{

printf("a[%d]:",i);

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

}

min\_max();

}

**o/p :**

Enter size of array : 4

Enter array elements :

a[0]:5

a[1]:2

a[2]:7

a[3]:3

Minimum element of array is : 2

Maximum element of array is : 7

**4. Implementation of Optimal merge patterns.**

→