Государственное образовательное учреждение высшего
профессионального образования
«Московский государственный технический университет имени Н. Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №4

на тему:

Алгоритмы сортировок

Студент ИУ7-54: Морозов И. А
Преподаватель: Погорелов. Д. А.

Москва 2018

Введение

Целью данной лабораторной работы является изучение и сравнение алгоритмов сортировки, а именно: алгоритм сортировки простыми вставками, алгоритм сортировки расческой, алгоритм сортировки с помощью двоичного дерева.

1. Аналитическая часть

1.1 Описание алгоритмов

1.1.1 Алгоритм сортировки простыми вставками

Описание алгоритма:

Простой алгоритм, часто применяемый на малых объемах. Массив разделен на 2 части: левая - упорядоченная, правая - нет.

На одном шаге:

1) Берется первый элемент правой части,
2) Вставляется на подходящее место в левой части

1.1.2. Алгоритм сортировки «расческой»

Описание алгоритма:

Данных алгоритм является улучшением алгоритма пузырька. Основная идея «расчёски» в том, чтобы первоначально брать достаточно большое расстояние между сравниваемыми элементами и по мере упорядочивания массива сужать это расстояние вплоть до минимального. Таким образом, мы как бы причёсываем массив, постепенно разглаживая на всё более аккуратные пряди. Первоначальный разрыв между сравниваемыми элементами лучше брать с учётом специальной величины, называемой фактором уменьшения, оптимальное значение которой равно примерно 1,247. Сначала расстояние между элементами равно размеру массива, разделённого на фактор уменьшения (результат округляется до ближайшего целого). Затем, пройдя массив с этим шагом, необходимо поделить шаг на фактор уменьшения и пройти по списку вновь. Так продолжается до тех пор, пока разность индексов не достигнет единицы. В этом случае массив «досортировывается» обычным пузырьком.

1.1.3 Алгоритм сортировки с помощью двоичного дерева

Описание алгоритма:

Идея алгоритма заключается в следующем:
1. Строим кучу на исходном массиве
2. N-1 раз достаем максимальный элемент, кладем его на освободившееся место в правой части

2. Конструкторская часть

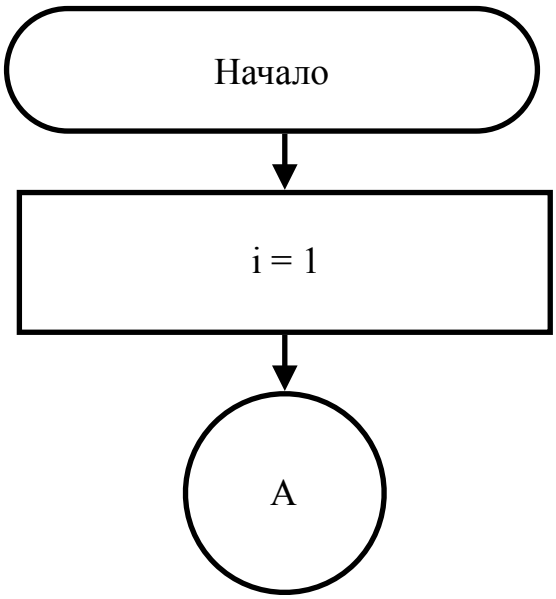
Для исследований будут и реализованы три алгоритма: алгоритм сортировки вставками, алгоритм сортировки «расческой» и алгоритм сортировки с помощью двоичного дерева.

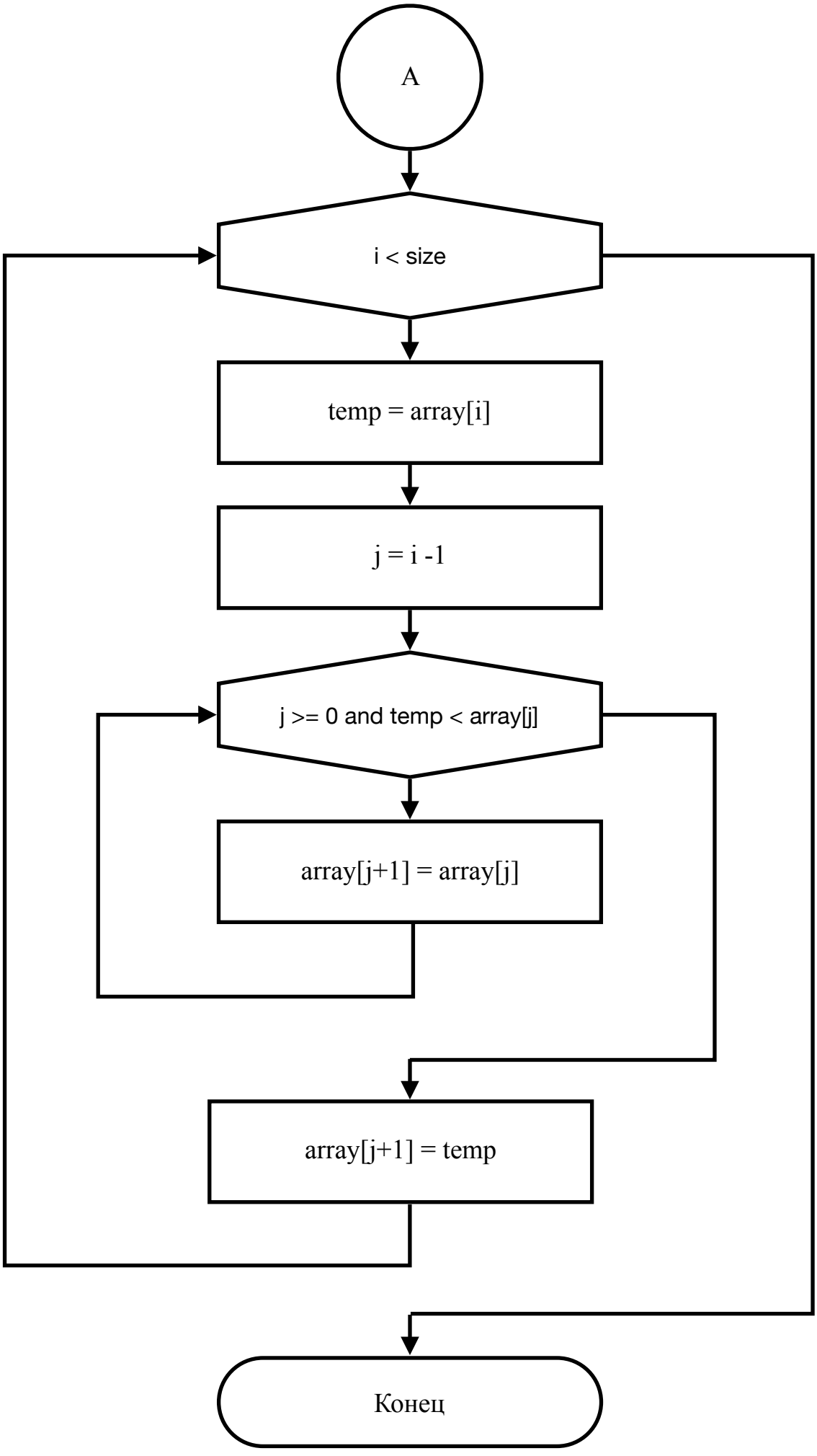
2.1 Разработка алгоритмов

2.1.1 Схема алгоритма сортировки простыми вставками

Входные данные:

array - входной массив
size - размер массива

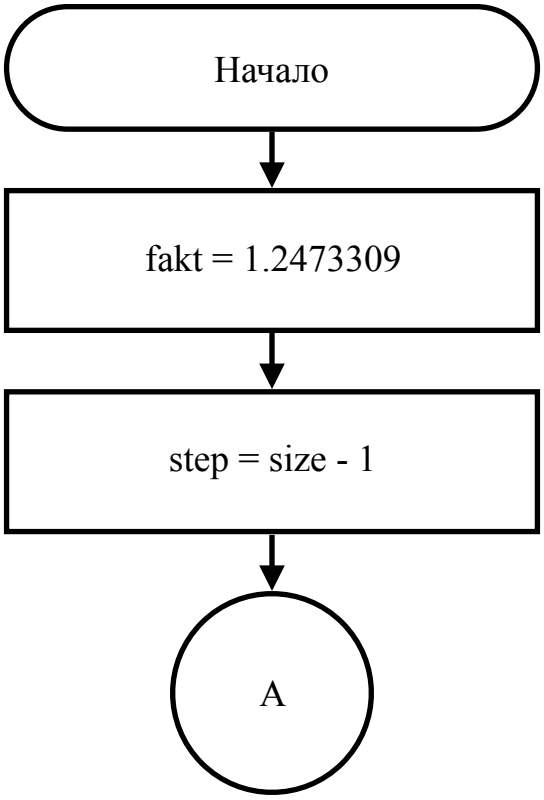


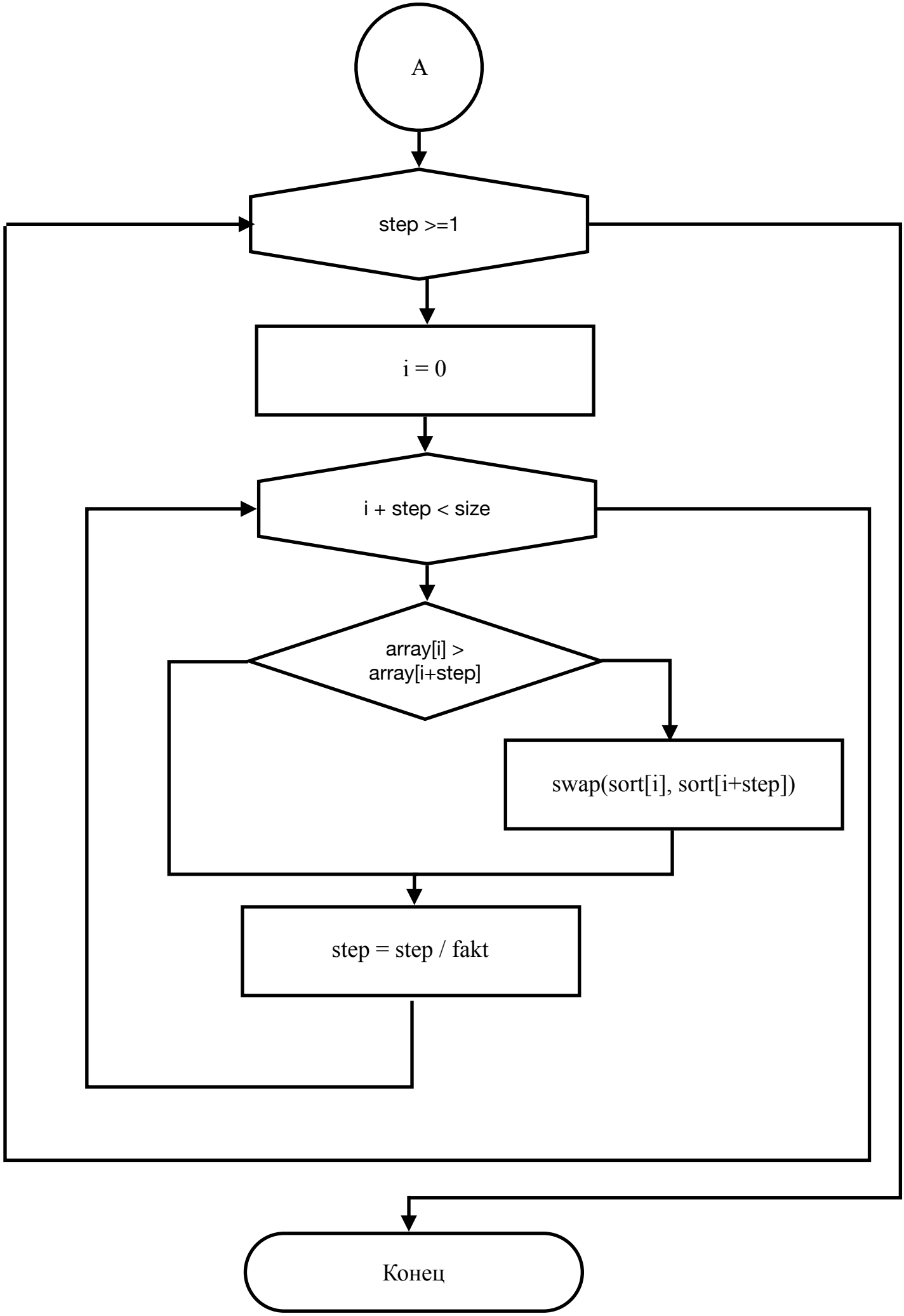

A
i < size
temp = array[i]
j = i -1
j >= 0 and temp < array[j]
array[j+1] = array[j]
array[j+1] = temp
Конец

2.1.2 Схема алгоритма сортировки расческой

Входные данные:

array - входной массив
size - размер массива




A
step >=1
i = 0
i + step < size
array[i] >
array[i+step]
swap(sort[i], sort[i+step])
step = step / fakt
Конец

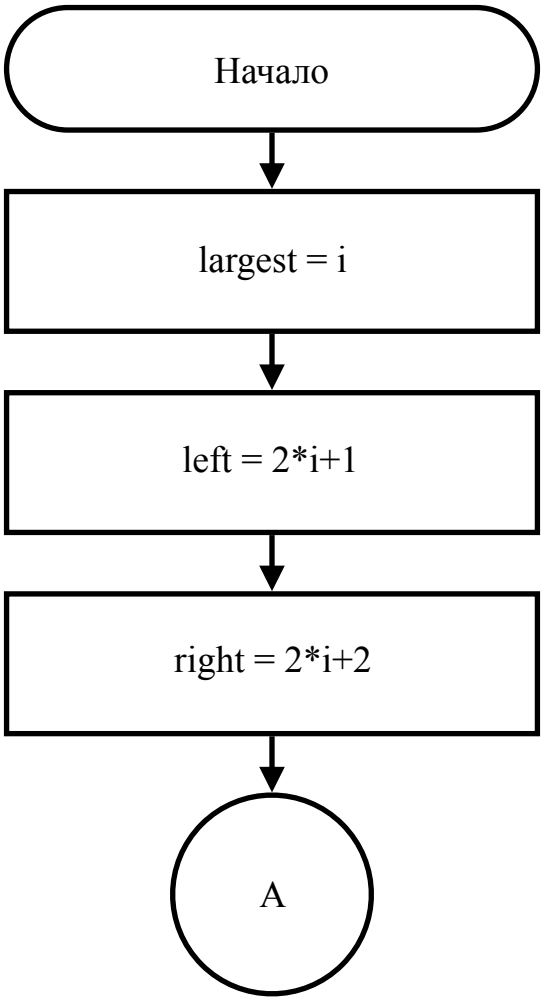
2.1.3 Схема алгоритма сортировки бинарной кучей

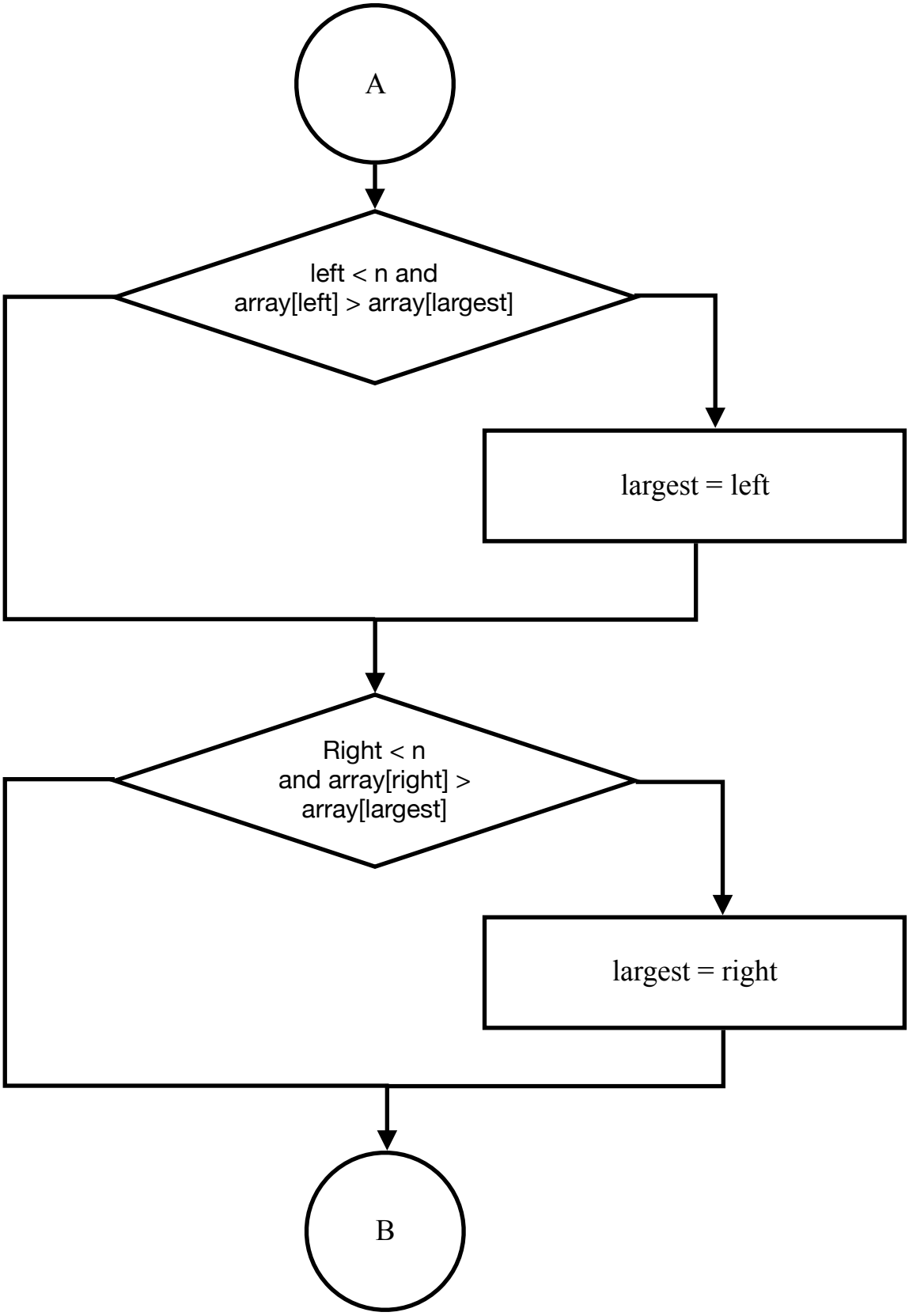
Данный алгоритм состоит из двух функций, одна из которых «просеивает» заданный элемент в куче, чтобы он занял свое место

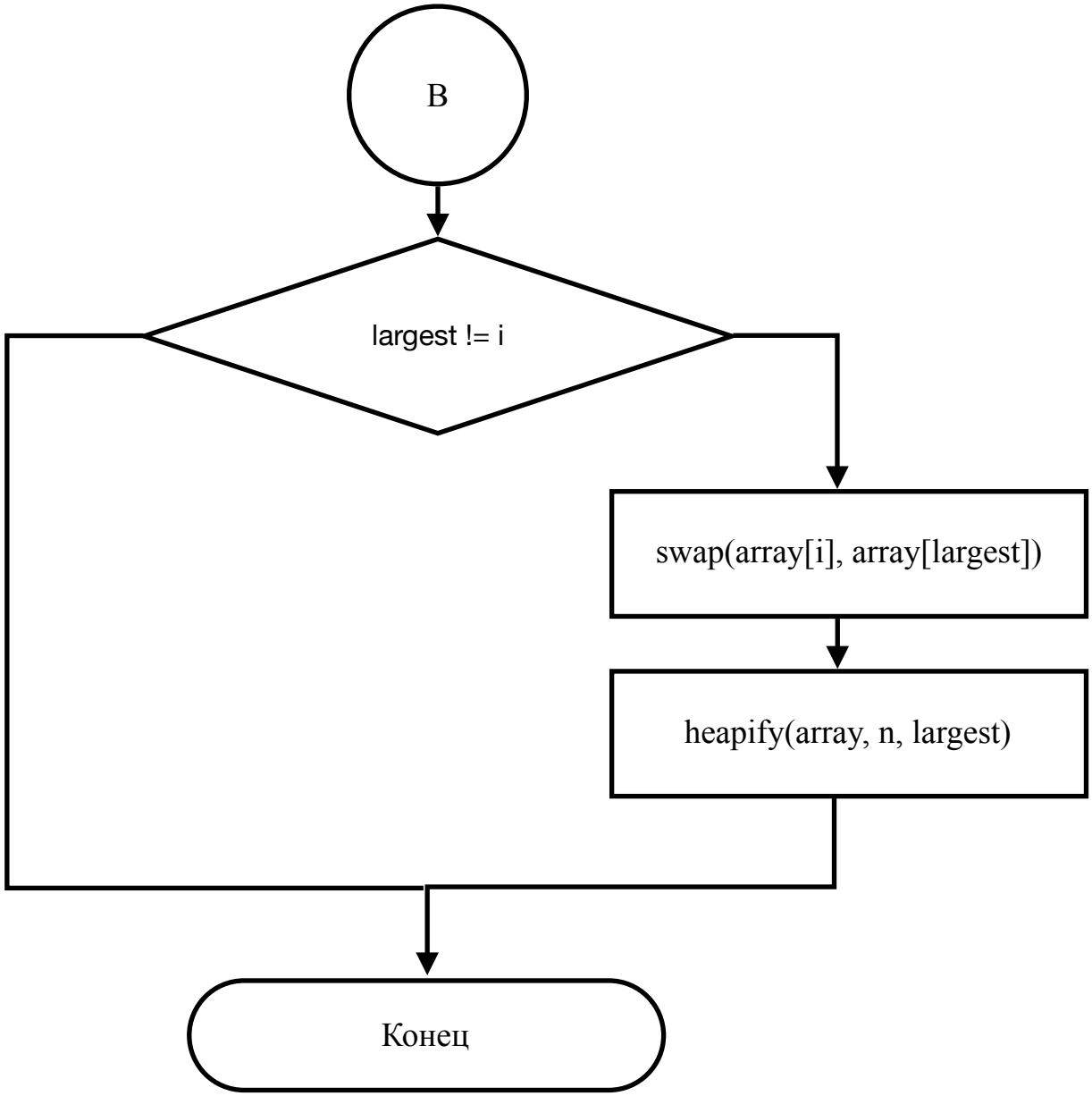
Функция просеивания heapify()

Входные данные:

array - массив
n - «просеиваемый» элемент
i - индекс элемента



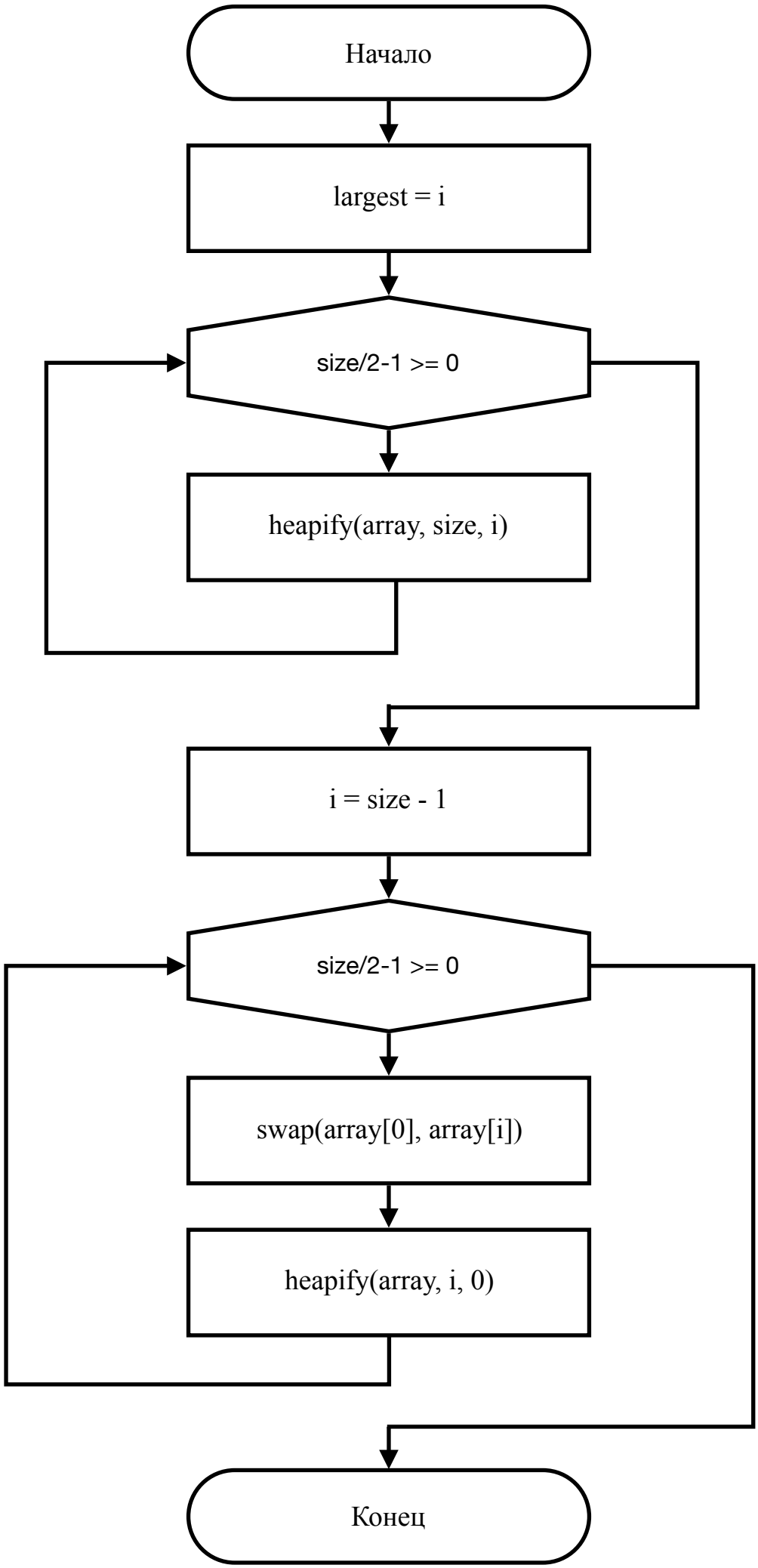

A
left < n and
array[left] > array[largest]
largest = left
Right < n
and array[right] >
array[largest]
largest = right
B



Функция heapSort()

Входные данные:

array - массив
size - размер массива


Начало
largest = i
size/2-1 >= 0
heapify(array, size, i)
i = size - 1
size/2-1 >= 0
swap(array[0], array[i])
heapify(array, i, 0)
Конец

2.2 Анализ сложности алгоритмов

2.2.1 Оценка сложности алгоритма вставками

Худший случай(n^2):

В худшем случае, внутренний цикл будет делать i-1 итераций. Таким образом:

$$2 + N(2 + 2 + 2 + 3 + (i - 1)(4 + 4) + 3) = 20N + 8iN + 2$$

Массив упорядочен по убыванию:
Выполняется $2(n-1) + \frac{n(n-1)}{2}$ копирований и $\frac{n(n-1)}{2}$ сравнений

Лучший случай(n):

В лучшем случае, внутренний цикл будет выходить по второму условию $tmp < a[j]$ и итераций не будет выполняться совсем. Таким образом:

$$2 + N(2 + 2 + 2 + 3 + 3) = 12N + 2$$

Массив упорядочен по возрастанию:
Выполняется $2(n-1)$ копирований и $n-1$ сравнений

2.2.2 Оценка сложности алгоритма сортировки расческой

Худший случай(n^2):

Худший случай будет возникать при обратно упорядоченном массиве. В данной ситуации будут производиться все «swap-ы».

Лучший случай($nlog(n)$):

Лучший случай будет возникать при отсортированном массиве, потому что мы не будем тратить время на «swap» элементов.

Справочная информация об оценке взята из https://en.wikipedia.org/wiki/Comb_sort

2.2.3 Оценка сложности алгоритма сортировки бинарной кучей

Алгоритм сортировки будет состоять из двух основных шагов:
1. Выстраивание элементов массива в виде сортирующего дерева
$Array[i] \geqslant Array[2i + 1]$
$Array[i] \geqslant Array[2i + 2]$
При $0 \leqslant i < \frac{n}{2}$
Этот шаг требует $O(n)$ операций
2. Будем удалять элементы из корня по одному за раз и перестраивать дерево. Процесс продолжается до тех пор пока в сортирующем дереве не останется один элемент. Этот шаг требует $nlog(n)$ операций

Таким образом данный алгоритм работает гарантированно за $nlog(n)$ в любом случае.

Справочная информация взята из https://ru.wikipedia.org/wiki/Пирамидальная_сортировка

3. Технологическая часть

В данном разделе будет представлено описание используемого языка программирования, а также будет показан листинг кода функций, работающих согласно указанным выше алгоритмам.

3.1 Требования к программному обеспечению

Данная программа была реализована на языке C++, компилятор которого поддерживается многими операционными системами

Компилятор: g++

3.2 Средства реализации

Программа была реализована на операционной системе MacOS в среде разработки Xcode

3.3 Листинг кода

3.3.1 Листинг алгоритма сортировки вставками

array - массив
size - размер массива

```
void InsertionSort(double* a, int n) {
    for(int i = 1; i < n; ++i) {
        int temp = a[i];
        int j = i - 1;
        for(; j >= 0 && temp < a[j]; --j) {
            a[j+1] = a[j];
        }

        a[j+1] = temp;
    }
}
```

3.3.2 Листинг алгоритма сортировки расческой

array - массив
size - размер массива

```
void CompSort(double* sort, int size)
{
    double fakt = 1.2473309; // фактор уменьшения
    int step = size - 1;

    while (step >= 1)
    {
        for (int i = 0; i + step < size; ++i)
        {
            if (sort[i] > sort[i + step])
            {
                std::swap(sort[i], sort[i + step]);
            }
        }
```

```
            step /= fakt;
        }
    }
```

3.3.3 Листинг алгоритма сортировки бинарной кучей

array - массив
size - размер массива

```
void heapify(double* arr, int n, int i)
{
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;

    if (l < n && arr[l] > arr[largest])
        largest = l;

    if (r < n && arr[r] > arr[largest])
        largest = r;

    if (largest != i)
    {
        std::swap(arr[i], arr[largest]);

        heapify(arr, n, largest);
    }
}

void heapSort(double* arr, int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i=n-1; i>=0; i--)
    {
        std::swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}
```

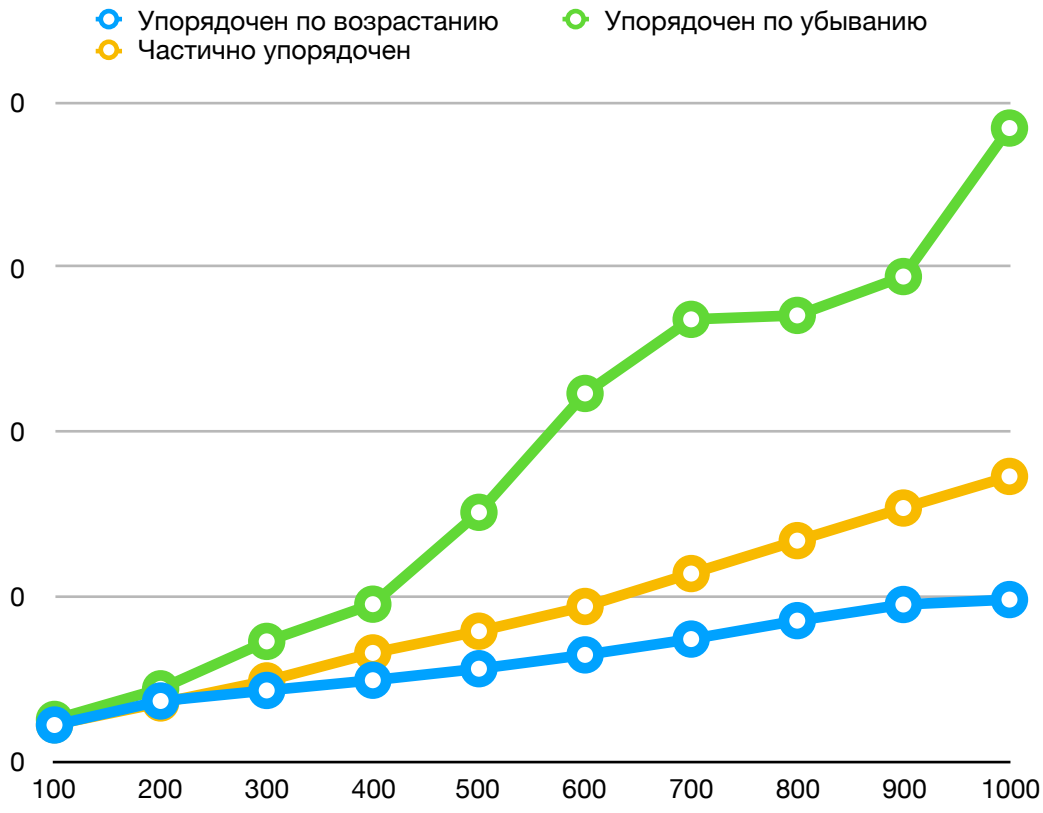
4. Экспериментальная часть

В данном разделе будут произведены сравнения сортировок для различных случаев расположения элементов в массиве (лучший, худший, случайный случаи)

4.1 Постановка эксперимента

4.1.1 Сортировка вставками

Замеры буду производиться для трех типов расположения элементов в массиве : частично упорядочены, упорядочены по возрастанию, упорядочены по убыванию. Размеры массива будут варьироваться от 100 до 1000 с шагом 100

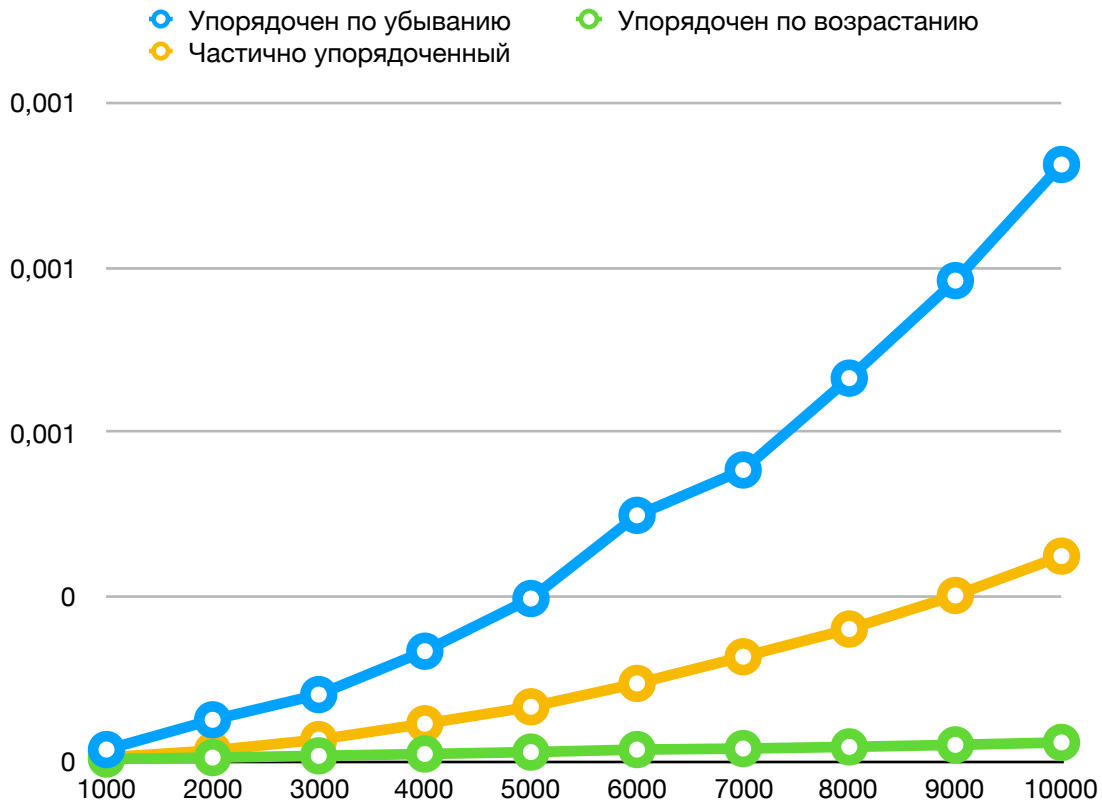


Можно заметить, что теоретические данные подтверждаются, а именно,

при массиве упорядоченно по убыванию, получается худший случай, а лучший случай, при массиве упорядоченном по возрастанию.

4.1.2 Сортировка расческой

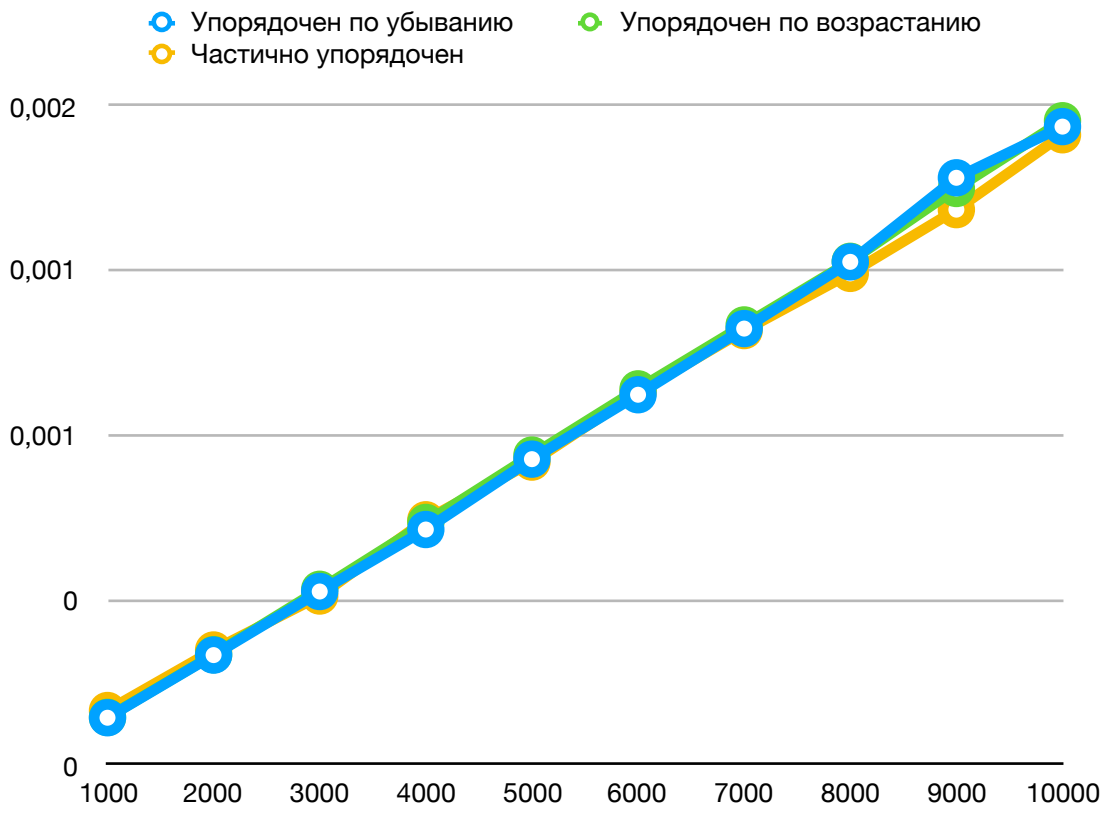
В данном эксперименте, замеры буду производиться на массивах трех типов:частично упорядоченный, упорядоченный по возрастанию(отсортированный) и упорядоченный по убыванию. Чтобы лучше была видна разница, замеры буду произведены на массивах с размерами от 1000 до 10000 с шагом 1000.



Можно заметить, что теоретические данный подтвердились, худший для данной сортировки случай наступает, когда массив отсортирован по убыванию и приходится делать большое количество «swap-ов».

4.1.2 Сортировка с помощью двоичного дерева

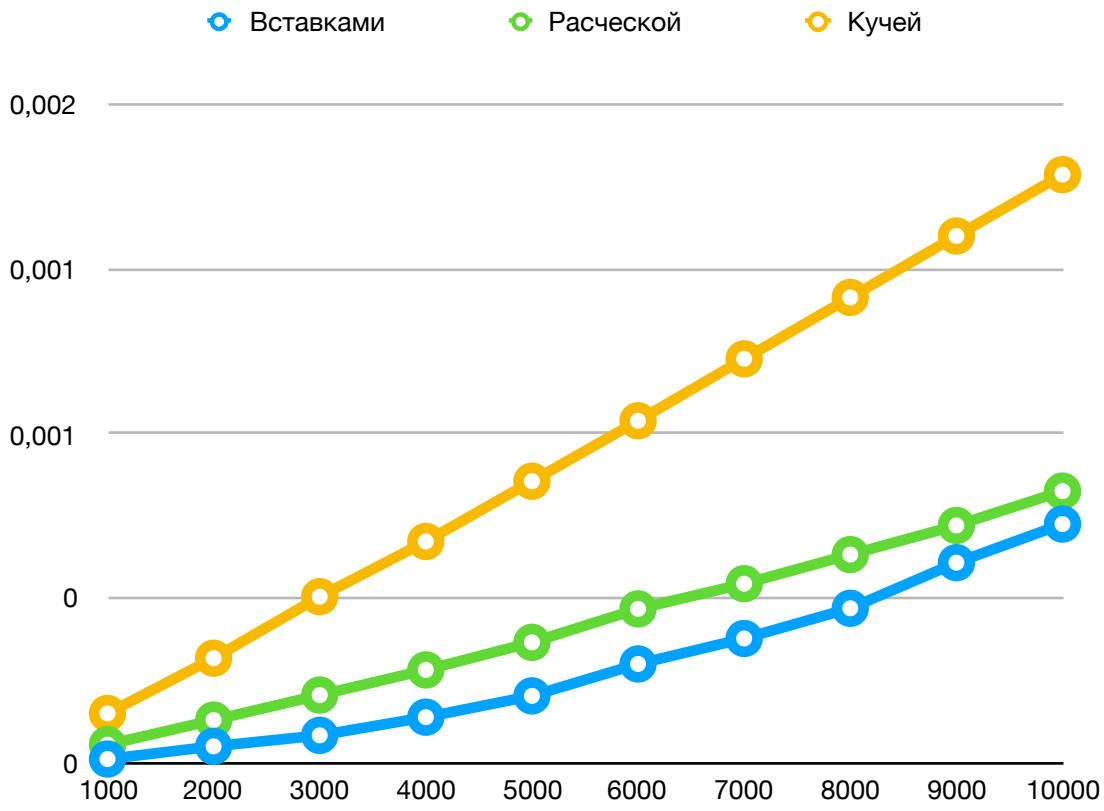
Замеры буду производиться для трех типов расположения элементов в массиве : частично упорядоченны, упорядочены по возрастанию, упорядочены по убыванию. Размеры массива будут варьироваться от 1000 до 10000 с шагом 1000



Как и было отмечено, данный алгоритм гарантировано работает за $nlog(n)$

4.1.3 Сравнение трех сортировок

В данном эксперименте будет произведено сравнение данных сортировок на случайных массивах с размерами от 1000 до 10000 с шагом 1000.



Можно сделать вывод, что сортировка вставками работает лучше, чем остальные сортировки, но сортировка кучей работает всегда за одинаковое время, что безусловно играет не малую роль, когда дело касается стабильности.

Заключение

В ходе лабораторной работы были изучены и реализованы алгоритмы сортировок массивов, а именно: сортировка вставками, сортировка расческой и сортировка двоичным деревом
Был проведен анализ данных алгоритмов. Сортировка вставками показала себя как наиболее быстрая сортировка, но наиболее устойчивой к входным данным оказалась сортировка двоичным деревом, что сходится с ожидаемым результатом