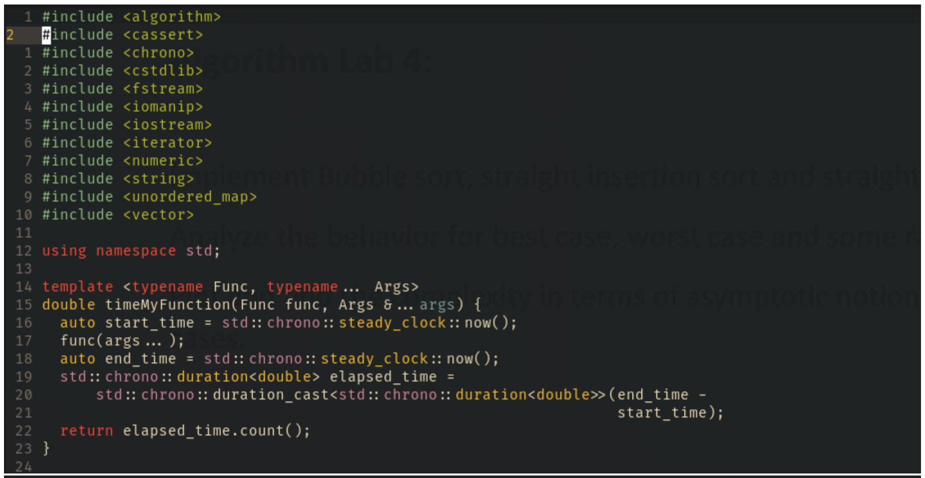
106119029, Dipesh Kafle, Lab4 Algos Lab

- Bubble Sort : Best Case is O(n) , Worst Case is O(nˆ2)
- Insertion Sort : Best Case is O(n) , Worst Case is O(nˆ2)
- Selection Sort : Best Case is O(nˆ2) , Worst Case is O(nˆ2)
- Some of the curves overlap so they are not visible that clearly

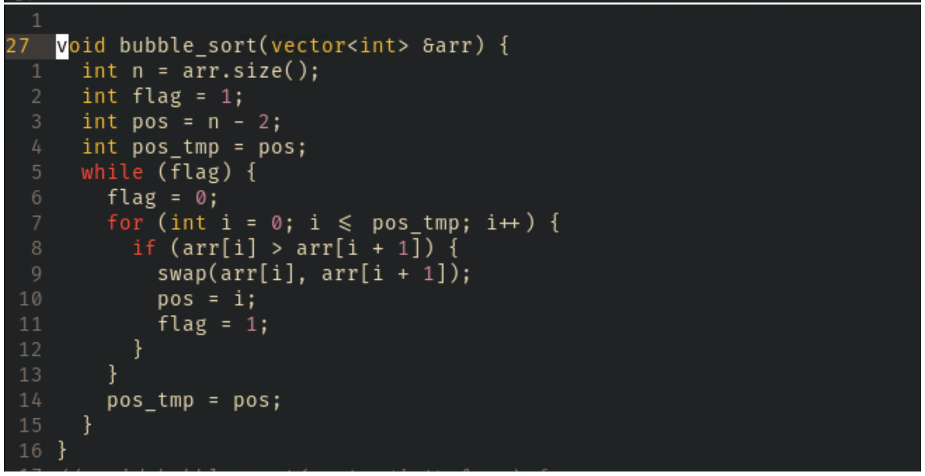
Question 1

Implement Bubble sort, straight insertion sort and straight selection sort

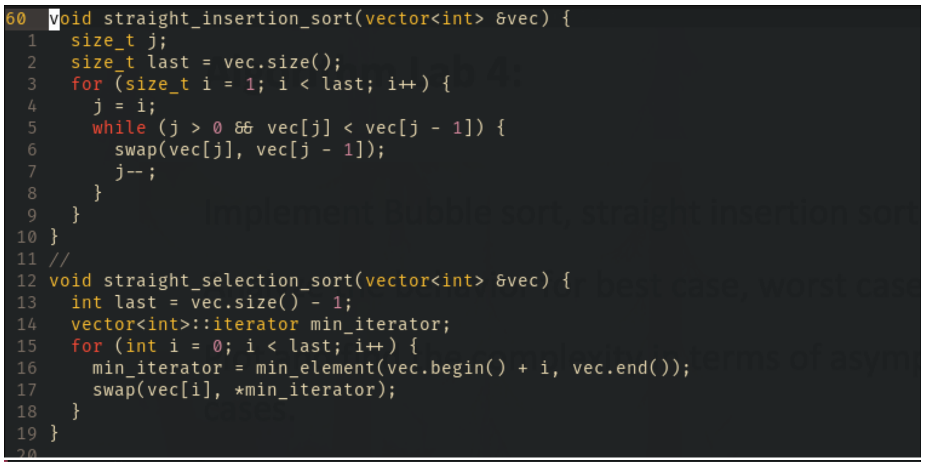
- Code



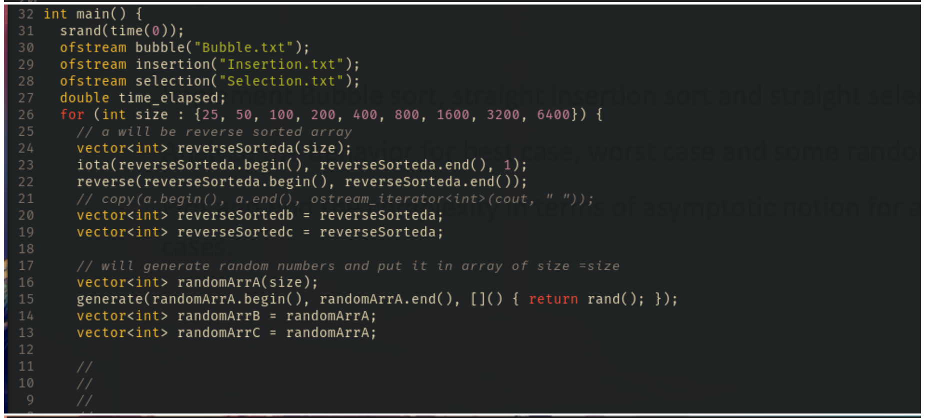
```cpp
#include <algorithm>
#include <cassert>
#include <chrono>
#include <cstdlib>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <iterator>
#include <numeric>
#include <string>
#include <unordered_map>
#include <vector>

using namespace std;

template <typename Func, typename... Args>
double timeMyFunction(Func func, Args &... args) {
  auto start_time = std::chrono::steady_clock::now();
  func(args... );
  auto end_time = std::chrono::steady_clock::now();
  std::chrono::duration<double> elapsed_time =
      std::chrono::duration_cast<std::chrono::duration<double>>(end_time -
                                                                 start_time);
  return elapsed_time.count();
}
```



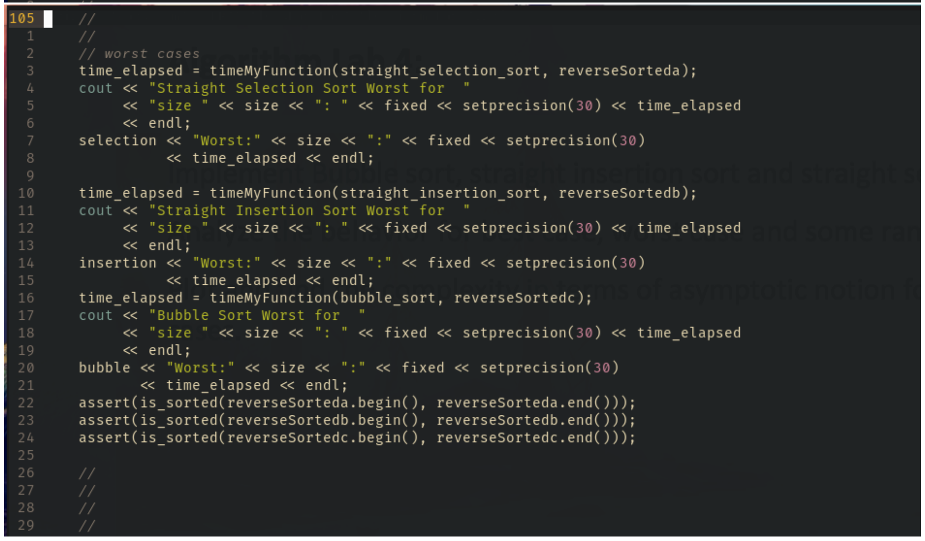
```cpp
void bubble_sort(vector<int> &arr) {
  int n = arr.size();
  int flag = 1;
  int pos = n - 2;
  int pos_tmp = pos;
  while (flag) {
    flag = 0;
    for (int i = 0; i <= pos_tmp; i++) {
      if (arr[i] > arr[i + 1]) {
        swap(arr[i], arr[i + 1]);
        pos = i;
        flag = 1;
      }
    }
    pos_tmp = pos;
  }
}
```



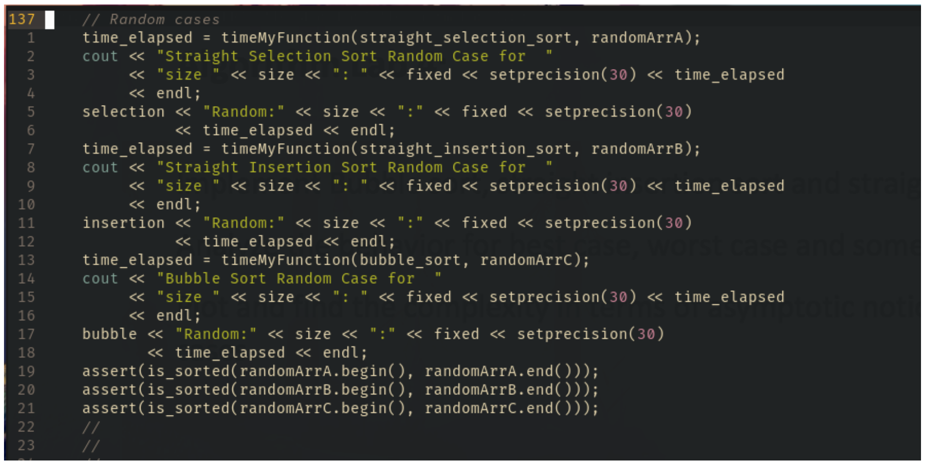
```cpp
void straight_insertion_sort(vector<int> &vec) {
  size_t j;
  size_t last = vec.size();
  for (size_t i = 1; i < last; i++) {
    j = i;
    while (j > 0 && vec[j] < vec[j - 1]) {
      swap(vec[j], vec[j - 1]);
      j--;
    }
  }
}
//
void straight_selection_sort(vector<int> &vec) {
  int last = vec.size() - 1;
  vector<int>::iterator min_iterator;
  for (int i = 0; i < last; i++) {
    min_iterator = min_element(vec.begin() + i, vec.end());
    swap(vec[i], *min_iterator);
  }
}
```



```cpp
int main() {
  srand(time(0));
  ofstream bubble("Bubble.txt");
  ofstream insertion("Insertion.txt");
  ofstream selection("Selection.txt");
  double time_elapsed;
  for (int size : {25, 50, 100, 200, 400, 800, 1600, 3200, 6400}) {
    // a will be reverse sorted array
    vector<int> reverseSorteda(size);
    iota(reverseSorteda.begin(), reverseSorteda.end(), 1);
    reverse(reverseSorteda.begin(), reverseSorteda.end());
    // copy(a.begin(), a.end(), ostream_iterator<int>(cout, " "));
    vector<int> reverseSortedb = reverseSorteda;
    vector<int> reverseSortedc = reverseSorteda;

    // will generate random numbers and put it in array of size =size
    vector<int> randomArrA(size);
    generate(randomArrA.begin(), randomArrA.end(), []() { return rand(); });
    vector<int> randomArrB = randomArrA;
    vector<int> randomArrC = randomArrA;

    //
    //
    //
```

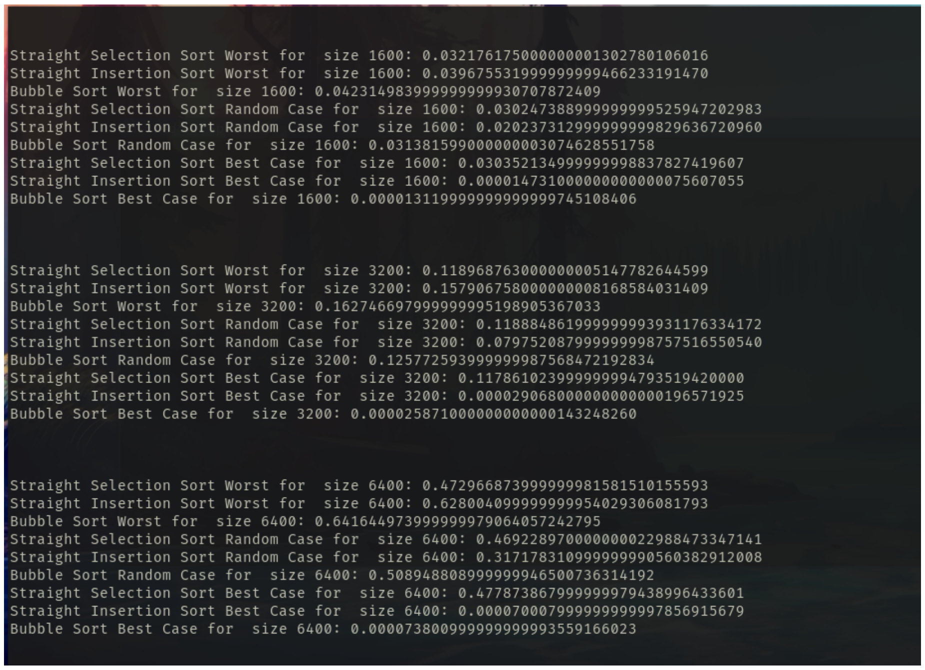


```cpp
    //
    //
    // worst cases
    time_elapsed = timeMyFunction(straight_selection_sort, reverseSorteda);
    cout << "Straight Selection Sort Worst for  "
         << "size " << size << ": " << fixed << setprecision(30) << time_elapsed
         << endl;
    selection << "Worst:" << size << ":" << fixed << setprecision(30)
              << time_elapsed << endl;

    time_elapsed = timeMyFunction(straight_insertion_sort, reverseSortedb);
    cout << "Straight Insertion Sort Worst for  "
         << "size " << size << ": " << fixed << setprecision(30) << time_elapsed
         << endl;
    insertion << "Worst:" << size << ":" << fixed << setprecision(30)
              << time_elapsed << endl;
    time_elapsed = timeMyFunction(bubble_sort, reverseSortedc);
    cout << "Bubble Sort Worst for  "
         << "size " << size << ": " << fixed << setprecision(30) << time_elapsed
         << endl;
    bubble << "Worst:" << size << ":" << fixed << setprecision(30)
           << time_elapsed << endl;
    assert(is_sorted(reverseSorteda.begin(), reverseSorteda.end()));
    assert(is_sorted(reverseSortedb.begin(), reverseSortedb.end()));
    assert(is_sorted(reverseSortedc.begin(), reverseSortedc.end()));

    //
    //
    //
    //
```

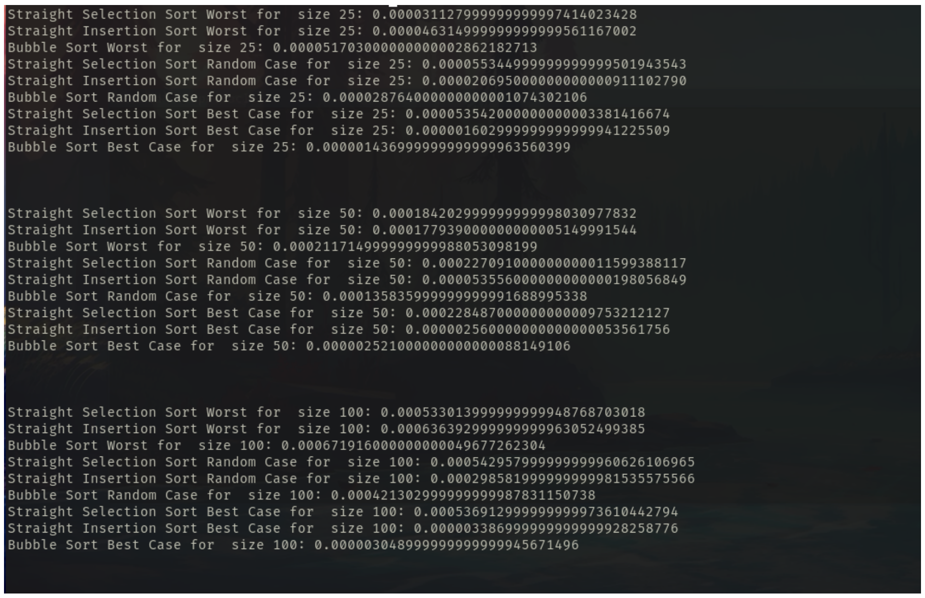


```cpp
// Random cases
time_elapsed = timeMyFunction(straight_selection_sort, randomArrA);
cout << "Straight Selection Sort Random Case for  "
     << "size " << size << ": " << fixed << setprecision(30) << time_elapsed
     << endl;
selection << "Random:" << size << ":" << fixed << setprecision(30)
          << time_elapsed << endl;
time_elapsed = timeMyFunction(straight_insertion_sort, randomArrB);
cout << "Straight Insertion Sort Random Case for  "
     << "size " << size << ": " << fixed << setprecision(30) << time_elapsed
     << endl;
insertion << "Random:" << size << ":" << fixed << setprecision(30)
          << time_elapsed << endl;
time_elapsed = timeMyFunction(bubble_sort, randomArrC);
cout << "Bubble Sort Random Case for  "
     << "size " << size << ": " << fixed << setprecision(30) << time_elapsed
     << endl;
bubble << "Random:" << size << ":" << fixed << setprecision(30)
       << time_elapsed << endl;
assert(is_sorted(randomArrA.begin(), randomArrA.end()));
assert(is_sorted(randomArrB.begin(), randomArrB.end()));
assert(is_sorted(randomArrC.begin(), randomArrC.end()));
//
//
```

- Output



```
Straight Selection Sort Worst for  size 1600: 0.032176175000000001302780106016
Straight Insertion Sort Worst for  size 1600: 0.039675531999999999466233191470
Bubble Sort Worst for  size 1600: 0.042314983999999999930707872409
Straight Selection Sort Random Case for  size 1600: 0.030247388999999999525947202983
Straight Insertion Sort Random Case for  size 1600: 0.020237312999999999829636720960
Bubble Sort Random Case for  size 1600: 0.031381599000000003074628551758
Straight Selection Sort Best Case for  size 1600: 0.030352134999999999883782741960 7
Straight Insertion Sort Best Case for  size 1600: 0.000014731000000000000075607055
Bubble Sort Best Case for  size 1600: 0.000013119999999999999745108406



Straight Selection Sort Worst for  size 3200: 0.118968763000000005147782644599
Straight Insertion Sort Worst for  size 3200: 0.157906758000000008168584031409
Bubble Sort Worst for  size 3200: 0.162746697999999995198905367033
Straight Selection Sort Random Case for  size 3200: 0.118884861999999993931176334172
Straight Insertion Sort Random Case for  size 3200: 0.079752087999999998757516550540
Bubble Sort Random Case for  size 3200: 0.125772593999999998756847219283 4
Straight Selection Sort Best Case for  size 3200: 0.117861023999999994793519420000
Straight Insertion Sort Best Case for  size 3200: 0.000029068000000000000196571925
Bubble Sort Best Case for  size 3200: 0.000025871000000000000143248260



Straight Selection Sort Worst for  size 6400: 0.472966873999999981581510155593
Straight Insertion Sort Worst for  size 6400: 0.628004099999999995402930608179 3
Bubble Sort Worst for  size 6400: 0.641644973999999979064057242795
Straight Selection Sort Random Case for  size 6400: 0.469228970000000022988473347141
Straight Insertion Sort Random Case for  size 6400: 0.317178310999999990560382912008
Bubble Sort Random Case for  size 6400: 0.508948808999999946500736314192
Straight Selection Sort Best Case for  size 6400: 0.477873867999999979438996433601
Straight Insertion Sort Best Case for  size 6400: 0.000070007999999999997856915679
Bubble Sort Best Case for  size 6400: 0.000073800999999999993559166023
```

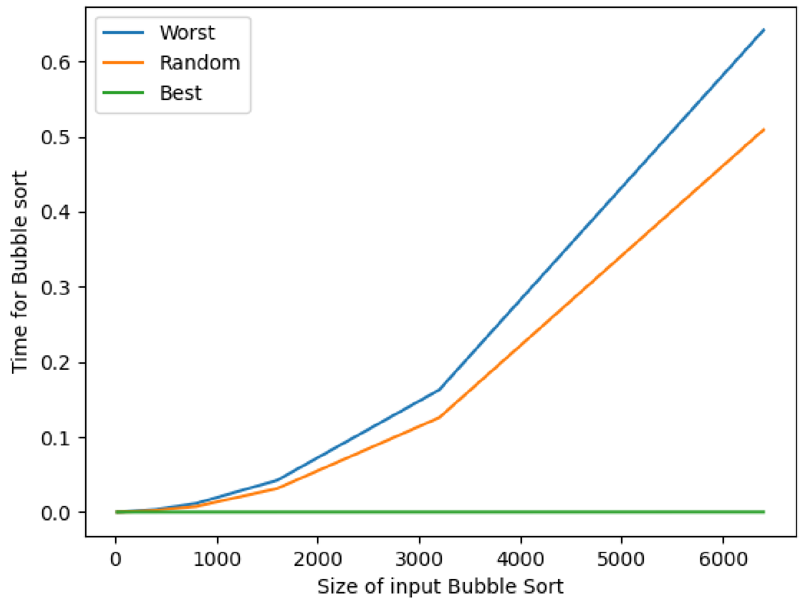


```
Straight Selection Sort Worst for  size 25: 0.000031127999999999997414023428
Straight Insertion Sort Worst for  size 25: 0.000046314999999999999561167002
Bubble Sort Worst for  size 25: 0.000051703000000000002862182713
Straight Selection Sort Random Case for  size 25: 0.000055344999999999999501943543
Straight Insertion Sort Random Case for  size 25: 0.000020695000000000000911102790
Bubble Sort Random Case for  size 25: 0.000028764000000000001074302106
Straight Selection Sort Best Case for  size 25: 0.000053542000000000003381416674
Straight Insertion Sort Best Case for  size 25: 0.000001602999999999999941225509
Bubble Sort Best Case for  size 25: 0.000001436999999999999963560399


Straight Selection Sort Worst for  size 50: 0.000184202999999999998030977832
Straight Insertion Sort Worst for  size 50: 0.000177939000000000005149991544
Bubble Sort Worst for  size 50: 0.000211714999999999988053098199
Straight Selection Sort Random Case for  size 50: 0.000227091000000000011599388117
Straight Insertion Sort Random Case for  size 50: 0.000053556000000000000198056849
Bubble Sort Random Case for  size 50: 0.000135835999999999991688995338
Straight Selection Sort Best Case for  size 50: 0.000228487000000000009753212127
Straight Insertion Sort Best Case for  size 50: 0.000002560000000000000053561756
Bubble Sort Best Case for  size 50: 0.000002521000000000000088149106


Straight Selection Sort Worst for  size 100: 0.000533013999999999948768703018
Straight Insertion Sort Worst for  size 100: 0.000636392999999999963052499385
Bubble Sort Worst for  size 100: 0.000671916000000000049677262304
Straight Selection Sort Random Case for  size 100: 0.000542957999999999960626106965
Straight Insertion Sort Random Case for  size 100: 0.000298581999999999981535575566
Bubble Sort Random Case for  size 100: 0.000421302999999999987831150738
Straight Selection Sort Best Case for  size 100: 0.000536912999999999973610442794
Straight Insertion Sort Best Case for  size 100: 0.000003386999999999999928258776
Bubble Sort Best Case for  size 100: 0.000003048999999999999945671496
```

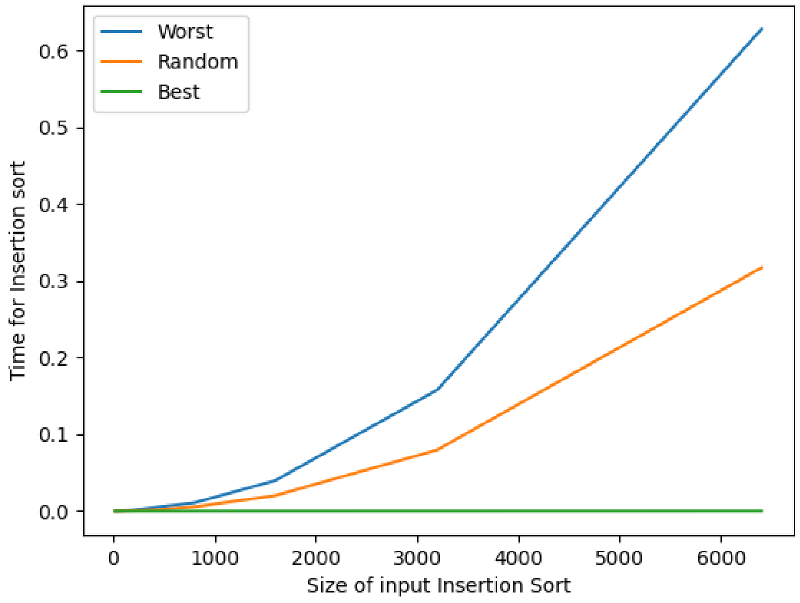
Question 2

Analyze the behavior for best case, worst case and some random cases

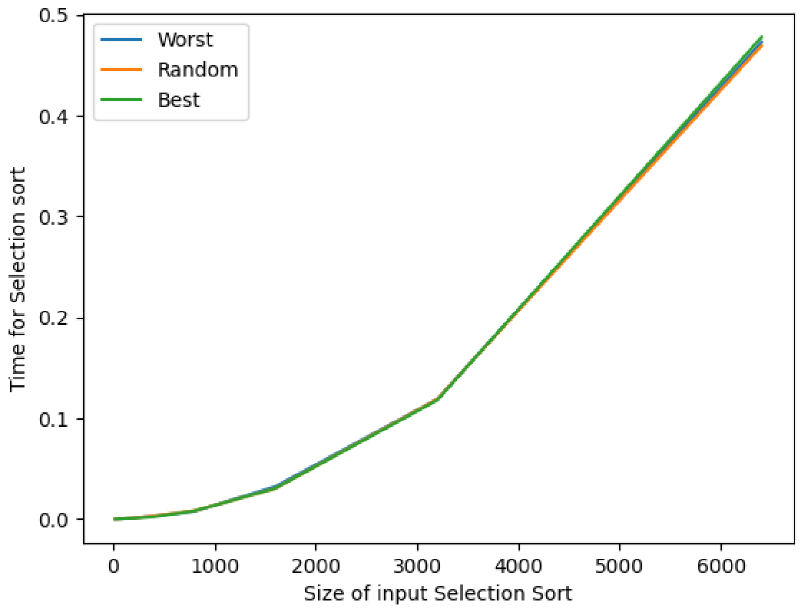
- Bubble Sort: Best , Worst and Random Case
  - Can be clearly seen that Worst case is O(nˆ2) and Best case is Linear



- Straight Insertion Sort: Best , Worst and Random Case
  - Can be clearly seen that Worst case is O(nˆ2) and Best case is Linear



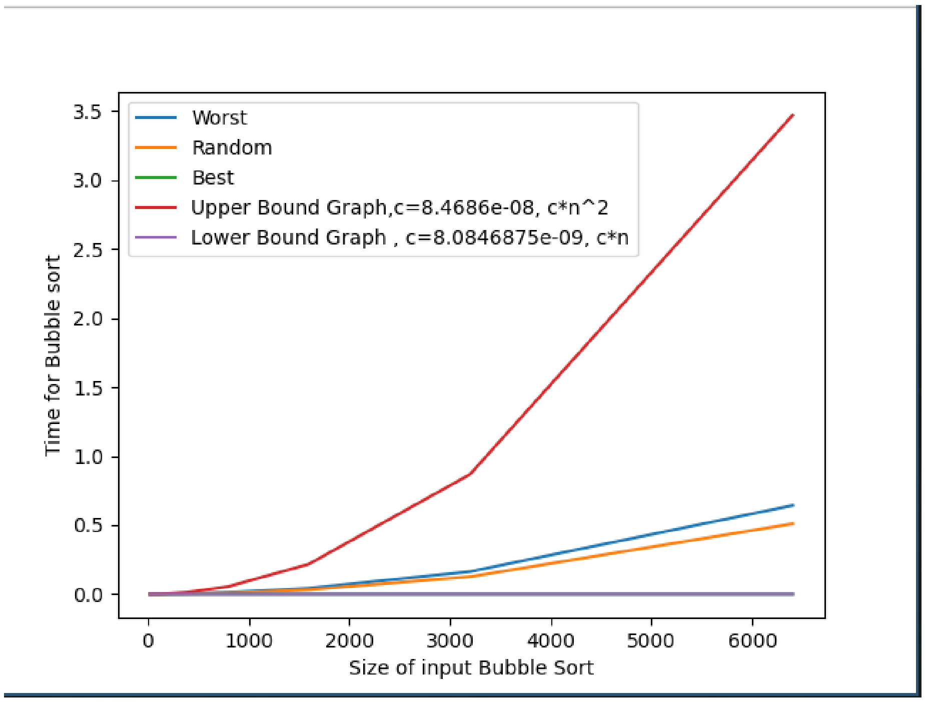
- Straight Selection Sort: Best , Worst and Random Case
  - Can be clearly seen that Worst case is O(nˆ2) and Best case is also O(nˆ2)



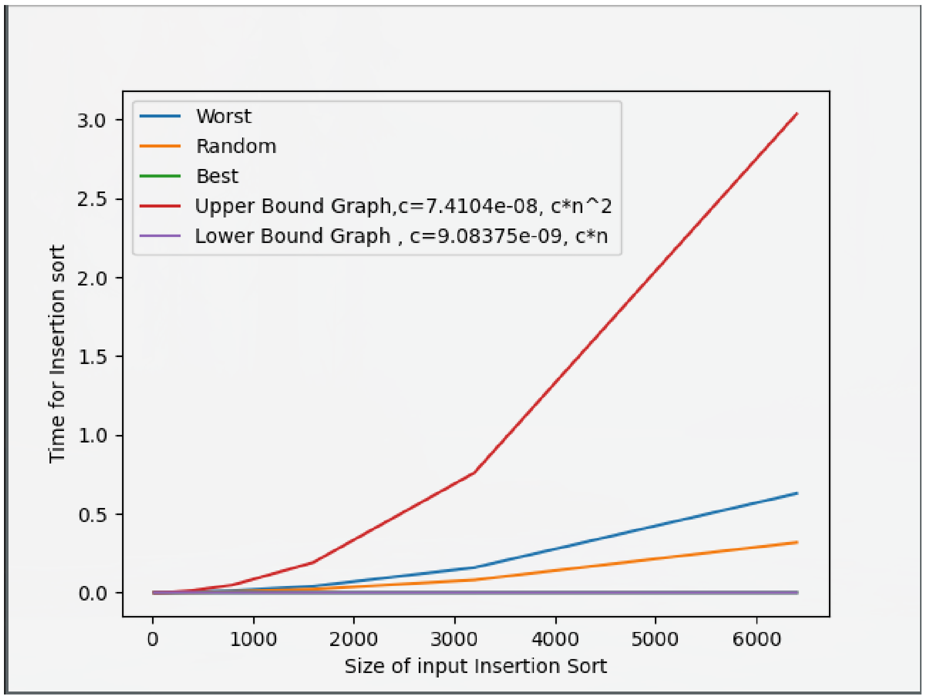
Question 3

Plot and find the complexity in terms of asymptotic notion for all these three

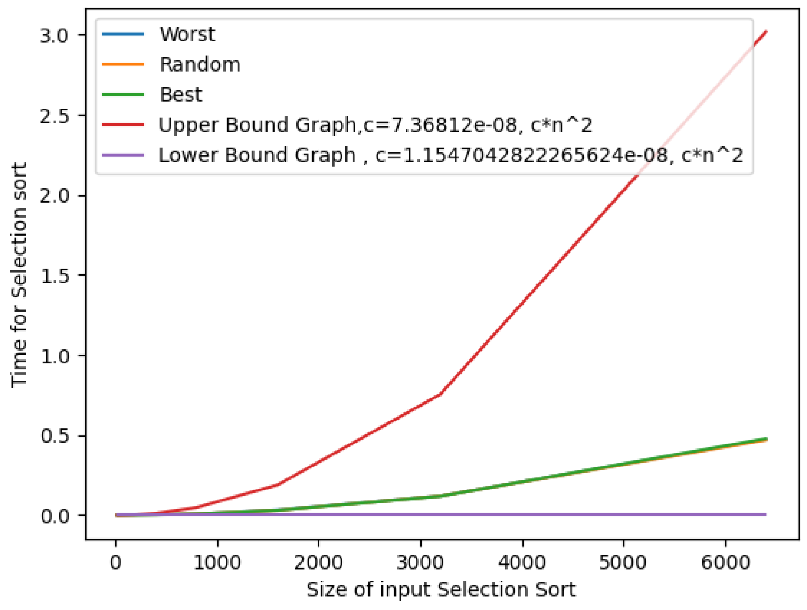
- Bubble
  - The constants have been shown in the curve for Upper and Lower Bound



- Insertion
  - The constants have been shown in the curve for Upper and Lower Bound



- Selection
  - The constants have been shown in the curve for Upper and Lower Bound



Observation

- We can see from the plots that Insertion and Bubble sort behave similarly in worst case. That is they both show linear complexity in the case when array is sorted(which is the best case). Selection sort however has best case O(nˆ2) only, making it the worst among these three.