ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Разработка алгоритмов сортировки»

Выполнил работу

Лапшин Максим Константинович

Академическая группа J3111

Принято

Ментор, Вершинин Владислав Константинович

Санкт-Петербург

2024

1. Введение

Цель данной работы заключается в разработке алгоритмов для сортировки массивов различной длинны.

Задачи – изучить алгоритмы сортировки массивов, реализовать алгоритмы, а также измерить время работы алгоритмов для различного размера данных, написать отчет

2. Теоретическая часть

Для реализации данной лабораторной работы потребовалось:

- Знание массивов
- Умение работы с рекурсивными алгоритмами для решения комбинаторных задач
- Знание библиотеки измерения времени

3. Реализация

- Ввод значений с клавиатуры
- Реализация сортировки, используя один из трех алгоритмов: сортировка шевелением / сортировка Шейла/ Bucket sort
- Вывод результата

алгоритмы

```
3oid cocktailsort(std::vector<float>& vec)//сортировка шевелением
{
  bool swapped = true;
  int start = 0;
  int end = vec.size() - 1;
  while (swapped)
  {
    swapped = false;
    for (int i = start; i < end; ++i)// Проход слева направо
    {
      if (vec[i] > vec[i + 1])
      {
```

```
            std::swap(vec[i], vec[i + 1]);
            swapped = true;
        }
    }
    if (!swapped)// Если не было обменов, массив отсортирован
    {
        break;
    }
    end--;
    swapped = false;
    for (int i = end; i > start; --i)// Проход справа налево
    {
        if (vec[i] < vec[i - 1])
        {
            std::swap(vec[i], vec[i - 1]);
            swapped = true;
        }
    }
    start++;
  }
}
```

3.2 `void shellsort(std::vector<float>& vec)//o(n^2) prost o(n)`

```
{
  int n = vec.size();
  for (int gap = n / 2; gap > 0; gap /= 2) {
    for (int i = gap; i < n; i++) {
      int temp = vec[i];
      int j;
      for (j = i; j >= gap && vec[j - gap] > temp; j -= gap) {
        vec[j] = vec[j - gap];
      }
      vec[j] = temp;
    }
  }
```

```
}
```

3.3
```
void bucketSort(std::vector<float>& arr) {
  int n = arr.size();
  if (n <= 0) return;
  std::vector<std::vector<float>> buckets(n);
  for (float num : arr) {
    int bucketIndex = static_cast<int>(n * num);
    if (bucketIndex >= n) {
      bucketIndex = n - 1;
    }
    buckets[bucketIndex].push_back(num);
  }
  arr.clear();
  for (int i = 0; i < n; i++) {
    shellsort(buckets[i]);
    for (float num : buckets[i]) {
      arr.push_back(num);
    }
  }
}
```

4. Экспериментальная часть

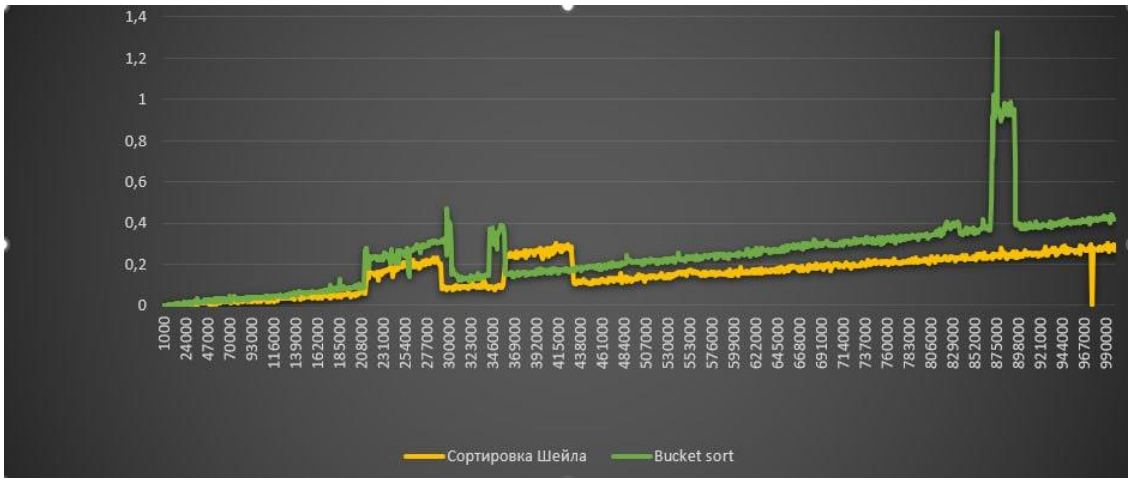
Согласно требованиям моего варианта, на вход к каждому алгоритму подаётся до 1000000 элементов. Теоретически заданная сложность задачи составляет O(N^2) для первого и второго алгоритма и O(N^K), где N - количество элементов начального массива, K – количество разрядов. Для тестирования алгоритма была собрана статистика.

Часть значений приведена в таблице №1.

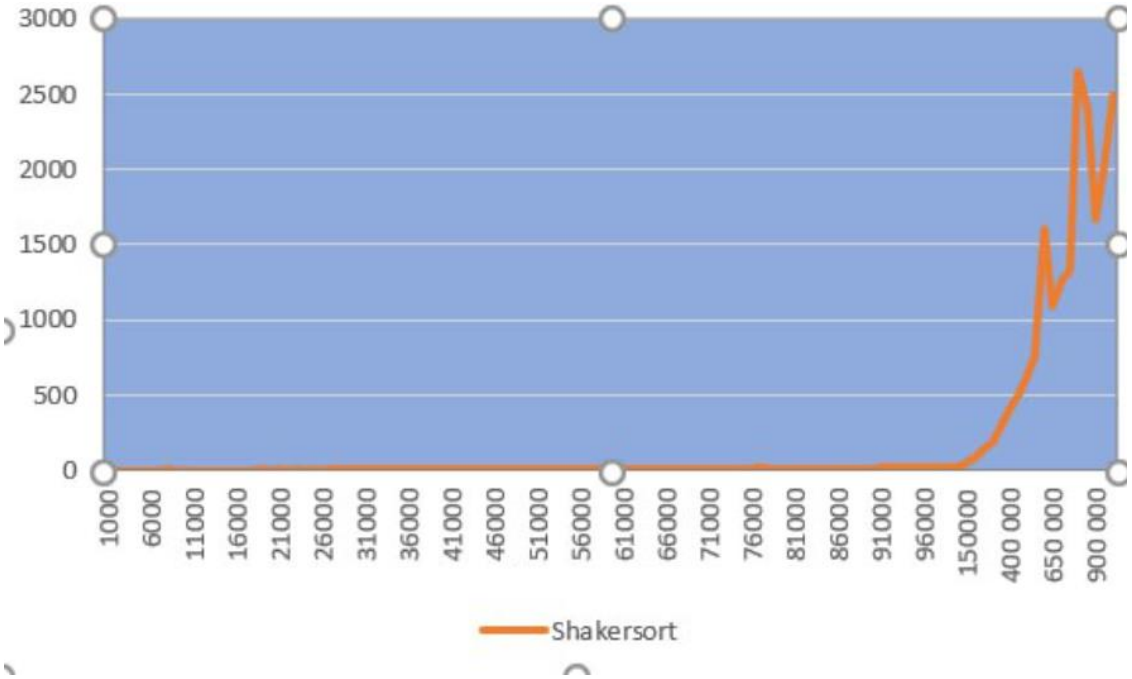
Размер входного набора	1000	5000	10000	100000	500000	1000000
Время выполнения “сортировка шевелением ”, сек	0,000003	0,007	0,057	20	621	2488
Время выполнения “сортировка Шейла”, сек	0,0002	0,001	0,001	0,018	0,152	0,266
Время выполнения “Bucket sort”, сек	0,000012	0,00003	0,007	0.04	0,206	0,415

Таблица №1 - Подсчёт сложности реализованного алгоритма

График представляющий визуально удобный формат всех данных представлен на изображении №1, 2.



Изображение 1 - подпись



Изображение 2

Для оценки вариабельности значений было выполнено по 50 тестов для трех алгоритмов. Тесты были разделены на две категории: одна группа с вводом 100000 элементов, другая – с вводом 10000 элементов. Результаты тестирования визуализированы с использованием двух boxplot.

Boxplot с вводом 10000 элементов

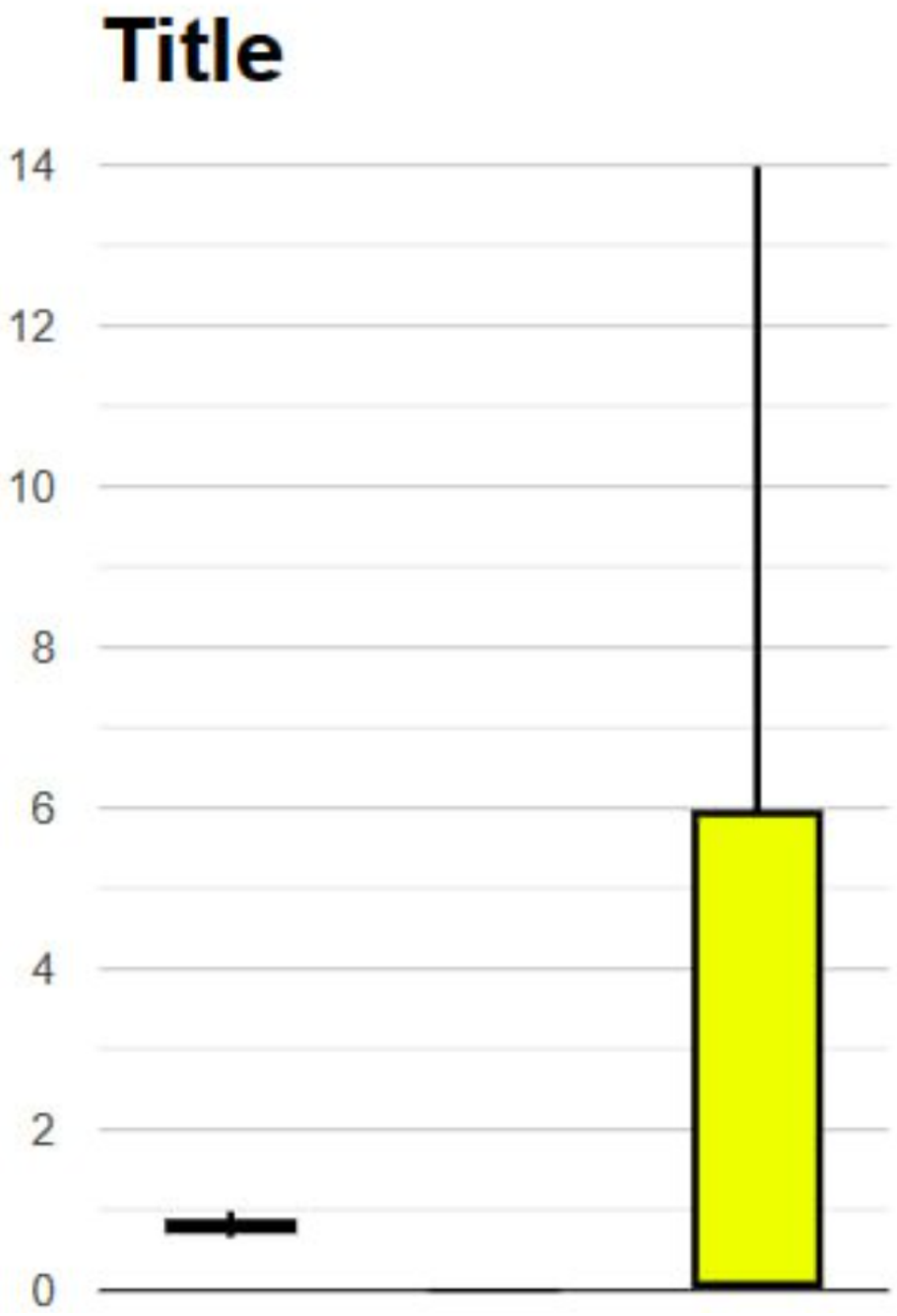


график зависемости времени от входных данных, здесь изображены сортировка Шейла/ Buckect sort\сортировка шевелением соотвественно

Boxplot с вводом 100000 элементов

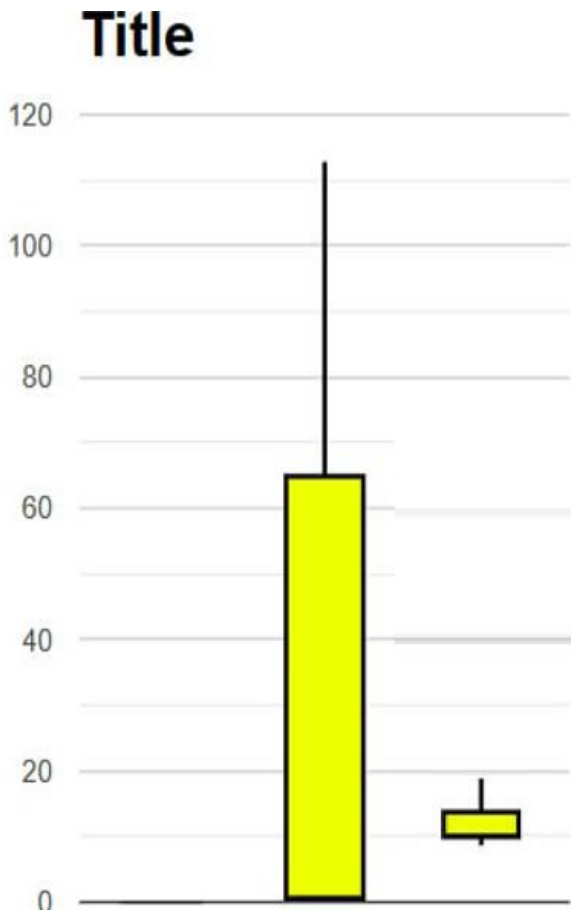


график зависемости времени от входных данных, здесь изображены Buckect sort\ сортировка шевелением / соотвественно сортировка Шейла

Заключение

В ходе выполнения работы мною было реализовано три алгоритма: “сортировка шеила”, "сортировка шевелением”, “bucket sort”.

Алгоритм и тестирующая система были реализованы на языке C++.

В качестве дальнейших исследований можно предложить исследование зависимости скорости выполнения алгоритма от внешних задач

Приложение А

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cassert>

void cocktailsort(std::vector<float>& vec)//сортировка шевелением
{
  bool swapped = true;
  int start = 0;
  int end = vec.size() - 1;
  while (swapped)
  {
    swapped = false;
    for (int i = start; i < end; ++i)// Проход слева направо
    {
      if (vec[i] > vec[i + 1])
      {
        std::swap(vec[i], vec[i + 1]);
        swapped = true;
      }
    }
    if (!swapped)// Если не было обменов, массив отсортирован
    {
      break;
    }
    end--;
    swapped = false;
    for (int i = end; i > start; --i)// Проход справа налево
    {
      if (vec[i] < vec[i - 1])
```

```
        {
            std::swap(vec[i], vec[i - 1]);
            swapped = true;
        }
    }
    start++;
  }
}

void shellsort(std::vector<float>& vec)//o(n^2) prost o(n)
{
  int n = vec.size();
  for (int gap = n / 2; gap > 0; gap /= 2) {
    for (int i = gap; i < n; i++) {
      int temp = vec[i];
      int j;
      for (j = i; j >= gap && vec[j - gap] > temp; j -= gap) {
        vec[j] = vec[j - gap];
      }
      vec[j] = temp;
    }
  }
}

void bucketSort(std::vector<float>& arr) {
  int n = arr.size();
  if (n <= 0) return;
  std::vector<std::vector<float>> buckets(n);
  for (float num : arr) {
    int bucketIndex = static_cast<int>(n * num);
    if (bucketIndex >= n) {
```

```
            bucketIndex = n - 1;
        }
        buckets[bucketIndex].push_back(num);
    }
    arr.clear();
    for (int i = 0; i < n; i++) {
        shellsort(buckets[i]);
        for (float num : buckets[i]) {
            arr.push_back(num);
        }
    }
}


int main()
{

    std::vector<float> bedtest = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    std::vector<float> midtest = { 3, 6, 1, 8, 4, 2, 5, 10, 9, 7 };
    std::vector<float> goodtest = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    std::vector<float> anstest = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    cocktailsort(bedtest);
    cocktailsort(midtest);
    cocktailsort(goodtest);
    assert(bedtest == anstest);
    assert(goodtest == anstest);
    assert(midtest == anstest);
    bedtest = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    midtest = { 3, 6, 1, 8, 4, 2, 5, 10, 9, 7 };
    goodtest = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    anstest = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
shellsort(bedtest);
shellsort(midtest);
shellsort(goodtest);
assert(bedtest == anstest);
assert(goodtest == anstest);
assert(midtest == anstest);
bedtest = { 5, 5, 5, 5, 5, 5, 5, 5, 5, 5};
midtest = { 0.1, 0.4, 0.3, 0.8, 0.9, 0.7, 0.2, 0.6};
goodtest = { 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0};
bucketSort(bedtest);
bucketSort(midtest);
bucketSort(goodtest);
std::vector<float> ans = { 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 };
assert(bedtest == ans);
ans = { 0.1, 0.2, 0.3, 0.4, 0.6, 0.7, 0.8, 0.9};
assert(midtest == ans);
ans = { 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 };
assert(goodtest == ans);
std::vector<float> vec;
int sizen;
std::cin >> sizen;
float g;
for (int i = 0; i < sizen; i++) {
   std::cin >> g;
   vec.push_back(g);
}
bucketSort(vec);
for (int i = 0; i < sizen; i++) {
   std::cout << vec[i] << " ";
}
return 0;
```

```
}
```