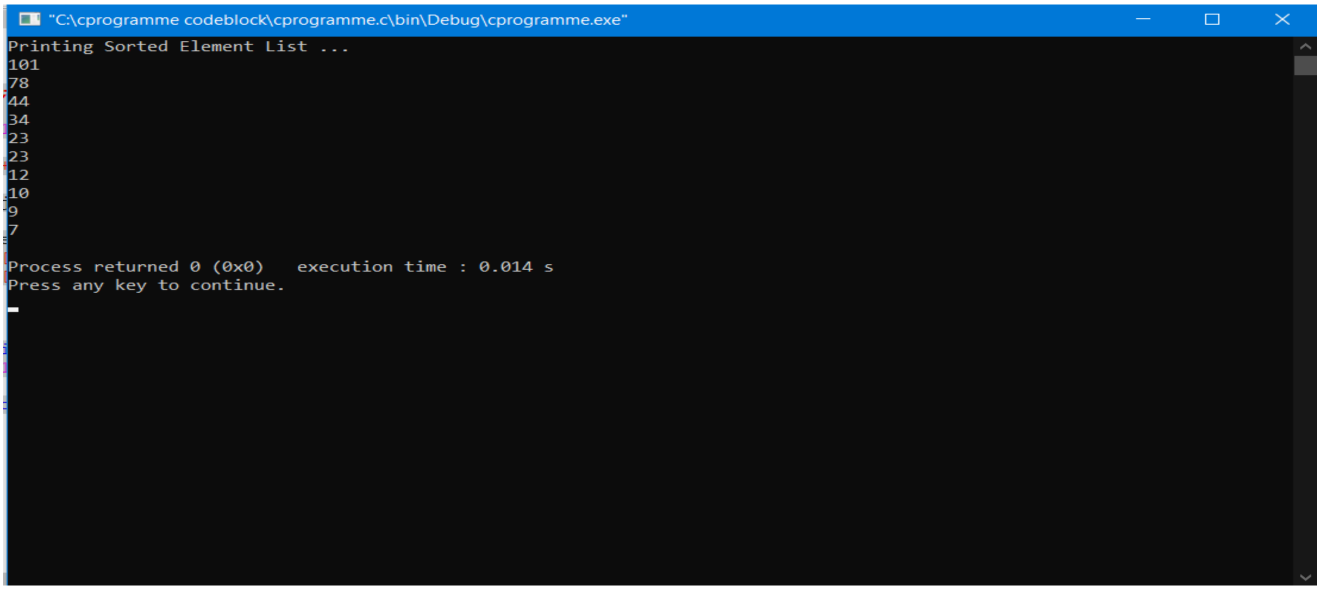
1.1 Implementation and Time analysis of Bubble sort.

➢ **Code :**

```
#include<stdio.h>
int main ()
{
   int i, j,temp;
   int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
   for(i = 0; i<10; i++)
   {
      for(j = i+1; j<10; j++)
      {
         if(a[j] > a[i])
         {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
         }
      }
   }
   printf("Printing Sorted Element List ...\n");
   for(i = 0; i<10; i++)
   {
      printf("%d\n",a[i]);
   }
   return 0;
}
```

➢ **Output :**



1.2 Implementation and Time analysis of Selection sort

- **Code :**

```
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;
        swap(&arr[min_idx], &arr[i]);
    }
}
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main()
{
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

- **Output :**

"C:\cprogramme codeblock\cprogramme.c\1-2.exe"

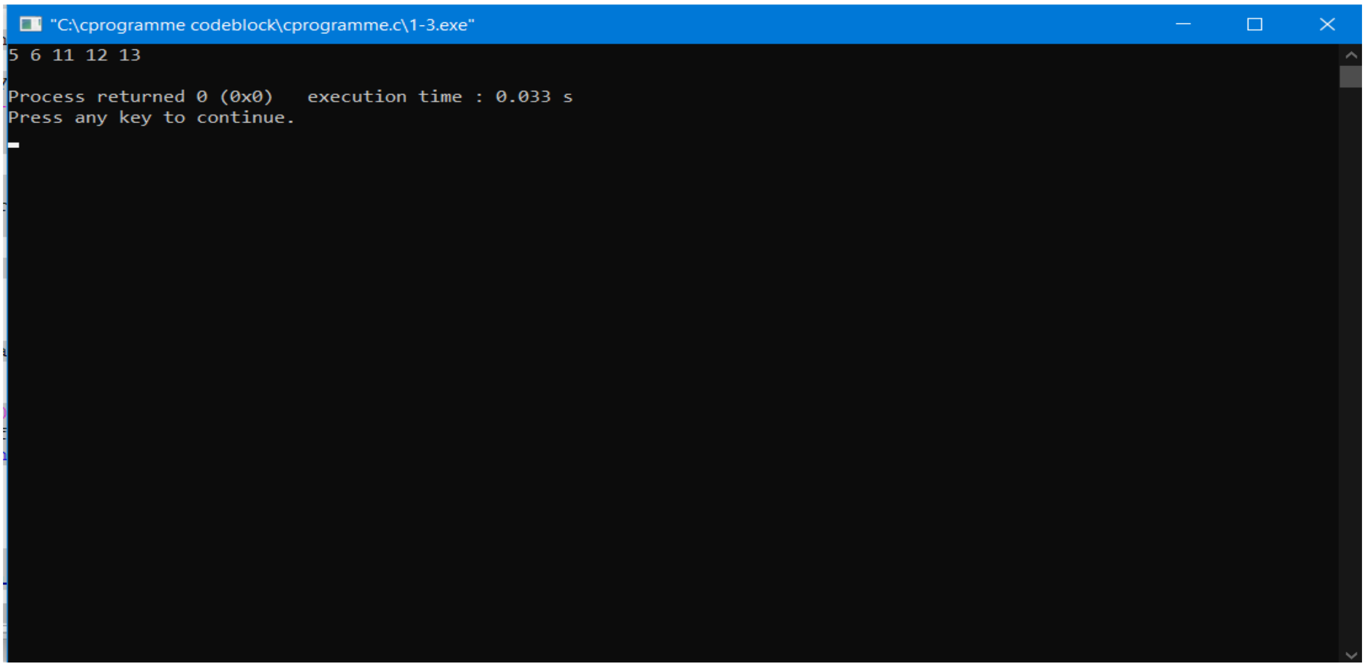
```
Sorted array:
11 12 22 25 64

Process returned 0 (0x0)   execution time : 0.042 s
Press any key to continue.
```

1.3	Implementation and Time analysis of Insertion sort

- **Code :**

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

void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    int arr[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

- **Output :**


"C:\cprogramme codeblock\cprogramme.c\1-3.exe"
5 6 11 12 13
Process returned 0 (0x0)   execution time : 0.033 s
Press any key to continue.

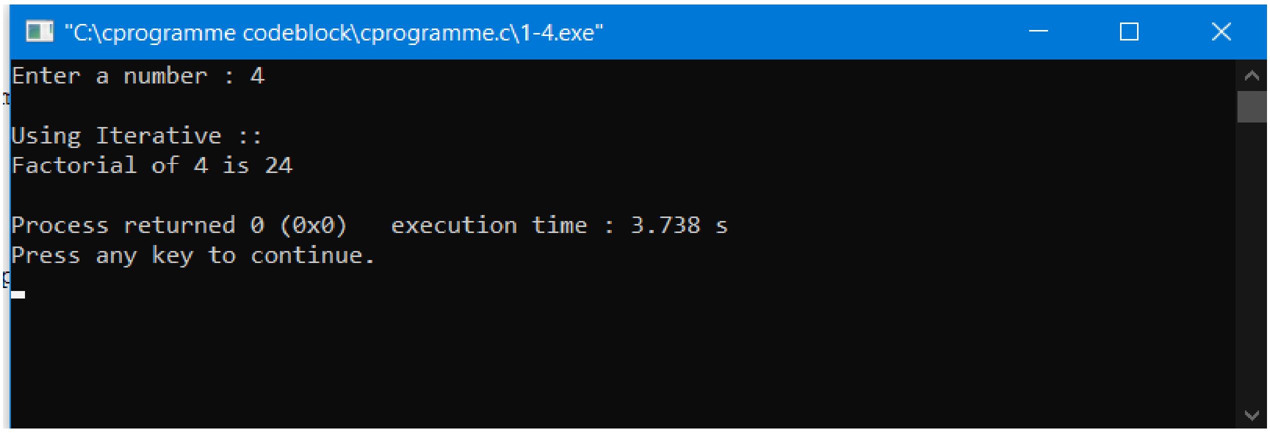
1.4	**Implementation and Time analysis of factorial program using iterative and recursive method.**

- **Using Iterative Method**

- **Code :**

```
#include<stdio.h>

long int Ifact(int n);

int main( )
{
	int num;
	printf("Enter a number : ");
	scanf("%d", &num);

    printf("\nUsing Iterative :: \n");

	if(num<0)
		printf("No factorial for negative number\n");
	else
		printf("Factorial of %d is %ld\n", num, Ifact(num) );

		return 0;
}
/*Using Iterative*/
long int Ifact(int n)
{
	long fact=1;
	while(n>0)
	{
		fact = fact*n;
		n--;
	}
	return fact;
}
```

- **Output :**

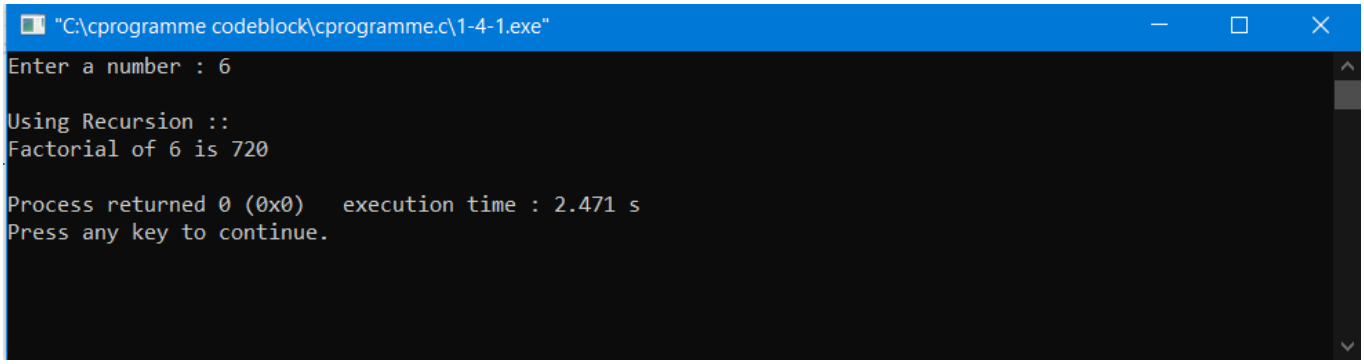


➢ Using Recursive Method

➢ Code :

```
#include<stdio.h>
long int fact(int n);

int main( )
{
	int num;
	printf("Enter a number : ");
	scanf("%d", &num);

	printf("\nUsing Recursion :: \n");
	if(num<0)
		printf("No factorial for negative number\n");
	else
		printf("Factorial of %d is %ld\n", num, fact(num) );
}

/*Using Recursive*/
long int fact(int n)
{
	if(n == 0)
		return(1);
	return(n * fact(n-1));
}
```

➢ Output :

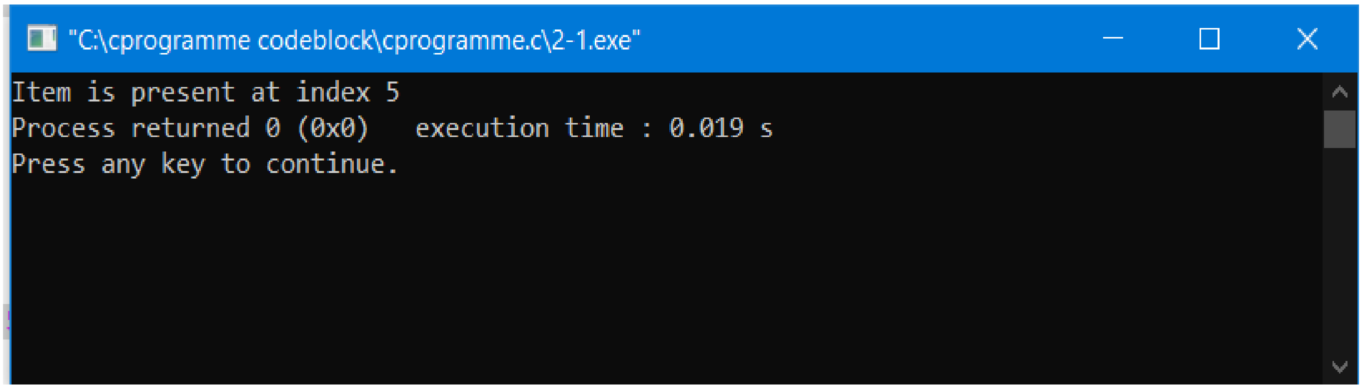


2.1 Implementation and Time analysis of Linear search algorithm.

- **Code :**

```
#include <stdio.h>

int LINEAR_SEARCH(int inp_arr[], int size, int val)
{

   for (int i = 0; i < size; i++)
      if (inp_arr[i] == val)
         return i;
   return -1;
}
int main(void)
{
   int arr[] = { 10, 20, 30, 40, 50, 100, 0 };
   int key = 100;
   int size = 10;
   int res = LINEAR_SEARCH(arr, size, key);
   if (res == -1)
   printf("ELEMENT NOT FOUND!!");
   else
   printf("Item is present at index %d", res);

   return 0;
}
```

- **Output :**



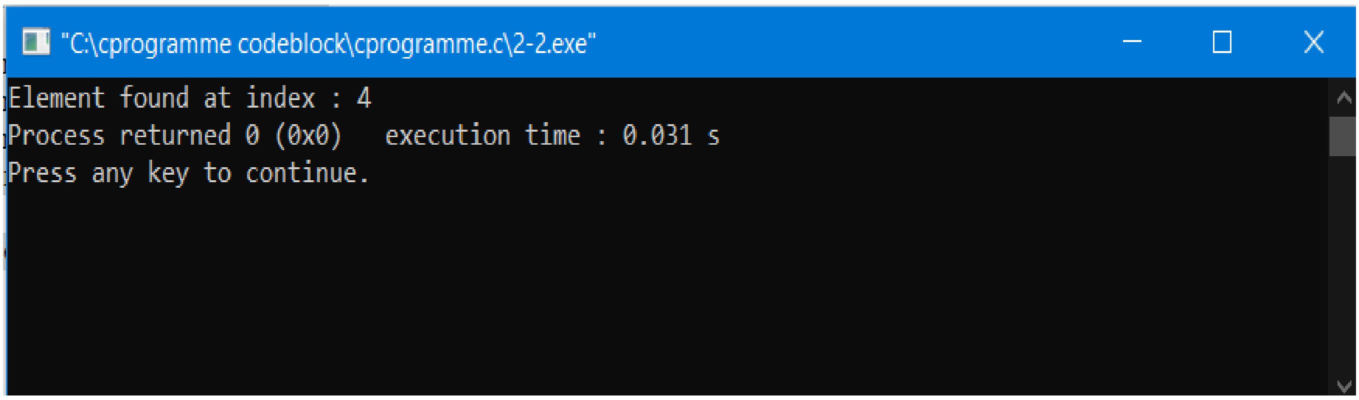
2.2 Implementation and Time analysis of Binary search algorithm.

- **Using Iterative Method :**

- **Code :**

```
#include <stdio.h>
int iterativeBinarySearch(int array[], int start_index, int end_index, int element){
   while (start_index <= end_index){
      int middle = start_index + (end_index- start_index )/2;
      if (array[middle] == element)
         return middle;
      if (array[middle] < element)
         start_index = middle + 1;
      else
         end_index = middle - 1;
   }
   return -1;
}
int main(void){
   int array[] = {1, 4, 7, 9, 16, 56, 70};
   int n = 7;
   int element = 16;
   int found_index = iterativeBinarySearch(array, 0, n-1, element);
   if(found_index == -1 ) {
      printf("Element not found in the array ");
   }
   else {
      printf("Element found at index : %d",found_index);
   }
   return 0;
}
```

- **Output :**



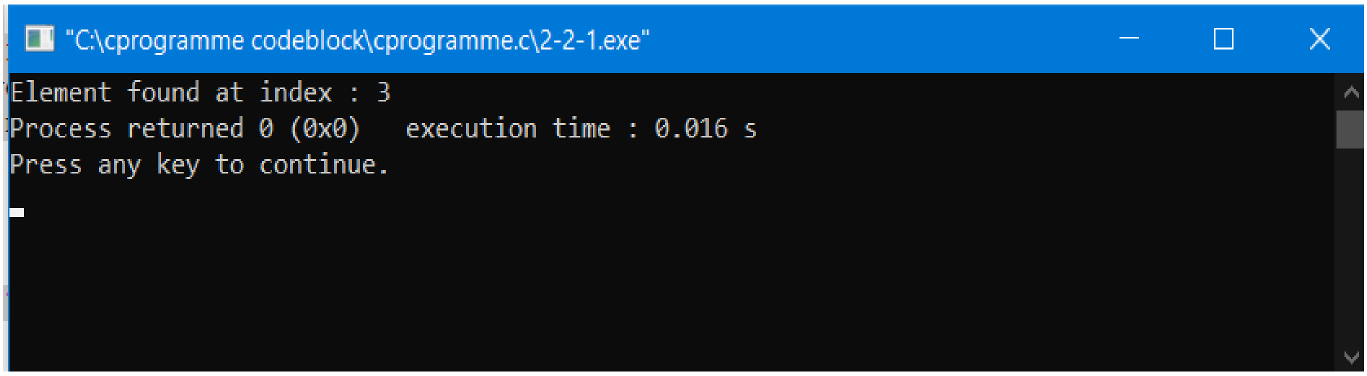
```
"C:\cprogramme codeblock\cprogramme.c\2-2.exe"
Element found at index : 4
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

- **Using Recursive Method :**

- **Code :**

```
#include <stdio.h>
int recursiveBinarySearch(int array[], int start_index, int end_index, int element){
  if (end_index >= start_index){
    int middle = start_index + (end_index - start_index )/2;
    if (array[middle] == element)
      return middle;
    if (array[middle] > element)
      return recursiveBinarySearch(array, start_index, middle-1, element);
    return recursiveBinarySearch(array, middle+1, end_index, element);
  }
  return -1;
}
int main(void){
  int array[] = {1, 4, 7, 9, 16, 56, 70};
  int n = 7;
  int element = 9;
  int found_index = recursiveBinarySearch(array, 0, n-1, element);
  if(found_index == -1 ) {
    printf("Element not found in the array ");
  }
  else {
    printf("Element found at index : %d",found_index);
  }
  return 0;
}
```

- **Output :**



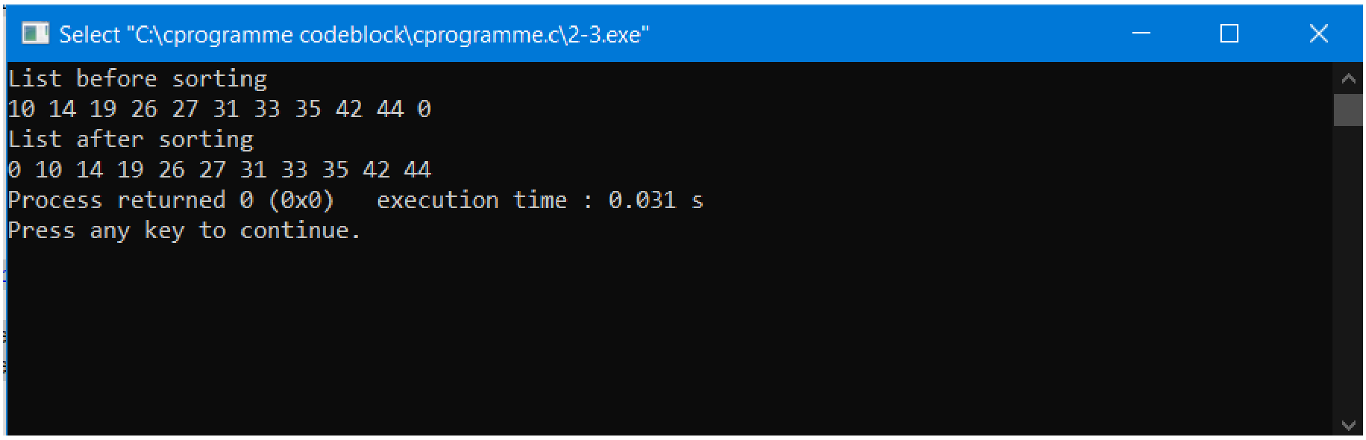
2.3	**Implementation and Time analysis of Merge sort**

- **Code :**

```
#include <stdio.h>

#define max 10

int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };
int b[10];

void merging(int low, int mid, int high)
{
   int l1, l2, i;

   for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
      if(a[l1] <= a[l2])
         b[i] = a[l1++];
      else
         b[i] = a[l2++];
   }

   while(l1 <= mid)
      b[i++] = a[l1++];

   while(l2 <= high)
      b[i++] = a[l2++];

   for(i = low; i <= high; i++)
      a[i] = b[i];
}

void sort(int low, int high) {
   int mid;

   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}

int main()
{
   int i;
   printf("List before sorting\n");

   for(i = 0; i <= max; i++)
      printf("%d ", a[i]);

   sort(0, max);
```

```
  printf("\nList after sorting\n");

  for(i = 0; i <= max; i++)
    printf("%d ", a[i]);
    return 0;
}
```

➢ **Output :**



2.4 Implementation and Time analysis of Quick sort

- **Code :**

```
#include<stdio.h>

void quicksort(int number[25],int first,int last)
{
int i, j, pivot, temp;
if(first<last)
{
pivot=first;
i=first;
j=last;
while(i<j)
{
while(number[i]<=number[pivot]&&i<last)
i++;
while(number[j]>number[pivot])
j--;
if(i<j)
{
temp=number[i];
number[i]=number[j];
number[j]=temp;
}
}
temp=number[pivot];
number[pivot]=number[j];
number[j]=temp;
quicksort(number,first,j-1);
quicksort(number,j+1,last);
}
}
int main()
{
int i, count, number[25];
printf("Enter some elements (Max. - 25): ");
scanf("%d",&count);
printf("Enter %d elements: ", count);
for(i=0;i<count;i++)
scanf("%d",&number[i]);
quicksort(number,0,count-1);
printf("The Sorted Order is: ");
for(i=0;i<count;i++)
printf(" %d",number[i]);
return 0;
}
```

- **Output :**

"C:\cprogramme codeblock\cprogramme.c\2-4.exe"

```
Enter some elements (Max. - 25): 6
Enter 6 elements: 5
2
45
85
65
3
The Sorted Order is:  2 3 5 45 65 85
Process returned 0 (0x0)   execution time : 9.933 s
Press any key to continue.
```

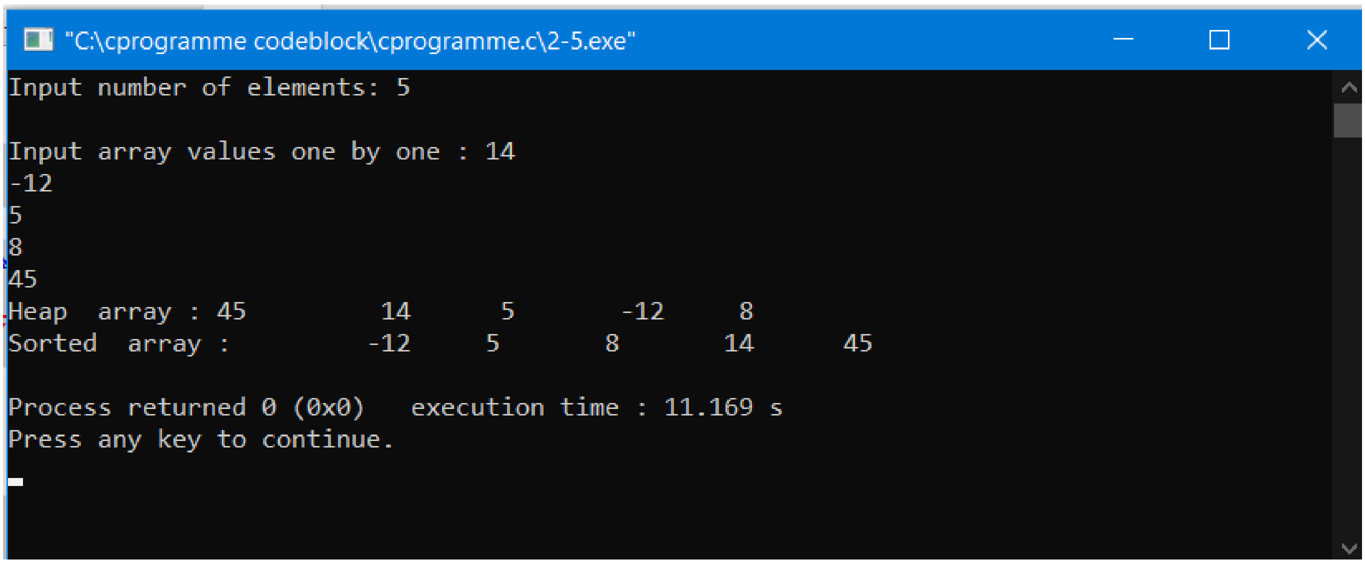
2.5	**Implementation of Max-Heap sort algorithm**

- **Code :**

```
#include <stdio.h>
 int main()
 {
 int arr[10], no,  i, j, c, heap_root, temp;
 printf("Input number of elements: ");
 scanf("%d", &no);
 printf("\nInput array values one by one : ");
 for (i = 0; i <  no; i++)
 scanf("%d", &arr[i]);
 for (i = 1; i <  no; i++)
 {
 c = i;
 do
 {
 heap_root  = (c - 1) / 2;
 /* to create MAX arr  array */
 if  (arr[heap_root] < arr[c])
 {
 temp =  arr[heap_root];
 arr[heap_root] = arr[c];
 arr[c]  = temp;
 }
 c =  heap_root;
 } while (c !=  0);
 }

 printf("Heap  array : ");
 for (i = 0; i <  no; i++)
 printf("%d\t ", arr[i]);
 for (j = no - 1; j  >= 0; j--)
 {
 temp = arr[0];
 arr[0] = arr[j];
 arr[j] = temp;
 heap_root = 0;
 do
 {
 c = 2 *  heap_root + 1;
 if  ((arr[c] < arr[c + 1]) && c < j-1)
 c++;
 if  (arr[heap_root]<arr[c] && c<j)
 {
 temp =  arr[heap_root];
 arr[heap_root] = arr[c];
 arr[c]  = temp;
 }
 heap_root  = c;
 } while (c  < j);
 }
 printf("\nSorted  array : ");
```

```
for (i = 0; i <  no; i++)
printf("\t%d", arr[i]);
printf("\n");
}
```

- **Output :**

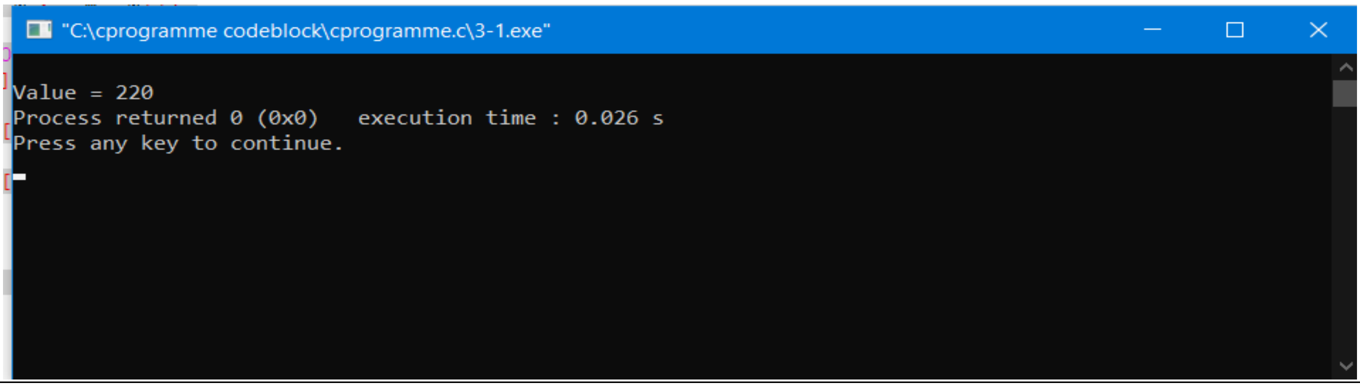


3.1 Implementation of a knapsack problem using dynamic programming.

➢ Code :

```
#include <stdio.h>

int max(int a, int b) { return (a > b)? a : b; }

int knapsack(int W, int wt[], int val[], int n)
{
  int i, w;
  int K[n+1][W+1];

  // Build table K[][] in bottom up manner
  for (i = 0; i <= n; i++)
  {
     for (w = 0; w <= W; w++)
     {
        if (i==0 || w==0)
           K[i][w] = 0;
        else if (wt[i-1] <= w)
              K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
        else
              K[i][w] = K[i-1][w];
     }
  }

  return K[n][W];
}

int main()
{
   int val[] = {60, 100, 120};
   int wt[] = {10, 20, 30};
   int  W = 50;
   int n = sizeof(val)/sizeof(val[0]);
   printf("\nValue = %d", knapsack(W, wt, val, n));
   return 0;
}
```

➢ Output :

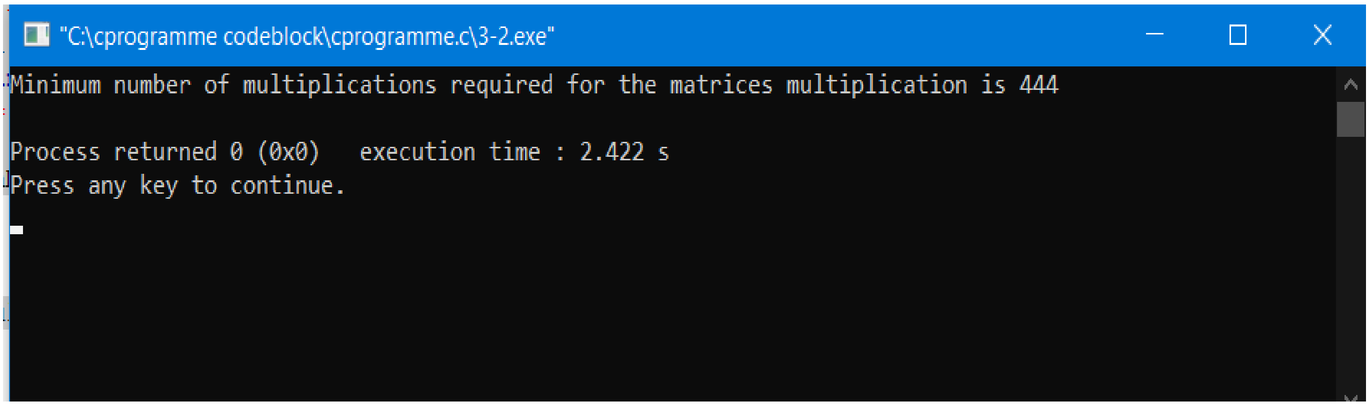


3.2	**Implementation of chain matrix multiplication using dynamic programming.**

➢ **Code :**

```
#include <stdio.h>
int MatrixChainMultuplication(int arr[], int n)
{
  int minMul[n][n];
  int j, q;
  for (int i = 1; i < n; i++)
    minMul[i][i] = 0;
  for (int L = 2; L < n; L++) {
    for (int i = 1; i < n - L + 1; i++) {
      j = i + L - 1;
      minMul[i][j] = 99999999;
      for (int k = i; k <= j - 1; k++) {
        q = minMul[i][k] + minMul[k + 1][j] + arr[i - 1] * arr[k] * arr[j];
        if (q < minMul[i][j])
        minMul[i][j] = q;
      }
    }
  }
  return minMul[1][n - 1];
}
int main()
{
  int arr[] = {3, 4, 5, 6, 7, 8};
  int size = sizeof(arr) / sizeof(arr[0]);
  printf("Minimum number of multiplications required for the matrices multiplication is %d ",
MatrixChainMultuplication(arr, size));
  getchar();
  return 0;
}
```

➢ **Output :**

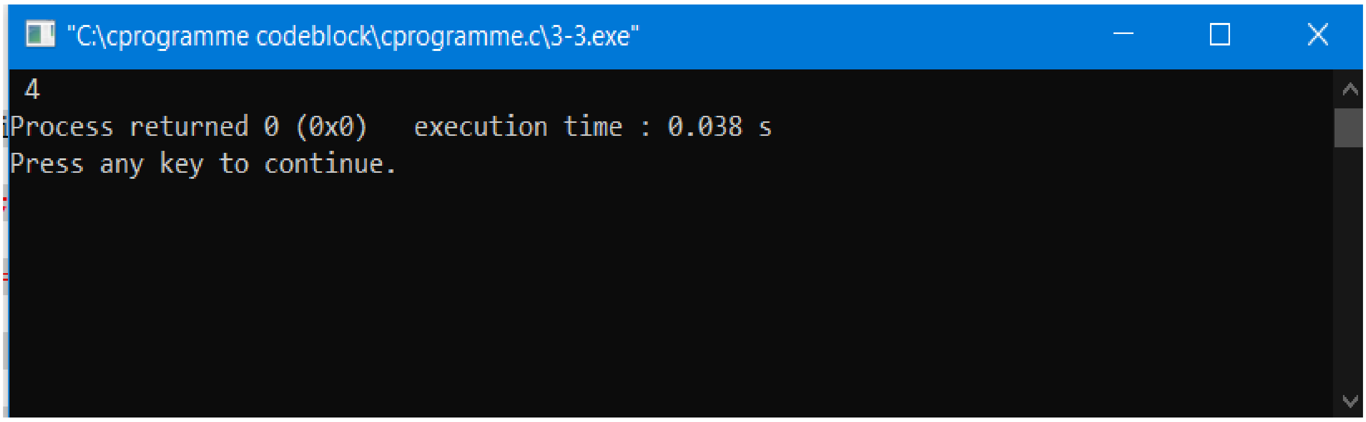


3.3 Implementation of making a change problem using dynamic programming

- **Code :**

```
#include<stdio.h>

int count( int S[], int m, int n )
{
    int i, j, x, y;

    int table[n+1][m];

    for (i=0; i<m; i++)
        table[0][i] = 1;

    for (i = 1; i < n+1; i++)
    {
        for (j = 0; j < m; j++)
        {
            x = (i-S[j] >= 0)? table[i - S[j]][j]: 0;

            y = (j >= 1)? table[i][j-1]: 0;

            table[i][j] = x + y;
        }
    }
    return table[n][m-1];
}

int main()
{
    int arr[] = {1, 2, 3};
    int m = sizeof(arr)/sizeof(arr[0]);
    int n = 4;
    printf(" %d ", count(arr, m, n));
    return 0;
}
```

- **Output :**



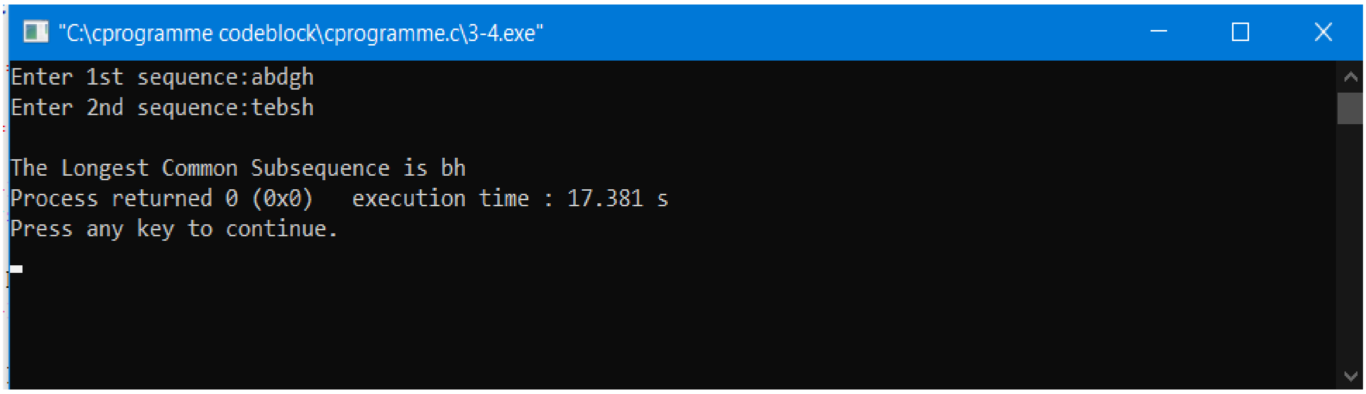
3.4	**Implement LCS problem.**

- **Code :**

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

int i,j,m,n,c[20][20];
char x[20],y[20],b[20][20];

void print(int i,int j)
{
        if(i==0 || j==0)
            return;
        if(b[i][j]=='c')
        {
            print(i-1,j-1);
            printf("%c",x[i-1]);
        }
            else if(b[i][j]=='u')
            print(i-1,j);
            else
            print(i,j-1);
}
void lcs()
{
            m=strlen(x);
            n=strlen(y);
            for(i=0;i<=m;i++)
                c[i][0]=0;
            for(i=0;i<=n;i++)
                c[0][i]=0;

            for(i=1;i<=m;i++)
                for(j=1;j<=n;j++)
                    {
                        if(x[i-1]==y[j-1])
                        {
                            c[i][j]=c[i-1][j-1]+1;
                            b[i][j]='c';
                        }
                            else if(c[i-1][j]>=c[i][j-1])
                        {
                            c[i][j]=c[i-1][j];
                            b[i][j]='u';
                        }
                            else
                        {
                            c[i][j]=c[i][j-1];
                            b[i][j]='l';
                        }
            }
}
int main()
```

```
{
                printf("Enter 1st sequence:");
                scanf("%s",x);
                printf("Enter 2nd sequence:");
                scanf("%s",y);
                printf("\nThe Longest Common Subsequence is ");
                lcs();
                print(m,n);
return 0;
}
```

- **Output :**



4.1	**Implementation of a Knapsack problem using greedy algorithm.**

- **Code :**

```
# include<stdio.h>

void knapsack(int n, float weight[], float profit[], float capacity) {
   float x[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("\nThe result vector is:- ");
   for (i = 0; i < n; i++)
      printf("%f\t", x[i]);

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

}

int main() {
   float weight[20], profit[20], capacity;
   int num, i, j;
   float ratio[20], temp;

   printf("\nEnter the no. of objects:- ");
   scanf("%d", &num);

   printf("\nEnter the wts and profits of each object:- ");
   for (i = 0; i < num; i++) {
      scanf("%f %f", &weight[i], &profit[i]);
   }

   printf("\nEnter the capacityacity of knapsack:- ");
   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 :**

```
"C:\cprogramme codeblock\cprogramme.c\4-1.exe"

Enter the no. of objects:- 7

Enter the wts and profits of each object:- 2
3
5
7
1
4
1
10
5
15
7
6
18
3

Enter the capacityacity of knapsack:- 15

The result vector is:- 1.000000 1.000000        1.000000        1.000000        1.000000        0.142857        0.000000

Maximum profit is:- 39.857143
Process returned 0 (0x0)   execution time : 41.294 s
Press any key to continue.
```

4.2	**Implement Prim's algorithm**

- **Code :**

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

#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()
{
int i,j,total_cost;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
total_cost=prims();
printf("\nspanning tree matrix:\n");
for(i=0;i<n;i++)
{
printf("\n");
for(j=0;j<n;j++)
printf("%d\t",spanning[i][j]);
}
printf("\n\nTotal cost of spanning tree=%d",total_cost);
return 0;
}

int prims()
{
int cost[MAX][MAX];
int u,v,min_distance,distance[MAX],from[MAX];
int visited[MAX],no_of_edges,i,min_cost,j;

for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(G[i][j]==0)
cost[i][j]=infinity;
else
cost[i][j]=G[i][j];
spanning[i][j]=0;
}

distance[0]=0;
visited[0]=1;
for(i=1;i<n;i++)
```

```
{
distance[i]=cost[0][i];
from[i]=0;
visited[i]=0;
}
min_cost=0;
no_of_edges=n-1;
while(no_of_edges>0)
{

min_distance=infinity;
for(i=1;i<n;i++)
if(visited[i]==0&&distance[i]<min_distance)
{
v=i;
min_distance=distance[i];
}
u=from[v];

spanning[u][v]=distance[v];
spanning[v][u]=distance[v];
no_of_edges--;
visited[v]=1;

for(i=1;i<n;i++)
if(visited[i]==0&&cost[i][v]<distance[i])
{
distance[i]=cost[i][v];
from[i]=v;
}
min_cost=min_cost+cost[u][v];
}
return(min_cost);
}
```

➢ **Output :**

"C:\cprogramme codeblock\cprogramme.c\4-2.exe"

```
Enter no. of vertices:6

Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

spanning tree matrix:

0       3       1       0       0       0
3       0       0       0       3       0
1       0       0       0       0       4
0       0       0       0       0       2
0       3       0       0       0       0
0       0       4       2       0       0

Total cost of spanning tree=13
Process returned 0 (0x0)   execution time : 89.614 s
Press any key to continue.
```

4.3	**Implement Kruskal's algorithm.**

- **Code :**

```
#include<stdio.h>

#define MAX 30

typedef struct edge
{
int u,v,w;
}edge;

typedef struct edgelist
{
edge data[MAX];
int n;
}edgelist;

edgelist elist;

int G[MAX][MAX],n;
edgelist spanlist;

void kruskal();
int find(int belongs[],int vertexno);
void union1(int belongs[],int c1,int c2);
void sort();
void print();

void main()
{
int i,j,total_cost;
printf("\nEnter number of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
kruskal();
print();
}

void kruskal()
{
int belongs[MAX],i,j,cno1,cno2;
elist.n=0;

for(i=1;i<n;i++)
for(j=0;j<i;j++)
{
if(G[i][j]!=0)
{
elist.data[elist.n].u=i;
```

	`` elist.data[elist.n].v=j; elist.data[elist.n].w=G[i][j]; elist.n++; } } sort(); for(i=0;i<n;i++) belongs[i]=i; spanlist.n=0; for(i=0;i<elist.n;i++) { cno1=find(belongs,elist.data[i].u); cno2=find(belongs,elist.data[i].v); if(cno1!=cno2) { spanlist.data[spanlist.n]=elist.data[i]; spanlist.n=spanlist.n+1; union1(belongs,cno1,cno2); } } } int find(int belongs[],int vertexno) { return(belongs[vertexno]); } void union1(int belongs[],int c1,int c2) { int i; for(i=0;i<n;i++) if(belongs[i]==c2) belongs[i]=c1; } void sort() { int i,j; edge temp; for(i=1;i<elist.n;i++) for(j=0;j<elist.n-1;j++) if(elist.data[j].w>elist.data[j+1].w) { temp=elist.data[j]; elist.data[j]=elist.data[j+1]; elist.data[j+1]=temp; } } void print() { int i,cost=0; for(i=0;i<spanlist.n;i++) { ```

```c
printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);
cost=cost+spanlist.data[i].w;
}

printf("\n\nCost of the spanning tree=%d",cost);
return 0;
}
```

- **Output :**

```
"C:\cprogramme codeblock\cprogramme.c\4-3.exe"

Enter number of vertices:6

Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

2       0       1
5       3       2
1       0       3
4       1       3
5       2       4

Cost of the spanning tree=13
Process returned 30 (0x1E)   execution time : 50.284 s
Press any key to continue.
```

5.1	Implementation of Graph and Searching : Breadth First Search

5.1 Implementation of Graph and Searching : Breadth First Search

- **Code :**

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 40

struct queue {
  int items[SIZE];
  int front;
  int rear;
};

struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);

struct node {
  int vertex;
  struct node* next;
};

struct node* createNode(int);

struct Graph {
  int numVertices;
  struct node** adjLists;
  int* visited;
};

void bfs(struct Graph* graph, int startVertex) {
  struct queue* q = createQueue();

  graph->visited[startVertex] = 1;
  enqueue(q, startVertex);

  while (!isEmpty(q)) {
    printQueue(q);
    int currentVertex = dequeue(q);
    printf("Visited %d\n", currentVertex);

    struct node* temp = graph->adjLists[currentVertex];

    while (temp) {
      int adjVertex = temp->vertex;

      if (graph->visited[adjVertex] == 0) {
        graph->visited[adjVertex] = 1;
        enqueue(q, adjVertex);
      }
```

```
      temp = temp->next;
    }
  }
}

struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));
  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }

  return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}

struct queue* createQueue() {
  struct queue* q = malloc(sizeof(struct queue));
  q->front = -1;
  q->rear = -1;
  return q;
}

int isEmpty(struct queue* q) {
  if (q->rear == -1)
    return 1;
  else
    return 0;
}

void enqueue(struct queue* q, int value) {
  if (q->rear == SIZE - 1)
    printf("\nQueue is Full!!");
```

```
  else {
    if (q->front == -1)
      q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
  }
}

int dequeue(struct queue* q) {
  int item;
  if (isEmpty(q)) {
    printf("Queue is empty");
    item = -1;
  } else {
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
      printf("Resetting queue ");
      q->front = q->rear = -1;
    }
  }
  return item;
}

void printQueue(struct queue* q) {
  int i = q->front;

  if (isEmpty(q)) {
    printf("Queue is empty");
  } else {
    printf("\nQueue contains \n");
    for (i = q->front; i < q->rear + 1; i++) {
      printf("%d ", q->items[i]);
    }
  }
}

int main() {
  struct Graph* graph = createGraph(6);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 1, 4);
  addEdge(graph, 1, 3);
  addEdge(graph, 2, 4);
  addEdge(graph, 3, 4);

  bfs(graph, 0);

  return 0;
}
```

➢ Output :

"C:\cprogramme codeblock\cprogramme.c\5-1.exe"

```
Queue contains
0 Resetting queue Visited 0

Queue contains
2 1 Visited 2

Queue contains
1 4 Visited 1

Queue contains
4 3 Visited 4

Queue contains
3 Resetting queue Visited 3

Process returned 0 (0x0)   execution time : 0.051 s
Press any key to continue.
```

5.2 Implementation of Graph and Searching : Depth First Search

- **Code :**

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

struct node {
  int vertex;
  struct node* next;
};

struct node* createNode(int v);

struct Graph {
  int numVertices;
  int* visited;
  struct node** adjLists;
};

void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;

  graph->visited[vertex] = 1;
  printf("Visited %d \n", vertex);

  while (temp != NULL) {
    int connectedVertex = temp->vertex;

    if (graph->visited[connectedVertex] == 0) {
      DFS(graph, connectedVertex);
    }
    temp = temp->next;
  }
}

struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
```

```
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}

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

  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;


  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}

void printGraph(struct Graph* graph) {
  int v;
  for (v = 0; v < graph->numVertices; v++) {
    struct node* temp = graph->adjLists[v];
    printf("\n Adjacency list of vertex %d\n ", v);
    while (temp) {
      printf("%d -> ", temp->vertex);
      temp = temp->next;
    }
    printf("\n");
  }
}

int main() {
  struct Graph* graph = createGraph(4);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 2, 3);

  printGraph(graph);

  DFS(graph, 2);

  return 0;
}
```

- **Output :**

```
"C:\cprogramme codeblock\cprogramme.c\5-2.exe"

 Adjacency list of vertex 0
 2 -> 1 ->

 Adjacency list of vertex 1
 2 -> 0 ->

 Adjacency list of vertex 2
 3 -> 1 -> 0 ->

 Adjacency list of vertex 3
 2 ->
Visited 2
Visited 3
Visited 1
Visited 0

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