

**Submitted By: Eman Amer Dar | 28692**

**Submitted To: Sir Hamdan**

Table of Contents

[1.1 1](#_Toc475011586)

[Searching 1](#_Toc475011587)

[Linear Search 1](#_Toc475011588)

[Binary Search 2](#_Toc475011589)

[Sorting 4](#_Toc475011590)

[Bubble Sort 4](#_Toc475011591)

[Selection Sort 5](#_Toc475011592)

[Insertion Sort 6](#_Toc475011593)

[1.2 8](#_Toc475011594)

[Cases 8](#_Toc475011595)

[Linear Search 8](#_Toc475011596)

[Binary Search 8](#_Toc475011597)

[Bubble Sort 8](#_Toc475011598)

[Selection Sort 9](#_Toc475011599)

[Insertion Sort 9](#_Toc475011600)

[1.3 11](#_Toc475011601)

[Recursion 11](#_Toc475011602)

[Base Case 11](#_Toc475011603)

[Recursive Case 11](#_Toc475011604)

[Merge Sorting 12](#_Toc475011605)

[(Factorial Recursion) 16](#_Toc475011606)

[(Binary Search Recursion) 18](#_Toc475011607)

[Comparison Between Recursion and Iteration 21](#_Toc475011608)

[2.1 24](#_Toc475011609)

[Function 24](#_Toc475011610)

[Linear Search 24](#_Toc475011611)

[Binary Search 28](#_Toc475011612)

[Bubble Sort 33](#_Toc475011613)

[Selection Sort 37](#_Toc475011614)

[Insertion Sort 41](#_Toc475011615)

[Minimum and Maximum Value 45](#_Toc475011616)

[Linked List Sorting 47](#_Toc475011617)

[Stack and Queue Infix to Postfix (Arithmetic Expression) 51](#_Toc475011618)

[2.2 60](#_Toc475011619)

[Errors Handling 60](#_Toc475011620)

[For MinMax 60](#_Toc475011621)

[For Queue and Stack 61](#_Toc475011622)

[Linked List Sorting 62](#_Toc475011623)

[2.3 65](#_Toc475011624)

[Actual and Expected Results 65](#_Toc475011625)

[Self-Criticism | Self-Evaluation 72](#_Toc475011626)

[Self-Criticism 72](#_Toc475011627)

[Self-Evaluation 72](#_Toc475011628)

[Bibliography 73](#_Toc475011629)

**Table of Figures**

[Recursive Trace Figure 1 (Prakash, 2015) 10](file:///Z:\Assignments\Semester-3%20Eman%20Dar%20HND%20Computing\Data%20Structure%20-%20AB1.docx#_Toc468717958)

# 1.1

## Searching

There are two types of searching:

* Linear Search
* Binary Search

### Linear Search

To move one by one through indexes is linear search. Linear search algorithm is done to make a program according to linear search and search one by one in the indexes to find a specific data/element stored in index.

**For Example**

If there is an organisation and they want to search any employee from their employee department so they will simply apply linear search and check one by one until they find the searched employee.

**Algorithm**

void function LinearSearch (int array [], int size, int key)

**Step1:** Declare and initialize i=1

**Step2:** Apply loop where i is lesser than array size if it is greater than size then go to step 7

**Step3:** If array [i] is equal to key then go to step 6

**Step4:** Apply increment i.e. i=i+1

**Step5:** To check go to step 2

**Step6:** Print key element is available at i index then go to step 8

**Step7:** Print key element not found

**Step8:** Exit

**Pseudo Code**

void function LinearSearch (int array [], int size, int key)

for item in array

if item==key

return item’s location

end if

end for

end function

### Binary Search

**Algorithm**

void function insertionsort (int num [], int size)

**Step1:** Declare and initialize i=0 in outer loop

**Step2:** Apply outer loop condition i is lesser than the size of array -1

**Step3:** Apply inner loop for (j=i+1; j>0 && num[j-1]>num[j]; j--)

**Step4:** If the condition is true apply swapping and go to step 3 until the swapping of indexes is done

**Step5:** Go to step 2 the increment is done and after increment if condition becomes false then go to step 6

**Step6:** End Function insertionsort

void function binarysearch(int num[], int size, int key)

**Step1:** initialize and declare first = 0, last = size-1 and mid = (first+last)/2

**Step2:** Apply for loop (;first<=last;)

**Step3:** Apply if condition (num[mid]<key)

**Step4:** Assign first = mid +1

**Step5:** Close if condition and apply else if condition (num[mid] == key)

**Step6:** print “number found” and apply break if condition is true

**Step7:** If condition is not true close else if condition and Assign last = mid-1

**Step8:** close condition

**Step9:** Close loop when first>last

**Step10:** If loop condition becomes false print “Number is not found”

**Step11:** Exit

**Pseudo Code**

void function insertionsort (int array [], int size)

for outer loop (int i=0; i<size-1; i++)

for inner loop(int j=i+1; j>0 && num[j-1]>num[j]; j--)

swapping

end for inner loop

end for outer loop

end function

void function binarysearch (int num[], int size, int key )

int first=0, last=size-1, mid= (first+last)/2;

for loop (;first<=last;)

if(num[mid]<key)

first = mid+1;

end if condition

else if (num[mid]==key)

Print number found;

break;

end else if condition

else

last=mid-1;

end else condition

end for loop

if (first>last)

Print number not found

end if condition

end function

## Sorting

Sorting done to sort the data stored in indexes in order i.e. ascending or descending and then makes a program to perform sorting. There are following types of sorting given below:

* Bubble Sort
* Selection Sort
* Insertion Sort

### Bubble Sort

Bubble sorting is done to sort the data by adjacent way.

**For Example**

If there are employees in any organisation and the person has to sort them on the basis of their designation then sorting algorithm will be made to arrange them in the following order i.e. ascending or descending by an adjacent way.

**Algorithm**

void function bubblesort (int num [], int size)

**Step1:** Declare and initialize i=size-1 in outer loop

**Step2:** Apply loop condition i>=0 if the condition becomes false go to step 7

**Step3:** Apply inner loop for (j=0; j<i; j++)

**Step4:** Apply condition if (num [j] < num [j+1]) in inner loop

**Step5:** If the condition is true apply swapping and go to step 3 until the swapping of indexes is done

**Step6:** Go to step 2 the increment is done

**Step7:** Exit

**Pseudo Code**

void function bubblesort (int array [], int size)

for outer loop (i=size-1; i>=0;i--)

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

if num [j] < num [j+1]

swapping

end if

end for inner loop

end for outer loop

end function

### Selection Sort

Selection sorting is done by selecting one by one.

**For Example**

If there are employees in any organisation and the person has to sort them on the basis of their designation then sorting algorithm will be made to arrange them in the following order i.e. ascending or descending by selecting one by one.

**Algorithm**

void function selectionsort (int num [], int size)

**Step1:** Declare min with the data type int

**Step2:** Declare and initialize i=0 in outer loop

**Step3:** Apply outer loop condition i is lesser than the size of array

**Step4:** Apply inner loop for (j=i+1; j<size; j++)

**Step5:** Apply condition if (num [j] < num [min]) in inner loop

**Step6:** If condition is true then the value of min is assigned

**Step7:** If inner loop condition j<size is false then break the inner loop

**Step8:** If the condition is true apply swapping and go to step 3 until the swapping of indexes is done

**Step9:** If outer loop conditioni<sizethen break the outer loopgo to step 10

**Step10:** Exit

**Pseudo Code**

void function selectionsort (int array [], int size)

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

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

if num [j] < num [min]

return index

end if

end for inner loop

swapping

end for outer loop

end function

### Insertion Sort

Insertion is done index by index

**Algorithm**

void function insertionsort (int num [], int size)

**Step1:** Declare and initialize i=0 in outer loop

**Step2:** Apply outer loop condition i is lesser than the size of array -1

**Step3:** Apply inner loop for (j=i+1; j>0 && num[j-1]>num[j]; j--)

**Step4:** If the condition is true apply swapping and go to step 3 until the swapping of indexes is done

**Step5:** Go to step 2 the increment is done and after increment if condition becomes false then go to step 6

**Step6:** Exit

**Pseudo Code**

void function insertionsort (int array [], int size)

for outer loop (int i=0; i<size-1; i++)

for inner loop(int j=i+1; j>0 && num[j-1]>num[j]; j--)

swapping

end for inner loop

end for outer loop

end function

# 1.2

## Cases

### Linear Search

**Best Case**

Searching 32

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 32 | 1 | 10 | 50 | 90 | 55 | 60 | 22 | 15 | 80 |

**Average Case**

Searching 32

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 25 | 50 | 60 | 32 | 90 | 70 | 22 | 15 | 80 |

**Worst Case**

Searching 32

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 25 | 50 | 60 | 80 | 90 | 70 | 22 | 15 | 32 |

Random numbers up to 10,000 are inserted and then the element is searched. The time efficiency was 14.713 which means it took this much time.

### Binary Search

**Best Case**

Ascending order Searching 22

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | 16 | 18 | 20 | 22 | 68 | 69 | 75 | 85 | 92 |

**Average Case**

Ascending order Searching 25

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 22 | 25 | 26 | 40 | 42 | 45 | 69 | 80 | 85 |

**Worst Case**

Ascending order Searching 90

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 22 | 25 | 40 | 45 | 50 | 55 | 65 | 70 | 90 |

Random numbers up to 10,000 are inserted and then are sorted in ascending order for binary search and then the element is searched. The time efficiency was 10.49 which means it took this much time.

### Bubble Sort

**Best Case**

Sort in Ascending Order

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 64 | 8 | 128 | 912 | 1028 | 2000 | 4000 | 6000 |

**Average Case**

Sort in Ascending Order

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 30 | 50 | 44 | 70 | 45 | 51 | 60 | 65 | 95 |

**Worst Case**

Sort in Ascending Order

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 90 | 30 | 20 | 80 | 50 | 85 | 75 | 70 | 21 | 22 |

Random numbers up to 10,000 are inserted and then are sorted in descending order. The time efficiency was 8.663 which means it took this much time.

### Selection Sort

**Best Case**

Sort in Ascending Order

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 8 | 4 | 64 | 108 | 128 | 192 |

**Average Case**

Sort in Ascending Order

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 8 | 4 | 128 | 108 | 164 | 192 |

**Worst Case**

Sort in Ascending Order

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 128 | 64 | 8 | 192 | 108 | 4 | 1 | 2 |

Random numbers up to 10,000 are inserted and then are sorted in ascending order. The time efficiency was 8.337 which means it took this much time.

### Insertion Sort

**Best Case**

Sort in Ascending Order

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 1 | 28 | 48 | 64 | 108 | 128 | 192 |

**Average Case**

Sort in Descending Order

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 192 | 108 | 64 | 54 | 60 | 4 | 3 | 2 |

**Worst Case**

Sort in Ascending Order

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 192 | 128 | 8 | 108 | 64 | 2 | 4 | 1 |

Random numbers up to 10,000 are inserted and then are sorted in ascending order. The time efficiency was 8.146 which means it took this much time.

# 1.3

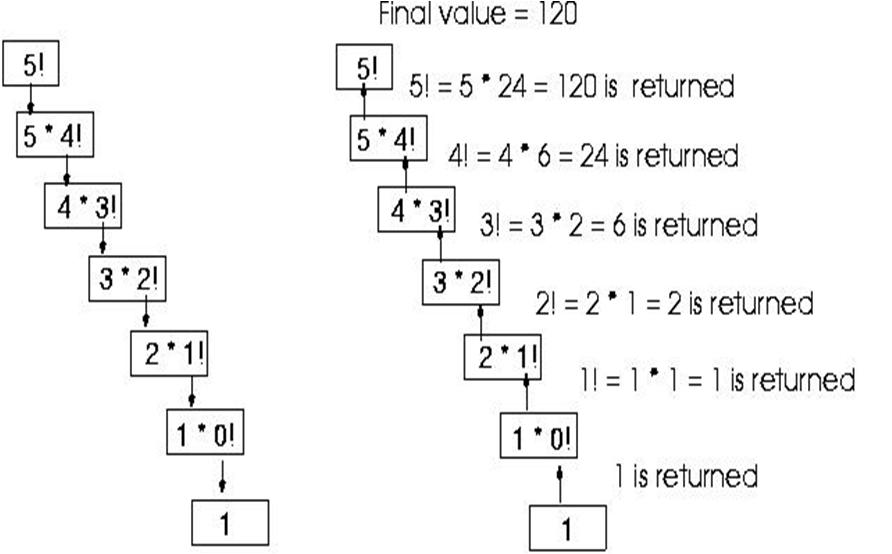
## Recursion

**“**The function which either calls itself or calls the function cycle that is mentioned in it is called Recursion. There are two types of cases in it that are base case and a recursive case.**”** (cs.wisc, 2016)

### Base Case

**“**When an answer is known for any case and also that is expressed without any recursion is known as base case.**”**

### Recursive Case

**“**When an answer is known for any case and it is expressed with the help of recursion is known as recursive case.**”**

Recursive Trace Figure 1 (Prakash, 2015)

### Merge Sorting

### 

### 

### 

#include <iostream>

#include <conio.h>

using namespace std;

void merge(int num[],int l,int m, int r)

{

int i,j,k;

int h1=m-l+1;

int h2=r-m;

int L[h1],R[h2];

for (i=0;i<h1;i++)

{

L[i]=num[l+i];

}

for (j=0;j<h2;j++)

{

R[j]=num[m+1+j];

}

i=0,j=0,k=l;

while (i<h1 && j<h2)

{

if (L[i]<=R[j])

{

num[k]=L[i];

i++;

}

else

{

num[k]=R[j];

j++;

}

k++;

}

while (i<h1)

{

num[k]=L[i];

i++;

k++;

}

while (j<h2)

{

num[k]=R[j];

j++;

k++;

}

}

void mergesort(int num[],int l,int r)

{

if (l<r)

{

int m=(l+r)/2;

mergesort(num,l,m);

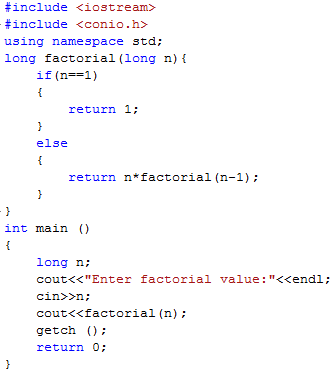
mergesort(num,m+1,r);

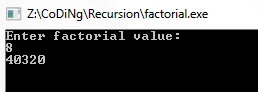
merge(num, l,m, r);

}

}

### (Factorial Recursion)

****

****

long factorial(long n){

if(n==1)

{

return 1;

}

else

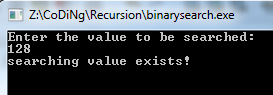
{

return n\*factorial(n-1);

}

}

### (Binary Search Recursion)

****

Apply insertion sort before binary search

long binarysearch(int num[], int key, int first, int last)

{

if (first<=last)

{

int mid=(first+last)/2;

if (num[mid]==key)

{

return mid;

}

else if (num[mid]>key)

{

return binarysearch(num,key,first,mid-1);

}

else

{

return binarysearch(num,key,mid+1,last);

}

}

else

{

return -1;

}

}

### Comparison Between Recursion and Iteration

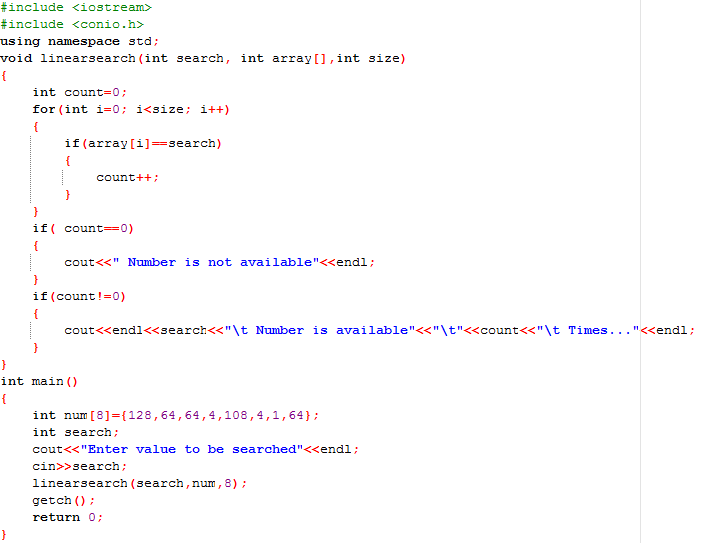
|  |  |  |
| --- | --- | --- |
| Factors | Recursion | Iteration |
| Type | Conditional | Incrementing and decrementing |
| Repetition | Within the function | Within the loop |
| Working | Base and Recursive case | Within the loop brackets |
| To execute infinite loop | If the base case is not available | If the condition stays true and does not becomes false |
| Way of representing | Two Cases i.e. two conditions | Just one loop with a condition |

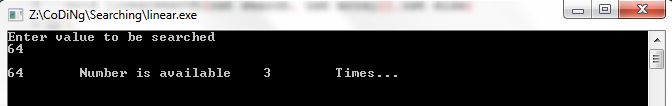
|  |  |  |
| --- | --- | --- |
| Example | Binary search: | Binary Search: |

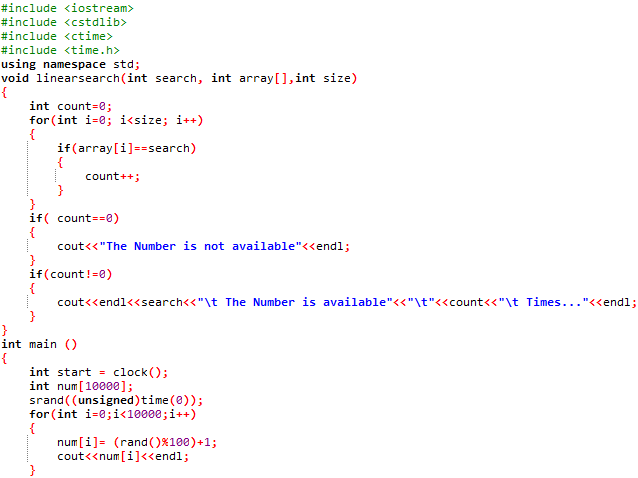
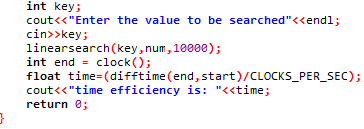
# 2.1

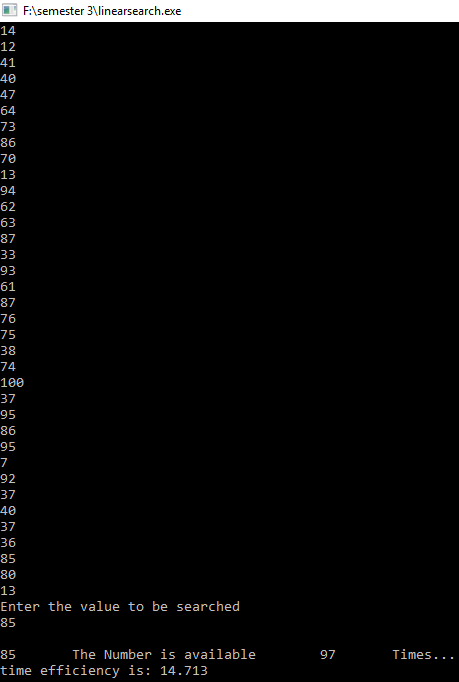
## Function

### Linear Search



**Random Numbers**

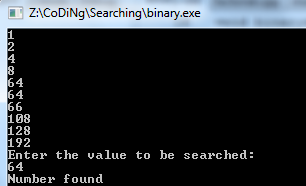


void linearsearch(int search, int array[],int size)  
{  
    int count=0;  
    for(int i=0; i<size; i++)  
    {  
        if(array[i]==search)  
        {  
            count++;  
        }  
    }  
    if( count==0)  
    {  
        cout<<**“**The Number is not available**”<<**endl;  
    }  
    if(count!=0)  
    {  
        cout<<endl<<search<<**“**\t The Number is available**”**<<**“**\t**”**<<count<<**“**\t Times...**”**<<endl;  
    }

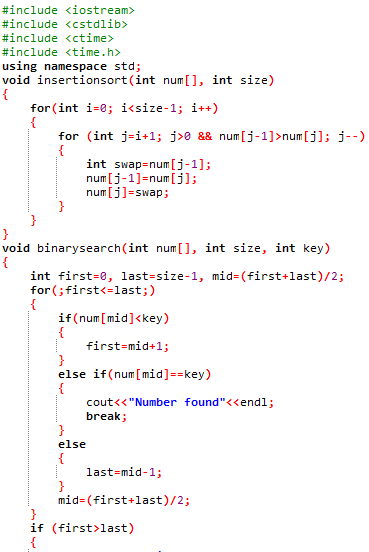
}

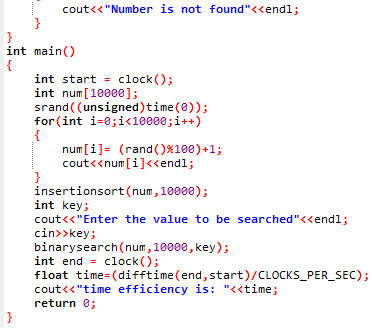
### Binary Search

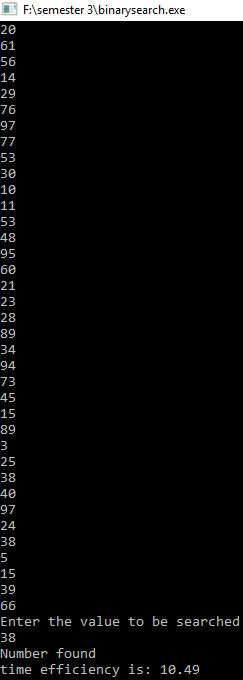




**Random Numbers**







Insertion sort .

.is applied before binary search i.e. void insertionsort(int num[], int size)

void binarysearch(int num[], int size, int key)

{

int first=0, last=size-1, mid=(first+last)/2;

for(;first<=last;)

{

if(num[mid]<key)

{

first=mid+1;

}

else if(num[mid]==key)

{

cout<<"Number found"<<endl;

break;

}

else

{

last=mid-1;

}

}

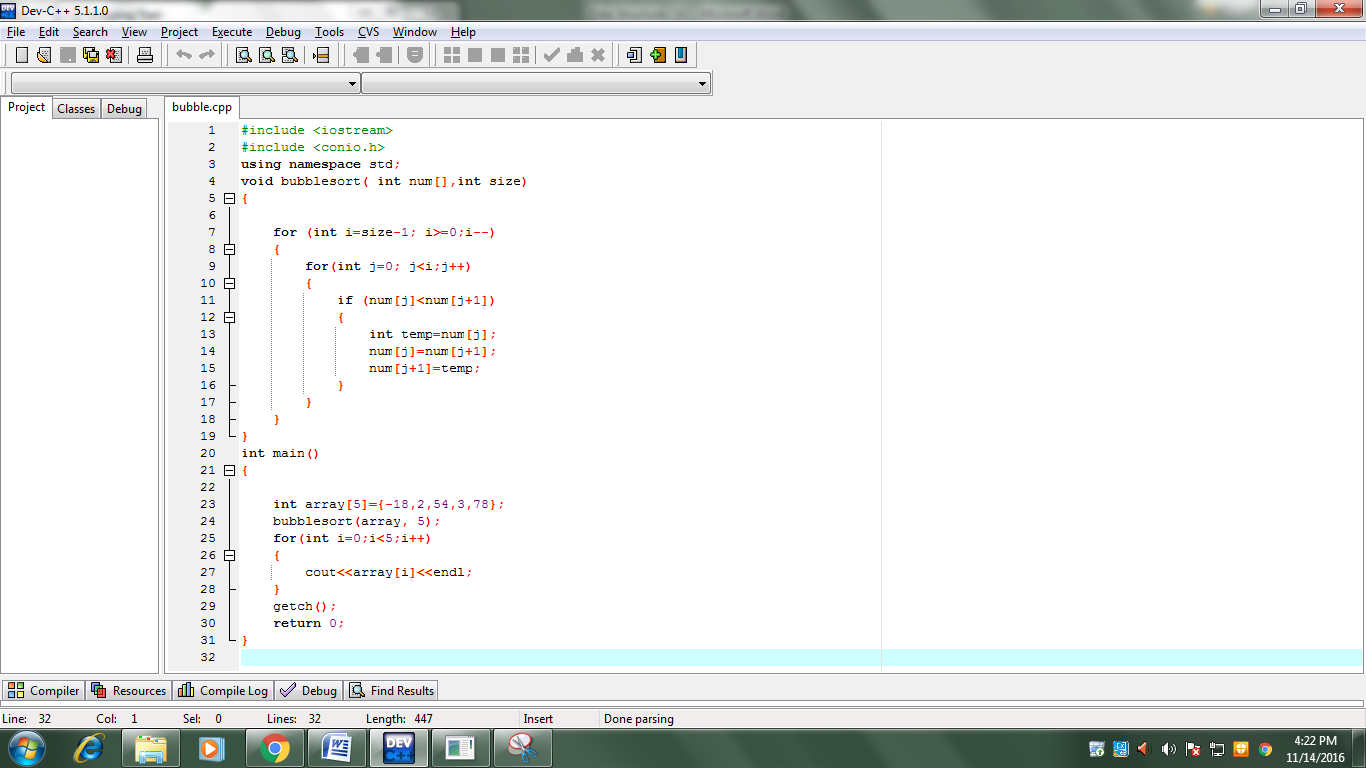
if (first>last)

{

cout<<"Number is not found"<<endl;

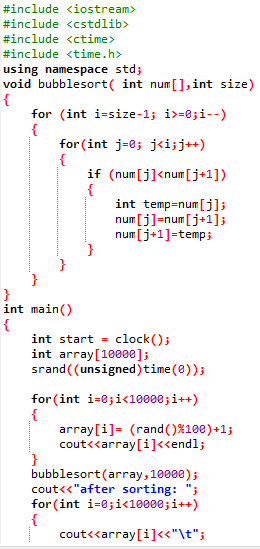
}}

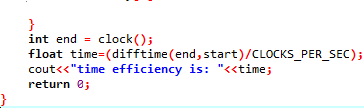
### Bubble Sort

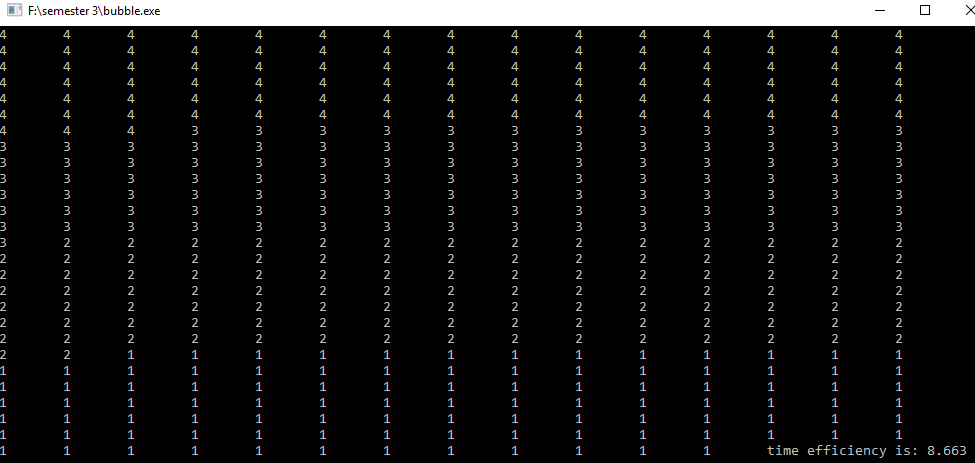




**Random Numbers**







void bubblesort( int num[],int size)

{

for (int i=size-1; i>=0;i--)

{

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

{

if (num[j]<num[j+1])

{

int temp=num[j];

num[j]=num[j+1];

num[j+1]=temp;

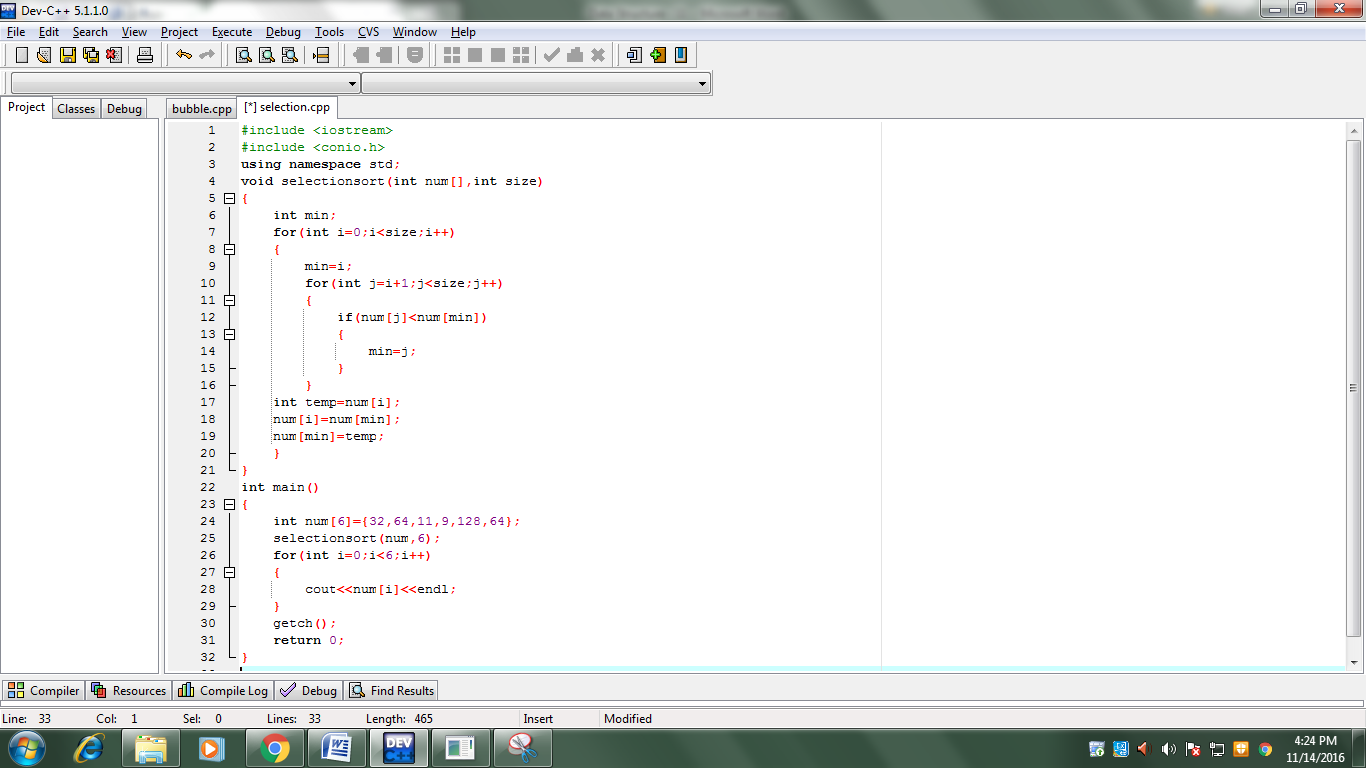
}

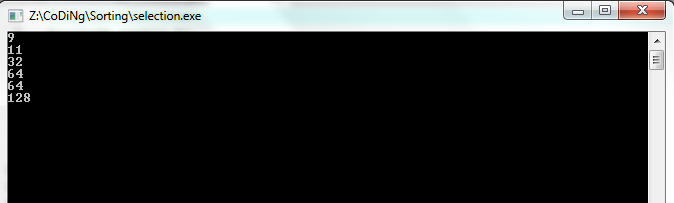
}

}

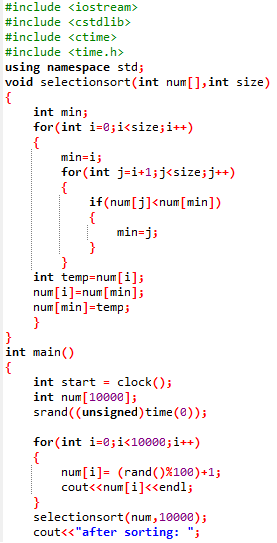
}

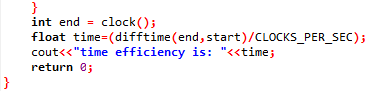
### Selection Sort

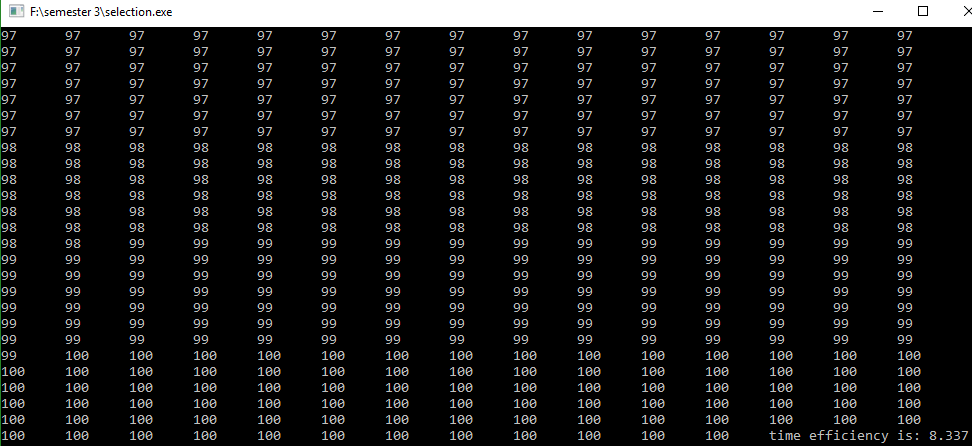




**Random Numbers**

****

****

****

void selectionsort(int num[],int size)

{

int min;

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

{

min=i;

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

{

if(num[j]<num[min])

{

min=j;

}

}

int temp=num[i];

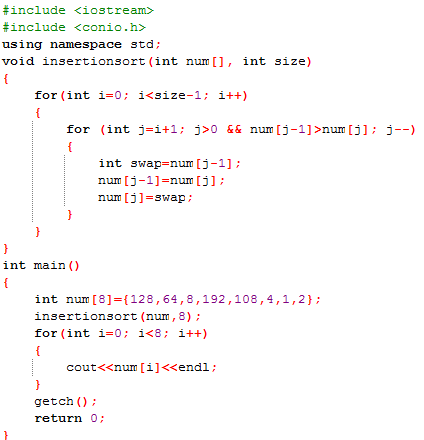
num[i]=num[min];

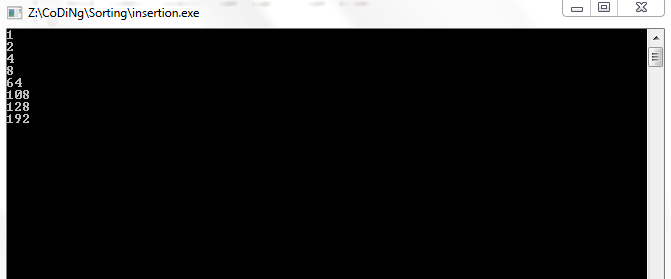
num[min]=temp;

}

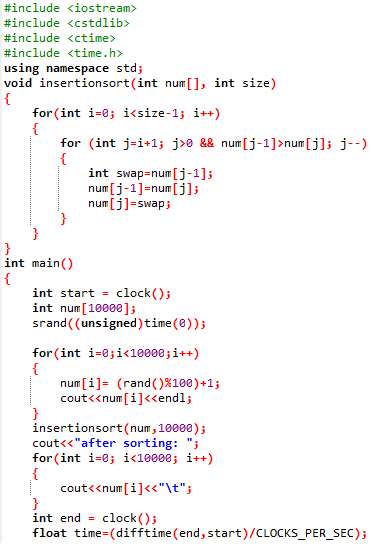
}

### Insertion Sort

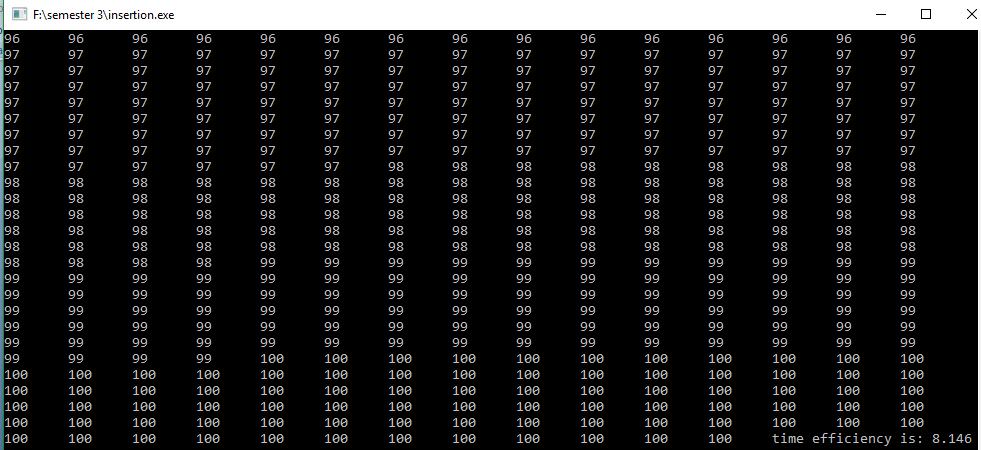




**Random Numbers**

****

****

****

void insertionsort(int num[], int size)

{

for(int i=0; i<size-1; i++)

{

for (int j=i+1; j>0 && num[j-1]>num[j]; j--)

{

int swap=num[j-1];

num[j-1]=num[j];

num[j]=swap;

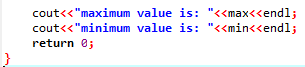
}

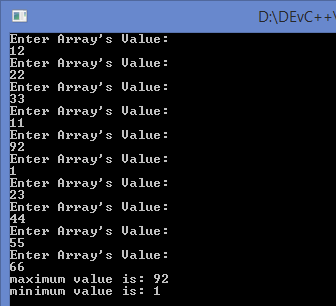
}

}

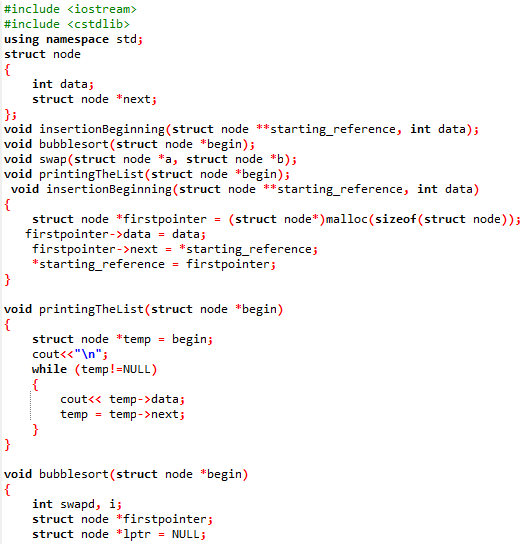
### Minimum and Maximum Value

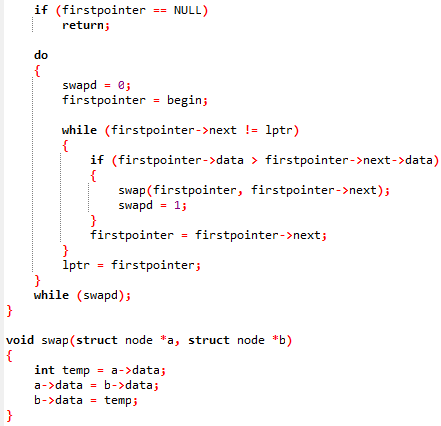
### 

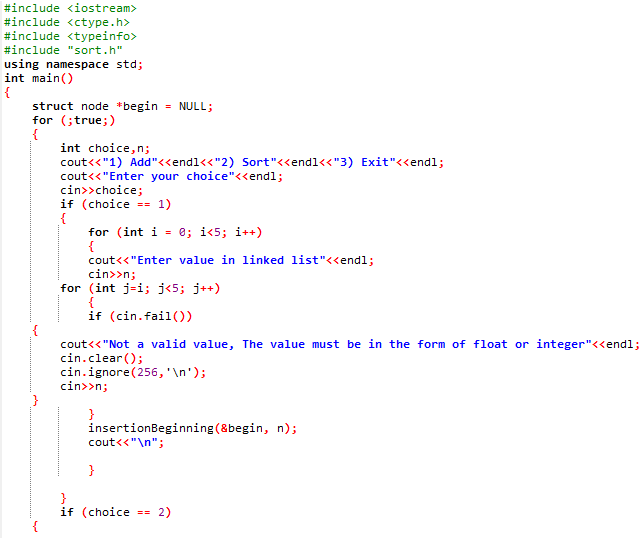


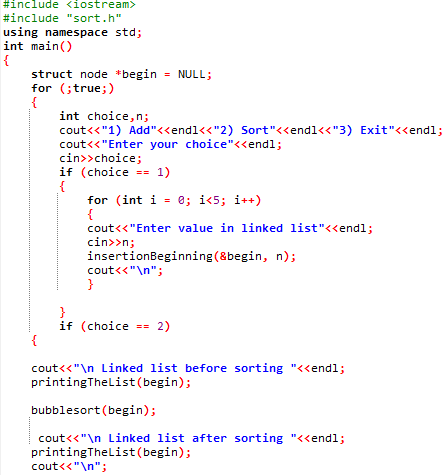


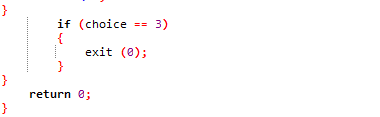
### Linked List Sorting

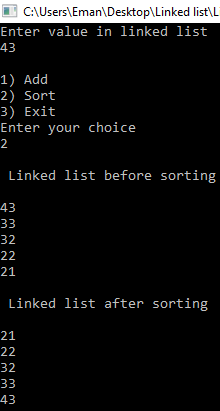




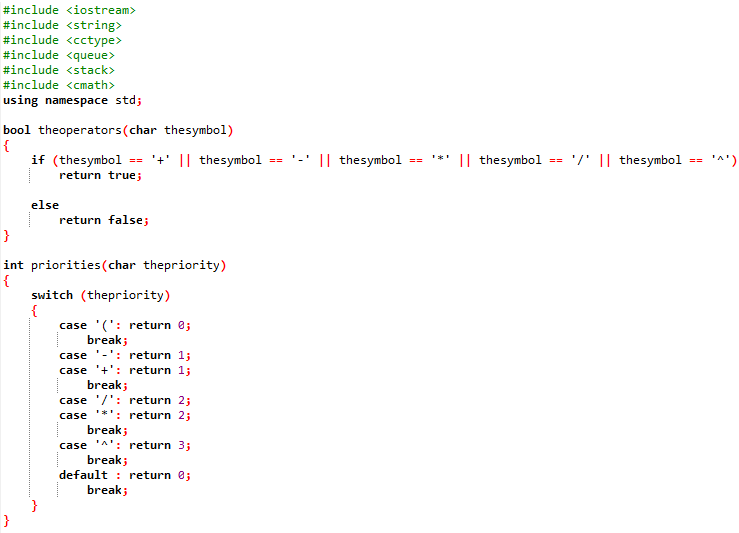
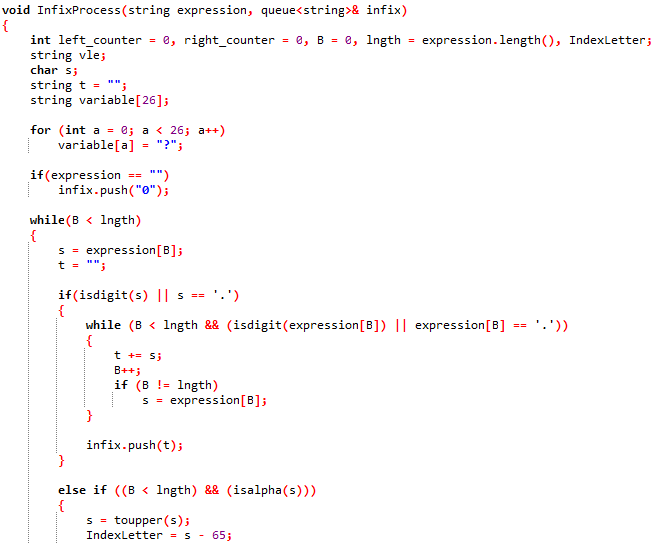
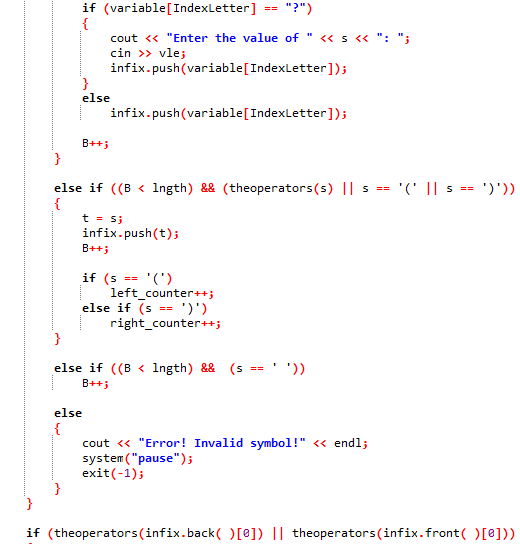


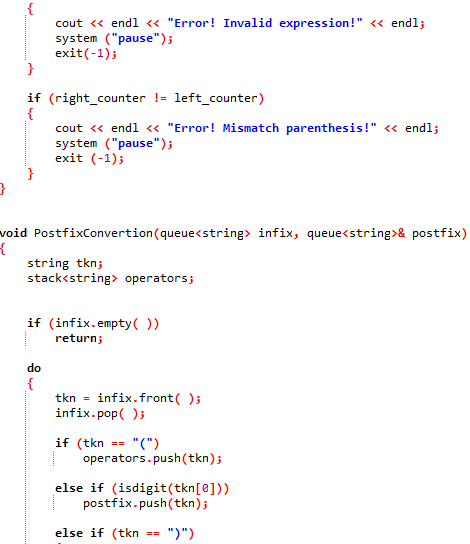


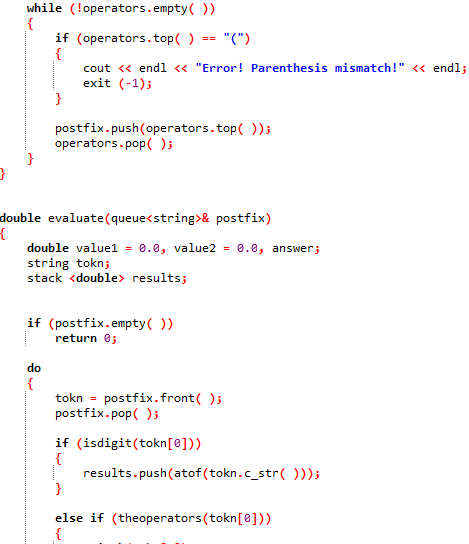
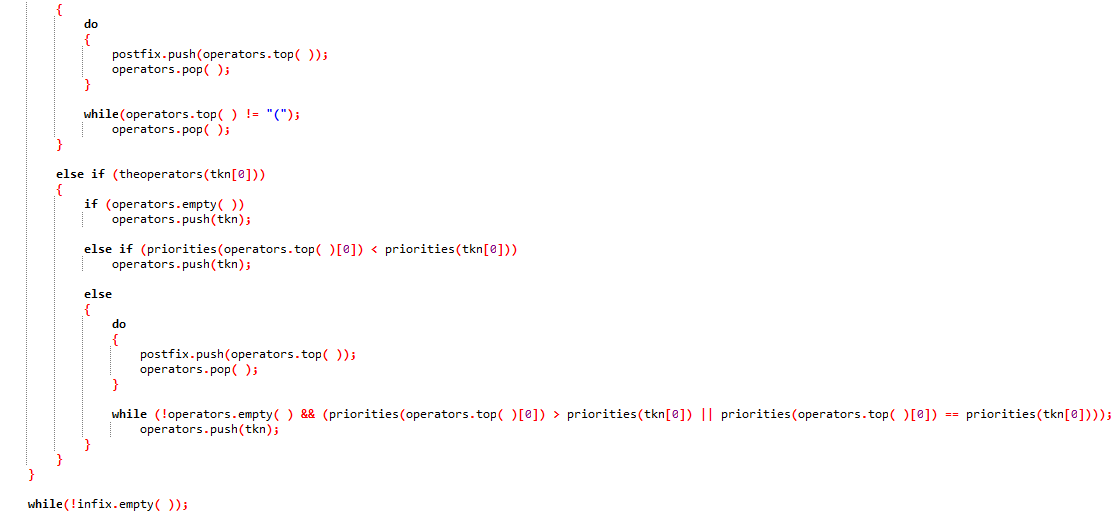


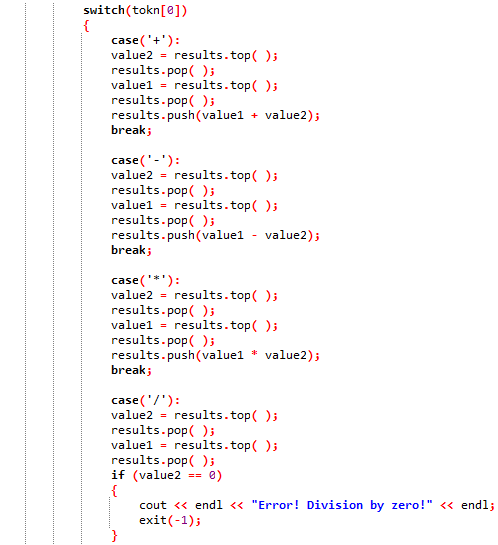


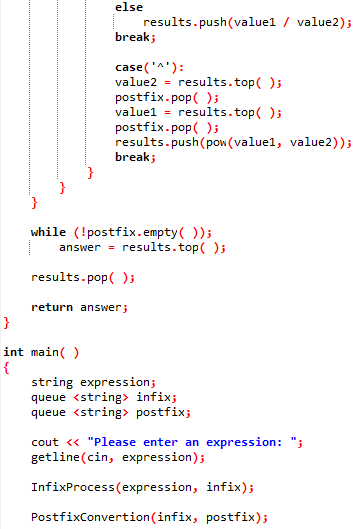
### Stack and Queue Infix to Postfix (Arithmetic Expression)

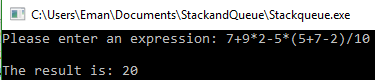










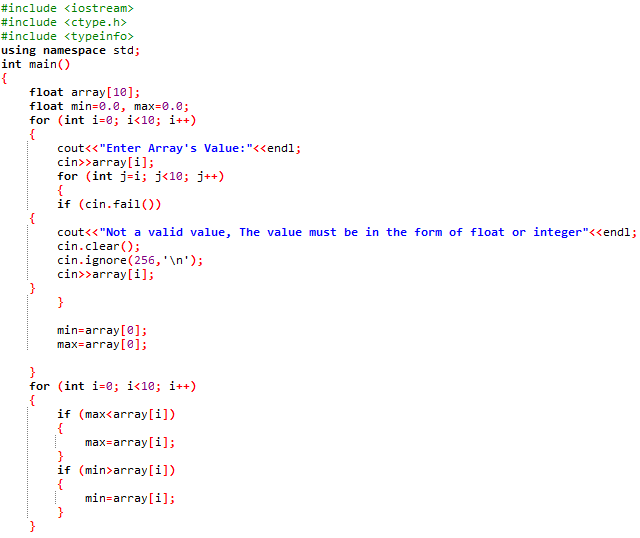


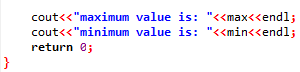
# 2.2

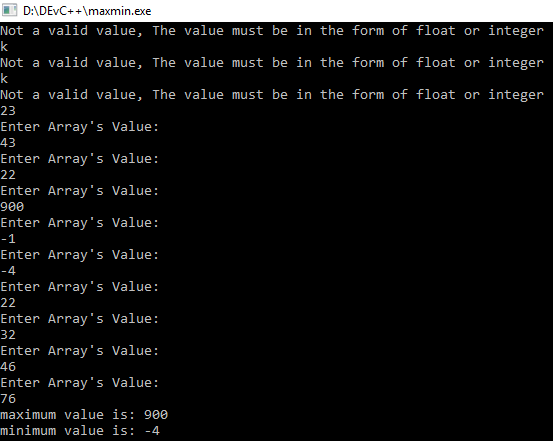
## Errors Handling

### For MinMax

The float is used instead of int because if the data is entered in float the int data type won’t display it and if the float data type is used the data whether mentioned in decimal or whole value i.e. int value the value will be displayed. To check the data type the these functions cin.fail() cin.clear() cin.ignore(256,'\n') are used so if the character is entered the error is displayed to enter the integer value.

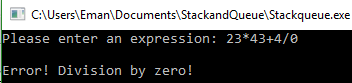


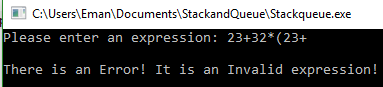


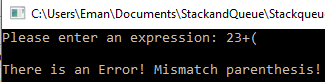


### For Queue and Stack

For error handling the functions are made like value if divide by zero is given the error will be displayed and if the parenthesis are wrong the parenthesis error will be displayed if the expression given is invalid then error will be displayed and if the character is mentioned in the expression then the program will ask the user enter the value of that character and if the symbol mentioned is wrong then the error with respect to symbol will be displayed.

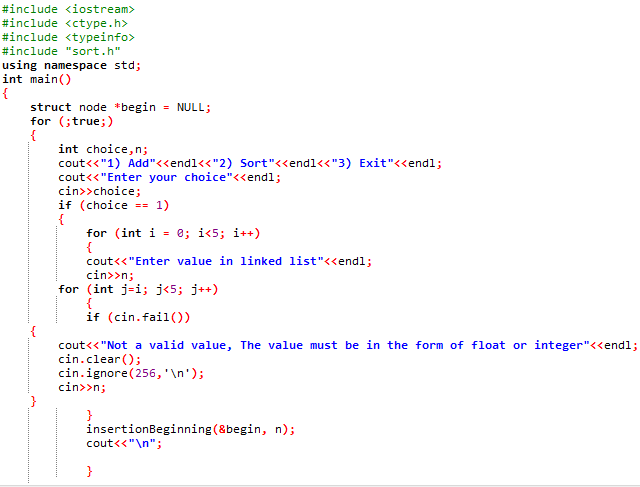


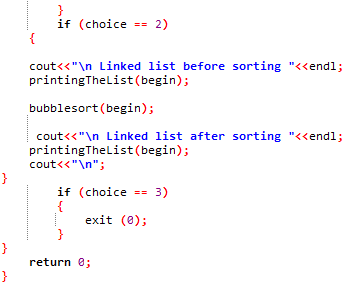


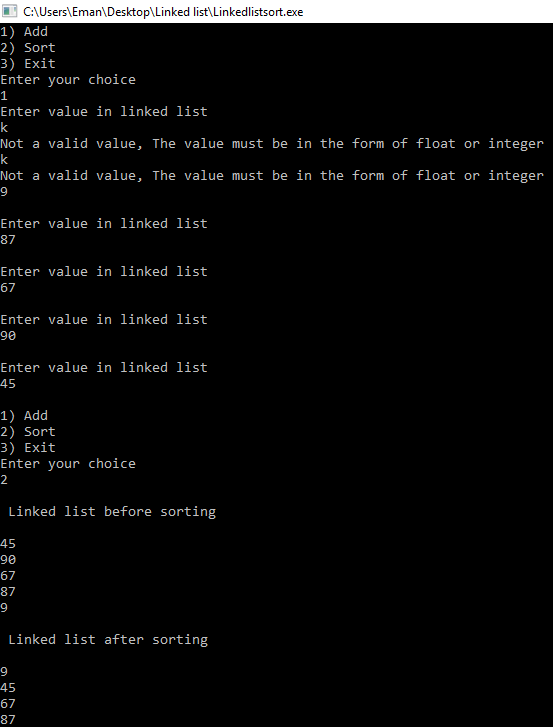


### Linked List Sorting

For error handling the functions to check the data type cin.fail() cin.clear() cin.ignore(256,'\n') are used so if the character is entered the error is displayed to enter the integer or float value.









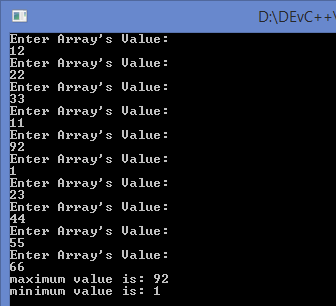
# 2.3

## Actual and Expected Results

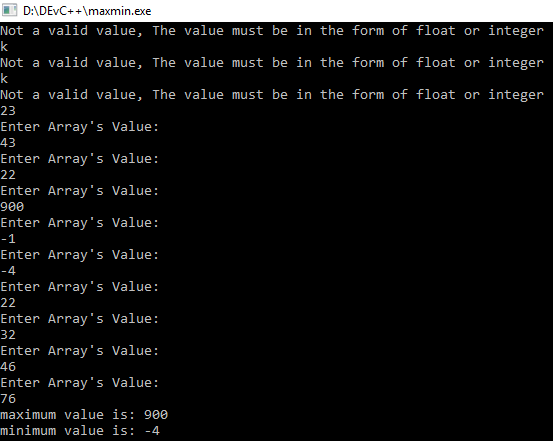
|  |  |  |  |
| --- | --- | --- | --- |
| Program | Type of Testing | Expected | Actual |
| MinMax | Checking max and min | Max = 92 Min = 1 | Max = 92 Min = 1 |
|  | Checking the error handling | Display error if character is entered | Error is displayed when character is entered |
| Stack and Queue Infix to Postfix (arithmetic expression) | Checking the result | Expression:  7+9\*2-5\*(5+7-2)/10  Result = 20 | Expression:  7+9\*2-5\*(5+7-2)/10  Result = 20 |
|  | Checking wrong expression error | Display error if expression is entered is wrong | Error is displayed when the expression entered is wrong |
|  | Checking parenthesis mismatch error | Display error if parenthesis is mismatched | Error is displayed when the parenthesis is mismatched |
|  | Checking division by zero error | Display error if division by zero is performed | Error is displayed when division by zero is performed |
| Linked List Sorting  (Ascending Order) | Checking the sorting | Values Inserted  1, 32, 23, 22, 67 and 90  Sequence:  90, 67, 22, 23, 32 and 1  Sorted in Ascending Order:  1, 22, 23, 32, 67, 90 | Values Inserted  1, 32, 23, 22, 67 and 90  Sequence:  90, 67, 22, 23, 32 and 1  Sorted in Ascending Order:  1, 22, 23, 32, 67, 90 |
|  | Checking the error handling | Display error if character is entered | Error is displayed when the character is entered |

**MINMAX**

**Checking Max and Min**

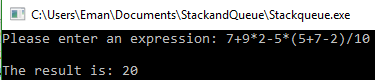


**Checking Error Handling**

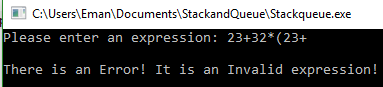
****

**Stack and Queue Infix to Postfix (arithmetic expression)**

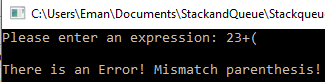
**Checking the Result**

****

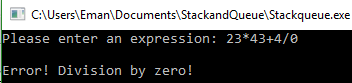
**Checking Wrong Expression Error**

****

**Checking Parenthesis Mismatch Error**

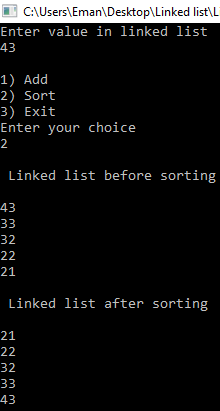
****

**Checking Division by Zero**

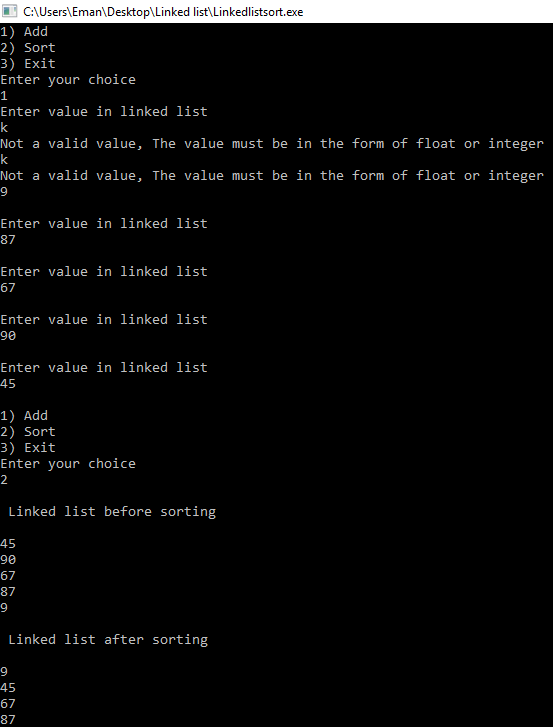
****

**Linked List Sorting (Ascending Order)**

**Checking the Sorting**

****

**Checking the Error Handling**

****

****

# Self-Criticism | Self-Evaluation

## Self-Criticism

Criticizing of the work is done very effectively. The algorithms and pseudo codes of the searching and sorting could be done more effectively. The software developer has given the algorithms and pseudo codes but they are not done effectively because of this the readers won’t be able to understand the algorithms and pseudo codes more efficiently and perfectly.

The cases that have been mentioned in the report could be done more efficiently like data entered in the cases could be more i.e. the software developer has mentioned like eight indexes he could mention about ten or more than ten cases so it could be understandable for the reader more efficiently.

The topic of recursion could be more explained than it is mentioned in the report by the software developer. Base case could be elaborated more than it is mentioned in the report. The functions mentioned in the report could be given in more detail so the reader could understand the topic easily.

The programs mentioned in the report could be more efficient so every person who reads the report could understand the topics and the working of the program easily and efficiently.

## Self-Evaluation

The topics mentioned in the report could be explained more efficiently. The software developer has used some articles, books and pdf files to write the report like mentioning the details about the topics according to the assignment brief. The software developer has done all the work according to scenario given to him.

The tools and techniques used for the report are very effective in writing the report according to the scenario given to him. More references could be added for more understanding of the reader. As we know that more the references and more the understanding is done. The references mentioned in the report are very authentic. If the references could be more then the report could be more authentic than before. According to the researcher the work for distinction and merit is done very effectively so he is sure about the distinction and merit. The programs given in the brief are done very efficiently so it can be useful for the reader to understand the programs easily. For merit random numbers are added in searching and sorting programs.

# Bibliography

<http://ijcsit.com/docs/Volume%205/vol5issue02/ijcsit20140502215.pdf>

<http://www.cplusplus.com/articles/D2N36Up4/>

[cs.wisc. (2016).](cs.wisc. (2016). RECURSION. Retrieved from cs.wisc: pages.cs.wisc.edu/~calvin/cs110/RECURSION.htmlPrakash, S. (2015, SEPTEMBER 14 ). C++ Programs. Retrieved from cppprogramz: http://2.bp.blogspot.com/-skGSb5QxmYQ/VfcWH-DCu5I/AAAAAAAAAYo/VT4xlZNahSI/s1600/recursive-example.jpg) *[RECURSION.](cs.wisc. (2016). RECURSION. Retrieved from cs.wisc: pages.cs.wisc.edu/~calvin/cs110/RECURSION.htmlPrakash, S. (2015, SEPTEMBER 14 ). C++ Programs. Retrieved from cppprogramz: http://2.bp.blogspot.com/-skGSb5QxmYQ/VfcWH-DCu5I/AAAAAAAAAYo/VT4xlZNahSI/s1600/recursive-example.jpg)* [Retrieved from cs.wisc: pages.cs.wisc.edu/~calvin/cs110/RECURSION.html](cs.wisc. (2016). RECURSION. Retrieved from cs.wisc: pages.cs.wisc.edu/~calvin/cs110/RECURSION.htmlPrakash, S. (2015, SEPTEMBER 14 ). C++ Programs. Retrieved from cppprogramz: http://2.bp.blogspot.com/-skGSb5QxmYQ/VfcWH-DCu5I/AAAAAAAAAYo/VT4xlZNahSI/s1600/recursive-example.jpg)

[Prakash, S. (2015, SEPTEMBER 14 ).](cs.wisc. (2016). RECURSION. Retrieved from cs.wisc: pages.cs.wisc.edu/~calvin/cs110/RECURSION.htmlPrakash, S. (2015, SEPTEMBER 14 ). C++ Programs. Retrieved from cppprogramz: http://2.bp.blogspot.com/-skGSb5QxmYQ/VfcWH-DCu5I/AAAAAAAAAYo/VT4xlZNahSI/s1600/recursive-example.jpg) *[C++ Programs.](cs.wisc. (2016). RECURSION. Retrieved from cs.wisc: pages.cs.wisc.edu/~calvin/cs110/RECURSION.htmlPrakash, S. (2015, SEPTEMBER 14 ). C++ Programs. Retrieved from cppprogramz: http://2.bp.blogspot.com/-skGSb5QxmYQ/VfcWH-DCu5I/AAAAAAAAAYo/VT4xlZNahSI/s1600/recursive-example.jpg)* [Retrieved from cppprogramz: http://2.bp.blogspot.com/-skGSb5QxmYQ/VfcWH-DCu5I/AAAAAAAAAYo/VT4xlZNahSI/s1600/recursive-example.jpg](cs.wisc. (2016). RECURSION. Retrieved from cs.wisc: pages.cs.wisc.edu/~calvin/cs110/RECURSION.htmlPrakash, S. (2015, SEPTEMBER 14 ). C++ Programs. Retrieved from cppprogramz: http://2.bp.blogspot.com/-skGSb5QxmYQ/VfcWH-DCu5I/AAAAAAAAAYo/VT4xlZNahSI/s1600/recursive-example.jpg)

<http://www.techtud.com/sites/default/files/public/share/algorithm-book-by-karumanchi.pdf>

<http://journalspub.com/journalspub/JournalsDetails.aspx?jid=104>