

1. Introduction

It aims to make a comparative analysis among the sorting algorithms requested which are Bubble sort, quick sort, insertion sort and selection sort, from us. For this, data obtained by executing each of these methods in different scenarios will be used through a program written in the C language. Each algorithm will be executed one time. Each algorithm will have its execution time. At the end of the work, we hope to have a clear understanding of what are the best algorithms for each type of situation.

2. Theoretical

There will be four sorting algorithms studied here, each with advantages, disadvantages, runtimes, complexities, and their codes. After briefly giving information about the algorithm, I will talk about the Time Complexities of the algorithms and then share their codes.

2.1 Bubble Sorting

It is based on the logic of comparing two items each time while moving on the array to be sorted and changing their places if the compared items are in the wrong order. Sorting is accomplished by stepping through all items one by one and comparing it to the adjacent item and changing it if necessary. Bubble sort compares all numbers and sorts them by size. The numbers will compare the number in the first index of the Array with the number in the second index. At the end of the match, the lower number will be placed in the previous index, and then the other will be placed. If the second number is less than the first number, the index numbers in the Array of Numbers will change and it will meet the third number in the row. This cycle will continue until the last element. If there are n elements, it will be repeated $n-1$ times.

First Number and second number match in Bubble sort algorithm with the number in the N . index ($n-1$). The number on the index is being compared. 2. In comparison, it continues until the end as ($n-2$) and ($n-3$). So When we collect the maximum number of comparisons and replacements for all elements of the array the total number of comparisons will be,

$$(n-1) + (n-2) + (n-3) + \dots + 3 + 2 + 1 = N * \left(\frac{N-1}{2}\right) = \sim (N^2)$$

Bubble Sorting Function in C Program and Output

```
//Bubble Sorting Algorithm
void bubbleSort(int bubbleArray[], int inputSize){
    int comp = 0, swp = 0;
    bool swapped;
    int i;
    for (i = 0; i < inputSize-1; i++) {
        swapped = false;
        for (j = 0; j < inputSize-i-1; j++) {
            if (bubbleArray[j] > bubbleArray[j+1]) {
                swap(&bubbleArray[j], &bubbleArray[j+1]);
                swapped = true;
                swp++;
            }
            comp++;
        }
        if (swapped == false) break;
    }
}
```

```

Assignment2
Please write c codes of bubble sort, quick sort, insertion sort, selection sort by supplying random numbers (100,1000, 10000).
Check their complexity and running times.Draw their separate graphs and decide which O notation they belong.
Enter 0 for EXIT :
Enter 1 for Bubble Sort :
Enter 2 for Selection Sort :
Enter 3 for Insertion Sort :
Enter 4 for Quick Sort :
Enter 5 for Compare The Algorithms, Bubble sort, quick sort, insertion sort and selection sort
1
Determine the size of the array you want to sort :
100
#####
Random Numbers [100]: 7 41 23 30 2 61 67 53 93 29 59 79 11 45 90 100 30 55 68 74 85 15 29 2 77 51 11 33 24 2 ....
Bubble Sort : 2 2 2 3 6 7 10 10 11 11 11 15 15 15 17 18 23 23 24 24 26 27 29 29 29 30 30 31 33 33 ....
The process took 0.000000 sec.
#####
If you want to view all values, please press 1 if NOT press any key.

Assignment2
1
Determine the size of the array you want to sort :
100
#####
Random Numbers [100]: 7 41 23 30 2 61 67 53 93 29 59 79 11 45 90 100 30 55 68 74 85 15 29 2 77 51 11 33 24 2 ....
Bubble Sort : 2 2 2 3 6 7 10 10 11 11 11 15 15 15 17 18 23 23 24 24 26 27 29 29 29 30 30 31 33 33 ....
The process took 0.000000 sec.
#####
If you want to view all values, please press 1 if NOT press any key.
1
#####
Random Numbers [100]: 7 41 23 30 2 61 67 53 93 29 59 79 11 45 90 100 30 55 68 74 85 15 29 2 77 51 11 33 24 2 23 100 38 38 70 81 89 69 94 60 60
40 46 71 73 51 63 18 67 53 64 53 94 73 24 98 10 39 33 17 51 11 69 27 33 94 37 63 76 70 72 75 48 65 89 94 47 73 47 92 15 10 44 15 41 91 26 45 92
6 86 54 62 31 97 3 29 67 88 43
Bubble Sort :2 2 2 3 6 7 10 10 11 11 11 15 15 15 17 18 23 23 24 24 26 27 29 29 29 30 30 31 33 33 33 37 38 38 39 40 41 41 43 44 45 45 46 47 47 4
8 51 51 51 53 53 53 54 55 59 60 60 61 62 63 63 64 65 67 67 68 69 69 70 70 71 72 73 73 73 74 75 76 77 79 81 85 86 88 89 89 90 91 92 92 93 94
94 94 94 97 98 100 100
#####
Press any key to Main Menu:

```

2.2.SELECTION SORTING

Selection sort algorithms called its name because, it selects the smallest number among the given values. This algorithm works as follows: Find the smallest element in the array and replace it with the first element, then find the other smallest element and replace it with the second element, find the smallest element each time and place it in the array until the array is completely sorted. To calculate the complexity of the Selection sorting algorithm, the comparisons and displacements made are calculated. The algorithm performs N-1 comparisons for the smallest element in an array with N elements, N-2 comparisons for the second smallest element, the number of comparisons for other elements is N-3, N-4,..., 2, 1, 0, and so on. compares 0 for the last element. If we add up all the comparisons, $(N-1) + (N-2) + (N-3) + (N-4) + \dots + 2 + 1 + 0 = N * \frac{N-1}{2}$ we get. A replacement is made for each element, a total of N replacements are made for the entire array. The asymptotic upper limit of $N(N-1) / 2 + N \sim N^2$ values obtained as a result of calculations gives $O(N^2)$ value. The Selection sorting algorithm works with $O * N^2$ complexity.

2.3.INSERTION SORTING

Insertion sort gets its name from the subsequent replacement of the selected element. The first selected element is copied to a separate place, the selected element is compared with the lower indexed elements, the elements larger than it are shifted one row to the right, and the selected element is placed in its place. When the last Number is added to the Array, the array is sorted.

To calculate the complexity of the Insertion sorting algorithm, the comparisons and displacements made are calculated. The algorithm performs at most 1 comparisons and 1 replacement for the second element in an array of N elements, 2 comparisons and 2 replacements for the third element, 3 comparisons and 3 replacements for the fourth element. This way, it performs at most N-1 comparisons and N-1 substitutions for the last element. When we collect the maximum number of comparisons and replacements for all elements of the array, We find

$$2*(1+2+3+4+...+N-2+N-1) = 2 (N (N-1) / 2) = 2N * \left(\frac{N-1}{2}\right) = N (N-1). N (N-1) \sim N^2$$

obtained as a result of calculations. The asymptotic upper limit of the value gives $O(N^2)$. The complexity of the insertion sorting algorithm is in the worst case (N^2) . If the sequence is sequential, it only performs $N-1$ comparisons and works with $O(N)$ complexity, with $O(N)$ * The average performance of the additive sorting algorithm is again $O(N^2)$ value.

2.3.1. Insertion Sorting Function in C Program

```
//Insertion Sort Algorithm
void insertionSort(int insertionArray[], int inputSize){
    int comp = 0, assg = 0, h;
    for (h = 1; h < inputSize; h++) {
        int key = insertionArray[h];
        int k = h - 1;
        while (k >= 0 && key < insertionArray[k]) {
            insertionArray[k + 1] = insertionArray[k];
            comp++;
            assg++;
            --k;
        }
        insertionArray[k + 1] = key;
    }
}
```

```
Assignment2
```

```
Please write c codes of bubble sort, quick sort, insertion sort, selection sort by supplying random numbers (100,1000, 10000).  
Check their complexity and running times.Draw their separate graphs and decide which O notation they belong.  
Enter 0 for EXIT :  
Enter 1 for Bubble Sort :  
Enter 2 for Selection Sort :  
Enter 3 for Insertion Sort :  
Enter 4 for Quick Sort :  
Enter 5 for Compare The Algorithms, Bubble sort, quick sort, insertion sort and selection sort  
3  
Determine the size of the array you want to sort : 10000  
  
*****  
>> Random Numbers [10000]: 1 91 81 13 46 7 95 91 22 44 39 87 59 6 92 97 14 6 54 100 74 8 32 47 27 58 25 30 11 5 ...  
>> Insertion Sort : 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 .....  
The process took 0.070000 sec.  
*****  
  
If you want to view all values, please press 1 if NOT press any key.
```

2.4. QUICK SORT

The Quicksort algorithm works with the logic of separating the array into two smaller parts and sorting these small parts into themselves. The algorithm selects any element from the array as a pivot (lock) element. Arrange the array so that all elements smaller than the pivot element are in front of the pivot element, and all elements larger than the pivot element are behind the pivot element. The numbers that are equal to the pivot element can pass on either side of the pivot element, depending on whether the sequence is small to large or large to small. The resulting small strings are reordered by calling the Quicksort algorithm recursive. This sorting algorithm continues until the 0th Index is in the array. When a sub-array with several elements is reached, the algorithm assumes that this array is sorted and the sorting process is completed. Complexity of the Quicksort algorithm is

$$((N + (N - 1) + (N - 2) + (N - 3) + + 2) = [N * (\frac{N-1}{2}) - 1] = N^2$$

2.4.1. Quicksort Sorting Function in C Program

```
//Quick Sort Algorithm
void quickSort(int quickArray[], int low, int inputSize){
    if (low < inputSize) {
        //returns the pivot -> some random middle value
        int pi = partitionMedian(quickArray, low, inputSize);
        quickSort(quickArray, low, pi - 1);
        quickSort(quickArray, pi + 1, inputSize);
    }
}

int partitionMedian (int arr[], int low, int high) {
    swap(&arr[(low + high)/2], &arr[high]);
    return partitionLast(arr, low, high);
}

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

```

Assignment2

Please write c codes of bubble sort, quick sort, insertion sort, selection sort by supplying random numbers (100,1000, 10000).
Check their complexity and running times. Draw their separate graphs and decide which O notation they belong.

Enter 0 for EXIT :
Enter 1 for Bubble Sort :
Enter 2 for Selection Sort :
Enter 3 for Insertion Sort :
Enter 4 for Quick Sort :
Enter 5 for Compare The Algorithms, Bubble sort, quick sort, insertion sort and selection sort
4
Determine the size of the array you want to sort : 100000

*****
>> Random Numbers [100000]: 58 32 67 43 91 79 6 32 84 67 78 73 25 81 68 45 78 92 80 99 30 16 56 57 5 49 35 75 49 3 ...
>> Quick Sort :      1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 .....
The process took 0.310000 sec.
*****

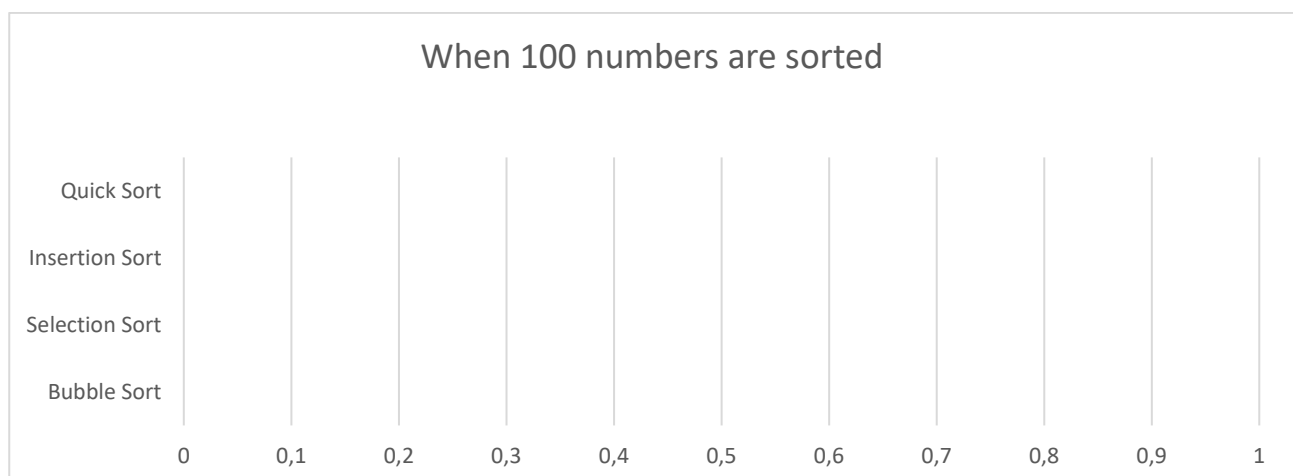
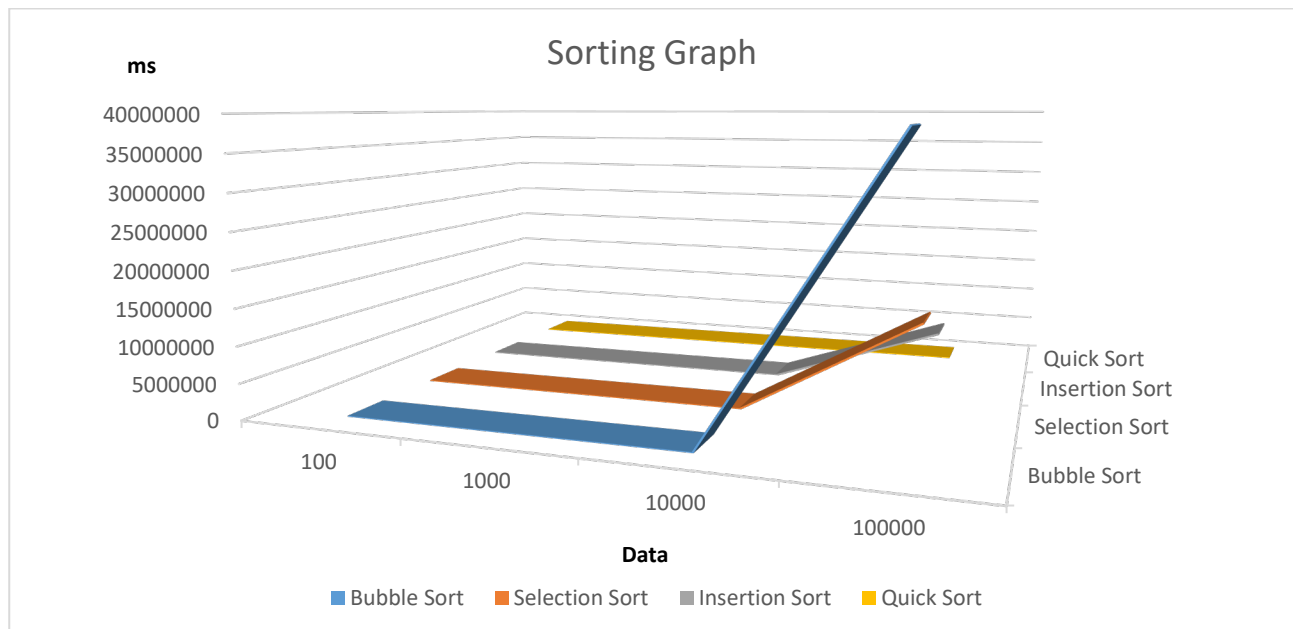
If you want to view all values, please press 1 if NOT press any key.

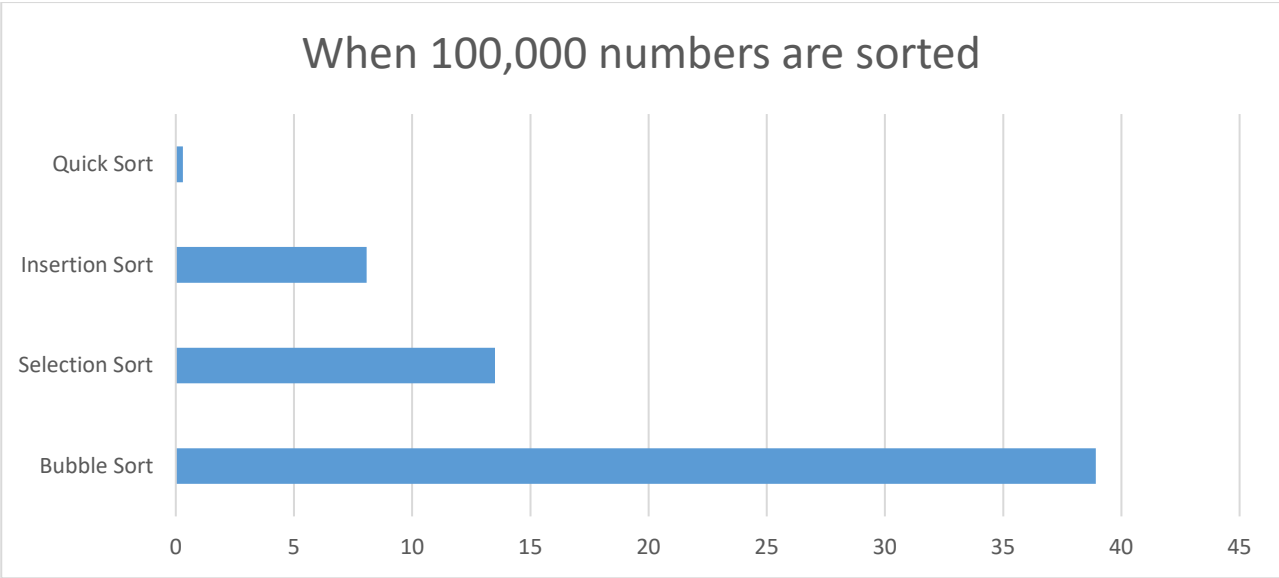
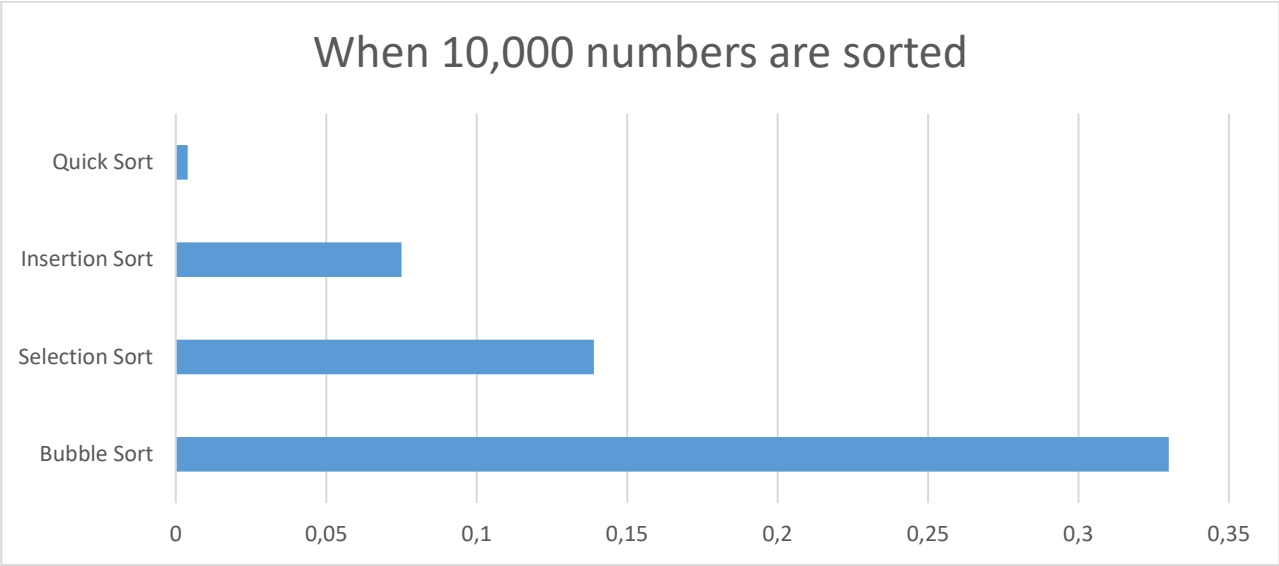
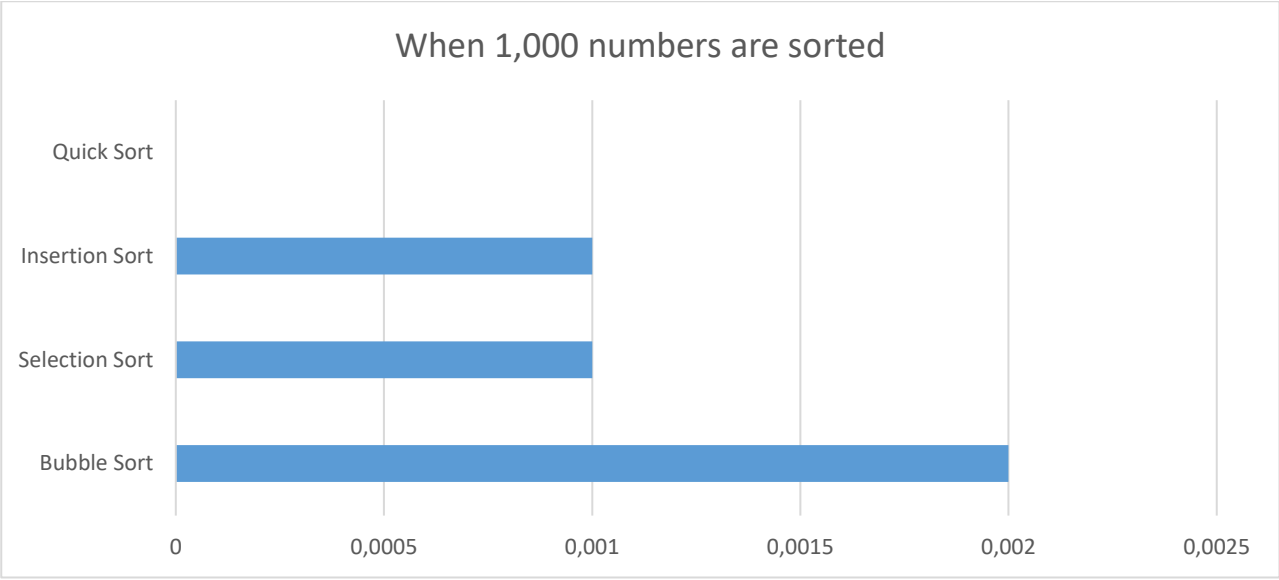
```

3. EVALUATION

In Bubble Sort algorithm Data is sorted in place, so there is very little memory overhead, and once sorted, data is in memory ready for processing. The Bubble Sort algorithm is the best choice when there is an already sorted array. Insertion Sort algorithm performs well when dealing with a small list and space requirement is minimal. Suitable for sorting a small number of items. The Pick Order is that no additional temporary storage is required and performs well in a small list. It works faster than the insertion sort algorithm. It is low efficiency when dealing with a large number of items. Quicksort algorithm is considered to be the best ranking algorithm. Quick Sort is much more efficient than selective sorting It copes well with a large number of item lists. No additional storage is required as it is sorted in place If the list is already sorted, Bubble sorting is much more efficient than quick sorting. In Bubble Sorting Data is sorted in place, so there is very little memory overhead and once sorted, the data is in memory ready for processing. The Bubble Sorting algorithm is the best choice when there is an already sorted array. Insertion sort algorithm performs well when dealing with a small list and space requirement is minimal. Suitable for sorting a small number of items. The Pick Order is that no additional temporary storage is required and performs well in a small list. It works faster than the insertion sort algorithm. It is low efficiency when dealing with a large number of items.

4. GRAPH



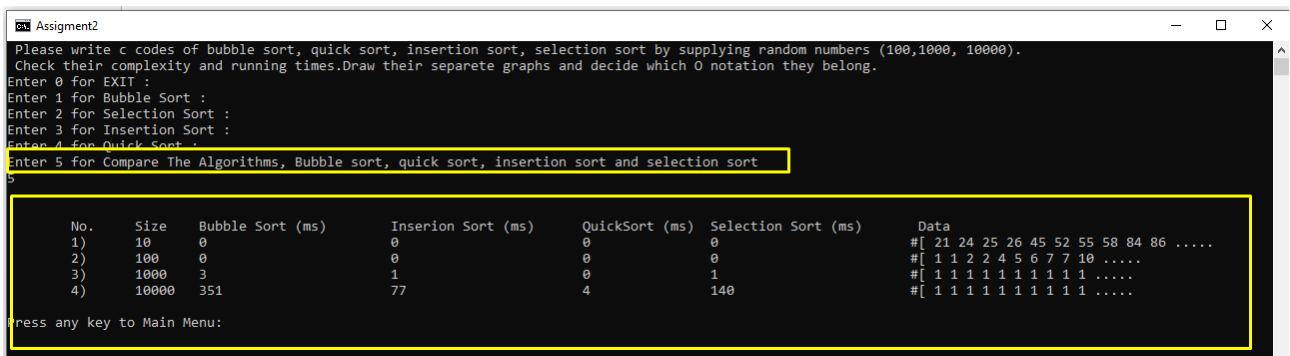


5. FINAL

It can be seen, from the foregoing, that there is no perfect sorting algorithm. None of the sorting methods analyzed here was able to present the best performance in all categories. In this context, the best choice always depends on the characteristics of the problem in question.

If you want to sort the 4 algorithms which is Bubble, Insertion, Quick and Selection sorting algorithms feeding randomly generated with 100,1000,10000 numbers and run the values then its run times of the algorithms at the same time there is an option for Compare the all algorithms.

When we want see all the listed numbers, the visuality distorts, so I printed only the first 10 values, Hocam I hope you like it all Thank you.



```
Assignment2
Please write c codes of bubble sort, quick sort, insertion sort, selection sort by supplying random numbers (100,1000, 10000).
Check their complexity and running times.Draw their separate graphs and decide which O notation they belong.
Enter 0 for EXIT :
Enter 1 for Bubble Sort :
Enter 2 for Selection Sort :
Enter 3 for Insertion Sort :
Enter 4 for Quick Sort :
Enter 5 for Compare The Algorithms, Bubble sort, quick sort, insertion sort and selection sort

5

```

No.	Size	Bubble Sort (ms)	Insertion Sort (ms)	QuickSort (ms)	Selection Sort (ms)	Data
1)	10	0	0	0	0	#[21 24 25 26 45 52 55 58 84 86
2)	100	0	0	0	0	#[1 1 2 2 4 5 6 7 7 10
3)	1000	3	1	0	1	#[1 1 1 1 1 1 1 1 1 1
4)	10000	351	77	4	140	#[1 1 1 1 1 1 1 1 1 1

Press any key to Main Menu:

6. SOURCE CODE OF THE PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <string.h>
void makeRandom(int[], int, int, int);
void showArray(int[], int);
void iter(int[], int[], int);
void bubbleSort(int[], int);
void selectionSort(int[], int);
void insertionSort(int[], int);
void quickSort(int[], int, int);
int partitionMedian (int[], int, int);
int partitionLast (int[], int, int);
void swap(int *xp, int *yp);

int i, j;
int main(int argc, char ** argv) {
    int n;
    while (1==1) {
        printf(" Please write c codes of bubble sort, quick sort, insertion sort,
selection sort by supplying random numbers (100,1000, 10000).\n Check their
complexity and running times.Draw their separate graphs and decide which O
notation they belong.  \n");
        options:
        printf("Enter 0 for EXIT :\n");
        printf("Enter 1 for Bubble Sort :\n");
        printf("Enter 2 for Selection Sort :\n");
        printf("Enter 3 for Insertion Sort :\n");
        printf("Enter 4 for Quick Sort :\n");
        printf("Enter 5 for Compare The Algorithms, Bubble sort, quick sort,
insertion sort and selection sort  \n");
        scanf("%d",&n);
```



```

switch(n)
{
case 0:
                                printf("Quitting the program...\n\n");
                                exit(5);

    break;
case 1:
{
    srand(time(0));
    clock_t start, stop;
    float timeTaken;
        int numOfInputs, minValue = 0, maxValue = 100;
        printf("Test icin giris boyutunu girin:\n ");
        scanf("%d", &numOfInputs);
        //Creating the arrays for testing
        int randNum[numOfInputs];
        int bubbleArray[numOfInputs];
        //Assigning random values
        makeRandom(randNum, numOfInputs, minValue, maxValue);

printf("#####\n\n");
        printf(" Random Numbers [%d]: ",numOfInputs);
        showArray(randNum, 30);
        printf("....\n");
        //Copying the random values into Testing arrays
        iter(randNum, bubbleArray, numOfInputs);
        start = clock();
        bubbleSort(bubbleArray, numOfInputs);
                                stop = clock();
                                timeTaken = (double) (stop-start)/CLOCKS_PER_SEC;
        printf(" Bubble Sort      : ");
        showArray(bubbleArray, 30);
        printf("....\n");
        printf(" The process took %f sec. \n",timeTaken);

printf("#####\n\n");
        printf("If you want to view all values, please press 1 if NOT press any
key.\n"); int x; scanf("%d",&x); if(x==1){

printf("#####\n\n");
        printf("\nRandom Numbers [%d]: ",numOfInputs);
        showArray(randNum, numOfInputs); printf("\n\nBubble Sort :");
        showArray(bubbleArray, numOfInputs);

printf("\n#####\n\n");
        }
    }

    printf("\n\nPress any key to Main Menu: ");
    while(getchar() == '\n');
    goto options;
    break;
case 2:
{
    srand(time(0));
        clock_t start, stop;
        float timeTaken;
        int numOfInputs, minValue = 0, maxValue = 100;
        printf("Test icin giris boyutunu girin: ");
        scanf("%d", &numOfInputs);
        int randNum[numOfInputs];

```

```

        int selectionArray[numOfInputs];
        makeRandom(randNum, numOfInputs, minValue, maxValue);

printf("#####\n");
        printf(">> Data to Sort [%d]: ",numOfInputs);
        showArray(randNum, 30);
        printf("...\n");
        iter(randNum, selectionArray, numOfInputs);
        start = clock();
        selectionSort(selectionArray, numOfInputs);
        stop = clock();
        timeTaken = (double)(stop-start)/CLOCKS_PER_SEC;
        printf(">> Selection Sort : ");
        showArray(selectionArray, 30);
        printf(".....\n");
        printf(" The process took %f sec. \n",timeTaken);

printf("#####\n\n");
        printf("If you want to view all values, please press 1 if NOT\n");
        int x; scanf("%d",&x); if(x==1){

printf("#####\n");
        printf("\nRandom Numbers [%d]: ",numOfInputs);
        showArray(randNum, numOfInputs); printf("\n\nBubble Sort :");
        showArray(selectionArray, numOfInputs);

printf("\n#####\n");
        }
    }
    printf("\n\nPress any key to Main Menu: ");
    while(getchar() == '\n');
    goto options;
    break;
    case 3:
    {
        srand(time(0));
        clock_t start, stop;
        float timeTaken;
        int numOfInputs, minValue = 0, maxValue = 100;
        printf("Test icin giris boyutunu girin: ");
        scanf("%d", &numOfInputs);
        int randNum[numOfInputs];
        int insertionArray[numOfInputs];
        makeRandom(randNum, numOfInputs, minValue, maxValue);

printf("#####\n");
        printf(">> Data to Sort [%d]: ",numOfInputs);
        showArray(randNum, 30);
        printf("...\n");
        iter(randNum, insertionArray, numOfInputs);
        start = clock();
        insertionSort(insertionArray, numOfInputs);
        stop = clock();
        timeTaken = (double)(stop-start)/CLOCKS_PER_SEC;
        printf(">> Insertion Sort : ");
        showArray(insertionArray, 30);
        printf(".....\n");
        printf(" The process took %f sec. \n",timeTaken);
    }
}

```

```

printf("#####\n\n");
        printf("If you want to view all values, please press 1 if NOT
press any key.\n"); int x; scanf("%d",&x); if(x==1){

printf("#####\n");
        printf("\nRandom Numbers [%d]: ",numOfInputs);
        showArray(randNum, numOfInputs); printf("\n\nBubble Sort :");
        showArray(insertionArray, numOfInputs);

printf("\n#####\n");
    }
    }
    printf("\n\nPress any key to Main Menu: ");
    while(getchar() == '\n');
    goto options;
    break;
case 4:
{
    srand(time(0));
    clock_t start, stop;
    float timeTaken;
    int numOfInputs, minValue = 0, maxValue = 100;
    printf("Test icin giris boyutunu girin: ");
    scanf("%d", &numOfInputs);
    int randNum[numOfInputs];
    int quickArray[numOfInputs];
    makeRandom(randNum, numOfInputs, minValue, maxValue);

printf("#####\n");
    printf(">> Data to Sort [%d]: ",numOfInputs);
    showArray(randNum, 30);
    printf("...\n");
    iter(randNum, quickArray, numOfInputs);
    start = clock();
    quickSort(quickArray, 0, numOfInputs-1);
    stop = clock();
    timeTaken = (double) (stop-start)/CLOCKS_PER_SEC;
    printf(">> Quick Sort : ");
    showArray(quickArray, 30);
    printf(".....\n");
    printf(" The process took %f sec. \n",timeTaken);

printf("#####\n\n");
    printf("If you want to view all values, please press 1 if NOT press
any key.\n"); int x; scanf("%d",&x); if(x==1){

printf("#####\n");
    printf("\nRandom Numbers [%d]: ",numOfInputs);
    showArray(randNum, numOfInputs); printf("\n\nBubble Sort :");
    showArray(quickArray, numOfInputs);

printf("\n#####\n");
    }
    }
    printf("\n\nPress any key to Main Menu: ");
    while(getchar() == '\n');

```

[illegible]

```

    }
    printf("\n");
}
void showArray(int randNum[], int numOfInputs){
    int i;
    for(i = 0; i < numOfInputs; i++){
        printf("%d ", randNum[i]);
    }
}
//Copying array
void iter(int randNum[], int resultArray[], int numOfInputs){
    int i;
    for(i = 0; i < numOfInputs; i++){
        resultArray[i] = randNum[i];
    }
}
//Swapping algorithm
void swap(int *xp, int *yp) {
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
//Bubble Sorting Algorithm
void bubbleSort(int bubbleArray[], int numOfInputs){
    int comp = 0, swp = 0;
    bool swapped;
    int i;
    for (i = 0; i < numOfInputs-1; i++) {
        swapped = false;
        for (j = 0; j < numOfInputs-i-1; j++) {
            if (bubbleArray[j] > bubbleArray[j+1]) {
                swap(&bubbleArray[j], &bubbleArray[j+1]);
                //swap numbers if Value > Next Value
                swapped = true;
                swp++;
            }
            comp++;
        }
        if (swapped == false) break; //Table sorted Algorithms done.
    }
}
//Selection Sort Algorithm
void selectionSort(int selectionArray[], int numOfInputs)
{
    int index = 1;
    if(index == numOfInputs)
        return;
    else
    {
        int highest = 0;
        int highIndex = 0;
        for(int i=0; i< numOfInputs; i++)
        {
            if(selectionArray[i] > highest)
            {
                highest = selectionArray[i];
                highIndex = i;
            }
        }
        int temp = selectionArray[numOfInputs-1];
        selectionArray[numOfInputs-1] = highest;
        selectionArray[highIndex] = temp;
        selectionSort(selectionArray, numOfInputs-1);
    }
}

```

```

}
//Insertion Sort Algorithm
void insertionSort(int insertionArray[], int numOfInputs){
    int comp = 0, assg = 0, h;
    //loop where the key is generated and the values are swapped likewise...
    for (h = 1; h < numOfInputs; h++) {
        int key = insertionArray[h];
        int k = h - 1;
        while (k >= 0 && key < insertionArray[k]) {
            insertionArray[k + 1] = insertionArray[k];
            comp++;
            assg++;
            --k;
        }
        insertionArray[k + 1] = key;
    }
}
//Quick Sort Algorithm
void quickSort(int quickArray[], int low, int numOfInputs){
    if (low < numOfInputs) {
        //returns the pivot -> some random middle value
        int pi = partitionMedian(quickArray, low, numOfInputs);
        quickSort(quickArray, low, pi - 1);
        quickSort(quickArray, pi + 1, numOfInputs);
    }
}
int partitionMedian (int arr[], int low, int high) {
    swap(&arr[(low + high)/2], &arr[high]);
    return partitionLast(arr, low, high);
}
int partitionLast (int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low, j;
    for (j = low; j <= high- 1; j++) {
        if (arr[j] <= pivot) {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[high]);
    return i;
}
}

```