Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

Выполнил студент группы .....КС-30..... Лихолат Полина Николаевна
Ссылка на репозиторий: .....https://github.com/MUCTR-IKT-CPP/Likholat_algorithms.git

Приняли: .....Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: 20.02.2023

Оглавление

Описание задачи.

Изучить и реализовать метод сортировки вставками. Провести серию тестов для всех значений N из списка (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000).

По окончанию всех тестов нанести все точки, полученные в результате замеров времени на график где на ось абсцисс(X) нанести N, а на ось ординат(Y) нанести значения времени на сортировку. По полученным точкам построить график лучшего (минимальное время для каждого N), худшего (максимальное время для каждого N) и среднего (среднее время для каждого N) случая.

В качестве дополнительного задания, необходимо построить график худшего случая, и график O(c * g(N)), где g(N) соответствует асимптотической сложности рассматриваемого метода сортировки, подобрав такое значение C, что бы начиная с N ~ 1000 график асимптотической сложности возрастал быстрее чем полученное худшее время, но при этом был различим на графике.

Описание метода/модели.

Сортировка вставками (англ. *Insertion sort*) — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Вычислительная сложность — $O(n^2)$

На вход алгоритма подаётся последовательность n чисел: $a_1, a_2, \dots, a_n$. Сортируемые числа также называют *ключами*. Входная последовательность на практике представляется в виде массива с n элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности $a'_1, a'_2, \dots, a'_n$, чтобы выполнялось следующее соотношение $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей.

Время выполнения алгоритма зависит от входных данных: чем большее множество нужно отсортировать, тем большее время потребуется для выполнения сортировки. Также на время выполнения влияет исходная упорядоченность массива. Время работы алгоритма для различных входных данных одинакового размера зависит от элементарных операций, или шагов, которые потребуется выполнить.

Выполнение задачи.

Реализация алгоритма выполнена с использованием языка C++.

Мы подключили нужные библиотеки, создали функции, необходимы для реализации алгоритма, проведения тестов и записи результатов. Запись результатов осуществляется в файл.

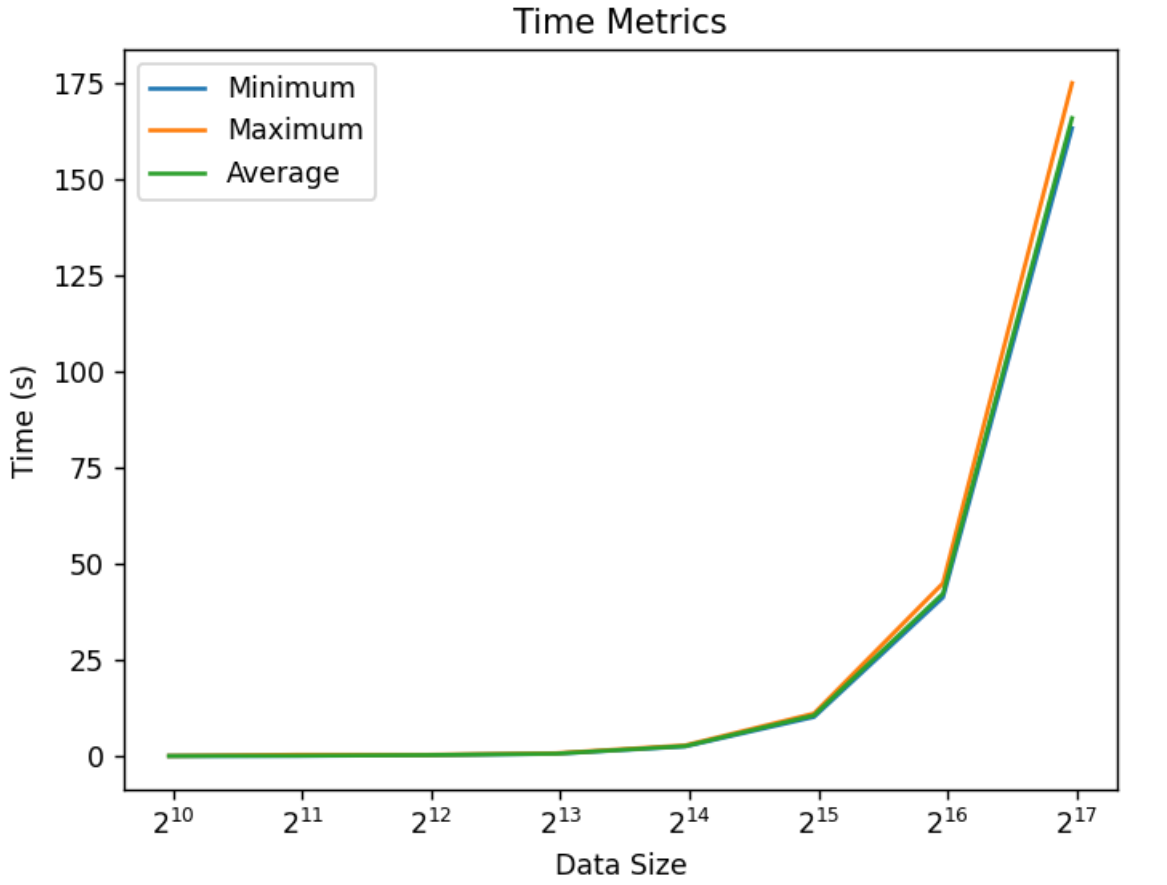
```
#include <iostream> // организации ввода-вывода
#include <vector> // реализует динамический массив
#include <random> // определяет средства для генерации случайных чисел
#include <chrono> // для определения классов и функций, которые представляют и управляют
длительностями времени и моментами времени
#include <fstream> // реализует вывод в файл
#include <numeric> // содержит коллекцию численных алгоритмов (reduce)

/*
* Сортировка вставками
* @param x - неотсортированный вектор
* @param N - длина вектора
* @return - отсортированный вектор
*/
std::vector<double> sort(std::vector<double> x, int N) {
    for (int i = 1; i < N; i++)
        for (int j = i; j > 0 && x[j - 1] > x[j]; j--) // пока j>0 и элемент j-1 > j, x-массив
int
            std::swap(x[j - 1], x[j]);   // меняем местами элементы j и j-1
    return x;
}

/*
* Генерация значений вектора
* @param v - ссылка на вектор
*/
void vectorGenerate(std::vector<double>& v)
{
    std::mt19937 engine(time(0));
    std::uniform_real_distribution<double> gen(-1.0, 1.0);
    for (auto& el : v)
        el = gen(engine);
}

/*
* Нахождение среднее значение каждого случая
*/
double getAverage(std::vector<double> const& v) {
    if (v.empty()) {
        return 0;
    }

    return std::reduce(v.begin(), v.end(), 0.0) / v.size();
}

int main()
{

    std::ofstream fout("result1.txt");

    std::vector<int> N = { 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000 };

    std::vector<double> min_time;
    std::vector<double> max_time;
    std::vector<double> avg_time;
    std::vector<double> time;

    for (auto it : N)
    {
        std::vector<double> timing;
```

```
        for (int i = 0; i < 20; i++)
        {
            std::vector<double> v(it);
            vectorGenerate(v);
            std::chrono::high_resolution_clock::time_point start =
std::chrono::high_resolution_clock::now();
            sort(v, it);
            std::chrono::high_resolution_clock::time_point end =
std::chrono::high_resolution_clock::now();
            std::chrono::duration<double> sec_diff = end - start;
            time.push_back(sec_diff.count());
            timing.push_back(sec_diff.count());
            std::cout << "Length: " << it << "  SIZE: " << size(v) << "  Time: " <<
sec_diff.count() << " sec." << std::endl;
            fout << "Length: " << it << "  SIZE: " << size(v) << "  Time: " <<
sec_diff.count() << " sec." << std::endl;
        }
        auto it = std::minmax_element(timing.begin(), timing.end());
        double min = *it.first;
        double max = *it.second;
        min_time.push_back(min);
        max_time.push_back(max);
        avg_time.push_back(getAverage(timing));
    }


    fout << "min time : ";
    std::cout << "min time : ";
    for (auto it : min_time)
    {
        fout << it << " ";
        std::cout << it << " ";
    }

    fout << "max time : ";
    std::cout << "max time : ";
    for (auto it : max_time)
    {
        fout << it << " ";
        std::cout << it << " ";
    }

    fout << "avg time : ";
    std::cout << "avg time : ";
    for (auto it : avg_time)
    {
        fout << it << " ";
        std::cout << it << " ";
    }

    fout << "time : ";
    std::cout << "time : ";
    for (auto it : time)
    {
        fout << it << " ";
        std::cout << it << " ";
    }


    fout.close();
  }
```

По полученным результатам строим графики с помощью Python.

```
import
matplotlib.pyplot
```

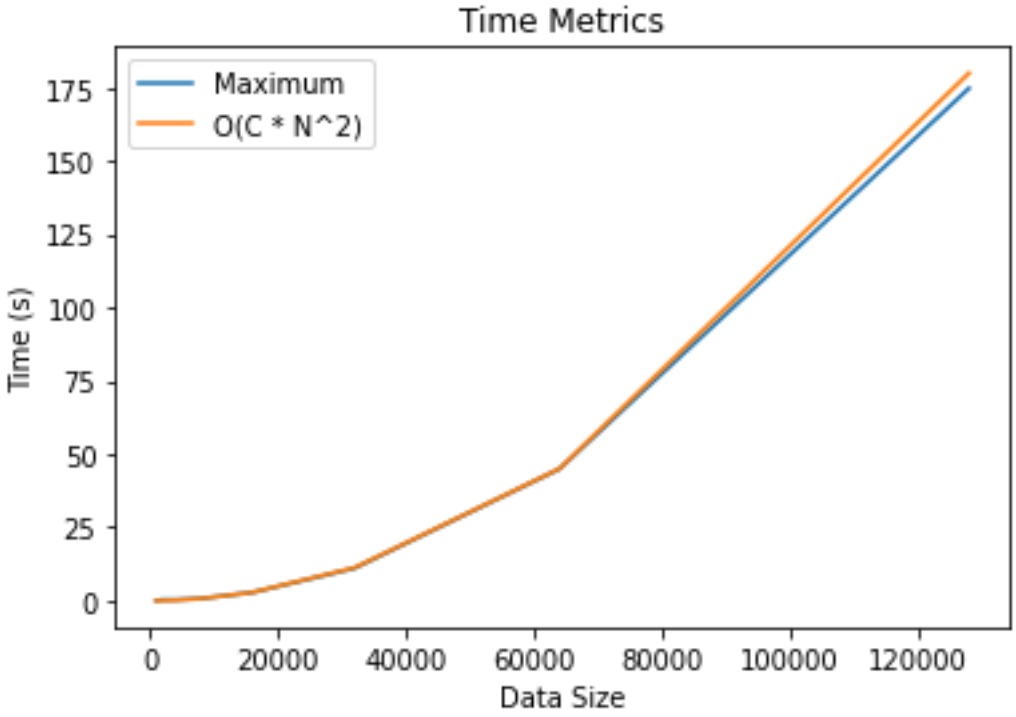
```
as plt
                    x = [1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000]
                    min_time = [0.0153646, 0.0645129, 0.224572, 0.605739, 2.49773, 10.1627, 41.2322,
                    163.104]
                    max_time = [0.0330504, 0.34036, 0.327446, 0.737365, 2.77065, 11.055, 44.9049, 174.81]
                    avg_time = [0.0174678, 0.101816, 0.259165, 0.630688, 2.59273, 10.5708, 42.21,
                    165.707]
                    plt.plot(x, min_time, label='Minimum')
                    plt.plot(x, max_time, label='Maximum')
                    plt.plot(x, avg_time, label='Average')
                    plt.xscale('log', base=2)
                    plt.xlabel('Data Size')
                    plt.ylabel('Time (s)')
                    plt.title('Time Metrics')
                    plt.legend()
                    plt.show()
```



Так же, используя Python, построим график для дополнительного задания.

```
import
matplotlib.pyplot
as plt
                    import numpy as np
                    x = [1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000]
                    max_time = [0.0330504, 0.34036, 0.327446, 0.737365, 2.77065, 11.055, 44.9049, 174.81]
                    C = 1.098727908132172e-08
                    plt.plot(x, max_time, label='Maximum')
                    plt.plot(x, C * np.array(x)* np.array(x), label='O(C * N^2)')
```

```
plt.xlabel('Data Size')
plt.ylabel('Time (s)')
plt.title('Time Metrics')
plt.legend()
plt.show()
```



Заключение.

Алгоритм сортировки вставками не самый лучший с точки зрения производительности, но традиционно более эффективен, чем большинство других простых $O(n^2)$ алгоритмов, таких как сортировка выбором или пузырьком. Также его преимуществом является то, что сортировать список можно по мере его получения. Скорость сортировки зависит от изначальной упорядоченности списка.