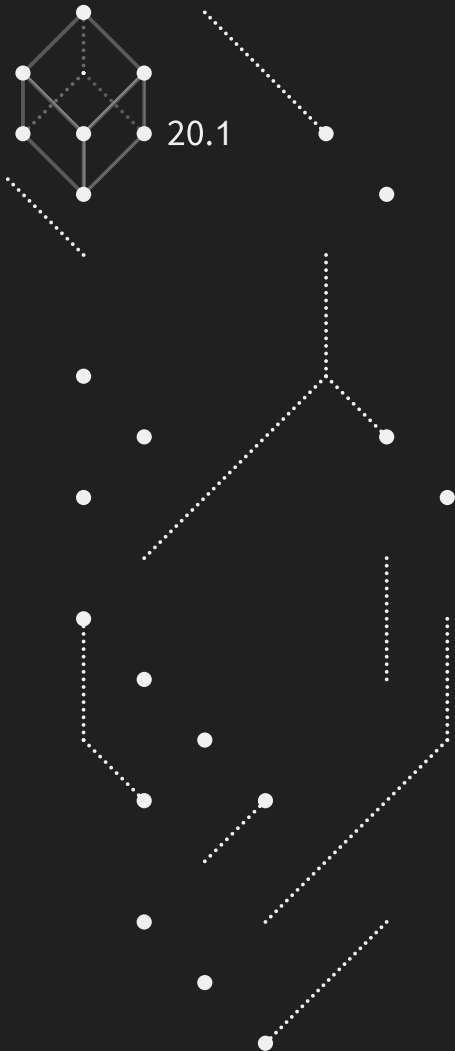




UNIVERSITY OF
ENGINEERING AND TECHNOLOGY
MARDAN



Name

A b d u l **H** a s e e b

Reg. No

2 2 M D S W E 1 9 **7**

Class. No

4

DSA Lab.

4

Submitted To.

E n g r . **S** o h a i l

1. Implement bubble sort as a function which takes a float array and the size of the array as parameters and returns the array sorted in descending order.
2. Implement Insertion sort as a function which takes a float array and the size of the array as parameters and returns the array sorted in descending order
3. Create a function named "InitArray" which takes an integer parameter N for the number of elements in the array and creates a float array of N elements randomly initializing the array with numbers between 0 and 1000. (Note: use rand() function).
4. Call both functions created in task 1 and task 2 from the main function with the float array in task 3 as parameter to both the functions. Display the sorted array returned by each function.
5. By adding the execution time calculation logic used in the Assignment # 1, Calculate the execution time for both algorithms and fill the values in the following table.
N=10 N=100 N=1000
Bubble Sort Execution Time
Insertion Sort Execution Time
6. Submit a printed report of the code snapshots (full screen) and output snapshots

1. Implement bubble sort as a function which takes a float array and the size of the array as parameters and returns the array sorted in descending order.

Bubble Sort

```
1
2
3  #include<iostream>
4
5  using namespace std;
6
7  void bubbleSort(int array[], int len){
8
9      int temp;
10
11     for(int i=0; i<len; i++){
12
13         for(int j=0; j<5-i-1; j++){
14
15             if(array[j] < array[j+1]){
16
17                 temp = array[j];
18                 array[j] = array[j+1];
19                 array[j+1] = temp;
20             }
21         }
22     }
23 }
24
25 }
26
27
28
29 int main(){
30
31     cout<<"Please enter the size of the array: " ;
32     int size;
33     cin>> size;
34
35     int array [size];
36     cout<<"\n Now enter the elements of the array: "<<endl;
37
38     for(int i=0; i<size; i++){
39         cin>>array[i];
40     }
41
42     bubbleSort(array, size);
43
44     cout<<"The sorted array is: "<<endl;
45
46     for(int i=0; i<size; i++){
47         cout<<"\n"<<array[i];
48     }
49 }
50
51 }
52
53
54
55
```

Output

```
Please enter the size of the array: 7

Now enter the elements of the array:
4
2
7
1
9
3
6
The sorted array is:

9
7
4
2
1
3
6
```

2. Implement Insertion sort as a function which takes a float array and the size of the array as parameters and returns the array sorted in descending order

Insertion Sort

```
1  #include<iostream>
2
3  using namespace std;
4
5
6
7
8  void insertionSort(int arr[], int len){
9
10     int i,j,temp;
11
12     for(int i=1; i<len; i++){
13
14         j=i;
15         temp = arr[i];
16
17         while(j>0 && temp> arr[j-1]){
18
19             arr[j] = arr[j -1];
20
21             j--;
22
23         }
24
25         arr[j] = temp;
26
27     }
28
29 }
30
31
32
33 int main(){
34
35
36     cout<<"Please enter the size of the array: " ;
37     int size;
38     cin>> size;
39
40     int array [size];
41     cout<<"\n Now enter the elements of the array: "<<endl;
42
43     for(int i=0; i<size; i++){
44         cin>>array[i];
45
46     }
47
48
49     insertionSort(array, size);
50
51
52
53     cout<<"after sorting the array: "<<endl;
54
55     for(int i=0; i<size; i++){
56         cout<<array[i];
57
58     }
59 }
60
```

Output

```
insertionSort }
Please enter the size of the array: 8

Now enter the elements of the array:
7
3
7
1
5
7
4
5
after sorting the array:
77755431
```

3. Create a function named “InitArray” which takes an integer parameter N for the number of elements in the array and creates a float array of N elements randomly initializing the array with numbers between 0 and 1000. (Note: use rand() function).

initArray

```
1  #include<iostream>
2  #include <cstdlib> // Required for rand() function
3  #include <ctime>
4  using namespace std;
5
6
7
8
9
10 void InitArray(float arr[], int N) {
11
12     for (int i = 0; i < N; i++) {
13         float randomFloat = (rand()%1000 ) *(0.9 - rand()%1);
14         arr[i] = randomFloat;
15     }
16 }
17
18
19
20
21
22
23 int main(){
24
25     cout<<"Enter the number of random floating points elements: ";
26
27     int size;
28     cin>> size;
29
30     float array[size];
31
32     InitArray(array, size);
33
34     for(int i=0; i<size; i++){
35         cout<<array[i]<<endl;
36     }
37
38
39
40
41
42
43
44 }
45
46
47
```

Output

```
.\3RandomGenerator }
Enter the number of random floating points elements: 11
420.3
450
651.6
322.2
417.6
130.5
744.3
441.9
847.8
392.4
543.6
```

4. Call both functions created in task 1 and task 2 from the main function with the float array in task 3 as parameter to both the functions. Display the sorted array returned by each function.

Float Sorting Algorithms

```
1 void bubbleSort(float array[], int len){
2
3     float temp;
4
5     for(int i=0; i<len-1; i++){
6
7
8
9         for(int j=0; j<len-i; j++){
10
11             if(array[j] < array[j+1]){
12
13                 temp = array[j];
14                 array[j] = array[j+1];
15                 array[j+1] = temp;
16             }
17         }
18     }
19 }
20
21 }
```

```
1 void insertionSort(float arr[], float len){
2
3     int i,j,temp;
4
5     for(int i=1; i<len; i++){
6
7         j=i;
8         temp = arr[i];
9
10        while(j>0 && temp> arr[j-1]){
11
12            arr[j] = arr[j-1];
13
14            j--;
15        }
16
17        arr[j] = temp;
18    }
19 }
20
21 }
```

Output [main() is on next page]

```
Please enter the size of the array: 6

Now enter the float elements of the array:
21
43
12
453
12
43
Sorted using bubble sort:
453
43
43
21
12
12

address of array0x7bdfdff6e0
```

```
Please enter the size of the array: 4

Now enter the float elements of the array:
12
43
12
65
Sorted using insertion sort:
65
43
12
12

address of array0x7bdfdff6e0
```

Main.cpp



```
1  #include "sortLibraries/bubble.h"
2  #include "sortLibraries/insertion.h"
3  #include<iostream>
4
5  using namespace std;
6
7  int main(){
8
9      cout<<"Please enter the size of the array: " ;
10     int size;
11     cin>> size;
12
13     float array [size];
14     cout<<"\n Now enter the float elements of the array: "<<endl;
15
16     for(int i=0; i<size; i++){
17         cin>>array[i];
18     }
19
20     bubbleSort(array, size);
21
22     cout<<"Sorted using bubble sort: \n";
23
24     for(int i=0; i<size; i++){
25         cout<<array[i]<<endl;
26     }
27
28     cout<<"\naddress of array"<<array<<"\n\n";
29
30
31
32     cout<<"\n\nPlease enter the size of the array: " ;
33     size;
34     cin>> size;
35
36     array [size];
37     cout<<"\n Now enter the float elements of the array: "<<endl;
38
39
40     for(int i=0; i<size; i++){
41         cin>>array[i];
42     }
43
44
45     cout<<"Sorted using insertion sort: \n";
46
47     insertionSort(array, size);
48
49     for(int i=0; i<size; i++){
50         cout<<array[i]<<endl;
51     }
52
53
54     cout<<"\naddress of array"<<array<<"\n";
55
56
57 }
```

5. By adding the execution time calculation logic used in the Assignment # 1, Calculate the execution time for both algorithms and fill the values in the following table.

N=10 N=100 N=1000

Bubble Sort Execution Time

Insertion Sort Execution Time

```
1
2 #include<iostream>
3 #include "sortLibraries/bubble.h"
4 #include "sortLibraries/insertion.h"
5 #include "sortLibraries/randomGenerator.h"
6 #include<chrono>
7 using namespace std;
8 using namespace chrono;
9 int main(){
10
11
12 float array [1000] ;
13
14 InitArray(array, 1000);
15
16
17 auto startTime = high_resolution_clock::now();
18
19 bubbleSort(array, 1000);
20
21 auto endTime = high_resolution_clock::now();
22
23 auto differenceTime = duration_cast<milliseconds>(endTime - startTime);
24
25 cout<<"Duration is "<<differenceTime.count();
26
27 cout<<"\n";
28
29
30 InitArray(array, 1000);
31
32
33 auto startTime = high_resolution_clock::now();
34
35 insertionSort(array, 1000);
36
37 auto endTime = high_resolution_clock::now();
38
39 auto differenceTime = duration_cast<milliseconds>(endTime - startTime);
40
41 cout<<"Duration is "<<differenceTime.count();
42
43 // cout<<"\n";
44 // for(int i=0; i<1000; i++){
45 //     cout<<array[i]<<" ";
46 // }
47
48
49 return 0;
50 }
```


5. By adding the execution time calculation logic used in the Assignment # 1, Calculate the execution time for both algorithms and fill the values in the following table.

N=10 N=100 N=1000

Bubble Sort Execution Time

Insertion Sort Execution Time

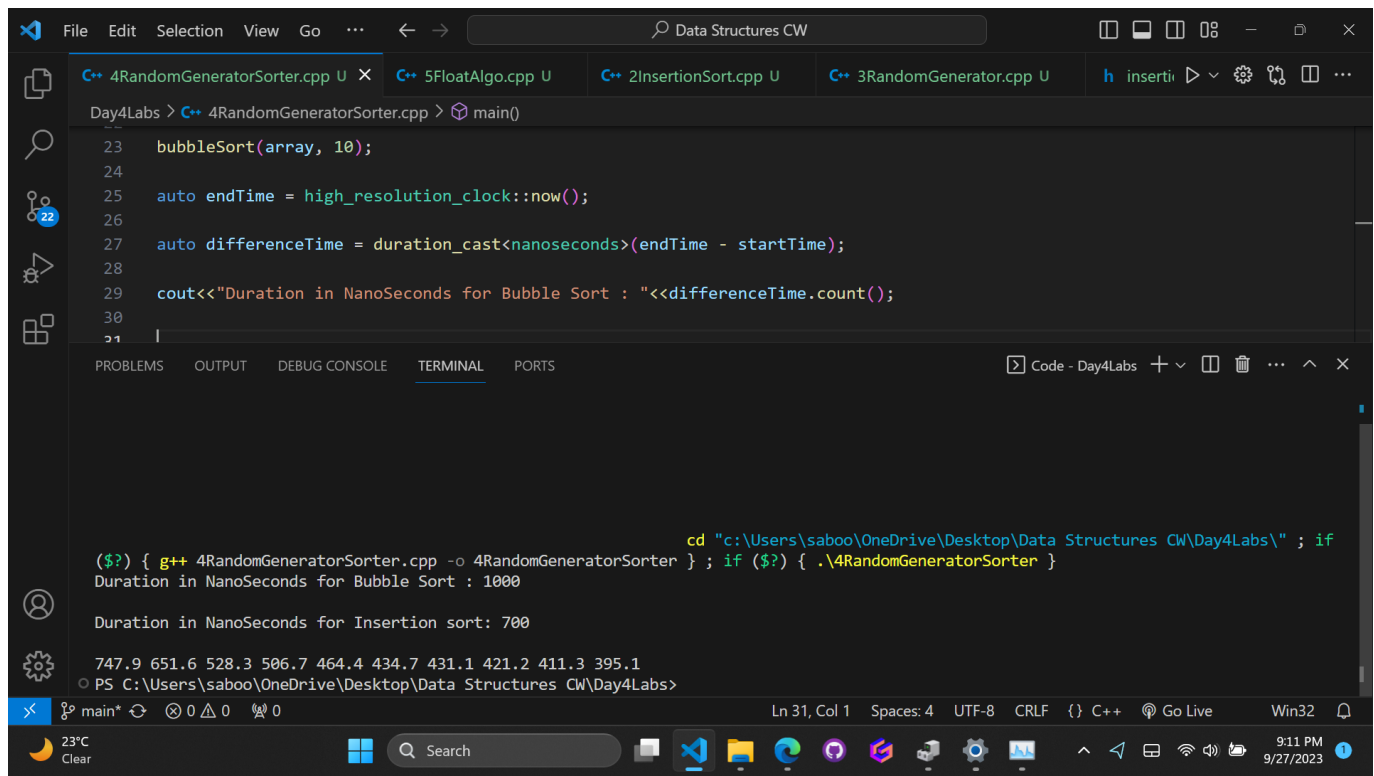
```
1
2 #include<iostream>
3 #include "sortLibraries/bubble.h"
4 #include "sortLibraries/insertion.h"
5 #include "sortLibraries/randomGenerator.h"
6 #include<chrono>
7 using namespace std;
8 using namespace chrono;
9 int main(){
10
11
12 float array [1000] ;
13
14 InitArray(array, 1000);
15
16
17 auto startTime = high_resolution_clock::now();
18
19 bubbleSort(array, 1000);
20
21 auto endTime = high_resolution_clock::now();
22
23 auto differenceTime = duration_cast<milliseconds>(endTime - startTime);
24
25 cout<<"Duration is "<<differenceTime.count();
26
27 cout<<"\n";
28
29
30 InitArray(array, 1000);
31
32
33 auto startTime = high_resolution_clock::now();
34
35 insertionSort(array, 1000);
36
37 auto endTime = high_resolution_clock::now();
38
39 auto differenceTime = duration_cast<milliseconds>(endTime - startTime);
40
41 cout<<"Duration is "<<differenceTime.count();
42
43 // cout<<"\n";
44 // for(int i=0; i<1000; i++){
45 //     cout<<array[i]<<" ";
46 // }
47
48
49 return 0;
50 }
```

6. Submit a printed report of the code snapshots (full screen) and output snapshots

For N=10

Duration in NanoSeconds for Bubble Sort : 1000

Duration in NanoSeconds for Insertion sort: 700



The screenshot shows a Visual Studio Code editor with a C++ project named "Data Structures CW". The active file is "4RandomGeneratorSorter.cpp". The code defines a `bubbleSort` function and a `main` function that measures the execution time of both `bubbleSort` and `insertionSort` for `N=10`. The terminal output shows the duration for Bubble Sort as 1000 and for Insertion sort as 700. Below these, a list of 20 random numbers generated by the program is displayed.

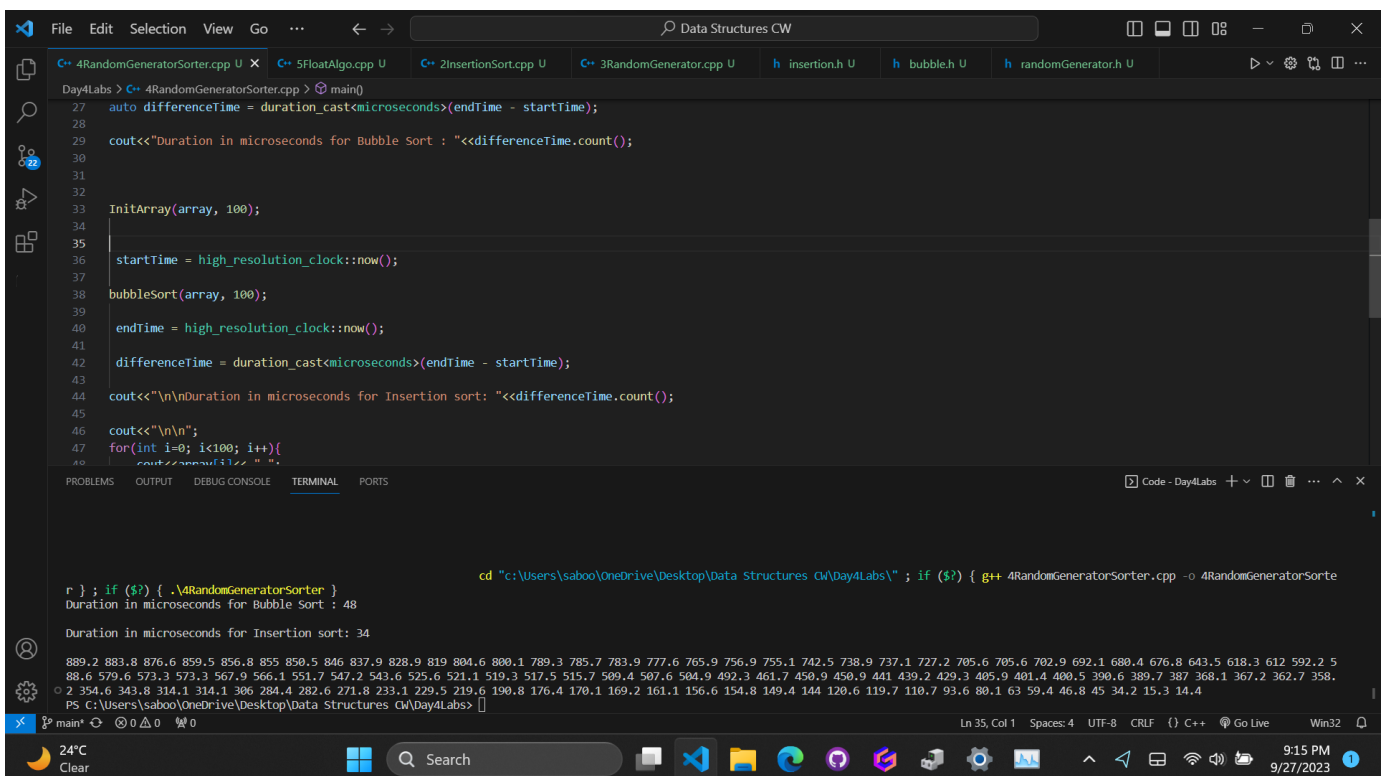
```
23 bubbleSort(array, 10);
24
25 auto endTime = high_resolution_clock::now();
26
27 auto differenceTime = duration_cast<nanoseconds>(endTime - startTime);
28
29 cout<<"Duration in NanoSeconds for Bubble Sort : "<<differenceTime.count();
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
cd "c:\Users\saboo\OneDrive\Desktop\Data Structures CW\Day4Labs\" ; if ($?) { g++ 4RandomGeneratorSorter.cpp -o 4RandomGeneratorSorter } ; if ($?) { .\4RandomGeneratorSorter }
Duration in NanoSeconds for Bubble Sort : 1000
Duration in NanoSeconds for Insertion sort: 700
747.9 651.6 528.3 506.7 464.4 434.7 431.1 421.2 411.3 395.1
PS C:\Users\saboo\OneDrive\Desktop\Data Structures CW\Day4Labs>
```

For N=100

Duration in microseconds for Bubble Sort : 48

Duration in microseconds for Insertion sort: 34



The screenshot shows a Visual Studio Code editor with a C++ project named "Data Structures CW". The active file is "4RandomGeneratorSorter.cpp". The code defines a `bubbleSort` function and a `main` function that measures the execution time of both `bubbleSort` and `insertionSort` for `N=100`. The terminal output shows the duration for Bubble Sort as 48 and for Insertion sort as 34. Below these, a list of 20 random numbers generated by the program is displayed.

```
27 auto differenceTime = duration_cast<microseconds>(endTime - startTime);
28
29 cout<<"Duration in microseconds for Bubble Sort : "<<differenceTime.count();
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

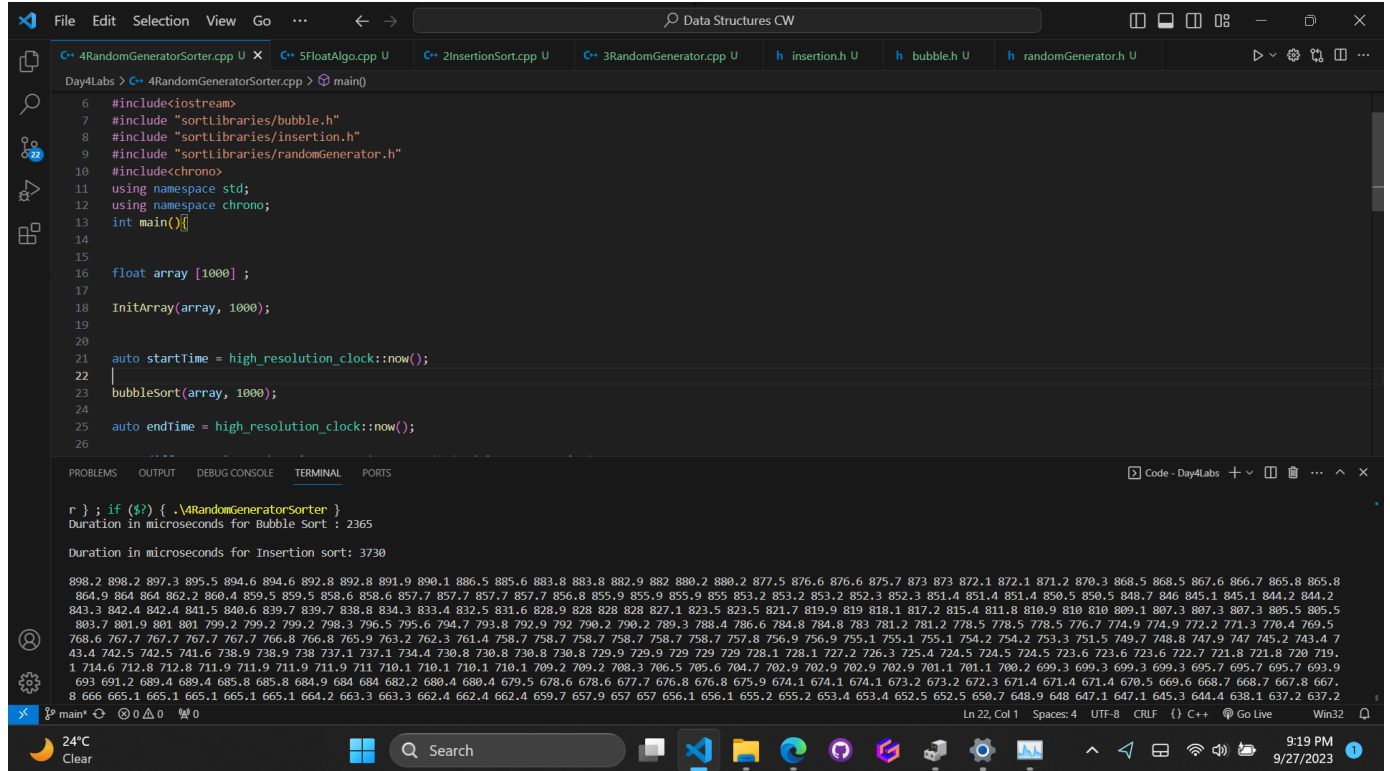
```
cd "c:\Users\saboo\OneDrive\Desktop\Data Structures CW\Day4Labs\" ; if ($?) { g++ 4RandomGeneratorSorter.cpp -o 4RandomGeneratorSorter } ; if ($?) { .\4RandomGeneratorSorter }
Duration in microseconds for Bubble Sort : 48
Duration in microseconds for Insertion sort: 34
889.2 883.8 876.6 859.5 856.8 855 850.5 846 837.9 828.9 819 804.6 800.1 789.3 785.7 783.9 777.6 765.9 756.9 755.1 742.5 738.9 737.1 727.2 705.6 705.6 702.9 692.1 680.4 676.8 643.5 618.3 612 592.2 5
88.6 579.6 573.3 573.3 567.9 566.1 551.7 547.2 543.6 525.6 521.1 519.3 517.5 515.7 509.4 507.6 504.9 492.3 461.7 450.9 450.9 441 439.2 429.3 405.9 401.4 400.5 390.6 389.7 387 368.1 367.2 362.7 358.
2 354.6 343.8 314.1 314.1 306 284.4 282.6 271.8 233.1 229.5 219.6 190.8 176.4 170.1 169.2 161.1 156.6 154.8 149.4 144 120.6 119.7 110.7 93.6 88.1 63 59.4 46.8 45 34.2 15.3 14.4
PS C:\Users\saboo\OneDrive\Desktop\Data Structures CW\Day4Labs>
```

6. Submit a printed report of the code snapshots (full screen) and output snapshots

For N=1000

Duration in microseconds for Bubble Sort : 2365

Duration in microseconds for Insertion sort: 3730



```
#include<iostream>
#include "sortlibraries/bubble.h"
#include "sortlibraries/insertion.h"
#include "sortlibraries/randomGenerator.h"
#include<chrono>
using namespace std;
using namespace chrono;
int main(){
    float array [1000];
    InitArray(array, 1000);
    auto startTime = high_resolution_clock::now();
    bubbleSort(array, 1000);
    auto endTime = high_resolution_clock::now();
    // ... (rest of the code) ...
    if ($?) { .\VRandomGeneratorSorter }
    Duration in microseconds for Bubble Sort : 2365
    Duration in microseconds for Insertion sort: 3730
    898.2 898.2 897.3 895.5 894.6 894.6 892.8 892.8 891.9 890.1 886.5 885.6 883.8 883.8 882.9 882 880.2 877.5 876.6 876.6 875.7 873 873 872.1 872.1 871.2 870.3 868.5 868.5 867.6 866.7 865.8 865.8
    864.9 864 864 862.2 860.4 859.5 859.5 858.6 858.6 857.7 857.7 857.7 857.7 856.8 855.9 855.9 855.9 855.9 853.2 853.2 852.3 852.3 851.4 851.4 851.4 850.5 850.5 848.7 846 845.1 845.1 844.2 844.2
    843.3 842.4 842.4 841.5 840.6 839.7 839.7 838.8 834.3 833.4 832.5 831.6 828.9 828 828 827.1 823.5 823.5 821.7 819.9 819 818.1 817.2 815.4 811.8 810.9 810 810 809.1 807.3 807.3 807.3 805.5 805.5
    803.7 801.9 801 801 799.2 799.2 799.2 798.3 796.5 795.6 794.7 793.8 792.9 792 790.2 789.3 788.4 786.6 784.8 784.8 783 781.2 781.2 778.5 778.5 778.5 776.7 774.9 774.9 772.2 771.3 770.4 769.5
    768.6 767.7 767.7 767.7 766.8 766.8 765.9 763.2 762.3 761.4 758.7 758.7 758.7 758.7 757.8 756.9 756.9 755.1 755.1 754.2 754.2 753.3 751.5 749.7 748.8 747.9 747 745.2 743.4 7
    43.4 742.5 742.5 741.6 738.9 738.9 738 737.1 737.1 734.4 730.8 730.8 730.8 729.9 729.9 729 729 728.1 728.1 727.2 726.3 725.4 724.5 724.5 724.5 723.6 723.6 722.7 721.8 721.8 720 719
    1 714.6 712.8 712.8 711.9 711.9 711.9 711.9 711.1 710.1 710.1 710.1 710.1 709.2 709.2 708.3 706.5 705.6 704.7 702.9 702.9 702.9 702.9 701.1 701.1 700.2 699.3 699.3 699.3 699.3 695.7 695.7 695.7 693.9
    693 691.2 689.4 689.4 685.8 685.8 684.9 684 684 682.2 680.4 680.4 679.5 678.6 678.6 677.7 676.8 676.8 675.9 674.1 674.1 674.1 673.2 673.2 672.3 671.4 671.4 671.4 670.5 669.6 668.7 668.7 667.8 667.8
    666 665.1 665.1 665.1 665.1 665.1 664.2 663.3 663.3 662.4 662.4 662.4 659.7 657.9 657 657 656.1 656.1 655.2 655.2 653.4 653.4 652.5 652.5 650.7 648.9 648 647.1 647.1 645.3 644.4 638.1 637.2 637.2
```

The End.

Submitted to : Engr. Sohail

Date: 27 / 9 / 2023

22MDSWE197

04 Abdul Haseeb