```
#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){
   int i = lo - 1, j = hi;
   int pivot_value = arr[hi];
   while(true){
      while(arr[++i] < pivot_value) if(i == hi) break;
      while(arr[--j] > pivot_value) if(j == lo) break;
      if(i >= j) break;
      swap(arr[i], arr[j]);
   }
   swap(arr[i], arr[hi]);
   return i;
}
```

```
void quick_sort(vector<int>& arr, int lo, int hi){
    const int CUTOFF_TO_INSERTION_SORT = 3;
    if(hi <= lo + CUTOFF_TO_INSERTION_SORT){
        partial_sort(arr, lo, hi);
        return;
    }
    int pivot_idx = median_of_three(arr, lo, hi);
    swap(arr[pivot_idx], arr[hi]);
    int new_pivot_idx = partition(arr, lo, hi);
    quick_sort(arr, lo, new_pivot_idx - 1);
    quick_sort(arr, new_pivot_idx + 1, hi);
}

void quick_sort(vector<int>& arr){
    quick_sort(arr, 0, arr.size() - 1);
}

int main() {

    string line;
    while(get_line("enters a positive integer: ", line)){
        int n = stoi(line);
        unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
        default_random_engine gen(seed);
        const int LOWER_BOUND = -5000;
        const int UPPER_BOUND = 5000;
        uniform_int_distribution<int> uniform_int_distribution(LOWER_BOUND, UPPER_BOUND);
        vector<int> a;
        for(int i = 0;i < n;i++)
            a.push_back(uniform_int_distribution(gen));
        const int TRIALS = 100;
        vector<double> insertion_sort_times;
        vector<double> quick_sort_times;
        for(int i = 0;i < TRIALS;i++){
            vector<int> insertion_sort_copy = a;
            auto start = std::chrono::steady_clock::now();
            insertion_sort(insertion_sort_copy);
            auto end = std::chrono::steady_clock::now();
            std::chrono::duration<double> elapsed_seconds = end - start;
            insertion_sort_times.emplace_back(elapsed_seconds.count());

            vector<int> quick_sort_copy = a;
            start = std::chrono::steady_clock::now();
            quick_sort(quick_sort_copy);
            end = std::chrono::steady_clock::now();
            elapsed_seconds = end - start;
            quick_sort_times.emplace_back(elapsed_seconds.count());
        }
        double in_sort_avg_runtime = accumulate(insertion_sort_times.begin(), insertion_sort_times.end(), 0.0) / TRIALS;
        double quick_sort_avg_runtime = accumulate(quick_sort_times.begin(), quick_sort_times.end(), 0.0) / TRIALS;

        cout << "The average runtime for insertion sort is: " << in_sort_avg_runtime << endl;
        cout << "The average runtime for quick sort is: " << quick_sort_avg_runtime << endl;

    }
}
```