Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе №3

Название: Алгоритмы сортировки массивов

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б _____ Хетагуров П.К

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподователь _____ Л.Л. Волкова

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

Введение

В данной лабораторной работе будут рассмотренны и проанализированы такие алгоритмы сортировки как:

1. сортировка пузырьком;
2. сортировка вставками;
3. сортировка выбором.

1 Аналитическая часть

В данном разделе будут поставлены цели и задачи работы, будут рассмотренны основные теоритические сведения связанные с алгоритмами сортировки.

1.1 Цель и задачи работы

Цель работы:

Реализовать и сравнить по трудоемкости алгоритмы сортировки.

Задачи работы:

1. дать описание реализуемых алгоритмов сортировок;
2. реализовать описанные алгоритмы;
3. провести эксперименты по замеру времени работы разработанных алгоритмов;
4. провести сравнения алгоритмов по затраченному времени;
5. дать оценку трудоемкости алгоритмов.

1.2 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз(где N - длина массива) или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.[1]

1.3 Сортировка вставками

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.[1]

1.4 Сортировка выбором

При каждой итерации максимальный элемент из не отсортированной части массива помещается в конец.[1]

1.5 Вывод

В данной части были поставлены задачи и цель работы, рассмотренны описания алгоритмов сортировки.

2 Конструкторская часть

В данном разделе будут рассмотренны схемы алгоритмов, требования к функциональности ПО и проведена оценка трудоемкости алгоритмов.

2.1 Требования к ПО

ПО должно иметь два режима работы, выбираемые из меню

1. Режим демонстрации. В этом режиме должен осуществляться ввод массива и демонстрация работы на нем всех реализованных алгоритмов.

2. Режим тестирования. В этом режиме должны проводится замеры времени выполнения реализованных алгоритмов. Должен осуществляться вывод затраченного процессорного времени на случайным образом сгенерированных данных.

2.2 Схемы алгоритмов

На рисунке [1] изображена схема алгоритма сортировки пузырьком.

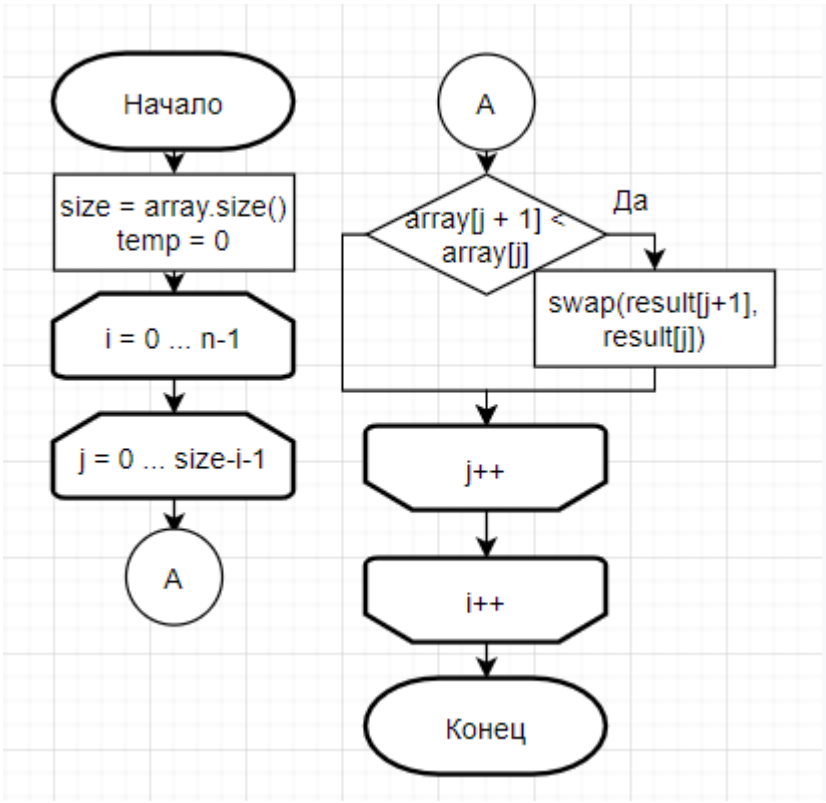

Рисунок 1 – Схема алгоритма сортировки пузырьком

На рисунке [2] изображена схема алгоритма сортировки вставками.

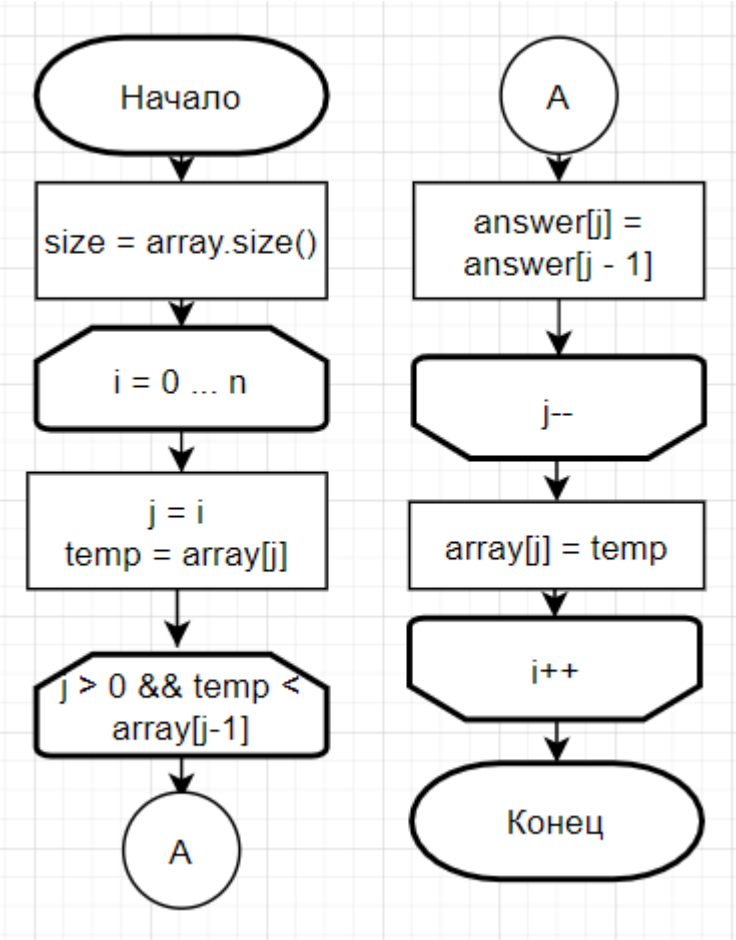

Рисунок 2 – Схема алгоритма сортировки вставками

На рисунке [3] изображена схема алгоритма сортировки выбором.

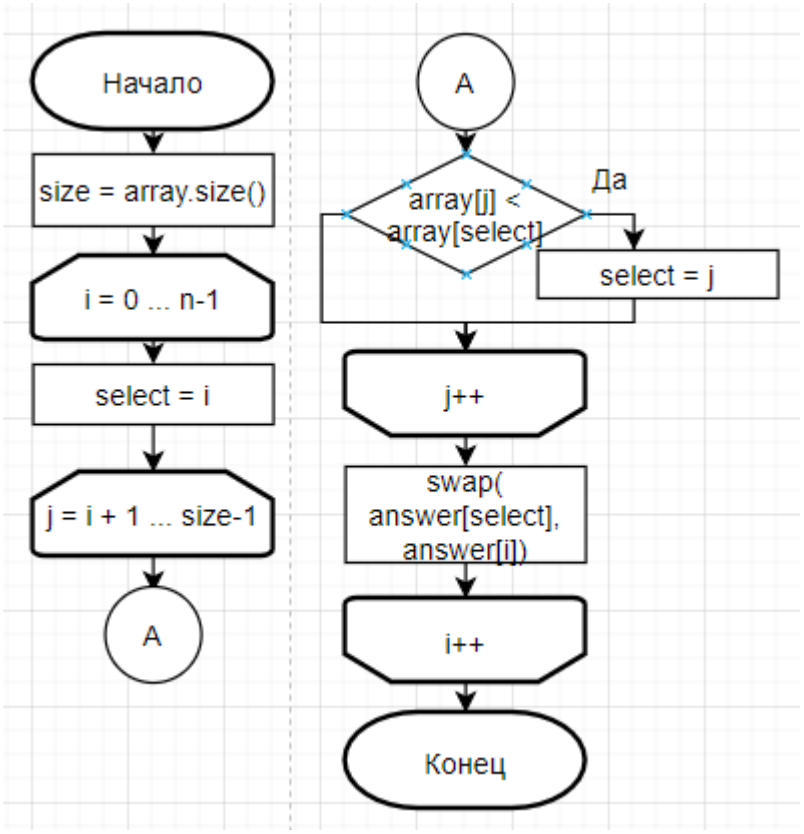

Рисунок 3 – Схема алгоритма сортировки выбором

2.3 Оценка трудоемкости

Модель трудоемкости для оценки алгоритмов:

Стоимость базовых операций единица:

$=, +, *, <, >, <=, >=, ==, !=, [], +=, -=, *=, /=, ++, --$

2.3.1 Трудоемкость swap

$f_{swap} = 3 + 4 = 7$

2.3.2 Трудоемкость сортировки пузырьком

Так как внутренний цикл будет выполняться: $n-0-1, n-1-1, n-2-1, ..., n-(n-1)$ раз, то количество выполненных раз можно записать как [1]:

$$int_{bubble} = \frac{(n-1)n}{2} \tag{1}$$

Тогда, если массив отсортирован в обратном порядке, происходит худший случай сортировки пузырьком [2]:

$$f_{bubble} = (3 + 1 + (2 + 1 + 2 + 1 + f_{swap}))int_{bubble}) + (2 + 1 + 1)(n-1) + 1 + 1 = 17int_{bubble} + 4(n-1) + 2 = \frac{17(n-1)n}{2} + 4(n-1) + 2 \approx \frac{17n^2}{2} \tag{2}$$

А если массив отсортирован, происходит лучший случай [3]:

$$f_{bubble} = (3+1+(2+1+1))int_{bubble})+(2+1+1)(n-1)+1+1 = 8int_{bubble}+4(n-1)+2 == \frac{8(n-1)n}{2}+4(n-1)+2 \approx 4n^2 \tag{3}$$

2.3.3 Трудоемкость сортировки выбором

Так как внутренний цикл будет выполняться: $n-1, n-2, n-3, ..., n-(n-1)$ раз, то количество выполненных раз можно записать как [4]:

$$int_{choose} = \frac{(n-1)n}{2} \tag{4}$$

Тогда, если массив отсортирован в обратном порядке, происходит худший случай [5]:

$$f_{choose} = ((2+4) * int_{choose}) + (3 + 1 + f_{swap}) * (n-1) + 2 = \frac{6 * (n-1)n}{2} + 11 * (n-1) + 2 \approx 3n^2 \tag{5}$$

А если массив отсортирован, происходит лучший случай [6]:

$$f_{choose} = (2+3) * int_{choose}) + (3 + 1 + f_{swap}) * (n-1) + 2 = \frac{5 * (n-1)n}{2} + 11 * (n-1) + 2 \approx \frac{5n^2}{2} \tag{6}$$

2.3.4 Трудоемкость сортировки вставками

Так как внутренний цикл будет выполняться: 1, 2, 3, ..., n-1 раз, то количество выполненных раз можно записать как [7]:

$$int_{insert} = \frac{(n-1)n}{2} \tag{7}$$

Тогда, если массив отсортирован в обратном порядке, происходит худший случай [8]:

$$f_{insert} = ((5+5) * int_{insert}) + (2+5) * (n-1) + 2 = \frac{10 * (n-1)n}{2} + 7 * (n-1) + 2 \approx 5n^2 \tag{8}$$

А если массив отсортирован, происходит лучший случай [9]:

$$f_{insert} = (2+5+4) * (n-1) + 2 = 11 * (n-1) + 2 \approx 11n \tag{9}$$

2.4 Вывод

В данном разделе были рассмотрены схемы и была рассчитана трудоемкость алгоритмов и обозначены требования к ПО.

3 Технологическая часть

Ниже будут представлены средствы реализации и листинги реализованной программы.

3.1 Средства реализации

Выбранный язык программирования - C++, так как требований по конкретнему языку не выдвигалось и он был изучен во время обучения. Среда разработки - Visual Studio Code.[2]

Функции вычисления процессорного времени - QueryPerfomanceCounter из библиотеки WinAPI.[3]

3.2 Реализации алгоритмов

Ниже представлены листинги реализаций алгоритмов. На листинге [1] представлен алгоритм сортировки пузырьком.

Листинг 1 – Алгоритм сортировки пузырьком

```
1  vector<int> sortBubble(vector<int> array)
2  {
3      int size = array.size();
4      int temp = 0;
5      vector<int> result = array;
6
7      for (int i = 0; i + 1 < size; i++)
8      {
9          for (int j = 0; j + 1 < size - i; j++)
10         {
11             if (result[j + 1] < result[j])
12             {
13                 temp = result[j];
14                 result[j] = result[j + 1];
15                 result[j + 1] = temp;
16             }
17         }
18     }
19
20     return result;
21 }
```

На листинге [2] Представлен алгоритм сортировки вставками.

Листинг 2 – Алгоритм сортировки вставками

```
1  vector<int> sortInsertion(vector<int> array)
2  {
```

```
    int size = array.size();
    int temp = 0;
    vector<int> result = array;

    for (int i = 1; i < size; i++)
    {
        int j = i;
        temp = result[i];
        while (j > 0 && temp < result[j - 1])
        {
            result[j] = result[j - 1];
            j--;
        }
        result[j] = temp;
    }

    return result;
}
```

На листинге [3] Представлен алгоритм сортировки выбором.

Листинг 3 – Алгоритм сортировки выбором

```
  vector<int> sortSelect(vector<int> array)
{
    int size = array.size();
    int temp = 0;
    vector<int> result = array;

    for (int i = 0; i < size - 1; i++)
    {
        int select = i;
        for (int j = i + 1; j < size; j++)
        {
            if (result[j] < result[select])
            {
                select = j;
            }
        }
        temp = result[select];
        result[select] = result[i];
        result[i] = temp;
    }
```

```
21
22      return result;
23  }
```

3.3 Вывод

В данном разделе были описаны средства реализации, были представлены листинги реализации сотрировок пузырьком, вставками и выбором.

4 Экспериментальная часть

В данной главе будут представлен пример работы программы, результат экспериментов по замеру времени и произведен сравнительный анализ алгоритмов по затрачиваемому времени.

4.1 Пример работы программы

Пример работы программы представлен на рисунке [4]

```
MENU:
0) Exit
1) Demonstration
2) Tests result
1
Input array size: 10
Input array with size: 10
-10 24 1 2 4 1 42 5 0 1234

Result select:
   -10     0     1     1     2     4     5    24    42  1234

Result bubble:
   -10     0     1     1     2     4     5    24    42  1234

Result insertion:
   -10     0     1     1     2     4     5    24    42  1234
```

Рисунок 4 – Пример работы программы

4.2 Сравнительный анализ алгоритмов по времени

Эксперименты проводятся на массивах размером от 2 до 252 с шагом 50 (результаты на рисунке [5]

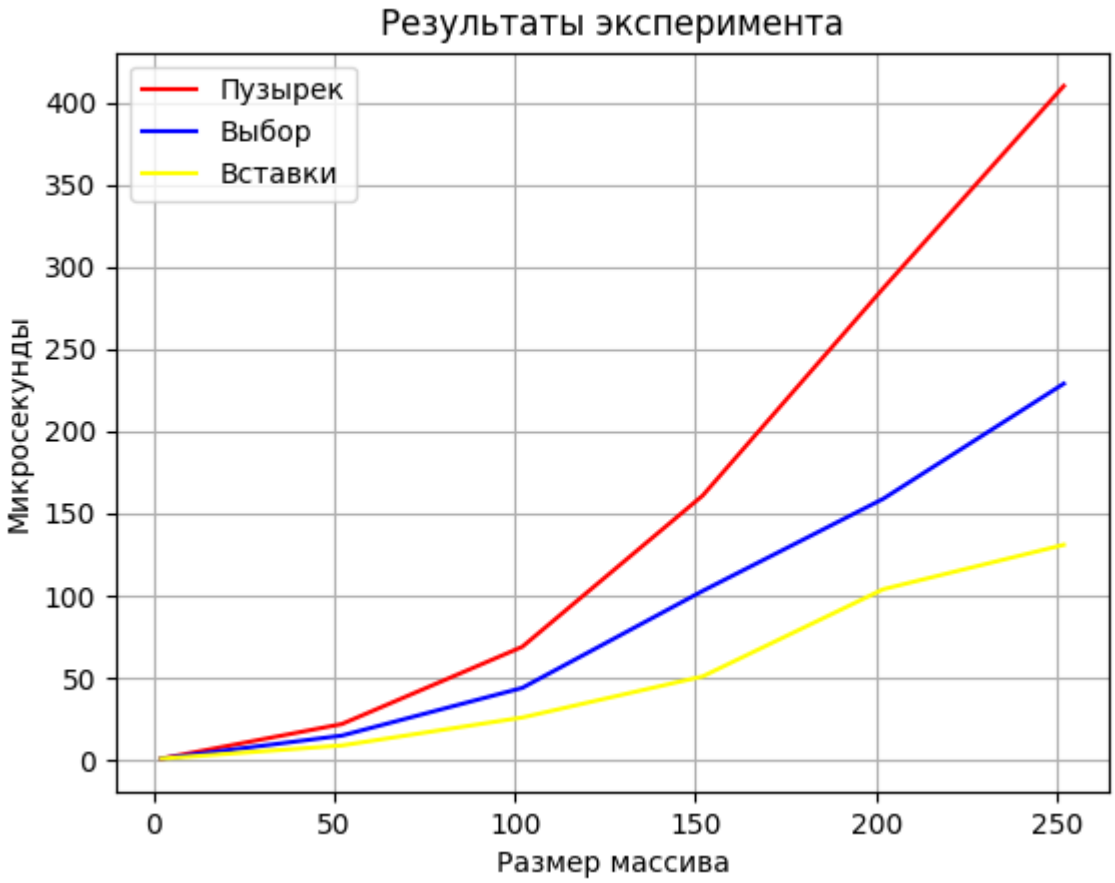

Рисунок 5 – Результаты на массивах размером от 2 до 252

4.3 Вывод

Как видно из графиков, самым долгим алгоритмом является алгоритм сортировки пузырьком. Самым быстрым является алгоритм сортировки вставками. Алгоритм сортировки выбором быстрее сортировки пузырьком, но уступает в скорости работы алгоритму сортировки вставками.

Заключение

В данной лабораторной работе были описаны алгоритмы сортировки пузырьком, вставками и выбором, они были реализованы, а также были проведены эксперименты по замеру времени работы реализованных алгоритмов и проведены сравнения алгоритмов по результатам эксперимента. Также была дана оценка трудоемкости.

Список литературы

[1] Базовые сортировки [Электронный ресурс]. Режим доступа: (дата обращения - 02.10.2020) Свободный. URL:https://tproger.ru/translations/sorting-for-beginners/

[2] Visual Studio Code [Электронный ресурс]. Режим доступа: (дата обращения - 02.10.2020) Свободный. URL: code.visualstudio.com

[3] WinAPI. Функция QueryPerformanceCounter [Электронный ресурс]. Режим доступа: (дата обращения - 02.10.2020) Свободный. URL: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter