File - /home/sergio/Desktop/EC_2/main.cpp

```cpp
#include <iostream>
#include <vector>
#include <random>
#include <chrono>

using std::getline;
using std::string;
using std::cout;
using std::endl;
using std::vector;
using std::cin;
using std::swap;
using std::default_random_engine;
using std::uniform_int_distribution;

bool get_line(const string& user_prompt, string& line){
    cout << user_prompt;
    getline(cin, line);
    return !line.empty();
}

void insertion_sort(vector<int>& arr){
    for(int i = 1;i < arr.size();i++){
        for(int j = i;j > 0 && arr[j] < arr[j - 1];j--){
            swap(arr[j], arr[j - 1]);
        }
    }
}

void partial_sort(vector<int>& arr, int lower_bound, int upper_bound){
    for(int i = lower_bound;i <= upper_bound;i++){
        for(int j = i;j > 0 && arr[j] < arr[j - 1];j--){
            swap(arr[j], arr[j - 1]);
        }
    }
}

int median_of_three(vector<int>& arr, int lo, int hi){
    int mid = (lo + hi) / 2;
    if(arr[lo] > arr[hi])
        swap(arr[lo], arr[hi]);
    if(arr[lo] > arr[mid])
        swap(arr[lo], arr[mid]);
    if(arr[mid] > arr[hi])
        swap(arr[mid], arr[hi]);
    return mid;
}

int partition(vector<int> &arr, int lo, int hi){
```

```
50      int left = lo - 1, right = hi;
51      int pivot_value = arr[hi];
52      while(true){
53          while(arr[++left] < pivot_value) if(left == hi)
    break;
54          while(arr[--right] > pivot_value) if(right == lo)
    break;
55          if(left >= right) break;
56          swap(arr[left], arr[right]);
57      }
58      swap(arr[left], arr[hi]);
59      return left;
60  }
61
62  void quick_sort(vector<int>& arr, int lo, int hi){
63      const int CUTOFF_TO_INSERTION_SORT = 3;
64      if(hi <= lo + CUTOFF_TO_INSERTION_SORT){
65          partial_sort(arr, lo, hi);
66          return;
67      }
68      int pivot_idx = median_of_three(arr, lo, hi);
69      swap(arr[pivot_idx], arr[hi]);
70      int new_pivot_idx = partition(arr, lo, hi);
71      quick_sort(arr, lo, new_pivot_idx - 1);
72      quick_sort(arr, new_pivot_idx + 1, hi);
73  }
74
75  void quick_sort(vector<int>& arr){
76      quick_sort(arr, 0, arr.size() - 1);
77  }
78
79  int main() {
80
81      string line;
82      while(get_line("enters a positive integer: ", line)){
83          int n = stoi(line);
84          unsigned seed = std::chrono::system_clock::now().
    time_since_epoch().count();
85          default_random_engine gen(seed);
86          const int LOWER_BOUND = -5000;
87          const int UPPER_BOUND = 5000;
88          const int TRIALS = 100;
89          vector<double> insertion_sort_times;
90          vector<double> quick_sort_times;
91          for(int i = 0;i < TRIALS;i++){
92              uniform_int_distribution<int>
    uniform_int_distribution(LOWER_BOUND, UPPER_BOUND);
93              vector<int> a;
94              for(int k = 0;i < n;i++)
95                  a.push_back(uniform_int_distribution(gen))
```

```
 95 ;
 96
 97             vector<int> insertion_sort_copy = a;
 98             auto start = std::chrono::steady_clock::now()
    ;
 99             insertion_sort(insertion_sort_copy);
100             auto end = std::chrono::steady_clock::now();
101             std::chrono::duration<double> elapsed_seconds
     = end - start;
102             insertion_sort_times.emplace_back(
    elapsed_seconds.count());
103
104             vector<int> quick_sort_copy = a;
105             start = std::chrono::steady_clock::now();
106             quick_sort(quick_sort_copy);
107             end = std::chrono::steady_clock::now();
108             elapsed_seconds = end - start;
109             quick_sort_times.emplace_back(elapsed_seconds
    .count());
110         }
111         double in_sort_avg_runtime = accumulate(
    insertion_sort_times.begin(), insertion_sort_times.end(),
     0.0) / TRIALS;
112         double quick_sort_avg_runtime = accumulate(
    quick_sort_times.begin(), quick_sort_times.end(), 0.0) /
    TRIALS;
113
114         cout << "The average runtime for insertion sort
    is: " << in_sort_avg_runtime << endl;
115         cout << "The average runtime for quick sort is: "
     << quick_sort_avg_runtime << endl;
116
117         cout << "Number of instructions computer can run
    in one second: " << (n*n  / in_sort_avg_runtime) << endl;
118     }
119 }
```

Insertion sort: $T(n) = \sum_{i=1}^{n-1} i = \frac{(n-1)\cdot n}{2} = O(n^2)$
Quicksort (avg time): $T(n) = 2 \cdot T(\frac{n}{2}) + O(n) = O(n \cdot log_2(n))$