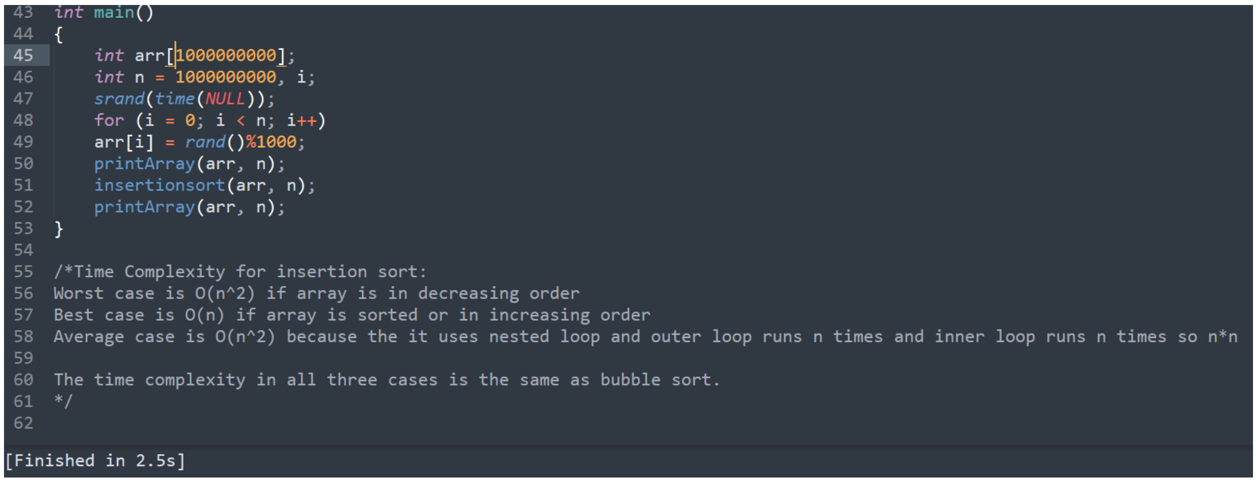
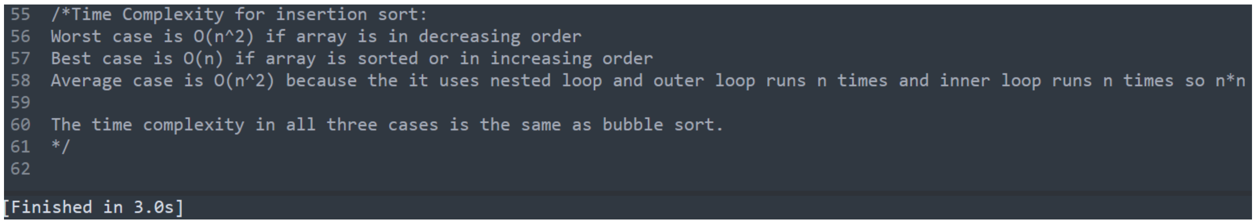
Insertion Sort:

1.



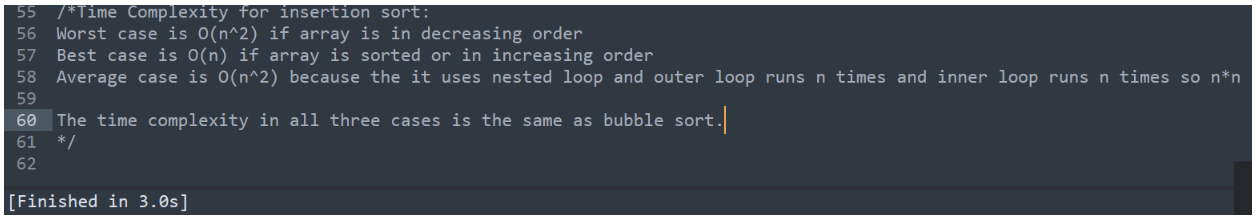
```
int main()
{
    int arr[1000000000];
    int n = 1000000000, i;
    srand(time(NULL));
    for (i = 0; i < n; i++)
    arr[i] = rand()%1000;
    printArray(arr, n);
    insertionsort(arr, n);
    printArray(arr, n);
}

/*Time Complexity for insertion sort:
Worst case is O(n^2) if array is in decreasing order
Best case is O(n) if array is sorted or in increasing order
Average case is O(n^2) because the it uses nested loop and outer loop runs n times and inner loop runs n times so n*n

The time complexity in all three cases is the same as bubble sort.
*/

[Finished in 2.5s]
```

2.



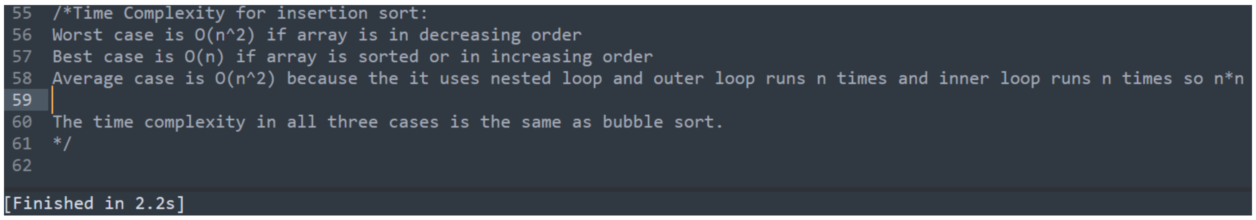
```
/*Time Complexity for insertion sort:
Worst case is O(n^2) if array is in decreasing order
Best case is O(n) if array is sorted or in increasing order
Average case is O(n^2) because the it uses nested loop and outer loop runs n times and inner loop runs n times so n*n

The time complexity in all three cases is the same as bubble sort.
*/

[Finished in 3.0s]
```

3.



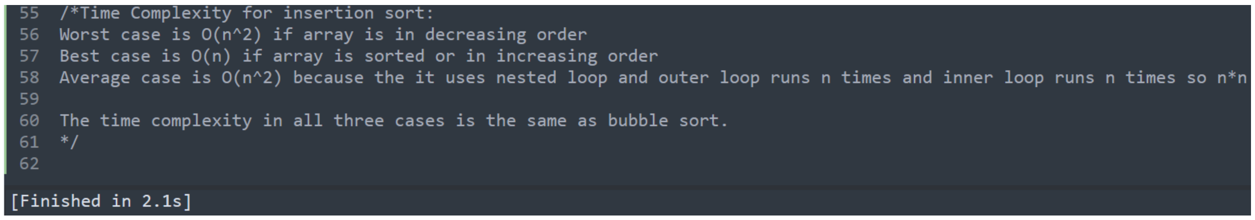
```
/*Time Complexity for insertion sort:
Worst case is O(n^2) if array is in decreasing order
Best case is O(n) if array is sorted or in increasing order
Average case is O(n^2) because the it uses nested loop and outer loop runs n times and inner loop runs n times so n*n

The time complexity in all three cases is the same as bubble sort.
*/

[Finished in 3.0s]
```

4.



```
/*Time Complexity for insertion sort:
Worst case is O(n^2) if array is in decreasing order
Best case is O(n) if array is sorted or in increasing order
Average case is O(n^2) because the it uses nested loop and outer loop runs n times and inner loop runs n times so n*n

The time complexity in all three cases is the same as bubble sort.
*/

[Finished in 2.2s]
```

5.

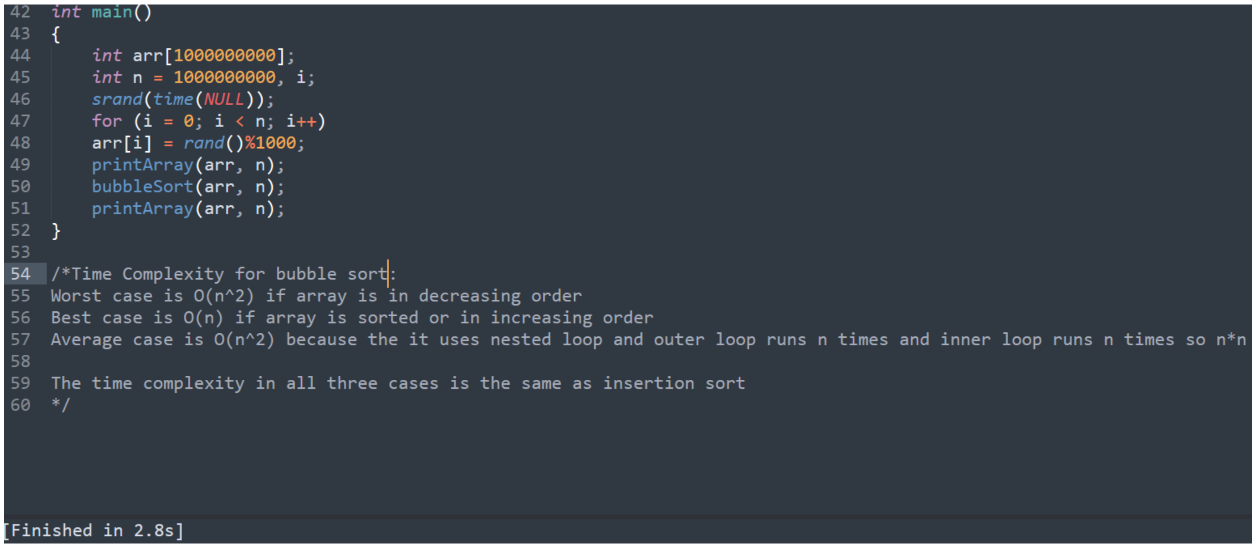


```
/*Time Complexity for insertion sort:
Worst case is O(n^2) if array is in decreasing order
Best case is O(n) if array is sorted or in increasing order
Average case is O(n^2) because the it uses nested loop and outer loop runs n times and inner loop runs n times so n*n

The time complexity in all three cases is the same as bubble sort.
*/

[Finished in 2.1s]
```

Average of all 5 is (2.5+3+3+2.2+2.1)/5 = 12.8/5 = 2.56

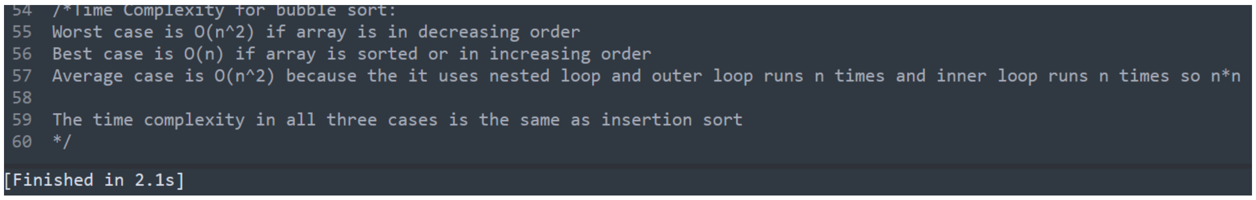
Bubble Sort:

1.



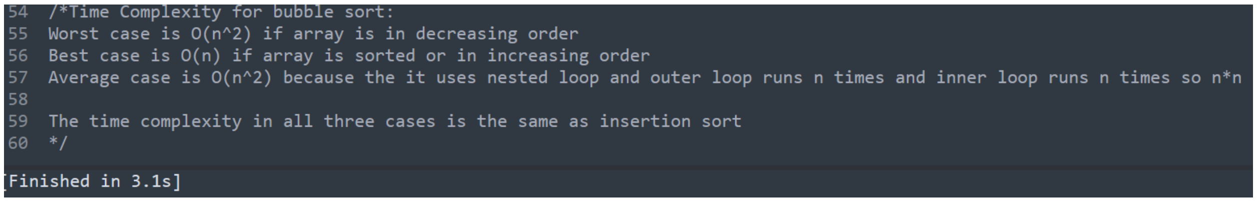
```
int main()
{
    int arr[1000000000];
    int n = 1000000000, i;
    srand(time(NULL));
    for (i = 0; i < n; i++)
    arr[i] = rand()%1000;
    printArray(arr, n);
    bubbleSort(arr, n);
    printArray(arr, n);
}

/*Time Complexity for bubble sort:
Worst case is O(n^2) if array is in decreasing order
Best case is O(n) if array is sorted or in increasing order
Average case is O(n^2) because the it uses nested loop and outer loop runs n times and inner loop runs n times so n*n

The time complexity in all three cases is the same as insertion sort
*/

[Finished in 2.8s]
```

2.



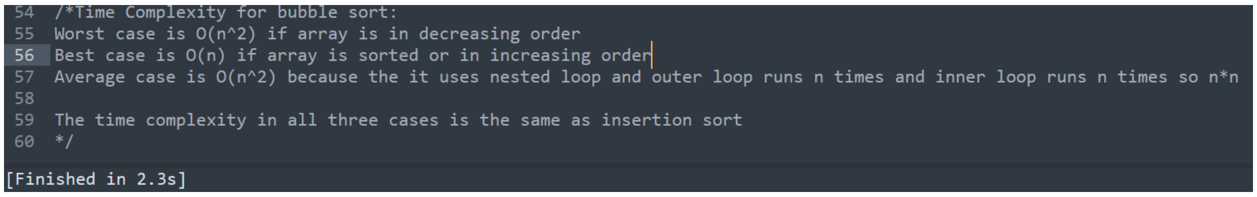
```
/*Time Complexity for bubble sort:
Worst case is O(n^2) if array is in decreasing order
Best case is O(n) if array is sorted or in increasing order
Average case is O(n^2) because the it uses nested loop and outer loop runs n times and inner loop runs n times so n*n

The time complexity in all three cases is the same as insertion sort
*/

[Finished in 2.1s]
```

3.



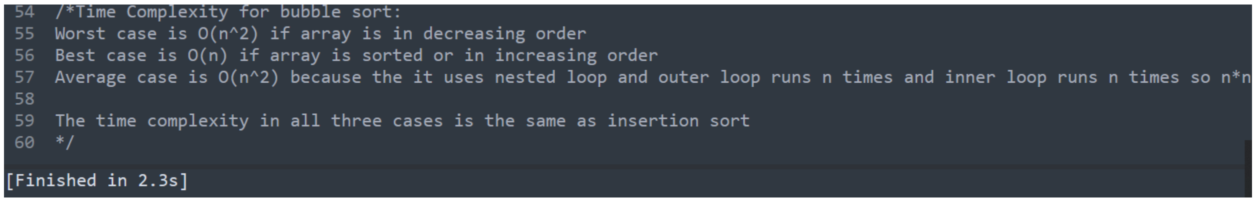
```
/*Time Complexity for bubble sort:
Worst case is O(n^2) if array is in decreasing order
Best case is O(n) if array is sorted or in increasing order
Average case is O(n^2) because the it uses nested loop and outer loop runs n times and inner loop runs n times so n*n

The time complexity in all three cases is the same as insertion sort
*/

[Finished in 3.1s]
```

4.



```
/*Time Complexity for bubble sort:
Worst case is O(n^2) if array is in decreasing order
Best case is O(n) if array is sorted or in increasing order
Average case is O(n^2) because the it uses nested loop and outer loop runs n times and inner loop runs n times so n*n

The time complexity in all three cases is the same as insertion sort
*/

[Finished in 2.3s]
```

5.



```
/*Time Complexity for bubble sort:
Worst case is O(n^2) if array is in decreasing order
Best case is O(n) if array is sorted or in increasing order
Average case is O(n^2) because the it uses nested loop and outer loop runs n times and inner loop runs n times so n*n

The time complexity in all three cases is the same as insertion sort
*/

[Finished in 2.3s]
```

Average of all 5 is (2.8+2.1+3.1+2.3+2.3)/5 = 12.6/5 = 2.52

Merge Sort:
I did the same thing as above and got the values as 2.2, 3.3, 2.4, 2.1, 2.6,
The average for this is 2.52.

Heap Sort:
2.1, 2.1, 2.5, 2.3, 2.0. The average for this is 2.2.

Quick Sort:
2.8, 2.3, 2.6, 2.3, 2.5. The average for this is 2.5.

Method: I have taken n as 1000000000 for all the sorting algorithms and took 5 outputs and took the average for them.

Insertion sort and bubble sort barely have any difference in their experimental data of time taken as the difference is only 0.04.

Merge sort, heap sort, and quick sort also barely have any difference in their experimental data of time taken as the difference is only 0.32 between max average and least average.

Note: I could have very easily got different values of averages for all the sorting algorithms because the array is randomly generated and even for the same array, the same sorting algorithm may take different time if we run it twice because of the way the compiler compiles every time.