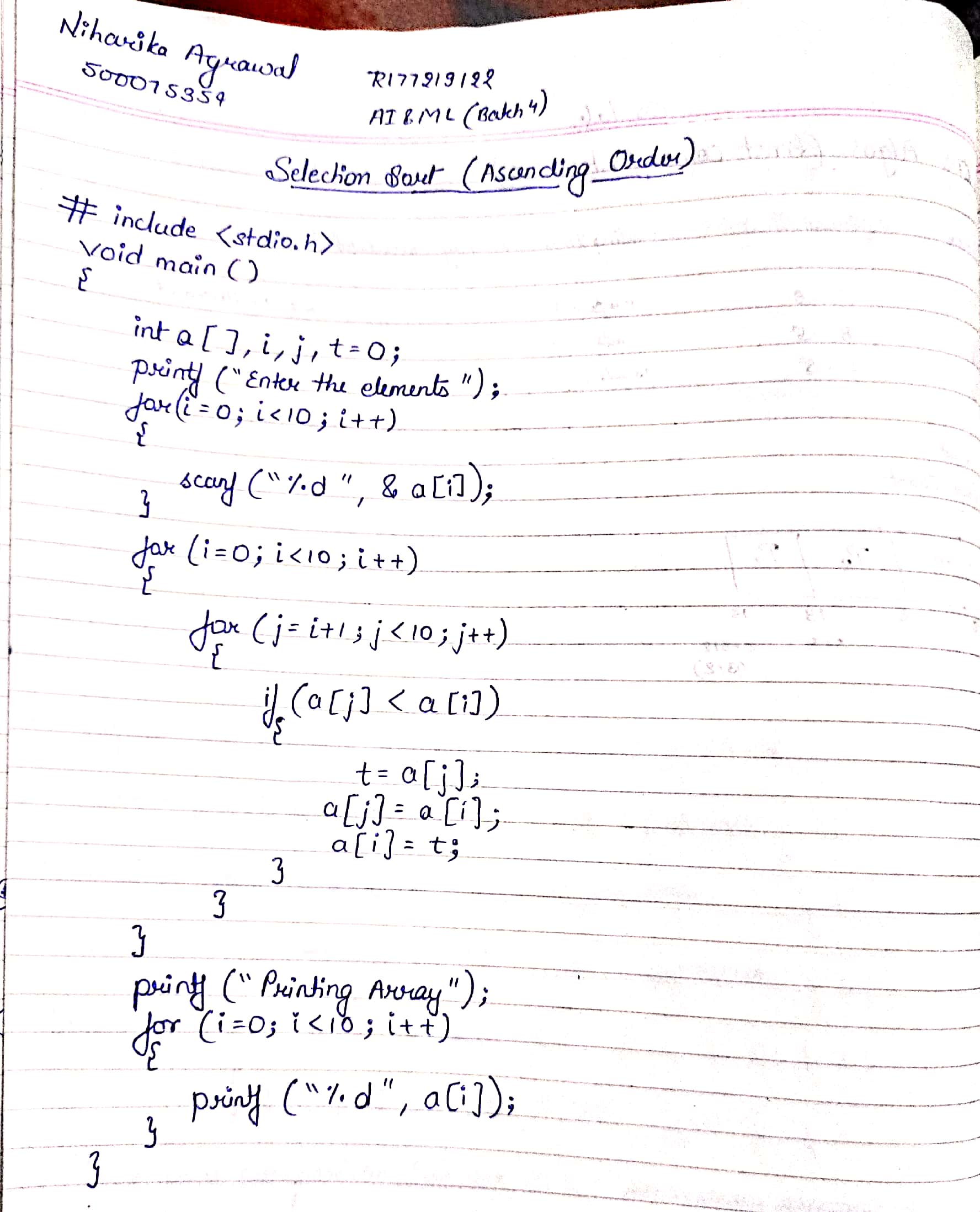
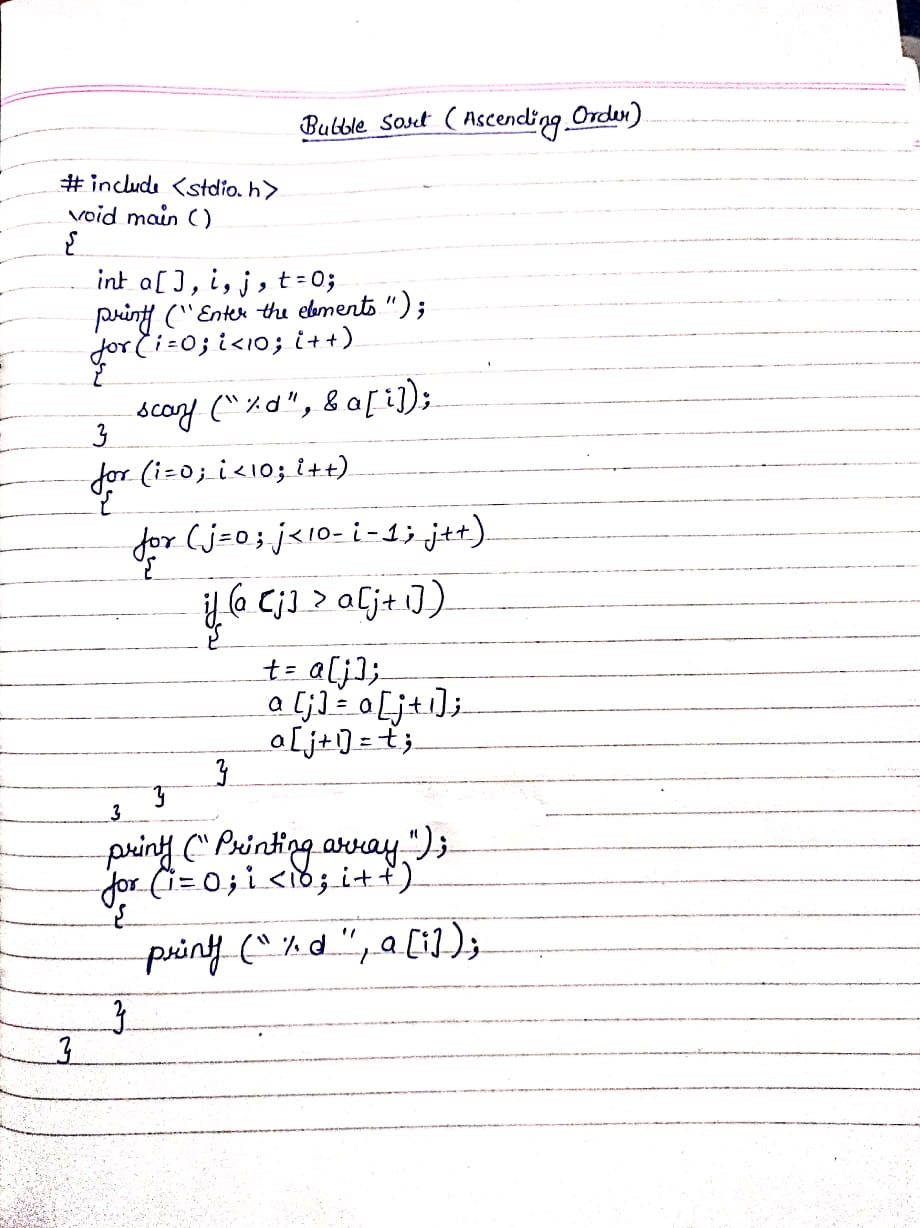
# **LAB RECORDS**

**SELECTION SORT**

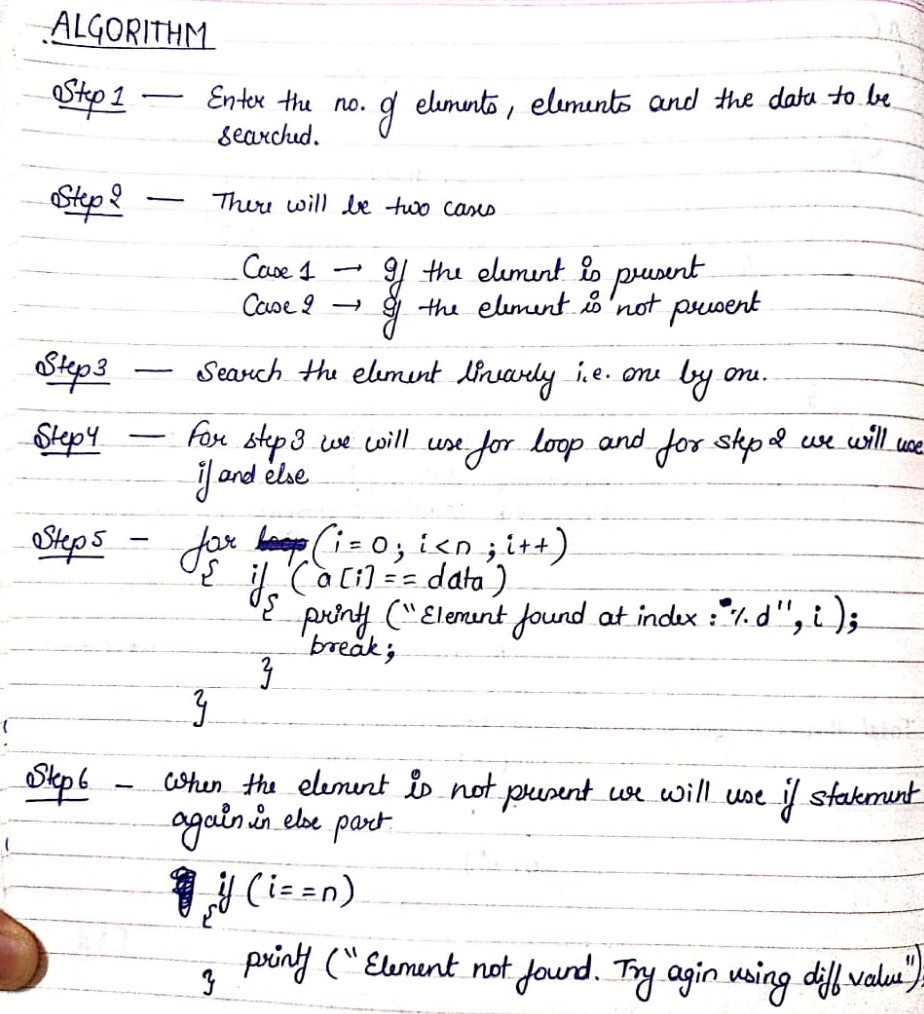
****

**BUBBLE SORT**

****

**LINEAR SEARCH**

**ALGORITHM**



**CODE**

#include<stdio.h>

int main()

{

int i,n,element;

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

scanf("%d", &n);

int array[n];

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

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

{

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

}

printf("Enter the element to be searched :");

scanf("%d", &element);

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

{

if(element == array[i])

{

printf("Element is present in array at position: %d",i);

break;

}

else

{

if(i==n)

printf("Element not found. try again using different value.");

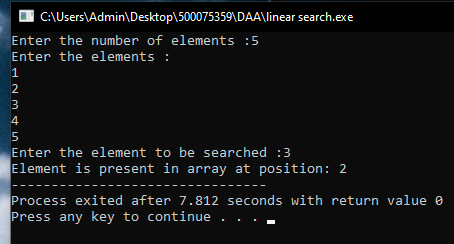
}

}

return 0;

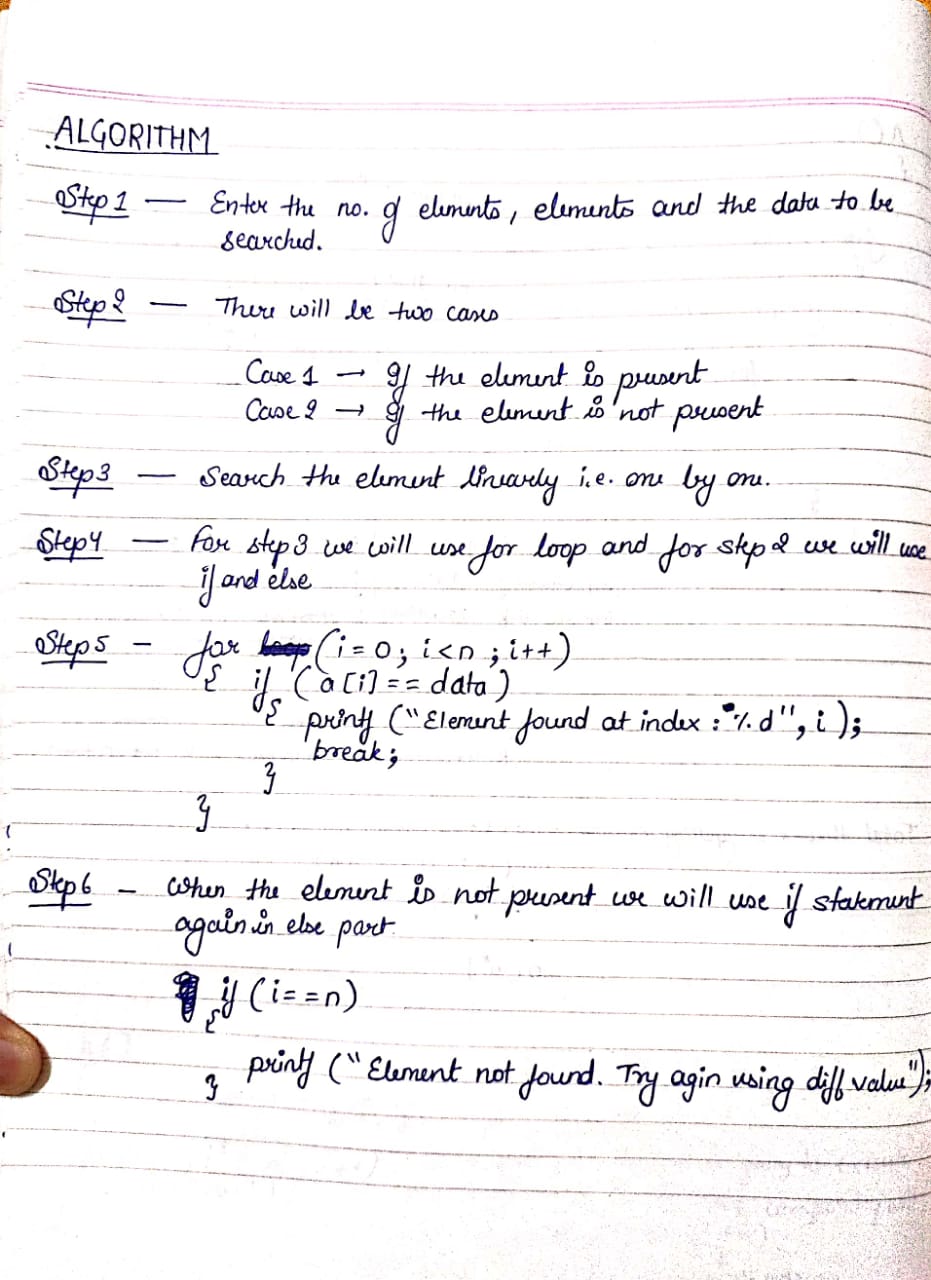
}

**OUTPUT**



**PATTERN MATCHING**

**ALGORITHM**



**CODE**

#include<stdio.h>

#include<string.h>

char t[100],p[50];

int brute\_force();

int main()

{

int pos;

printf("Enter the Source String: ");

scanf("%s",t);

printf("Enter the pattern: ");

scanf("%s",p);

pos = brute\_force ();

if(pos==-1)

printf("%s pattern not found in text",p);

else

printf("%s pattern found at index: %d",p,pos);

}

int brute\_force()

{

int n,j,m,i;

n=strlen(t);

m=strlen(p);

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

{

j=0;

while(j<m && t[i+j]==p[j])

{

j++;

if(j==m)

return i+1; //pattern found

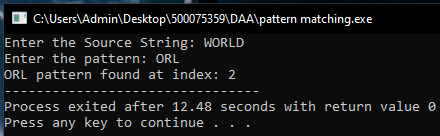
}

}

return -1; //pattern not found

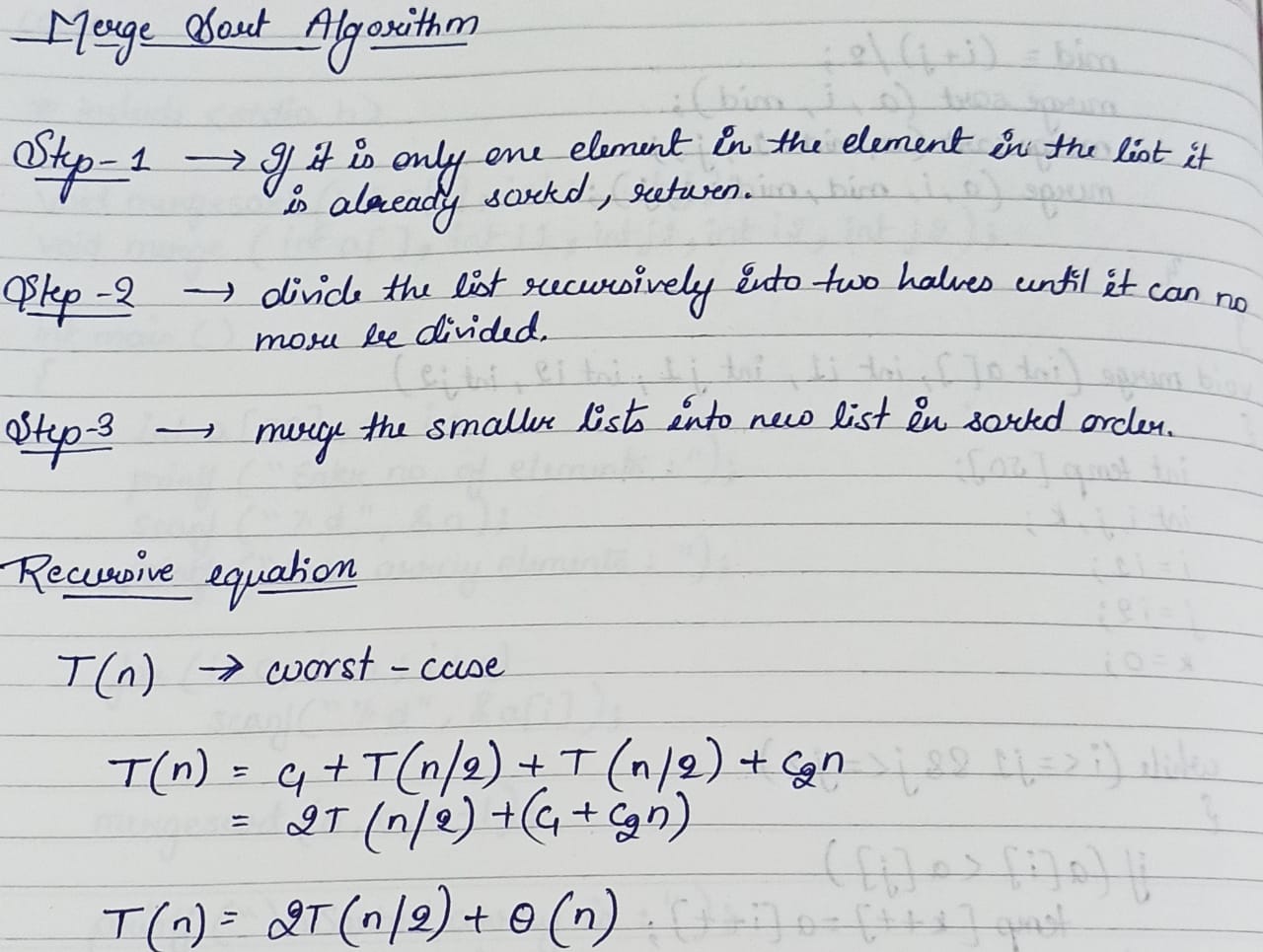
}

**OUTPUT**



**MERGE SORT**

**ALGORITHM**

****

**CODE**

#include<stdio.h>

void mergesort(int a[],int i,int j);

void merge(int a[],int i1,int j1,int i2,int j2);

int main()

{

int a[30],n,i;

printf("Enter no of elements:");

scanf("%d",&n);

printf("Enter array elements:");

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

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

mergesort(a,0,n-1);

printf("\nSorted array is :");

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

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

return 0;

}

void mergesort(int a[],int i,int j)

{

int mid;

if(i<j)

{

mid=(i+j)/2;

mergesort(a,i,mid);

mergesort(a,mid+1,j);

merge(a,i,mid,mid+1,j);

}

}

void merge(int a[],int i1,int j1,int i2,int j2)

{

int temp[50];

int i,j,k;

i=i1;

j=i2;

k=0;

while(i<=j1 && j<=j2)

{

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

temp[k++]=a[i++];

else

temp[k++]=a[j++];

}

while(i<=j1)

temp[k++]=a[i++];

while(j<=j2)

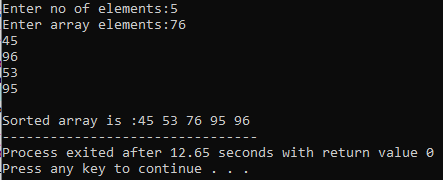
temp[k++]=a[j++];

for(i=i1,j=0;i<=j2;i++,j++)

a[i]=temp[j];

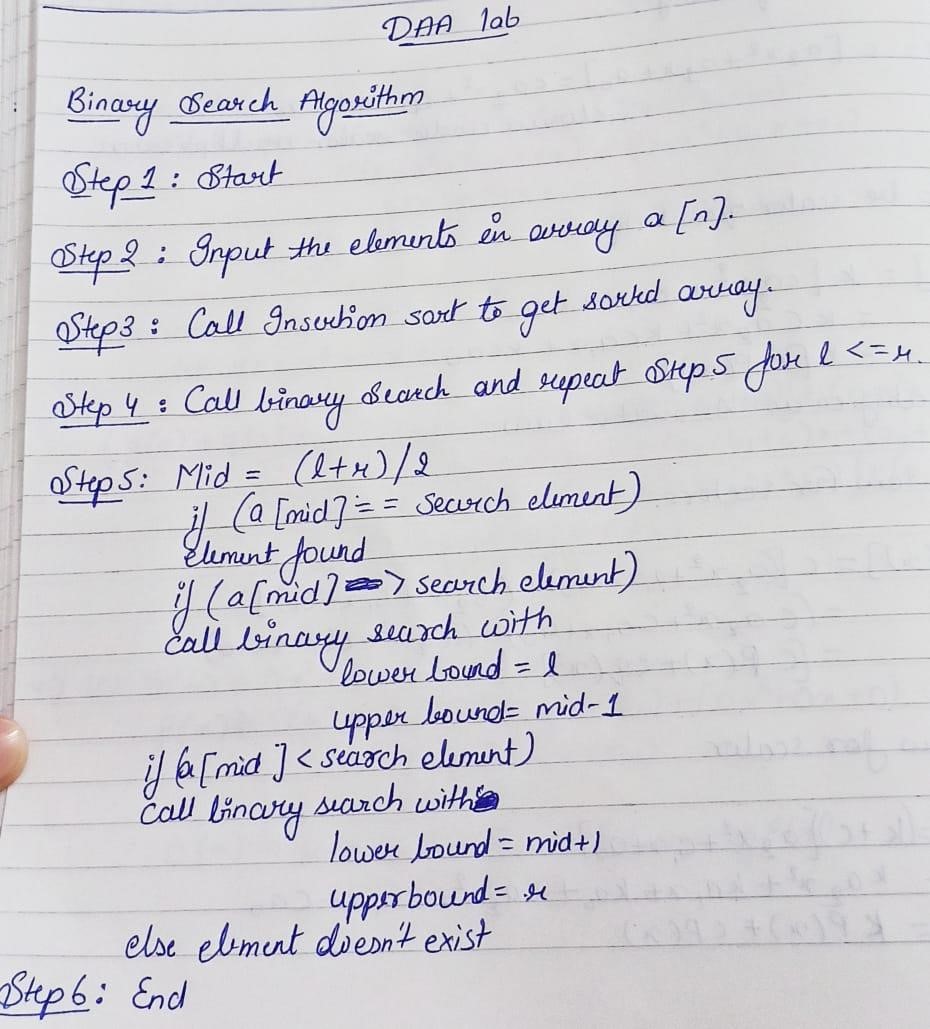
}

**OUTPUT**

****

**BINARY SEARCH**

**ALGORITHM**



**CODE**

#include<iostream> using namespace std; void insertion\_sort(int a[],int n){ int k,j; for (int i=1;i<n;i++){ k=a[i]; j=i-1; while(j>=0&&a[j]>k){

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

j--;

}

a[j+1]=k;

} } int binary\_search(int l,int a[],int r,int no){

if(l<=r){

int mid=(l+r)/2;

if(a[mid]==no) return mid; if(a[mid]>no)

return binary\_search(l,a,mid-1,no);

if(a[mid]<no)

return binary\_search(mid+1,a,r,no);

}

else

return -1;

}

int main(){

int n;

cout<<"Enter the no. of elements: ";

cin>>n; int a[n]; cout<<"Enter the elements in the array: ";

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

cin>>a[i];

}

cout<<"Unsorted array: ";

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

cout<<a[i]<<" ";

}

insertion\_sort(a,n); cout<<"\nSorted array: "; for(int

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

cout<<a[i]<<" ";

}

int no;

cout<<"\nEnter the number you want to search for: ";

cin>>no;

int t=binary\_search(0,a,n-1,no); if(t==-1) cout<<"No such element exist in the array";

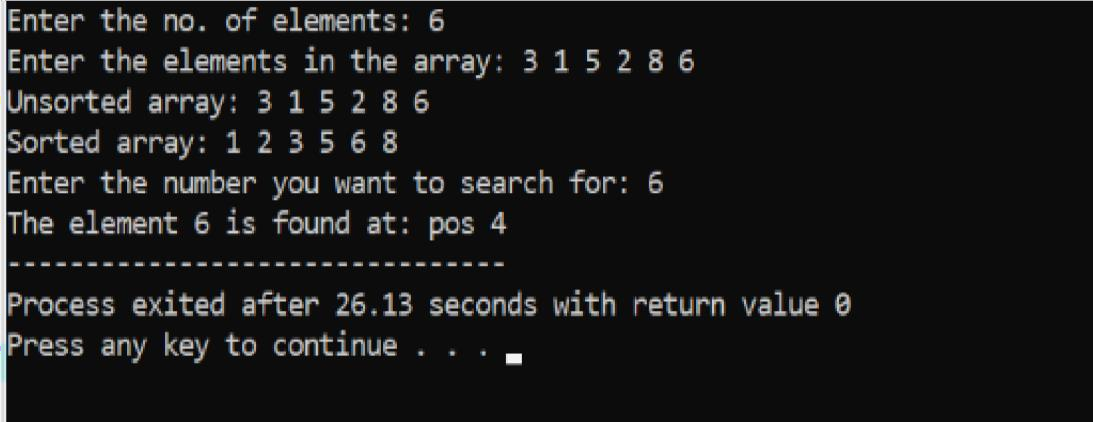
else

cout<<"The element "<<no<<" is found at: pos "<<t;}

**OUTPUT**

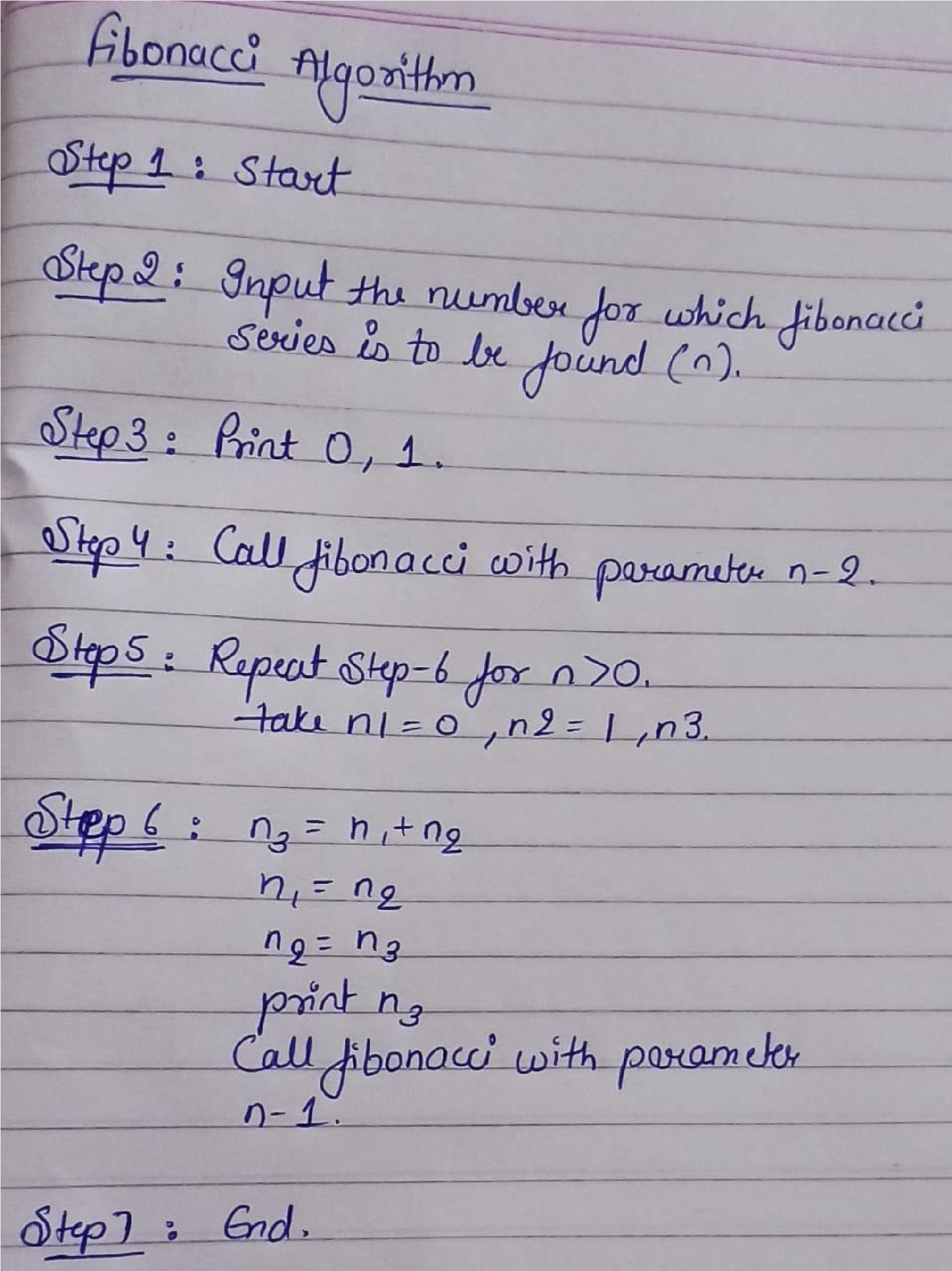
**Recurrence Relation:**

T(n) = 1+ T(n/2)



**FIBIONACCI SERIES**

**ALGORITHM**



**CODE**

#include<iostream> using namespace std;

int fibonacci(int n)

{ if(n==0) return 0; else if(n==1) return 1; else return fibonacci(n-1)+fibonacci(n-2);

}

int main ()

{ int n;

cout<<"Enter the number for which you want to find fibonacci series: ";

cin>>n;

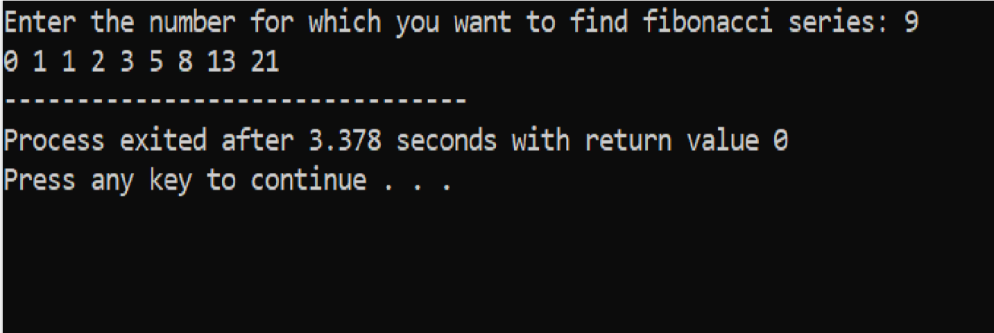
for(int i=0;i<n;i++){

cout<<fibonacci(i)<<" ";

}

}

**OUTPUT**



**Recurrence Relation:** T(n) = 1+ T(n-1) + T(n-2)

**STRASSEN’S MULTIPLICATION MATRIX**

**CODE**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int power(int);

int strassen(int \*, int \*, int \*, int, int);

int main()

{

int i,j,k,n1,n2,n3,n,m;

printf("Enter the order of the 1st Matrix (rows and columns):");

scanf("%d%d",&n1,&n2);

int A[n1][n2];

printf("\nNow enter the first matrix");

for(i=0;i<n1;i++){

printf("\nEnter the elements of the %d-th row:",i+1);

for(j=0;j<n2;j++)

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

}

printf("\nEnter the no. of columns of the 2nd Matrix:");

scanf("%d",&n3);

int B[n2][n3];

printf("\nNow enter the second matrix");

for(i=0;i<n2;i++){

printf("\nEnter the elements of the %d-th row:",i+1);

for(j=0;j<n3;j++)

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

}

if(n1>=n2 && n1>=n3)

n=n1;

else if(n2>=n1 && n2>=n3)

n=n2;

else

n=n3;

int o=1;

while(n>power(o))

o=o+1;

m=power(o);

int a[m][m],b[m][m],C[m][m];

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

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

a[i][j]=0;

b[i][j]=0;

}

}

for(i=0;i<n1;i++)

for(j=0;j<n2;j++)

a[i][j]=A[i][j];

for(i=0;i<n2;i++)

for(j=0;j<n3;j++)

b[i][j]=B[i][j];

printf("\nThis is the first matrix:");

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

printf("\n\n\n");

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

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

}

printf("\n\n\nThis is the second matrix:");

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

printf("\n\n\n");

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

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

}

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

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

C[i][j]=0;

strassen(a,b,C,m,m);

printf("\n\n\nThis is the final matrix:");

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

printf("\n\n\n");

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

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

}

}

int power(int n){

int i,p=1;

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

p=2\*p;

return(p);

}

int strassen(int \*A, int \*B, int \*C, int m, int n){

if(m==2){

int P=(\*A+\*(A+n+1))\*(\*B+\*(B+n+1));

int Q=(\*(A+n)+\*(A+n+1))\*(\*B);

int R=(\*A)\*(\*(B+1)-\*(B+n+1));

int S=(\*(A+n+1))\*(\*(B+n)-\*B);

int T=(\*A+\*(A+1))\*(\*(B+n+1));

int U=(\*(A+n)-\*A)\*(\*B+\*(B+1));

int V=(\*(A+1)-\*(A+n+1))\*(\*(B+n)+\*(B+n+1));

\*C=\*C+P+S-T+V;

\*(C+1)=\*(C+1)+R+T;

\*(C+n)=\*(C+n)+Q+S;

\*(C+n+1)=\*(C+n+1)+P+R-Q+U;

}

else{

m=m/2;

strassen(A,B,C,m,n);

strassen(A,B+m,C+m,m,n);

strassen(A+m,B+m\*n,C,m,n);

strassen(A+m,B+m\*(n+1),C+m,m,n);

strassen(A+m\*n,B,C+m\*n,m,n);

strassen(A+m\*n,B+m,C+m\*(n+1),m,n);

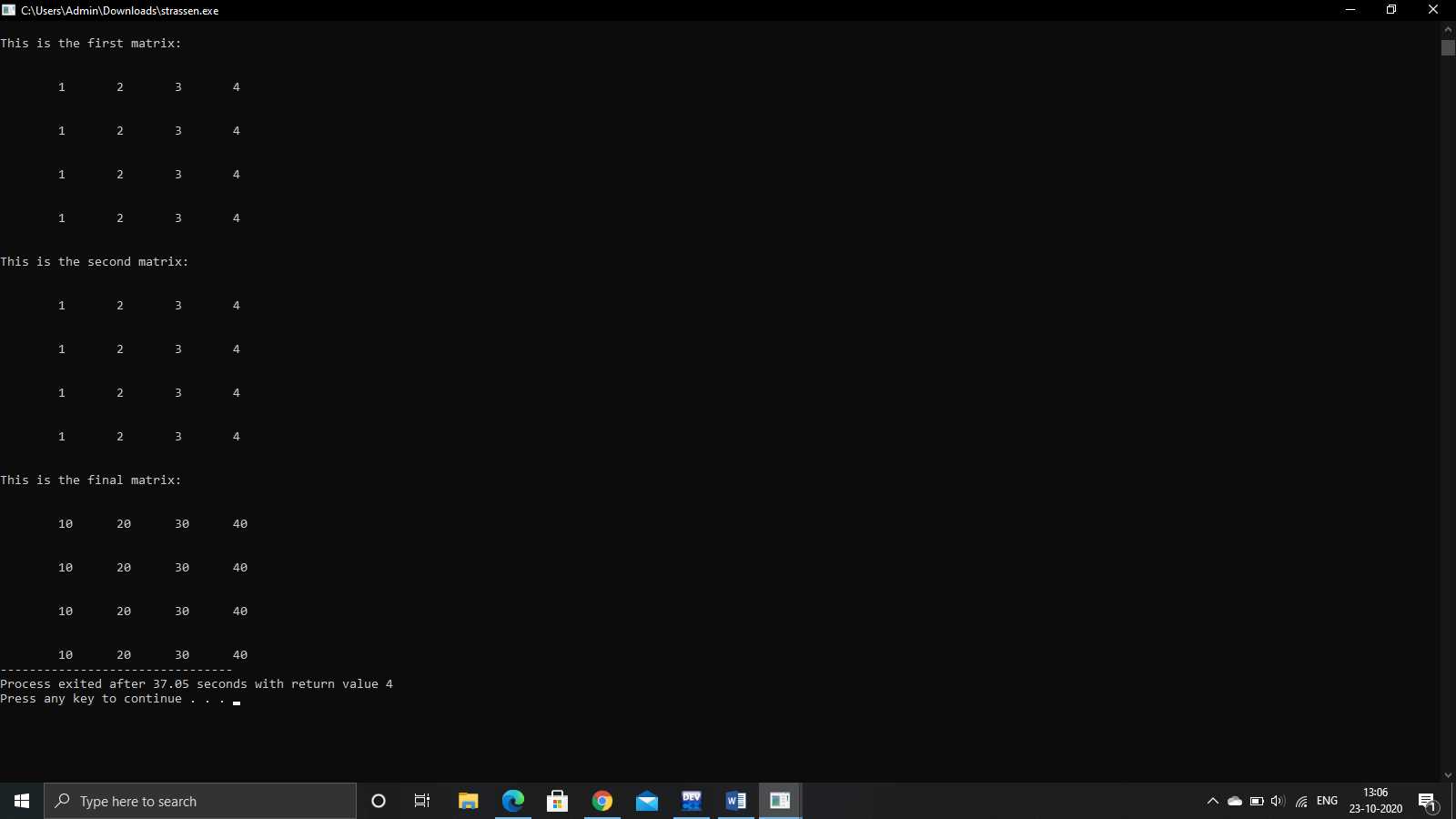
strassen(A+m\*(n+1),B+m\*n,C+m\*n,m,n);

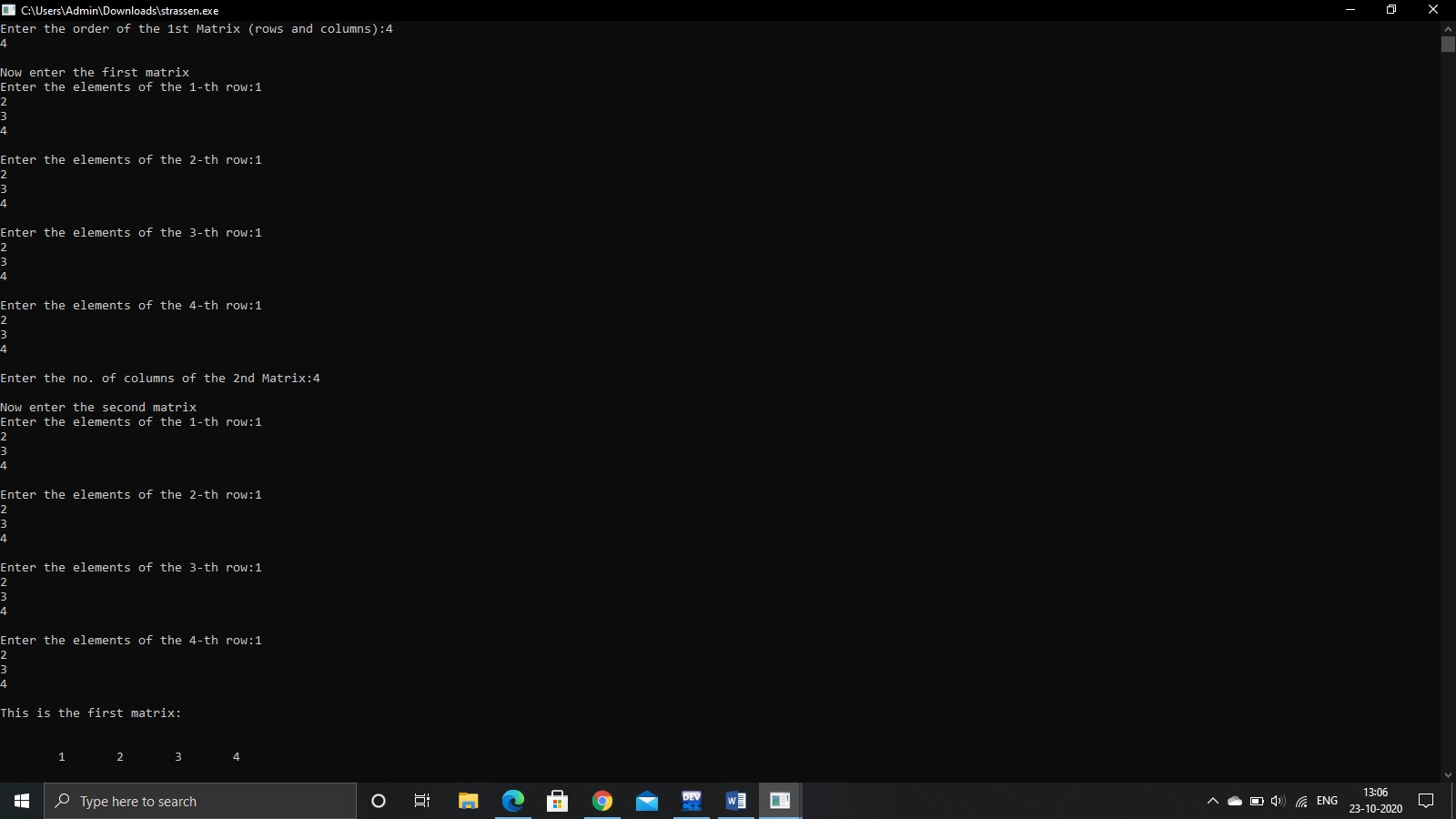
strassen(A+m\*(n+1),B+m\*(n+1),C+m\*(n+1),m,n);

}

}

**OUTPUT**





**MAX SUB ARRAY**

**CODE**

#include<stdio.h>

#define NEGINF -1000000000

#define MAX(a,b) a>b?a:b

int a[9]={-2,1,-3,4,-1,2,1,-5,4};

int max\_across(int low,int mid,int high){

int i;

int leftsum,rightsum,sum,maxsum;

leftsum=NEGINF;

sum=0;

for(i=mid;i>=low;i--){

sum=sum+a[i];

if(sum>leftsum)

leftsum=sum;

}

rightsum=NEGINF;

for(i=mid+1;i<=high;i++){

sum=sum+a[i];

if(sum>rightsum)

rightsum=sum;

}

maxsum=leftsum+rightsum;

return maxsum;

}

int max\_sum(int low,int high){

int left\_sum,right\_sum,across\_sum,final\_sum;

int mid=low+(high-low)/2;

if(low==high) return a[low];

left\_sum=max\_sum(low,mid);

right\_sum=max\_sum(mid+1,high);

across\_sum=max\_across(low,mid,high);

final\_sum=MAX(MAX(left\_sum,right\_sum),across\_sum);

return final\_sum;

}

int main(){

int sum;

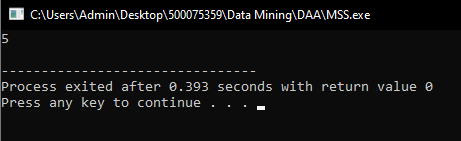
sum=max\_sum(0,9);

printf("%d\n",sum);

return 0;

}

**OUTPUT**



**GRAPH IMPLEMENTATION**

**CODE**

#include <stdio.h>

#include <stdlib.h>

struct AdjListNode

{

int dest;

struct AdjListNode\* next;

};

struct AdjList

{

struct AdjListNode \*head;

};

struct Graph

{

int V;

struct AdjList\* array;

};

struct AdjListNode\* newAdjListNode(int dest)

{

struct AdjListNode\* newNode =

(struct AdjListNode\*) malloc(sizeof(struct AdjListNode));

newNode->dest = dest;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int V)

{

struct Graph\* graph =

(struct Graph\*) malloc(sizeof(struct Graph));

graph->V = V;

graph->array =

(struct AdjList\*) malloc(V \* sizeof(struct AdjList));

int i;

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

graph->array[i].head = NULL;

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

struct AdjListNode\* newNode = newAdjListNode(dest);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

newNode = newAdjListNode(src);

newNode->next = graph->array[dest].head;

graph->array[dest].head = newNode;

}

void printGraph(struct Graph\* graph)

{

int v;

for (v = 0; v < graph->V; ++v)

{

struct AdjListNode\* pCrawl = graph->array[v].head;

printf("\n Adjacency list of vertex %d\n head ", v);

while (pCrawl)

{

printf("-> %d", pCrawl->dest);

pCrawl = pCrawl->next;

}

printf("\n");

}

}

int main()

{

int V = 5;

struct Graph\* graph = createGraph(V);

addEdge(graph, 0, 1);

addEdge(graph, 0, 4);

addEdge(graph, 1, 2);

addEdge(graph, 1, 3);

addEdge(graph, 1, 4);

addEdge(graph, 2, 3);

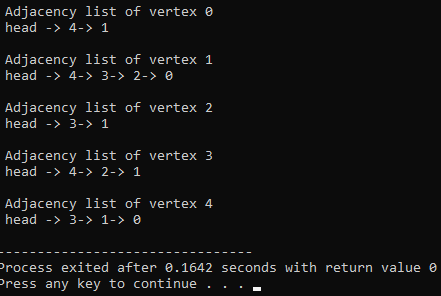
addEdge(graph, 3, 4);

printGraph(graph);

return 0;

}

**OUTPUT**



**ACTIVITY SELECTION**

**CODE**

#include<stdio.h>

void activities(int s[], int f[], int n)

{

int i, j;

printf ("Selected Activities are:\n");

i = 1;

printf("A%d ", i);

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

{

if (s[j] >= f[i])

{

printf ("A%d ", j+1);

i = j;

}

}

}

int main()

{

int s[] = {1, 3, 0, 5, 3, 5, 6, 8, 8, 2, 12};

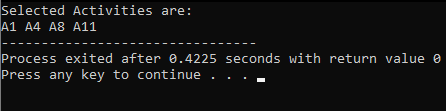
int f[] = {4, 5, 6, 7, 9, 9, 10, 11, 12, 14, 16};

int n = sizeof(s)/sizeof(s[0]);

activities(s, f, n);

}

**OUTPUT**

****