

# Algorithm Analysis and Design (Assignment –II)

*Submitted in partial fulfilment of the requirements for the degree of*

**Master of Technology in Information Technology**

by

Vijayananda D Mohire

(Enrolment No.921DMTE0113)



Information Technology Department

Karnataka State Open University

Manasagangotri, Mysore – 570006

Karnataka, India

(2009)

# Algorithm Analysis and Design



**CERTIFICATE**

This is to certify that the Assignment-II entitled (Algorithm Analysis and Design, subject code: MT13) submitted by Vijayananda D Mohire having Roll Number 921DMTE0113 for the partial fulfilment of the requirements of Master of Technology in Information Technology degree of Karnataka State Open University, Mysore, embodies the bonafide work done by him under my supervision.

**Place:** \_\_\_\_\_**Signature of the Internal Supervisor****Name****Date:** \_\_\_\_\_**Designation**

**For Evaluation**

<b>Question Number</b>	<b>Maximum Marks</b>	<b>Marks awarded</b>	<b>Comments, if any</b>
1	5		
2	5		
<b>TOTAL</b>	10		

Evaluator's Name and Signature

Date

**Preface**

This document has been prepared specially for the assignments of M.Tech – IT I Semester. This is mainly intended for evaluation of assignment of the academic M.Tech - IT, I semester. I have made a sincere attempt to gather and study the best answers to the assignment questions and have attempted the responses to the questions. I am confident that the evaluator's will find this submission informative and evaluate based on the provide content.

For clarity and ease of use there is a Table of contents and Evaluators section to make easier navigation and recording of the marks. A list of references has been provided in the last page – Bibliography that provides the source of information both internal and external. Evaluator's are welcome to provide the necessary comments against each response, suitable space has been provided at the end of each response.

I am grateful to the Infysys academy, Koramangala, Bangalore in making this a big success. Many thanks for the timely help and attention in making this possible within specified timeframe. Special thanks to Mr. Vivek and Mr. Prakash for their timely help and guidance.

Candidate's Name and Signature

Date

## TABLE OF CONTENTS

FOR EVALUATION .....	4
PREFACE.....	5
QUESTION 1 .....	9
ANSWER 1 (A) .....	9
ANSWER 1 (B).....	12
QUESTION 2 .....	15
ANSWER 2.....	15
BIBLIOGRAPHY .....	38

## Table of Figures

<b>Figure 1</b> Depth First Search Algorithm Output (Schildt, 2000).....	10
<b>Figure 2</b> Depth First Search explanation (Schildt, 2000) .....	11
<b>Figure 3</b> DFS Code Snippet (Schildt, 2000) .....	12
<b>Figure 4</b> Breadth First Search Output (Schildt, 2000).....	13
<b>Figure 5</b> Breadth First Search Explanation (Schildt, 2000).....	13
<b>Figure 6</b> Breadth First Search Code Snippet (Schildt, 2000) .....	14
<b>Figure 7</b> Merge Sort Input (Anonymous, 2009) .....	15
<b>Figure 8</b> Merge Sort Output (Anonymous, 2009) .....	15

***ALGORITHM ANALYSIS AND DESIGN  
RESPONSE TO ASSIGNMENT - II***



**Question 1** Write the C/C++ codes for DFS & BFS?

**Answer 1(a) Depth First Search Algorithm**

Input:

Let us now consider a problem that we will use various searches to solve. Imagine that you are a travel agent and a rather quarrelsome customer wants you to book a flight from New York to Los Angeles with XYZ Airlines. You try to tell the customer that XYZ does not have a direct flight from New York to Los Angeles, but the customer insists that XYZ is the only airline that he will fly.

XYZ's scheduled flights are as follows:

**Flight Distance**

New York to Chicago 1,000 miles

Chicago to Denver 1,000 miles

New York to Toronto 800 miles

New York to Denver 1,900 miles

Toronto to Calgary 1,500 miles

Toronto to Los Angeles 1,800 miles

Toronto to Chicago 500 miles

Denver to Urbana 1,000 miles

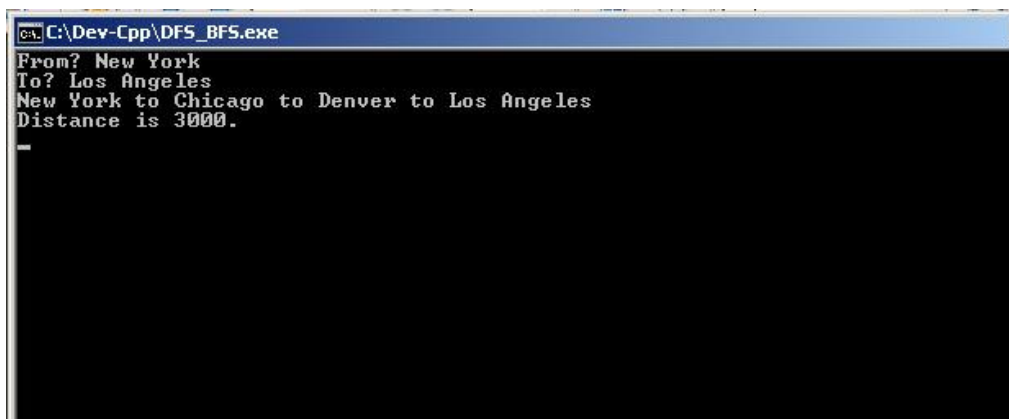
Denver to Houston 1,500 miles

Houston to Los Angeles 1,500 miles

Denver to Los Angeles 1,000 miles

You quickly see that there is a way to fly from New York to Los Angeles by using XYZ if you book connecting flights, and you book the fellow his flights.

Output:

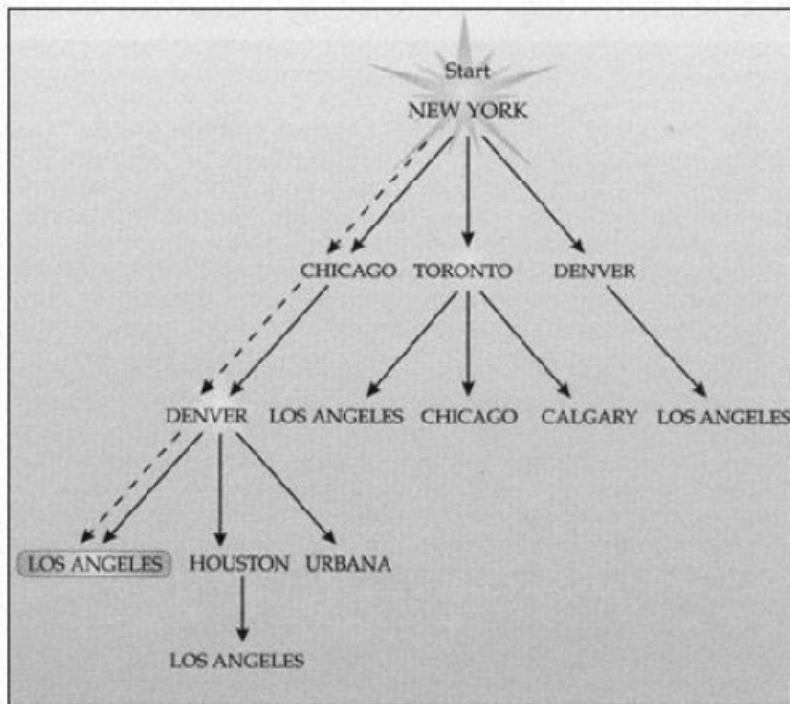


```
C:\Dev-Cpp\DFS_BFS.exe
From? New York
To? Los Angeles
New York to Chicago to Denver to Los Angeles
Distance is 3000.

```

**Figure 1** Depth First Search Algorithm Output (Schildt, 2000)

Explanation:



**Figure 2** Depth First Search explanation (Schildt, 2000)

If you refer to Figure 2, you see that this is indeed the first solution that would be found by a depth-first search. It is not the optimal solution— which is New York to Chicago to Denver to Los Angeles with a distance of 3,000 miles— but it is not bad.

Lets looks at the key code SNIPPET as below:

```

/* Determine if there is a route between from and to. */
void isflight(char *from, char *to)
{
    int d, dist;
    char anywhere[20];
    /* see if at destination */
    if(d=match(from, to)) {
        push(from, to, d);
        return;
    }
    /* try another connection */
    if(dist=find(from, anywhere)) {
        push(from, to, dist);
        isflight(anywhere, to);
    }
    else if(tos > 0) {
        /* backtrack */
        pop(from, to, &dist);
        isflight(from, to);
    }
}
}

```

**Figure 3** DFS Code Snippet (Schildt, 2000)

Full code available in Appendix

### **Answer 1(b)** Breadth First Search Algorithm

Input: Same as in Depth First Search Algorithm

Output:

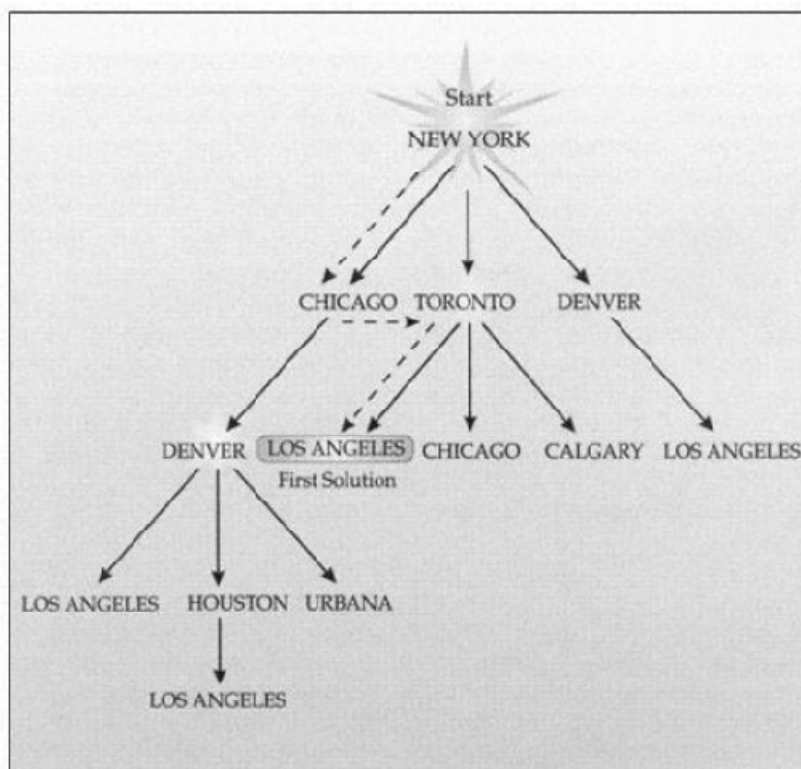
```

C:\Dev-Cpp\BFS.exe
From? New York
To? Los Angeles
New York to Toronto to Los Angeles
Distance is 2600.

```

**Figure 4** Breadth First Search Output (Schildt, 2000)

Explanation:



**Figure 5** Breadth First Search Explanation (Schildt, 2000)

Let's look into the code snippet as below:

```

/* Determine if there is a route between from and to. */
void isflight(char *from, char *to)
{
    int d, dist;
    char anywhere[20];
    while(dist=find(from, anywhere)) {
        /* breadth-first modification */
        if(d=match(anywhere, to)) {
            push(from, to, dist);
            push(anywhere, to, d); ← Extra code Line for BFS algorithm
            return;
        }
    }
    /* try any connection */
    if(dist=find(from, anywhere)) {
        push(from, to, dist);
        isflight(anywhere, to);
    }
    else if(tos>0) {
        pop(from, to, &dist);
        isflight(from, to);
    }
}
}

```

**Figure 6** Breadth First Search Code Snippet (Schildt, 2000)

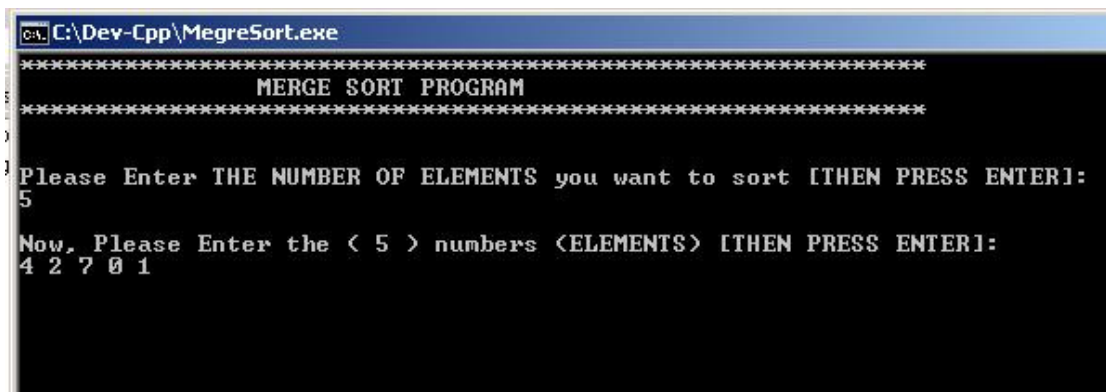
Full code available in Appendix:

Evaluator's Comments if any:

## Question 2 Write the C/C++ codes for merge sort?

### Answer 2

Input:



```

C:\Dev-Cpp\MegreSort.exe
*****
MERGE SORT PROGRAM
*****

Please Enter THE NUMBER OF ELEMENTS you want to sort [THEN PRESS ENTER]:
5

Now, Please Enter the < 5 > numbers <ELEMENTS> [THEN PRESS ENTER]:
4 2 7 0 1
  
```

**Figure 7** Merge Sort Input (Anonymous, 2009)

Output:



```

C:\Dev-Cpp\MegreSort.exe
*****
MERGE SORT PROGRAM
*****

Please Enter THE NUMBER OF ELEMENTS you want to sort [THEN PRESS ENTER]:
5

Now, Please Enter the < 5 > numbers <ELEMENTS> [THEN PRESS ENTER]:
4 2 7 0 1

So, the sorted list <using MERGE SORT> will be :

0      1      2      4      7
  
```

**Figure 8** Merge Sort Output (Anonymous, 2009)

Explanation:

```
void merge_sort(int low,int high)
{
    int mid;
    if (low<high)
    {
        mid=(low+high)/2;
        merge_sort (low,mid);
        merge_sort (mid+1,high);
        merge (low,mid,high);
    }
}
```

Full code is available in Appendix

Evaluator's Comments if any:



## Appendix

```

/* Depth-first search. */

#include <stdio.h>

#include <string.h>

#include <conio.h>


#define MAX 100


/* structure of the flight database */

struct FL {

    char from[20];

    char to[20];

    int distance;

    char skip; /* used in backtracking */

};


struct FL flight[MAX]; /* array of db structures */

int f_pos = 0; /* number of entries in flight db */

int find_pos = 0; /* index for searching flight db */

int tos = 0; /* top of stack */

```

```
struct stack {  
  
    char from[20];  
  
    char to[20];  
  
    int dist;  
  
};  
  
struct stack bt_stack[MAX]; /* backtrack stack */  
  
void setup(void), route(char *to);  
  
void assert_flight(char *from, char *to, int dist);  
  
void push(char *from, char *to, int dist);  
  
void pop(char *from, char *to, int *dist);  
  
void isflight(char *from, char *to);  
  
int find(char *from, char *anywhere);  
  
int match(char *from, char *to);  
  
int main(void)  
  
{  
  
    char from[20], to[20];  
  
    setup();
```

```
printf("From? ");

gets(from);

printf("To? ");

gets(to);

isflight(from,to);

route(to);

getch();

return 0;

}

/* Initialize the flight database. */

void setup(void)

{

    assert_flight("New York", "Chicago", 1000);

    assert_flight("Chicago", "Denver", 1000);

    assert_flight("New York", "Toronto", 800);

    assert_flight("New York", "Denver", 1900);

    assert_flight("Toronto", "Calgary", 1500);

    assert_flight("Toronto", "Los Angeles", 1800);

    assert_flight("Toronto", "Chicago", 500);

    assert_flight("Denver", "Urbana", 1000);
```

```

assert_flight("Denver", "Houston", 1500);

assert_flight("Houston", "Los Angeles", 1500);

assert_flight("Denver", "Los Angeles", 1000);

}

/* Put facts into the database. */

void assert_flight(char *from, char *to, int dist)

{

    if(f_pos < MAX) {

        strcpy(flight[f_pos].from, from);

        strcpy(flight[f_pos].to, to);

        flight[f_pos].distance = dist;

        flight[f_pos].skip = 0;

        f_pos++;

    }

    else printf("Flight database full.\n");

}


/* Show the route and total distance. */

void route(char *to)

{

```

```

int dist, t;

dist = 0;

t = 0;

while(t < tos) {

printf("%s to ", bt_stack[t].from);

dist += bt_stack[t].dist;

t++;

}

printf("%s\n", to);

printf("Distance is %d.\n", dist);

}

/* If flight between from and to, then return

the distance of flight; otherwise, return 0. */

int match(char *from, char *to)

{

    register int t;

    for(t=f_pos-1; t > -1; t--)

        if(!strcmp(flight[t].from, from) &&

!strcmp(flight[t].to, to)) return flight[t].distance;

    return 0; /* not found */

```

```

}

/* Given from, find anywhere. */

int find(char *from, char *anywhere)

{
    find_pos = 0;

    while(find_pos < f_pos) {

        if(!strcmp(flight[find_pos].from,from) &&
        !flight[find_pos].skip) {

            strcpy(anywhere,flight[find_pos].to);

            flight[find_pos].skip = 1; /* make active */

            return flight[find_pos].distance;

        }

        find_pos++;

    }

    return 0;

}

/* Determine if there is a route between from and to. */

void isflight(char *from, char *to)

{

    int d, dist;

```

```

char anywhere[20];

/* see if at destination */

if(d=match(from, to)) {

    push(from, to, d);

    return;

}

/* try another connection */

if(dist=find(from, anywhere)) {

    push(from, to, dist);

    isflight(anywhere, to);

}

else if(tos > 0) {

    /* backtrack */

    pop(from, to, &dist);

    isflight(from, to);

}

}

/* Stack Routines */

void push(char *from, char *to, int dist)

{

```

```
if(tos < MAX) {  
  
    strcpy(bt_stack[tos].from,from);  
  
    strcpy(bt_stack[tos].to,to);  
  
    bt_stack[tos].dist = dist;  
  
    tos++;  
  
}  
  
else printf("Stack full.\n");  
  
}  
  
void pop(char *from, char *to, int *dist)  
  
{  
  
    if(tos > 0) {  
  
        tos--;  
  
        strcpy(from,bt_stack[tos].from);  
  
        strcpy(to,bt_stack[tos].to);  
  
        *dist = bt_stack[tos].dist;  
  
    }  
  
    else printf ("Stack underflow.\n");  
  
}
```



```

/* Breadth-first search. */

#include <stdio.h>

#include <string.h>

#include <conio.h>


#define MAX 100


/* structure of the flight database */

struct FL {

    char from[20];

    char to[20];

    int distance;

    char skip; /* used in backtracking */

};


struct FL flight[MAX]; /* array of db structures */

int f_pos = 0; /* number of entries in flight db */

int find_pos = 0; /* index for searching flight db */

int tos = 0; /* top of stack */

struct stack {

```

```
char from[20];

char to[20];

int dist;

};

struct stack bt_stack[MAX]; /* backtrack stack */

void setup(void), route(char *to);

void assert_flight(char *from, char *to, int dist);

void push(char *from, char *to, int dist);

void pop(char *from, char *to, int *dist);

void isflight(char *from, char *to);

int find(char *from, char *anywhere);

int match(char *from, char *to);


int main(void)

{

    char from[20], to[20];

    setup();

    printf("From? ");

    gets(from);
```

```
printf("To? ");

gets(to);

isflight(from,to);

route(to);

getch();

return 0;

}

/* Initialize the flight database. */

void setup(void)

{

    assert_flight("New York", "Chicago", 1000);

    assert_flight("Chicago", "Denver", 1000);

    assert_flight("New York", "Toronto", 800);

    assert_flight("New York", "Denver", 1900);

    assert_flight("Toronto", "Calgary", 1500);

    assert_flight("Toronto", "Los Angeles", 1800);

    assert_flight("Toronto", "Chicago", 500);

    assert_flight("Denver", "Urbana", 1000);

    assert_flight("Denver", "Houston", 1500);

    assert_flight("Houston", "Los Angeles", 1500);
```

```

assert_flight("Denver", "Los Angeles", 1000);

}

/* Put facts into the database. */

void assert_flight(char *from, char *to, int dist)
{
    if(f_pos < MAX) {

        strcpy(flight[f_pos].from, from);

        strcpy(flight[f_pos].to, to);

        flight[f_pos].distance = dist;

        flight[f_pos].skip = 0;

        f_pos++;

    }

    else printf("Flight database full.\n");

}


/* Show the route and total distance. */

void route(char *to)
{
    int dist, t;

    dist = 0;

```

```

t = 0;

while(t < tos) {

    printf("%s to ", bt_stack[t].from);

    dist += bt_stack[t].dist;

    t++;

}

printf("%s\n", to);

printf("Distance is %d.\n", dist);

}

/* If flight between from and to, then return
the distance of flight; otherwise, return 0. */

int match(char *from, char *to)

{

    register int t;

    for(t=f_pos-1; t > -1; t--)

        if(!strcmp(flight[t].from, from) &&

!strcmp(flight[t].to, to)) return flight[t].distance;

    return 0; /* not found */

}

/* Given from, find anywhere. */

```

```

int find(char *from, char *anywhere)

{

    find_pos = 0;

    while(find_pos < f_pos) {

        if(!strcmp(flight[find_pos].from,from) &&

        !flight[find_pos].skip) {

            strcpy(anywhere,flight[find_pos].to);

            flight[find_pos].skip = 1; /* make active */

            return flight[find_pos].distance;

        }

        find_pos++;

    }

    return 0;

}

/* Determine if there is a route between from and to. */

void isflight(char *from, char *to)

{

    int d, dist;

    char anywhere[20];

```

```

while(dist=find(from, anywhere)) {

    /* breadth-first modification */

    if(d=match(anywhere, to)) {

        push(from, to, dist);

        push(anywhere, to, d);

        return;

    }

}

/* try any connection */

if(dist=find(from, anywhere)) {

    push(from, to, dist);

    isflight(anywhere, to);

}

else if(tos>0) {

    pop(from, to, &dist);

    isflight(from, to);

}

}

/* Stack Routines */

```

```
void push(char *from, char *to, int dist)

{

if(tos < MAX) {

strcpy(bt_stack[tos].from,from);

strcpy(bt_stack[tos].to,to);

bt_stack[tos].dist = dist;

tos++;

}

else printf("Stack full.\n");

}

void pop(char *from, char *to, int *dist)

{

if(tos > 0) {

tos--;

strcpy(from,bt_stack[tos].from);

strcpy(to,bt_stack[tos].to);

*dist = bt_stack[tos].dist;

}

else printf ("Stack underflow.\n");

}
```



```
//Merge Sort
```

```
#include <iostream.h>
```

```
#include <conio.h>
```

```
int a[50];
```

```
void merge(int,int,int);
```

```
void merge_sort(int low,int high)
```

```
{
```

```
    int mid;
```

```
    if(low<high)
```

```
    {
```

```
        mid=(low+high)/2;
```

```
        merge_sort(low,mid);
```

```
        merge_sort(mid+1,high);
```

```
        merge(low,mid,high);
```

```
    }
```

```
}
```

```
void merge(int low,int mid,int high)
```

```
{  
  
    int h,i,j,b[50],k;  
  
    h=low;  
  
    i=low;  
  
    j=mid+1;  
  
  
    while((h<=mid)&&(j<=high))  
  
    {  
  
        if(a[h]<=a[j])  
  
        {  
  
            b[i]=a[h];  
  
            h++;  
  
        }  
  
        else  
  
        {  
  
            b[i]=a[j];  
  
            j++;  
  
        }  
  
        i++;  
  
    }  
}
```

```
if(h>mid)

{

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

    {

        b[i]=a[k];

        i++;

    }

}

else

{

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

    {

        b[i]=a[k];

        i++;

    }

}

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

}

int main()

{
```

```

int num,i;

cout<<"*****"<<endl;

cout<<"          MERGE SORT PROGRAM          "<<endl;

cout<<"*****"<<endl;

cout<<endl<<endl;

cout<<"Please Enter THE NUMBER OF ELEMENTS you want to sort [THEN
PRESS ENTER]:"<<endl;

cin>>num;

cout<<endl;

cout<<"Now, Please Enter the ( "<< num <<" ) numbers (ELEMENTS)
[THEN PRESS ENTER]:"<<endl;

for(i=1;i<=num;i++)
{
    cin>>a[i] ;
}

merge_sort(1,num);

cout<<endl;

cout<<"So, the sorted list (using MERGE SORT) will be :"<<endl;

cout<<endl<<endl;

```

```
for(i=1;i<=num;i++)  
    cout<<a[i]<<"    ";  
  
    cout<<endl<<endl<<endl<<endl;  
  
    getch();  
  
    return 0;  
}
```

## **Bibliography**

Anonymous. (2009). *C++ > Algorithms sample source codes, Merge Sort*. Retrieved 2009, from Happy Codings: <http://www.cplusplus.happycodings.com/Algorithms/code17.html>

Schildt, H. (2000). *C, The Complete Reference*. New York: McGraw-Hill.