PROJECT REPORT

ON

"Binary tree implementation & comparison of sorting algorithms"

(FOR DMSRDE)

DEFENCE MATERIALS & STORES RESEARCH & DEVELOPMENTESTABLISHMENT, KANPUR

PROJECT GUIDE:

Mr. Sanjeev Kumar (Sc. 'E')

SUBMITTED BY:

IRTEZA ALI NAQVI
B.Tech II Year
(Roll No 1605210021)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INSTITUTE OF ENGINEERING AND TECHNOLOGY, LUCKNOW

Preface

The innovation of computers has drastically effected organization of all kinds. Dependence on computers is increasing day by day with the exponential growth in IT industry, increase in E- commerce, E-Payment, Data Science etc.

Data structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way. Data structures are about rendering data elements in terms of some relationship, for better organization and storage.

Sorting is nothing but storage of data in a sorted manner. Sorting arranges data in a manner which makes searching easier. *I have been assigned the project topic "Binary tree implementation & comparison of sorting algorithms" in this esteemed Organization.*

.

ACKNOWLEDGEMENT

I feel greatly indebted to all those who helped me during the project work. I avail this opportunity for thanking them. I sincerely express my indebtedness and gratitude to **Dr N. Eshwara Prasad,** Director DMSRDE (DRDO), for giving me an opportunity to enhance my skills and allowing me to join this esteemed organization.

This acknowledgement would be incomplete if I fail to express deep gratitude to Mr**. Sanjeev Kumar** for his valuable guidance, patience and consummate support. Finally, I would like to thank Technical Coordination Section and whole staff of Administration D.M.S.R.D.E., Kanpur for the valuable support they rendered and for providing such a stimulating environment in which the project development was done with style and verve.

Phone : 2451759-78
Gram : LABDEV
Fax : 0512-2450404
Fax : 0512-2404774

No. PR/0573/
GOVERNMENT OF INDIA,MINISTRY
OF DEFENCE RESEARCH &
DEVELOPMENT ORGANISATION
DEFENCE MATERIALS AND STORES
RESEARCH AND DEVELOPMENT
ESTABLISHMENT DMSRDE
POST OFFICE, G. T. ROAD,
KANPUR-208 013

CERTIFICATE

This is to certify that **Mr. IRTEZA ALI NAQVI B.Tech II Year** (CS) have undergone project training from 4 weeks on the project entitled: **"Binary tree implementation & comparison of sorting algorithms"** at **D.M.S.R.D.E** (**DRDO**), Kanpur.

He has successfully completed their project under the guidance of me.

Sanjeev Kumar (Sc. 'E')
Networking Cell
Govt. of India, Min. of Defence
DMSRDE, Kanpur

ORGANIZATION PROFILE

The origins of the Defence Materials & Stores Research & Development Establishment (DMSRDE) can be traced back to 1929, when it was known as the Inspectorate of General Stores in the Harness & Saddlery Factor in Cawnpore (present day Kanpur). After 1929, the establishment underwent a number of reorganizations. Finally, in 1976, the Defence Materials & Stores Research & Development Establishment was established through the amalgamation of three separate establishments.

The DMSRDE is involved in interdisciplinary research and development of materials for the military services, including various types of chemical protective clothing and equipment. Work at the DMSRDE laboratory in Kanpur includes the synthesis and development of polymers and composite materials, as well as research and development of protective clothing and equipment against hazardous and toxic chemicals. In partnership with the Defence Research and Development Establishment (DRDE), the DMSRDE has produced five types of protective systems and equipment that have been introduced into the services. These include nuclear, biological, and chemical (NBC) individual protection equipment, NBC collective protection system, NBC medical protection equipment, NBC detection equipment, and a NBC decontamination system.

The DMSRDE Kanpur laboratory interacts with the three military services, as well as a number of academic institutions, including the Indian Institute of Technology, Kanpur, and the Indian Institute of Technology, Mumbai.

INTRODUCTION

Organization requires the services of a large number of personnel. D.M.S.R.D.E involves in R&D, is a DRDO lab working in non-metallic materials, textiles and critical scientific and technological area. It has strong manpower consisting of around 200 officers and 600 staffs. These personnel occupy the various position created through the process of organizing. Each position of the organization has certain specific contribution to achieve organization objectives.

Now to provide various facilities to these employees is the responsibility of Government of India and management of the former is the duty of the organization. Organization has been authorized to hold its Employee Record.

D.M.S.R.D.E. also manages the records of offered courses, Higher Degree, Junior Research Fellowship, Senior Research Fellowship etc. The management of data involves both the definition of data structures for the storage of information and provision of mechanism for the information.

Sorting is a process of arranging data in a way that makes it easier to search data within these stored records.

All the above mentioned work can be done using computer to endeavor fast and secure retrieval of information. For the above work, I have used most common open source platform named CodeBlocks. A GCC compiler compiles the code written in C to give the desired output.

TOOLS AND PLATFORM: -

Designed by using …

Open source IDE:	CodeBlocks
Operating System:	MS Windows's 10
Processor:	Intel i7

Some Attractive features of project

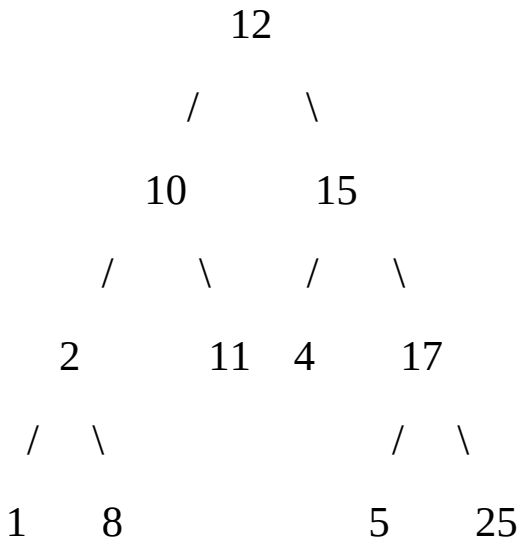
- *File input / output
- *Sort up to 1 lakh items.
- *Time efficient
- *More flexible & reliable

Binary tree

In a normal tree, every node can have any number of children. Binary tree is a special type of tree data structure in which every node can have maximum of 2 children. One known as the left child and other known as the right child. In computing, binary trees are seldom solely used for their structure. Storage of data in a hierarchical manner is possible using binary trees.

As we keep on entering the elements, the left sub child holds lower value as compared to parent node and right sub child holds greater value as compared to the parent node. In this manner the whole tree is designed.

EXAMPLE:

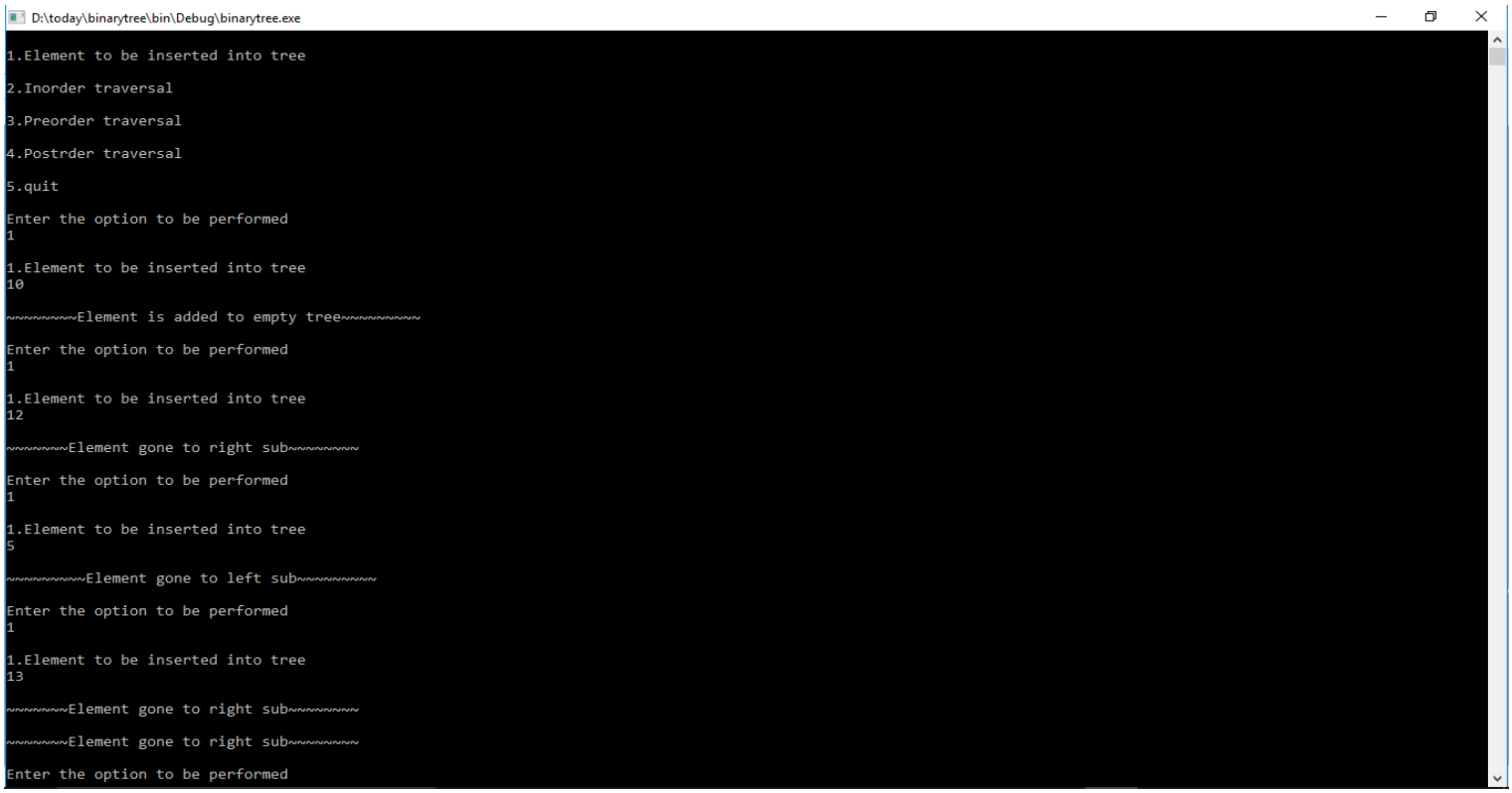


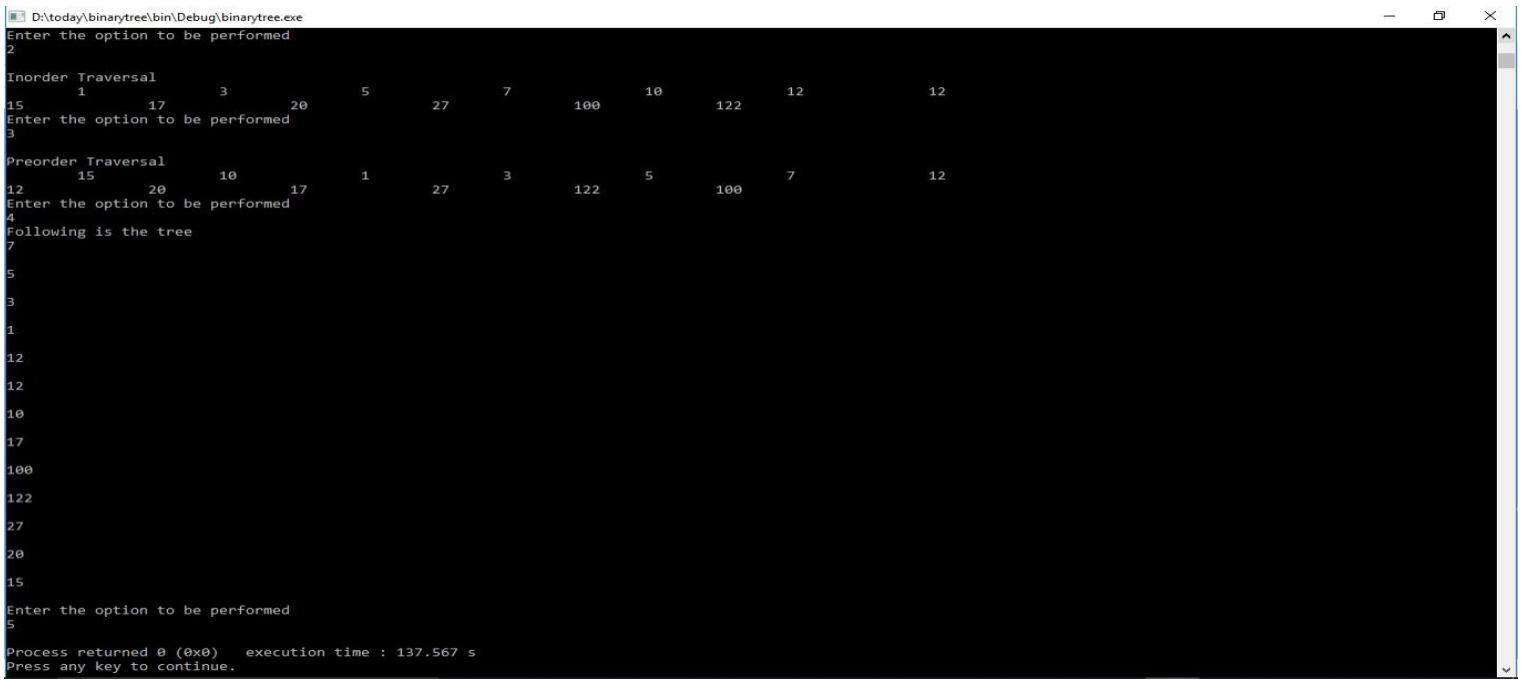
Code in C

```
#include <stdio.h>
#include <stdlib.h>
struct tree
{ int value;
 struct tree *left;
 struct tree *right; };
struct tree *root,*temp;
void insert(int element);
void inorder(struct tree *p);
void preorder(struct tree *p);
void postorder(struct tree *p);
int main()
{ root=NULL;
  int option,data,i=0;
        printf("\n1.Element to be inserted into tree\n");
        printf("\n2.Inorder traversal\n");
        printf("\n3.Preorder traversal\n");
        printf("\n4.Postrder traversal\n");
        printf("\n5.quit \n");
 do{
     printf("\nEnter the option to be performed\n");
     scanf("%d",&option);
        switch(option)
        {
        case 1 :
            {
                printf("\n1.Element to be inserted into tree\n");
                scanf("%d",&data);
                insert(data);
                break;
             }
        case 2 :
            {   temp=root;
                printf("Following is the tree");
                inorder(temp);
                break;
            }
            case 3 :
            {   temp=root;
                printf("Following is the tree");
                preorder(temp);
                break;
            }
            case 4 :
            {   temp=root;
                printf("Following is the tree");
                postorder(temp);
                break;
            }
        case 5 :
            {
                exit(0);
                break;
            }
        }
          i++;  } while(i<10000);

}
```

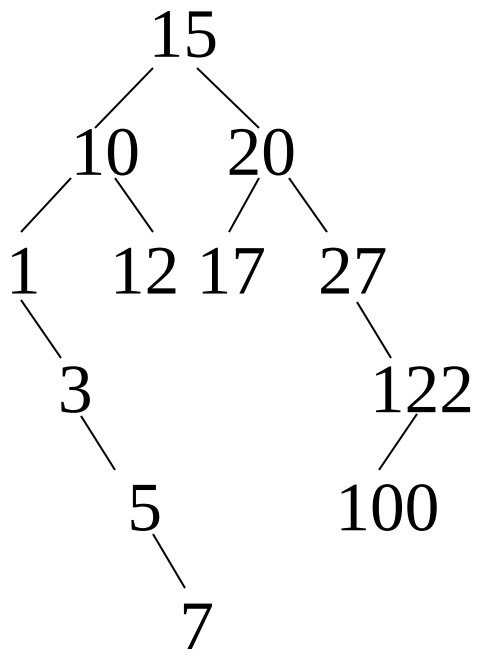
```
        temp=(struct tree*)malloc(sizeof(struct tree));

        if(temp==NULL)
        {
            printf("Memory allocation failed");

        }
        temp->value= element;
        temp->left=NULL;
        temp->right=NULL;

        if (root==NULL)
        {
            printf("\n~~~~~~~~Element is added to empty tree~~~~~~~~~\n");
            root=temp;
        }
        else
        {
            p=root;
            while(p!=NULL)
            {
                r=p;
                if(p->value==element)
                {
                    printf("\n~~~~~~~~Element already exists~~~~~~~~\n");
                      break;

                }
             else if(p->value<element)
            {
                printf("\n~~~~~~~Element gone to right sub~~~~~~~~\n");
            p=p->right;
            }
            else{
                printf("\n~~~~~~~~~Element gone to left sub~~~~~~~~~\n");
            p=p->left;
            }
        }
            if(r->value>element)
        {
            r->left=temp;

        }
        else{
            r->right=temp;
        }

        }

    }
    void inorder(struct tree *p)

    {  if(p==NULL) return;
           inorder(p->left);
        printf("\n%d\n",p->value);
        inorder(p->right);
    }
void preorder(struct tree *p)

    {  if(p==NULL) return;
           printf("\n%d\n",p->value);
           preorder(p->left);
     preorder(p->right);
    }
```

```
    void postorder(struct tree *p)

    {   if(p==NULL) return;
            postorder(p->left);
            postorder(p->right);
         printf("\n%d\n",p->value);

    }

}
```





Tree formed for current example:



Inorder: 1 ,3 ,5, 7, 10, 12, 15, 17, 20, 27 100, 122

Preorder : 15, 10, 1, 5, 7, 12, 20 ,17, 27,122, 100

Postorder: 7,5,3,1,12,10,17, 100, 122, 27, 20,15

Sorting

The arrangement of data in either ascending or descending manner in order to make searching of data easier is called sorting of data.

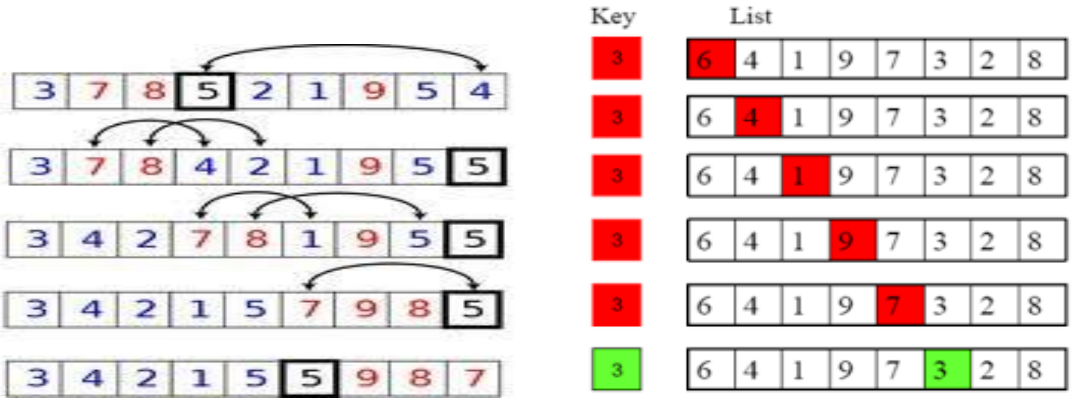
Various sorting algorithms take different time in their execution on the basis of technique used and number of comparisons done.

Comparing different sorting algorithms for time performance has always been amusing. Apart from fun, this comparison is very useful in real life. Companies and organizations need to sort and search data all the time. Thus it makes sense to compare and find the best suitable sorting method.

In the project following sorting algorithms have been chosen:

- Quick Sort
- Bubble Sort
- Insertion Sort
- Selection Sort

Sorting algorithms were mostly used during development of user interface and functionality--for ease of programming and debugging.



QUICK SORT ANALYSIS

Quick Sort is a Divide-and-Conquer Algorithm. It means, the given problem is divided into many sub-problems and we solve the sub-problems to get the solution of the actual problem.

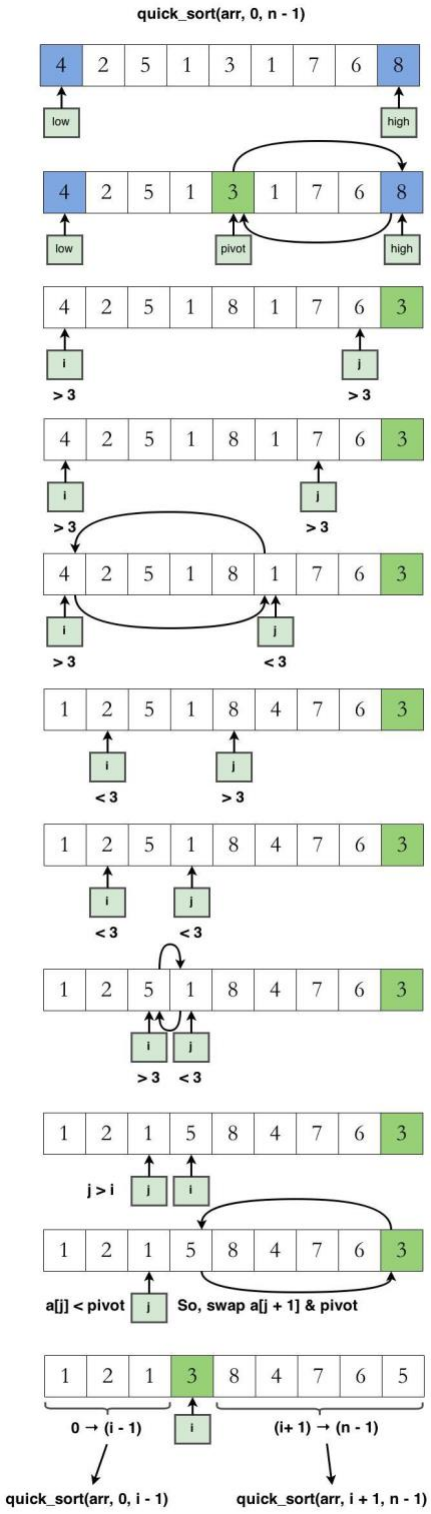
It is an in-place sorting algorithm, which means it works on the given array itself and does not need any additional space.

It is an unstable sorting algorithm, which means that the relative position of equal elements may not be maintained.

Quick Sort algorithm has the following complexities –

- Worst Case Time Complexity – $O(n^2)$
- Average Case Time Complexity – $O(n \log_2 n)$
- Best Case Time Complexity – $O(n \log_2 n)$
- Space Complexity – O(n)

The technique of quick sort is rather weird but it is straight-forward.



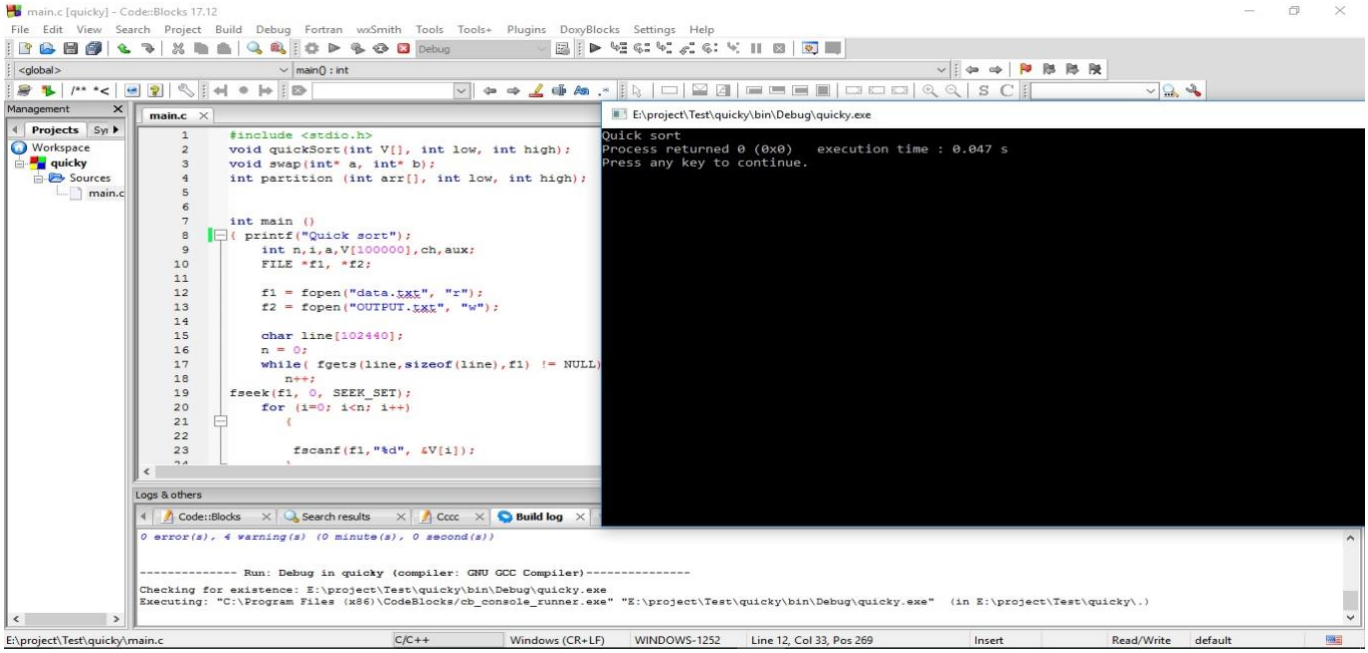
Code in C:

```c
#include <stdio.h>
void quickSort(int V[], int low,
int high);
void swap(int* a, int* b);
int partition (int arr[], int
low, int high);


int main ()
{
    int n,i,a,V[100000],ch,aux;
    FILE *f1, *f2;

    f1 = fopen("data.txt", "r");
    f2 = fopen("OUTPUT.txt",
"w");

    char line[102440];
    n = 0;
    while(
fgets(line,sizeof(line),f1) !=
NULL)
       n++;
fseek(f1, 0, SEEK_SET);
    for (i=0; i<n; i++)
       {
        fscanf(f1,"%d", &V[i]);
       }
    quickSort(V, 0, n-1);
     for (i=0; i<n; i++)
        fprintf(f2, "%d\n",
V[i]);
    fclose(f1);
    fclose(f2);
}
    void quickSort(int V[], int
low, int high)
{
    if (low < high)
    {
        int pi = partition(V,
low, high);
        quickSort(V, low, pi -
1);
        quickSort(V, pi + 1,
high);
    }
}
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
```

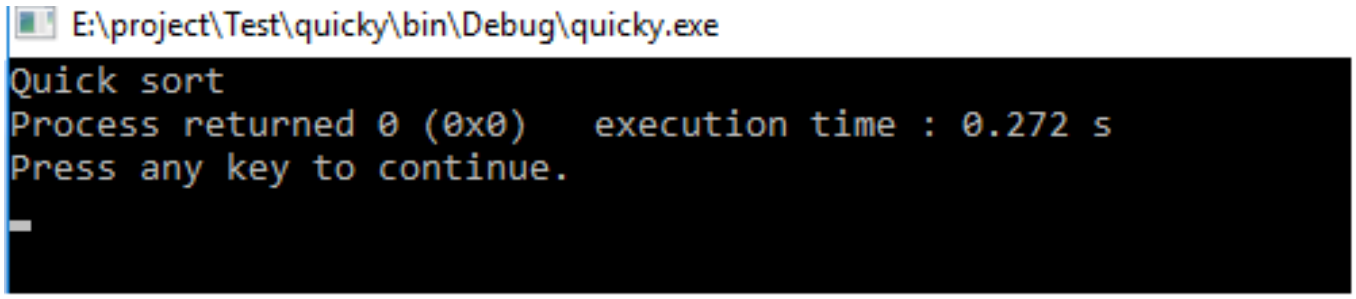
```c
int partition (int V[], int low, int
high)
{   int j;
    int pivot = V[high];
    int i = (low - 1);

    for (j = low; j <= high- 1; j++)
    {
        if (V[j] <= pivot)
        {
            i++;
            swap(&V[i], &V[j]);
        }
    }
    swap(&V[i + 1], &V[high]);
    return (i + 1);
}
```

The program inputs a file data.txt containing 100, 1000, 10,000, 15,000 etc. random integers at various intervals.The output is again in the form of a file which contains a sorted list of the integers that were input .The total time taken to sort the elements is noted down.

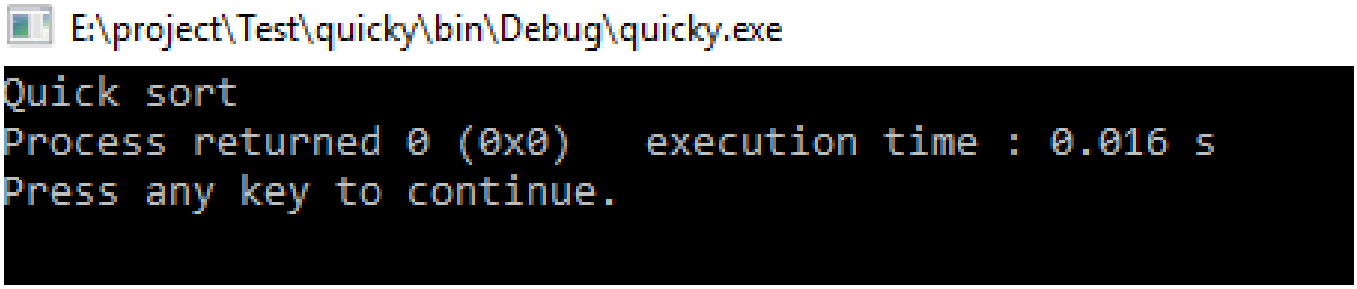
Clips of running program:



0.042 seconds for 15,000 elements



0.272 seconds for 50,000 elements



0.016 seconds for 100 elements

BUBBLE SORT ANALYSIS

Bubble sort algorithm starts by comparing the first two elements of an array and swapping if necessary, i.e., if you want to sort the elements of array in ascending order and if the first element is greater than second then, you need to swap the elements but, if the first element is smaller than second, you mustn't swap the element. Then, again second and third elements are compared and swapped if it is necessary and this process go on until last and second last element is compared and swapped. This completes the first step of bubble sort.

If there are n elements to be sorted then, the process mentioned above should be repeated n-1 times to get required result. But, for better performance, in second step, last and second last elements are not compared because; the proper element is automatically placed at last after first step. Similarly, in third step, last and second last and second last and third last elements are not compared and so on.

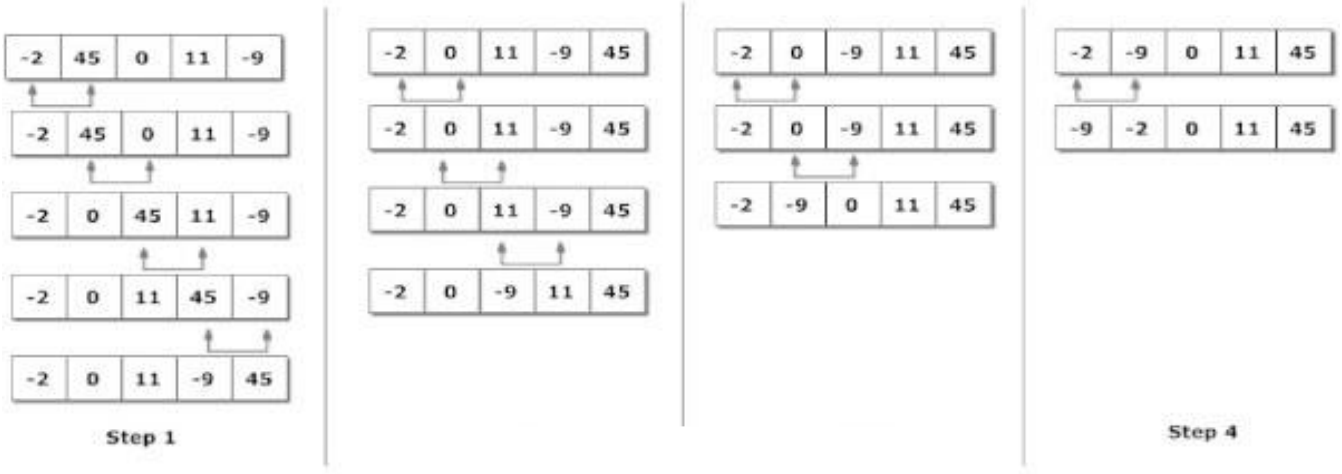


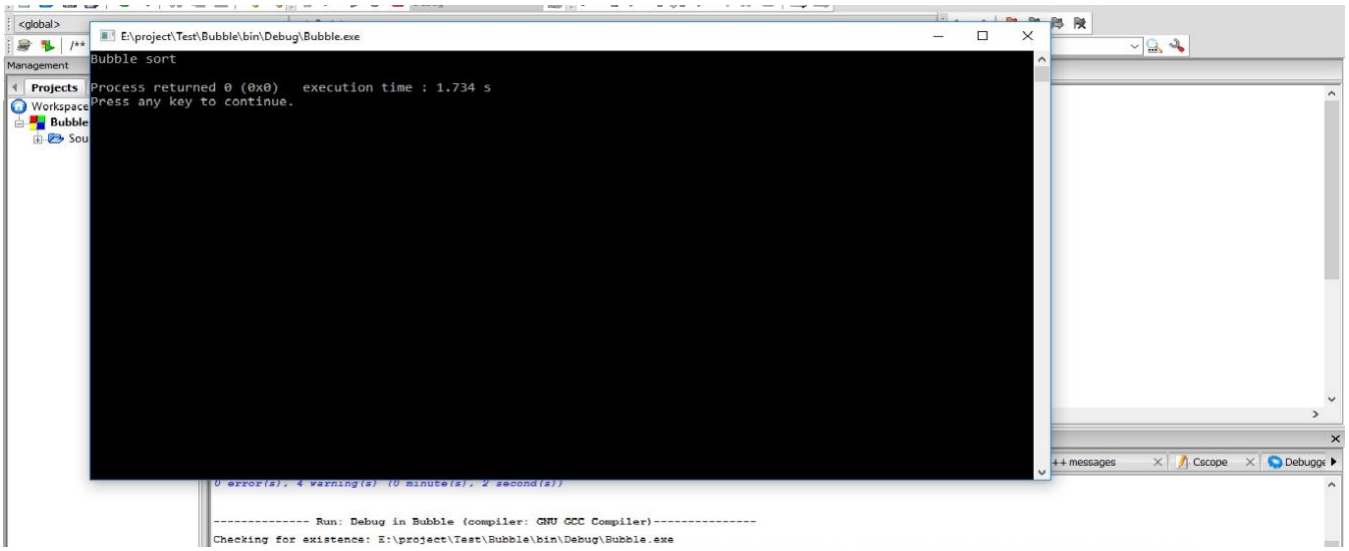
Figure: Working of Bubble sort algorithm

Code in C:

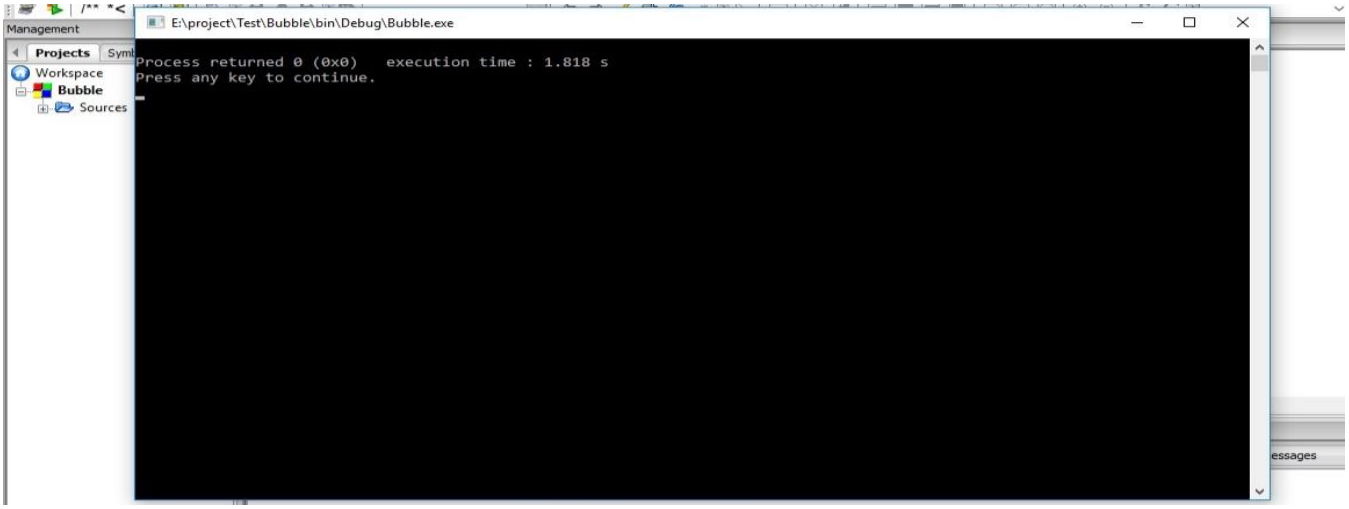
```
  #include <stdio.h>

int main ()
{ printf("Bubble sort\n");
    long int n,i,a,V[100000],ch,aux,d;
    FILE *f1, *f2;
    f1 = fopen("data.txt", "r");
    f2 = fopen("OUTPUT_BUBBLE.txt", "w");
    char line[102440];
    d=0;
    n = 0;
    while( fgets(line,sizeof(line),f1) != NULL)
       n++;
fseek(f1, 0, SEEK_SET);
    for (i=0; i<n; i++)
        fscanf(f1,"%d", &V[i]);
    do {
            d++;
        ch=0;
        for (i=0; i<n-1; i++)
            if (V[i]>V[i+1])
            {
                aux=V[i]; V[i]=V[i+1]; V[i+1]=aux; ch=1;
            }
    } while (ch);
    for (i=0; i<n; i++)
        fprintf(f2, "%d\n", V[i]);
    fclose(f1);
    fclose(f2);
    printf("\n%d\n",d);
  }
```

The program inputs a file data.txt containing 100, 1000, 10,000, 15,000 etc. random integers at various intervals.The output is again in the form of a file which contains a sorted list of the integers that were input .The total time taken to sort the elements is noted down.

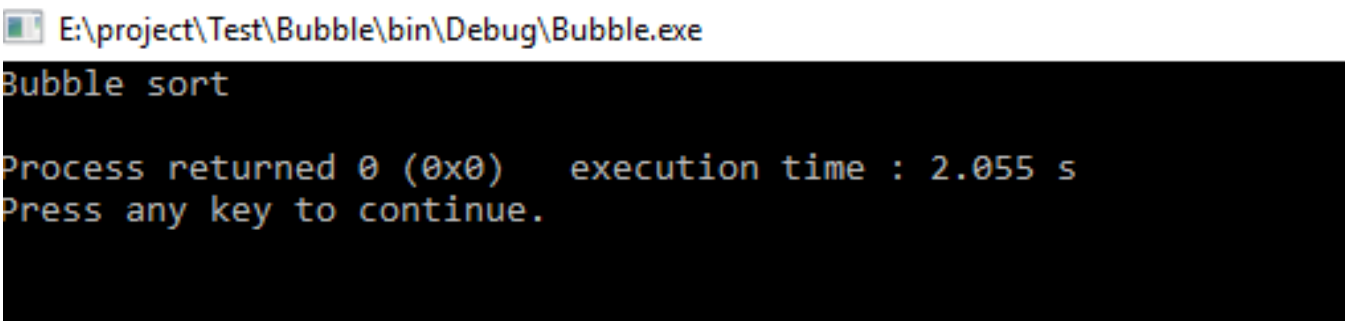
Clips of running program:



1.734 seconds for 15000 elements



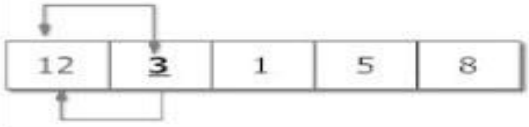
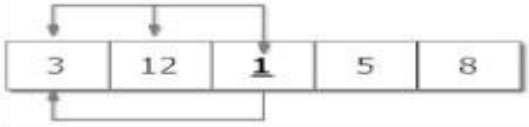
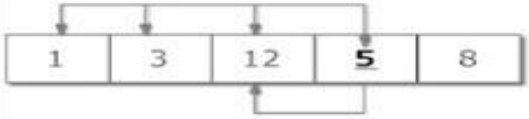
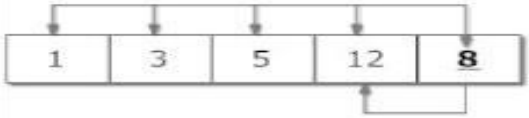
1.818 seconds for 20000 elements



2.055 seconds for 40000 elements

INSERTION SORT ANALYSIS

1. Step 1: The second element of an array is compared with the elements that appear before it (only first element in this case). If the second element is smaller than first element, second element is inserted in the position of first element. After first step, first two elements of an array will be sorted.
2. Step 2: The third element of an array is compared with the elements that appears before it (first and second element). If third element is smaller than first element, it is inserted in the position of first element. If third element is larger than first element but, smaller than second element, it is inserted in the position of second element. If third element is larger than both the elements, it is kept in the position as it is. After second step, first three elements of an array will be sorted.
3. Step 3: Similarly, the fourth element of an array is compared with the elements that appear before it (first, second and third element) and the same procedure is applied and that element is inserted in the proper position. After third step, first four elements of an array will be sorted.

Step 1		**Checking second element of array with element before it and inserting it in proper position. In this case, 3 is inserted in position of 12.**
Step 2		**Checking third element of array with elements before it and inserting it in proper position. In this case, 1 is inserted in position of 3.**
Step 3		**Checking fourth element of array with elements before it and inserting it in proper position. In this case, 5 is inserted in position of 12.**
Step 4		**Checking fifth element of array with elements before it and inserting it in proper position. In this case, 8 is inserted in position of 12.**
		**Sorted Array in Ascending Order**

Code in C:

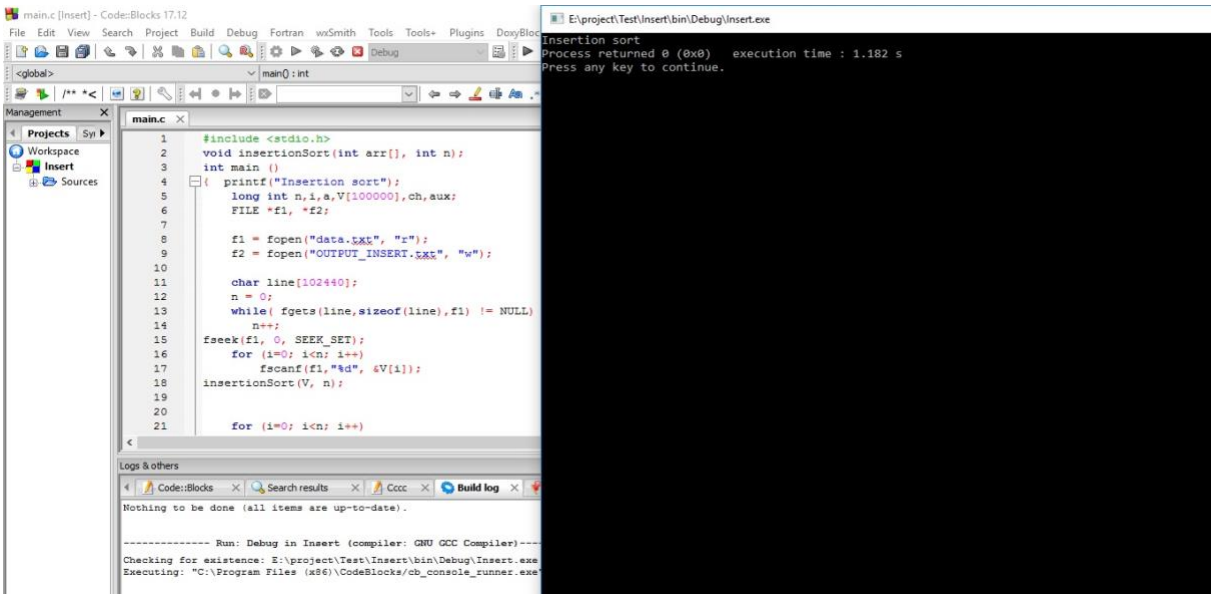
```c
#include<stdio.h>

void insertionSort(int arr[], int n);
int main ()

{
    printf("Insertion sort");
    int n,i,a,V[100000],ch,aux;

    FILE *f1, *f2;
    f1 = fopen("data.txt", "r");
    f2 = fopen("OUTPUT_INSERT.txt", "w");

    char line[102440];
    n = 0;

    while( fgets(line,sizeof(line),f1) != NULL)
       n++;

    fseek(f1, 0, SEEK_SET);

    for (i=0; i<n; i++)
        fscanf(f1,"%d", &V[i]);

        insertionSort(V, n);

    for (i=0; i<n; i++)
        fprintf(f2, "%d\n", V[i]);

    fclose(f1);
    fclose(f2);

}
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;
   }
}
```

The program inputs a file data.txt containing 100, 1000, 10,000, 15,000 etc. random integers at various intervals. The output is again in the form of a file which contains a sorted list of the integers that were input. The total time taken to sort the elements is noted down.

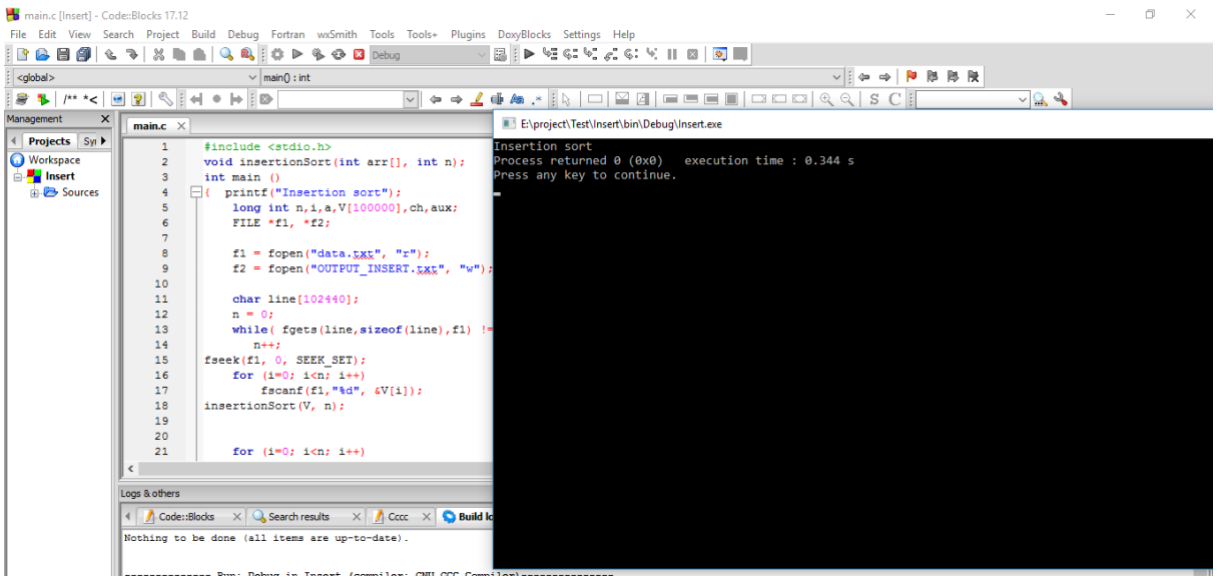
E:\project\Test\Insert\bin\Debug\Insert.exe

```
Insertion sort
Process returned 0 (0x0)   execution time : 0.573 s
Press any key to continue.
```

0.573 seconds for 25000 elements



1.182 seconds for 45000 elements



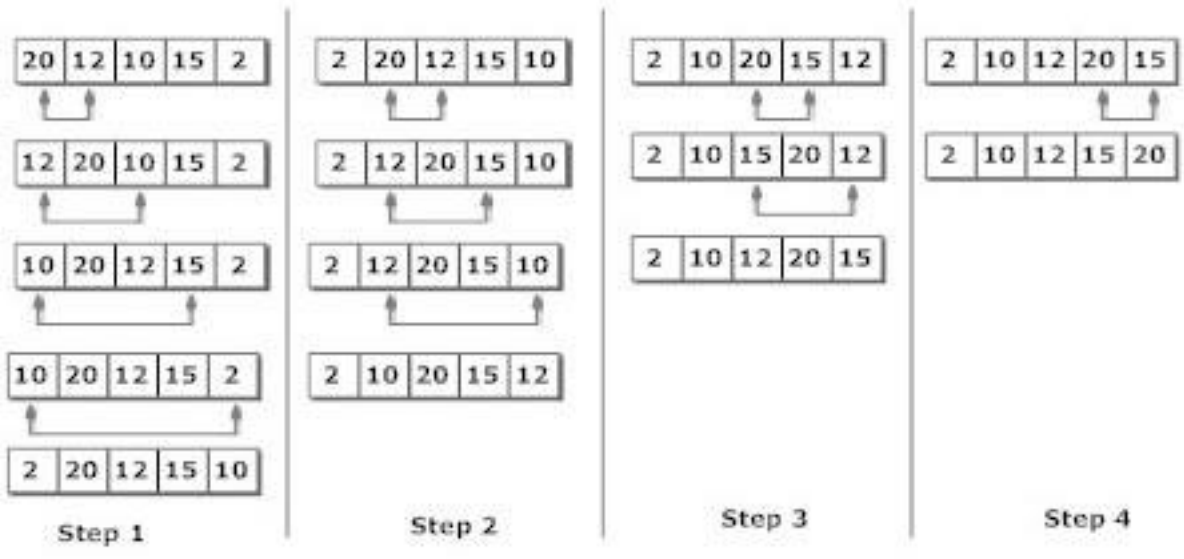
0.344 seconds for 15000 elements

SELECTION SORT ANALYSIS

Selection sort algorithm starts by comparing first two elements of an array and swapping if necessary, i.e., if you want to sort the elements of array in ascending order and if the first element is greater than second then, you need to swap the elements but, if the first element is smaller than second, leave the elements as it is.

Then, again first element and third element are compared and swapped if necessary. This process goes on until first and last element of an array is compared. This completes the first step of selection sort.

If there are n elements to be sorted then, the process mentioned above should be repeated n-1 times to get required result. But, for better performance, in second step, comparison starts from second element because after first step, the required number is automatically placed at the first

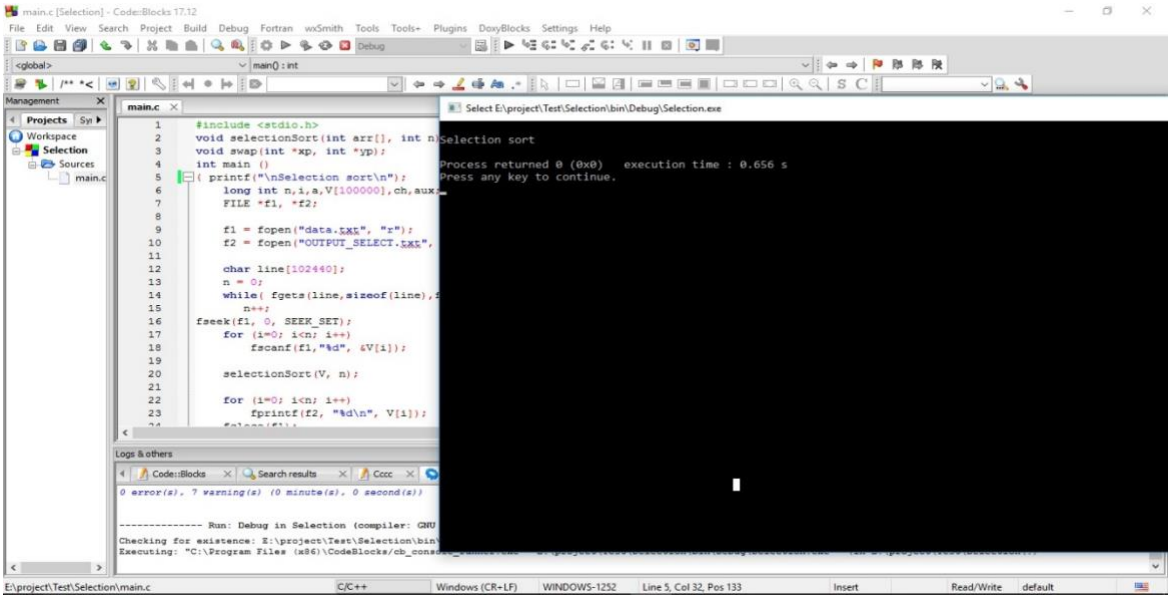


Code in C:

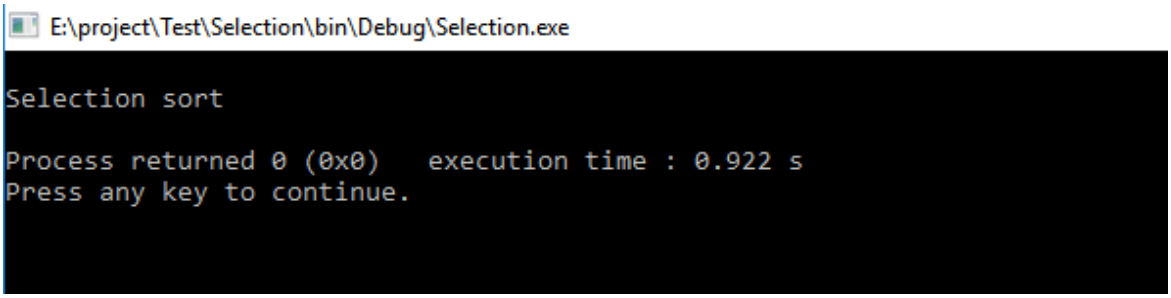
```c
#include <stdio.h>
void selectionSort(int arr[], int n);
void swap(int *xp, int *yp);
int main ()
{ printf("\nSelection sort\n");
    long int n,i,a,V[100000],ch,aux;
    FILE *f1, *f2;

    f1 = fopen("data.txt", "r");
    f2 = fopen("OUTPUT_SELECT.txt", "w");

    char line[102440];
    n = 0;
    while( fgets(line,sizeof(line),f1) != NULL)
       n++;
fseek(f1, 0, SEEK_SET);
    for (i=0; i<n; i++)
        fscanf(f1,"%d", &V[i]);

    selectionSort(V, n);

    for (i=0; i<n; i++)
        fprintf(f2, "%d\n", V[i]);
    fclose(f1);
    fclose(f2);

}
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 swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
```

The program inputs a file data.txt containing 100, 1000, 10,000, 15,000 etc. random integers at various intervals. The output is again in the form of a file which contains a sorted list of the integers that were input. The total time taken to sort the elements is noted down.

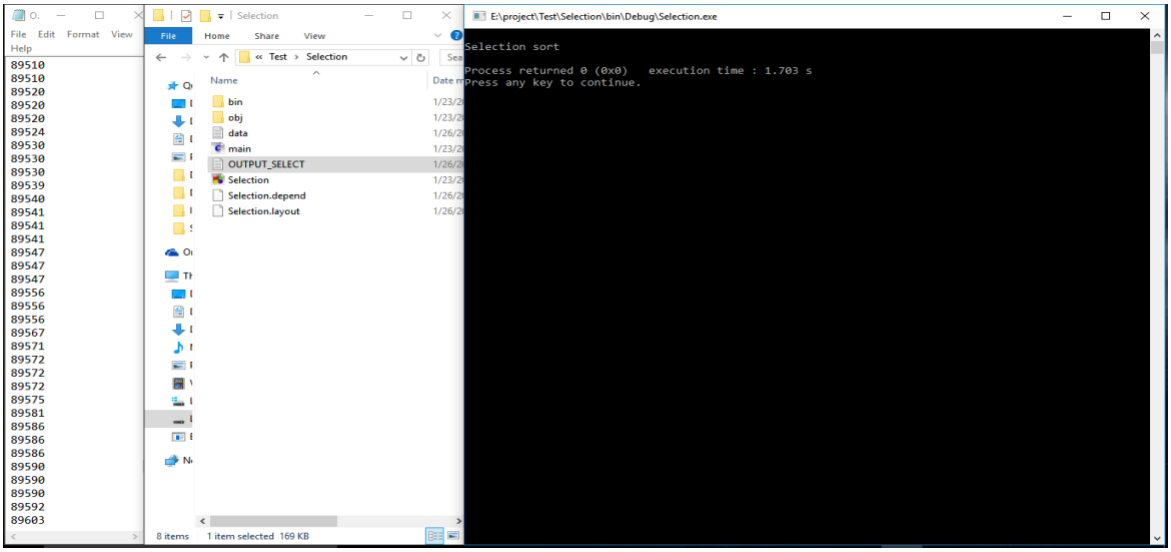
Clips of running program:



0.656 seconds for 15000 elements



0.922 seconds for 25000 elements

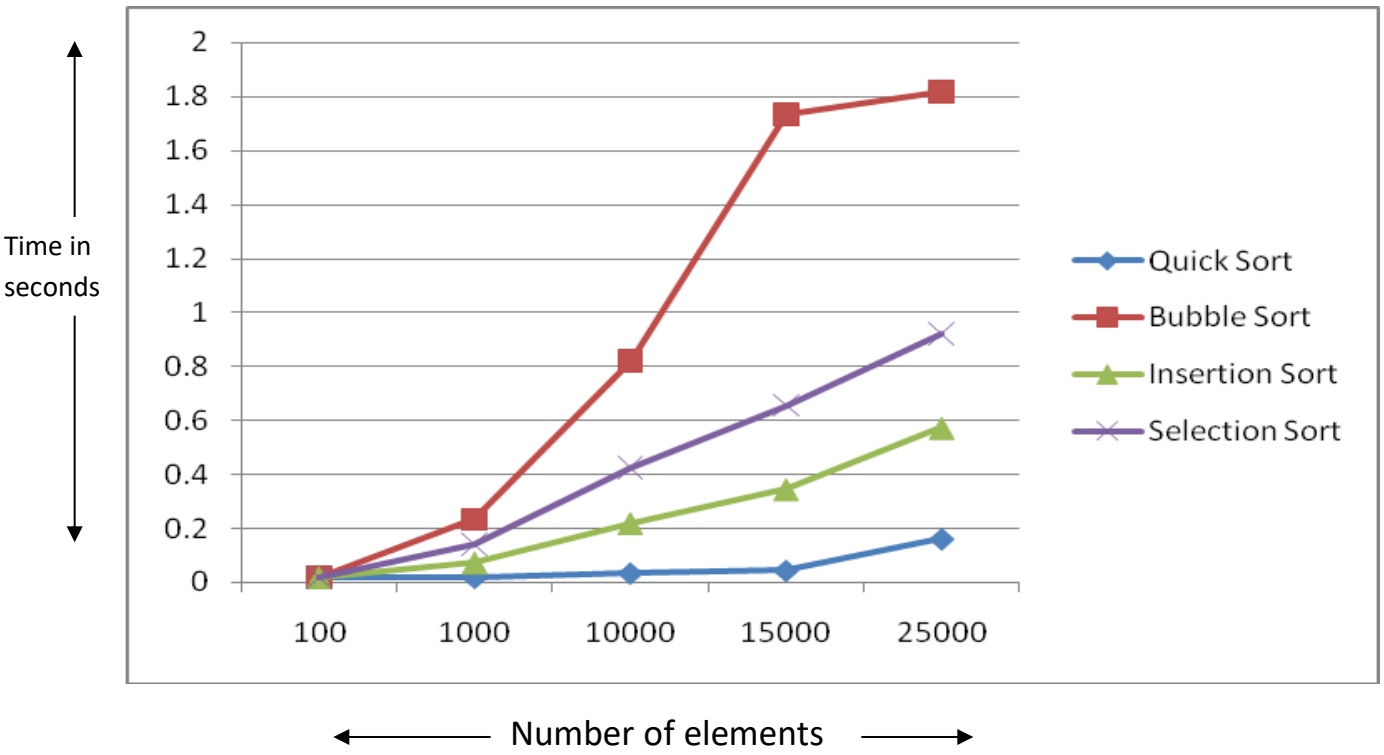


1.703 seconds for 45000 elements

COMPARISON TABLE:

No. of elements	Quick Sort	Bubble Sort	Insertion Sort	Selection Sort
100	0.016	0.016	0.016	0.016
1000	0.016	0.232	0.071	0.141
10000	0.032	0.818	0.218	0.423
15000	0.042	1.734	0.344	0.656
25000	0.161	1.818	0.573	0.922
45000	0.242	2.142	1.182	1.703

Graph for the given data:



From the given graph we can clearly see that the quick sort algorithm takes the lowest amount of time to compile and give the output as compared to the other sorts. Thus quick sort algorithm is considered best amongst other sorts. (Blue line)

On the contrary bubble sort is the easiest to understand but it's not a good sorting algorithm as it takes lots of time for execution. (red line)

Next comes Insertion sort, its efficiency is a little better than bubble sort but less than quick sort. (Green line)

Selection sort's efficiency lies in between insertion sort and bubble sort. (Purple line)

Conclusion:

- Bubble Sort is not suitable in any circumstance. It is an $O(n^2)$ algorithm with a large constant. In simple words, time required to perform bubble sort on ‘n’ numbers increase as square of ‘n’. Thus it is quite slow.

- Insertion Sort is suitable for small files, but again it is an $O(n^2)$ algorithm, but with a small constant. Also note that it works best when the file(/numbers) is already almost sorted.

- Quick Sort is an O(n*log(n)) algorithm on an average case and an $O(n^2)$ algorithm in the worst case scenario. (Quick sort’s worst case occurs when the numbers are already sorted!!) The graph speaks it all. You need this algorithm when the list is large and time is premium.

- Selection sort requires only n write operations. If we have a system where write operations are extremely expensive and read operations are not, then selection sort could be ideal (Assuming write time outweighs processing time).

References:

Books: ***Data Structures in C by Schaum's series***

C in Depth by *S.K.* ***Srivastava***

Websites*:* ***www. google.com***

www.btechsmartclass.com