

Описание задачи. 2

Описание метода/модели. 2

Выполнение задачи. 2

Заключение. 7

Описание задачи.

В рамках лабораторной работы необходимо изучить и реализовать метод сортировки перемешиванием. Для реализованного метода сортировки необходимо провести серию тестов для всех значений N из списка (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000), при этом:

- в каждом тесте необходимо по 20 раз генерировать вектор, состоящий из N элементов;
- каждый элемент массива заполняется случайным числом с плавающей запятой от -1 до 1;
- каждый массив после генерации необходимо отсортировать и замерить время, требуемое на сортировку;
- результат замера для каждой попытки каждого теста записать в файл общий файл.

По окончании всех тестов необходимо 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.

Описание метода/модели.

Сортировка перемешиванием схожа с сортировкой “пузырьком”. В ней пределы той части массива в которой есть перестановки, сужаются. Также внутренние циклы проходят по массиву то в одну, то в другую сторону, перемещая наименьший элемент в конец, а наибольший элемент в начало за одну итерацию внешнего цикла.

На выполнение сортировки данным методом понадобится в 2 раза меньше времени, чем сортировкой “пузырьком”. Также сортировка перемешиванием является методом без привлечения дополнительной памяти.

Из преимуществ стоит отметить, что данная сортировка работает за меньшее количество итераций, по сравнению с сортировкой пузырьком. Также достаточно простая реализация алгоритма. Однако, так как количество сравнений такое же, поэтому скорость работы программы увеличится не намного. Поэтому главный минус этого метода в скорости работы.

Выполнение задачи.

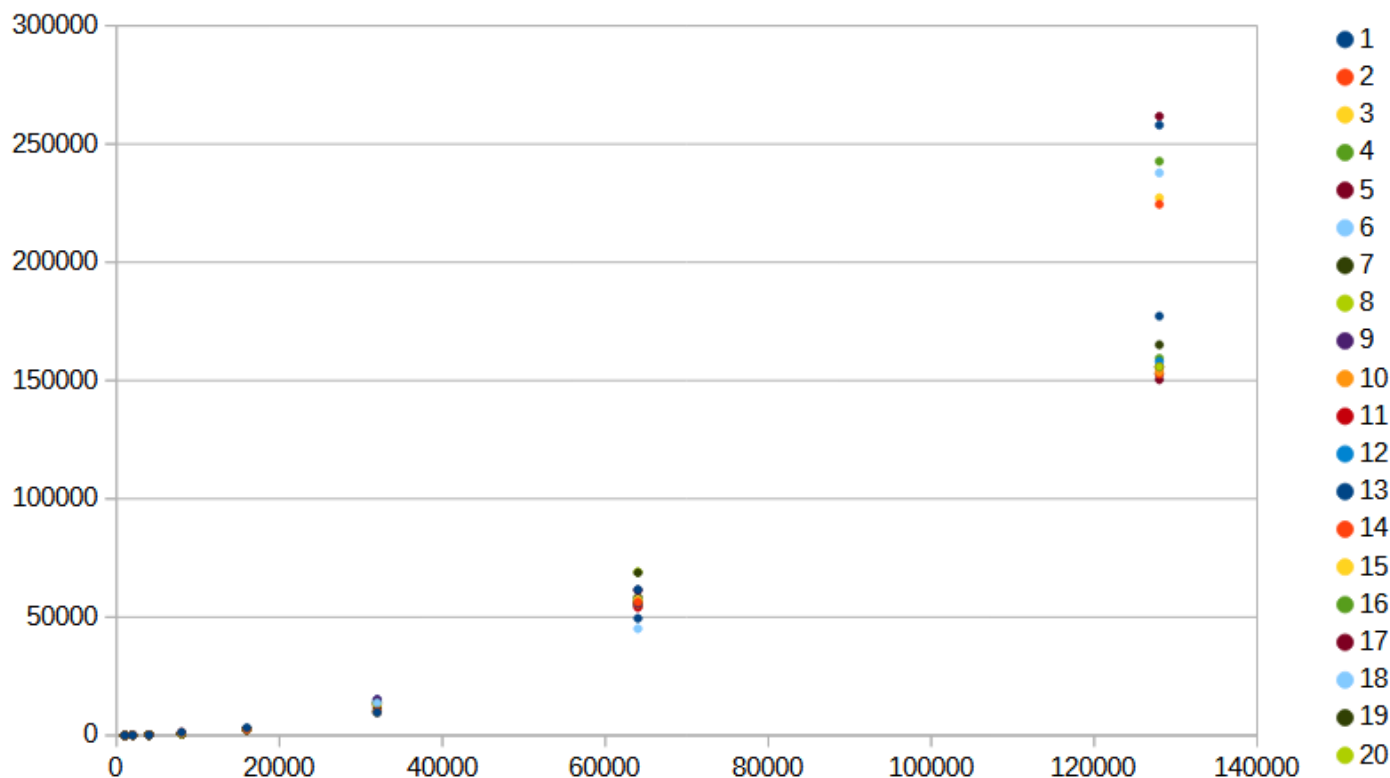
Для реализации данного метода сортировки использовался язык программирования C++.

Изначально мы идем с начала в конец. Берем первый элемент и сравниваем его со следующим и при необходимости делаем перестановку. Далее берем второй элемент и совершаем те же действия. Таким образом, мы идем до конца массива. После этого мы начинаем новый цикл с конца в начало и таким же методом делаем перестановку. Эти действия мы продолжаем до тех пор пока массив не будет отсортирован.

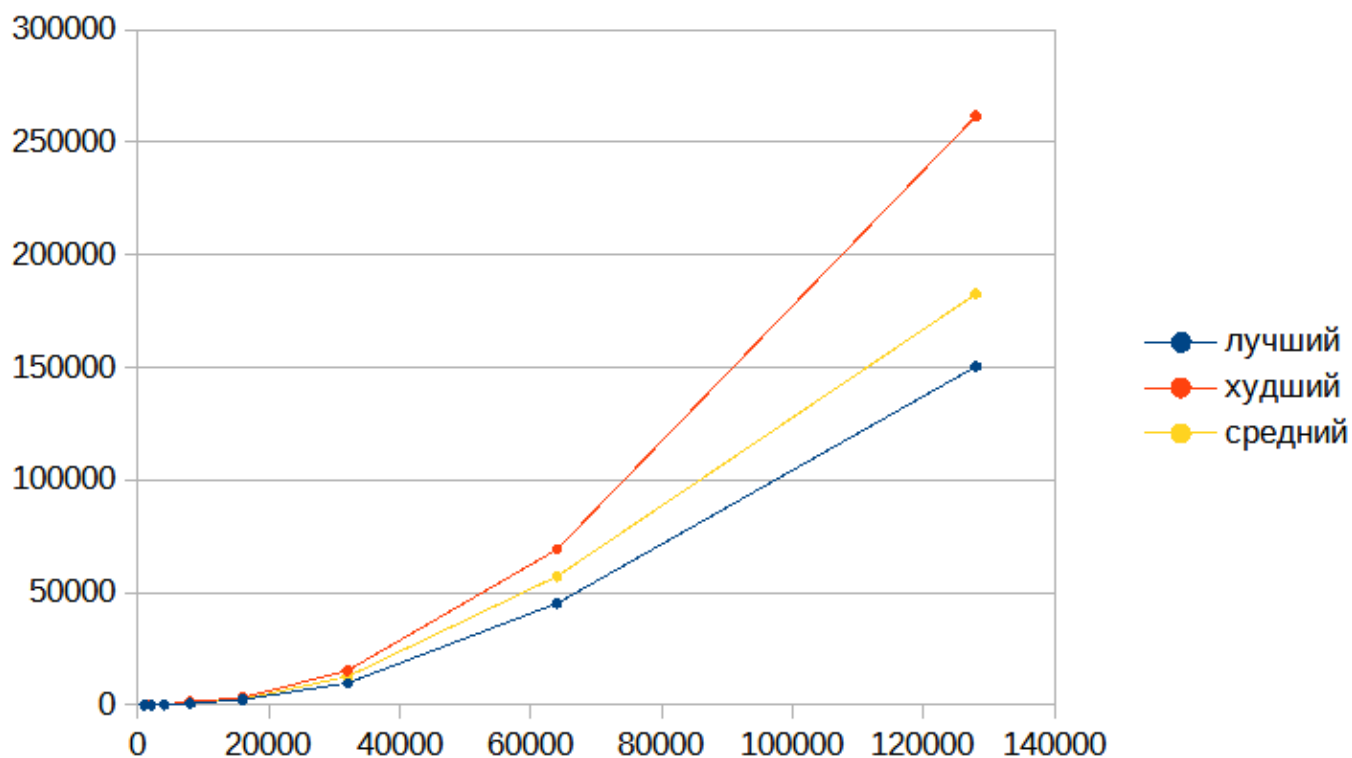
Нам необходимо провести по 20 тестов на массивы с различной длиной и записать время, затраченное на сортировку. Поэтому перед запуском нам надо запустить таймер и остановить его в конце. Чтобы сделать тесты на массивах с разной длиной, мы создаем массив N с размерностями и

выделять динамически память под массив чисел.

Проведя все необходимые тесты, получился график со всеми значениями времени для каждой N.



Для того, чтобы получить лучший и худший случаи, необходимо соединить точки с наименьшим и наибольшим временем соответственно. Также, посчитав среднее значение времени для каждого N, можно получить средний случай.

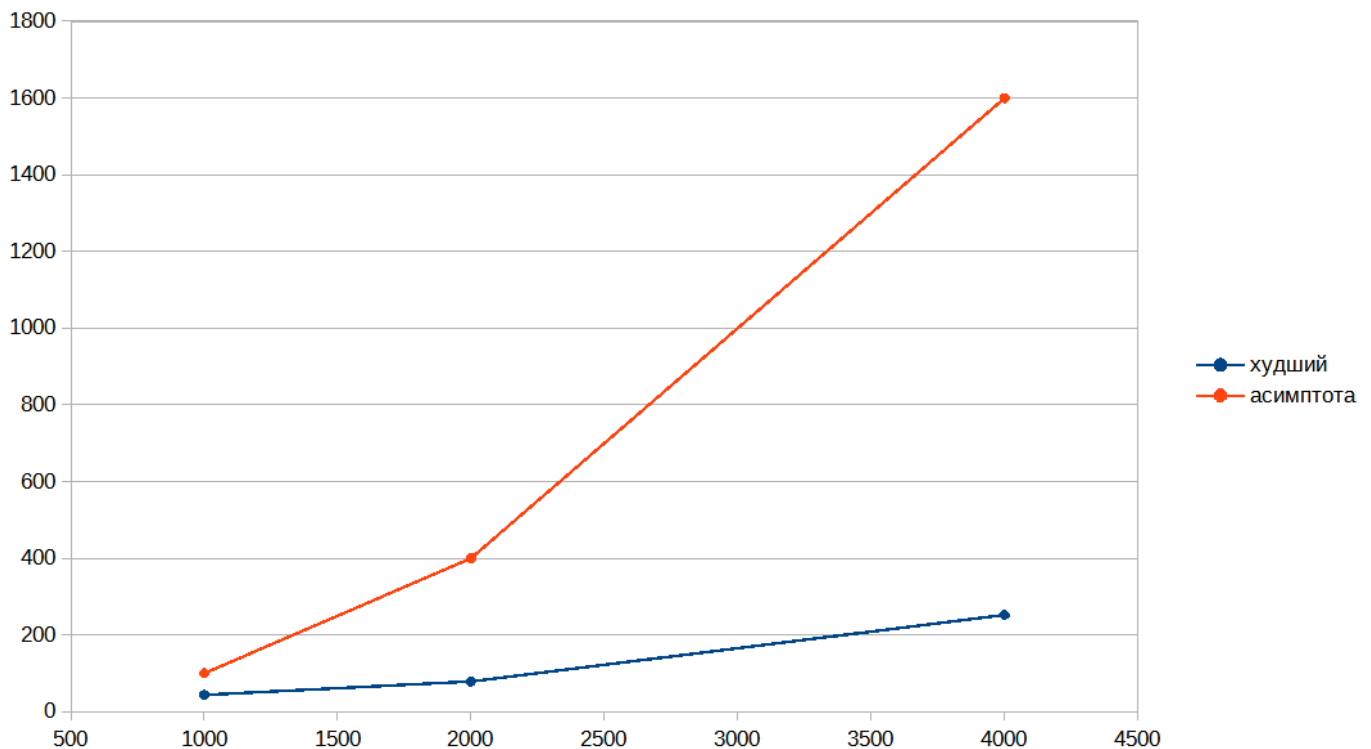


Таким образом, мы получили график времени, который показывает за какое минимальное, максимальное и среднее время будет отработан алгоритм.

Также были построены график худшего случая, и график $O(c * g(N))$, где $g(N)$ соответствует асимптотической сложности рассматриваемого метода сортировки. Значение C было подобрано так,

что начиная с $N \sim 1000$ график асимптотической сложности возрастал быстрее, чем полученное худшее время, но при этом был различим на графике.

Таким образом, мы получили асимптотическую функцию $0,0001 * N^2$



Далее представлен код на языке C++:

```
#include <iostream>
#include <random>
#include <chrono>
#include <fstream>
```

```
const int M = 20; // Количество тестов
```

```
using namespace std;
```

```
/* Очистка файла перед записью */
```

```
void clearFile() {
    ofstream out;
    out.open("C:\\Users\\evgen\\Desktop\\Алгоритмы\\data1.txt");
}
```

```
/* Сортировка перемешиванием
```

```
*
```

```
* @param begin начало массива
```

```

* @param end конец массива
* @param swapped проверка необходимости перестановки
* @param v массив
* @param N количество элементов в массиве
*/

```

```

void shakerSort(double* v, int N) {
    int begin = -1; // сдвиг на 1 влево
    int end = N - 1; // сдвиг на 1 влево
    bool swapped = 1;
    while (swapped) {
        swapped = false;
        begin++;
        for (int i = begin; i < end; i++) { // сортировка с начала
            if (v[i] > v[i + 1]) { // сравнение двух элементов
                swap(v[i], v[i + 1]); // перестановка
                swapped = true;
            }
        }
        if (!swapped) break; // отсортирован ли массив
        swapped = false;
        end--;
        for (int i = end - 1; i >= begin; i--) { // сортировка с конца
            if (v[i] > v[i + 1]) { // сравнение двух элементов
                swap(v[i], v[i + 1]); // перестановка
                swapped = true;
            }
        }
    }
}

```

```

/* Запись результата в файл */

```

```

void writer(int i, int N, double msec) {
    ofstream out;
    out.open("C:\\Users\\evgen\\Desktop\\Алгоритмы\\data1.txt", ios::app);
    if (out.is_open())
    {
        out << "Test №" << i << " N = " << N << " Time: " << msec << endl;
    }
}

```

```
}  
}
```

```
/* Отделение части записи */
```

```
void piece() {  
    ofstream out;  
    out.open("C:\\Users\\evgen\\Desktop\\Алгоритмы\\data1.txt", ios::app);  
    if (out.is_open())  
    {  
        out << "-----" << endl;  
    }  
}
```

```
/* Таймер
```

```
*
```

```
* @param i номер теста
```

```
* @param v массив
```

```
* @param N количество элементов в массиве
```

```
*/
```

```
void timer(double* v, int N, int i) {  
    chrono::high_resolution_clock::time_point start = chrono::high_resolution_clock::now(); // стартовое  
время сортировки  
    shakerSort(v, N); // сортировка  
    chrono::high_resolution_clock::time_point end = chrono::high_resolution_clock::now(); // конечное  
время сортировки  
    chrono::duration<double, nano> nano_diff = end - start; // время в наносекундах  
    chrono::duration<double, micro> micro_diff = end - start; // время в микросекундах  
    chrono::duration<double, milli> milli_diff = end - start; // время в миллисекундах  
    chrono::duration<double> sec_diff = end - start; // время в секундах  
    writer((i + 1), N, milli_diff.count()); // запись в файл  
}
```

```
int main()
```

```
{
```

```
    int N[] = { 1000 }; // массив различных длин массива
```

```

double* v; // массив
clearFile(); // очистка файла
for (int j = 0; j < size(N); j++) {
    v = new double[N[j]]; // выделение памяти
    for (int i = 0; i < M; i++) {
        // генерация случайных чисел от -1 до 1
        mt19937 engine(time(0));
        uniform_real_distribution<double> gen(-1.0, 1.0);
        for (int el = 0; el < N[j]; el++) {
            v[el] = gen(engine);
        }
        timer(v, N[j], i); // таймер
    }
    piece();
    delete[] v; // очистка памяти
}
cout << "Programm end!";
}

```

Заключение.

В заключении можно заметить, что метод перемешивания является измененной версией метода “пузырька”. Однако он работает несколько быстрее и тратит в 2 раза меньше итераций, так как алгоритм идет в обе стороны массива. Как и метод “пузырька”, данный метод достаточно прост в реализации и не требует выделения дополнительной памяти. Однако шейкерный метод все же очень медленный и сортировка крупного массива может занять довольно много времени.