Assignment-2

Name – Ujjwal Lehri
Roll No. – 60216403224
Course – B.Tech CSE 4th sem

Q. Write a program in C/C++, show its output, plot the graph, and calculate it's time complexity for

1. Bubble Sort
2. Selection Sort
3. Insertion Sort

Code:

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

int getRandom(int min, int max);
int* GenArr(int size);
void swap(int i, int j, int *Arr);

void BubbleSort(int *Arr, int size);
void SelectionSort(int *Arr, int size);
void InsertionSort(int *Arr, int size);

int main(){
   LARGE_INTEGER freq, start, end;
   QueryPerformanceFrequency(&freq);
   srand(time(0));

   int Sizes[] = {100, 1000, 10000, 100000};
   int n = sizeof(Sizes)/sizeof(Sizes[0]);

   printf("Bubble sort: \n");
   printf("Input size \t   time taken\n");
   printf("----------------------------------\n");
   for(int i=0; i<n; i++){
      int size = Sizes[i];
      int *Arr = GenArr(size);
```

```
        QueryPerformanceCounter(&start);
        BubbleSort(Arr, size);
        QueryPerformanceCounter(&end);

        double time_taken = (double)(end.QuadPart- start.QuadPart) * 1e9 / freq.QuadPart;
        printf("%-18d  %-5.2lf ns\n", size, time_taken);

        free(Arr);
    }

    printf("-------------------------------------------------------------------------------\n");
    printf("Selection sort: \n");
    printf("Input size \t   time taken\n");
    printf("-----------------------------------\n");
    for(int i=0; i<n; i++){
        int size = Sizes[i];
        int *Arr = GenArr(size);

        QueryPerformanceCounter(&start);
        SelectionSort(Arr, size);
        QueryPerformanceCounter(&end);

        double time_taken = (double)(end.QuadPart- start.QuadPart) * 1e9 / freq.QuadPart;
        printf("%-18d  %-5.2lf ns\n", size, time_taken);

        free(Arr);
    }

    printf("-------------------------------------------------------------------------------\n");
    printf("Insertion sort: \n");
    printf("Input size \t   time taken\n");
    printf("-----------------------------------\n");

    for(int i=0; i<n; i++){
        int size = Sizes[i];
        int *Arr = GenArr(size);

        QueryPerformanceCounter(&start);
        InsertionSort(Arr, size);
        QueryPerformanceCounter(&end);

        double time_taken = (double)(end.QuadPart- start.QuadPart) * 1e9 / freq.QuadPart;
        printf("%-18d  %-5.2lf ns\n", size, time_taken);

        free(Arr);
    }

}
```

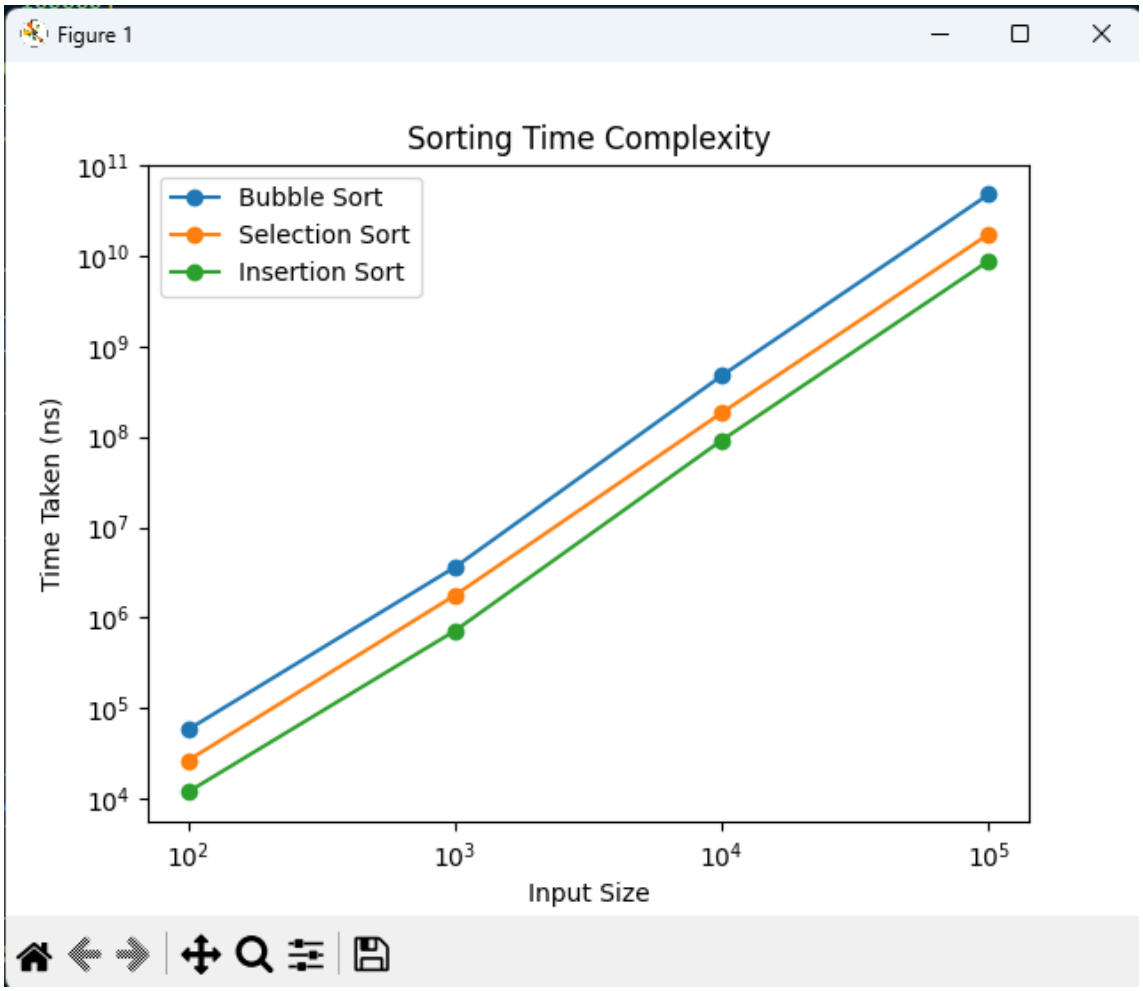
```
/*****************************************************************************/
/*                                  Bubble Sort                              */
/*****************************************************************************/

void BubbleSort(int *Arr, int size){
   for(int i = size-1; i > 0; i--){
      int flag = 0;
      for(int j = 0; j < i; j++){
         if(Arr[j] > Arr[j+1]){
            flag = 1;
            swap(j, j+1, Arr);
         }
      }
      if(!flag) break;
   }
}

/*****************************************************************************/
/*                                  Selection Sort                           */
/*****************************************************************************/

void SelectionSort(int *Arr, int size){
   for(int i = 0; i < size; i++){
      int minIndex = i;
      for(int j = i+1; j<size; j++){
         if(Arr[j] < Arr[minIndex]) minIndex = j;
      }
      if(minIndex != i) swap(i, minIndex, Arr);
   }
}

/*****************************************************************************/
/*                                  Insertion Sort                           */
/*****************************************************************************/

void InsertionSort(int *Arr, int size){
   for(int i = 1; i< size; i++){
      int val = Arr[i];
      int j = i- 1;
      while(j >-1 && val < Arr[j]){
         Arr[j+1] = Arr[j];
         j--;
      }
      Arr[++j] = val;
   }
}
```

```
/*****************************************************************************/
/*                                Helper functions                           */
/*****************************************************************************/

int getRandom(int min, int max){
   return min + rand()%(max-min);
}

int* GenArr(int size){
   int min = 1, max = 999;
   int* Arr = (int*)malloc(sizeof(int) * size);
   for (int i = 0; i < size; i++) {
      Arr[i] = getRandom(min, max);
   }
   return Arr;
}

void swap(int i, int j, int *Arr){
   int temp = Arr[i];
   Arr[i] = Arr[j];
   Arr[j] = temp;
}
```

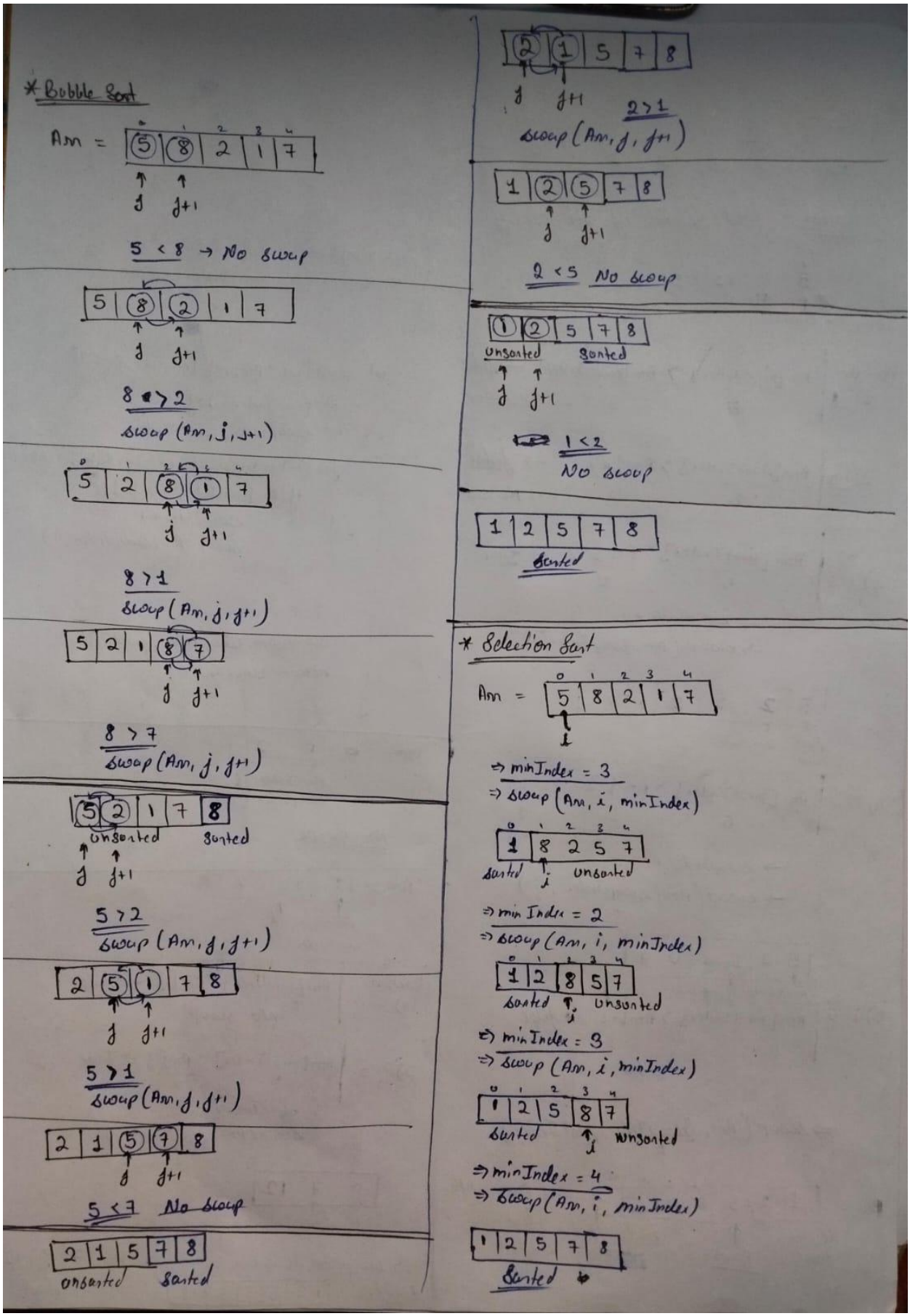
Output:

```
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\SortingAlgo> cd "c:\Users\Ujjwal\Desktop
go }
Bubble sort:
Input size          time taken
-----------------------------------
100                  57700.00 ns
1000                 3609400.00 ns
10000                474083600.00 ns
100000               47940138200.00 ns
-------------------------------------------------------------------------------
Selection sort:
Input size          time taken
-----------------------------------
100                  26300.00 ns
1000                 1764200.00 ns
10000                184330800.00 ns
100000               17425517200.00 ns
-------------------------------------------------------------------------------
Insertion sort:
Input size          time taken
-----------------------------------
100                  11800.00 ns
1000                 714300.00 ns
10000                91490000.00 ns
100000               8817683500.00 ns
```

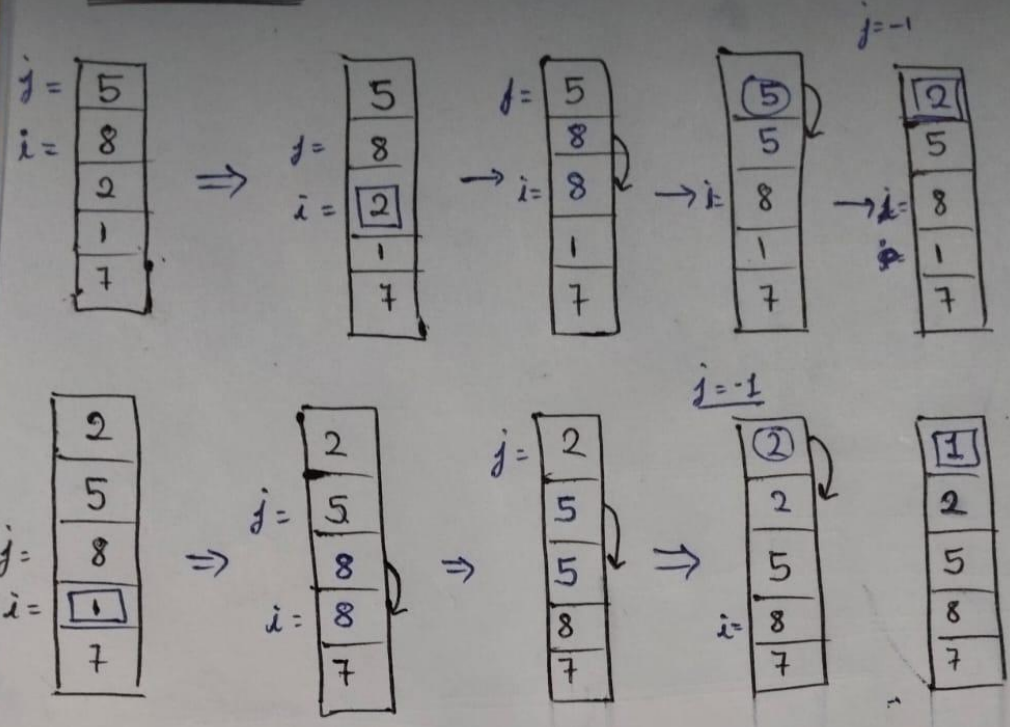
Graph: the graph compares the performance of Direct Search (using hashing), Linear Search, and Binary Search across different array sizes.



Dry Run: Dry run on array {5,8,2,1,7} using Bubble Sort, Selection Sort and Insertion Sort.



* Insertion Sort



~~Selection Sort~~