МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №1.3

Тема:

Определение эффективного алгоритма сортировки на основе эмпирического и асимптотического методов анализа

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Враженко Д.О.

Группа: ИКБО-10-23

Вариант: 10

Москва – 2024

ЦЕЛЬ РАБОТЫ

Получить навыки по анализу вычислительной сложности алгоритмов сортировки и определению наиболее эффективного алгоритма.

ХОД РАБОТЫ

1. Задание 1

Эмпирическая оценка эффективности алгоритмов

1.1 Cocktail sort:

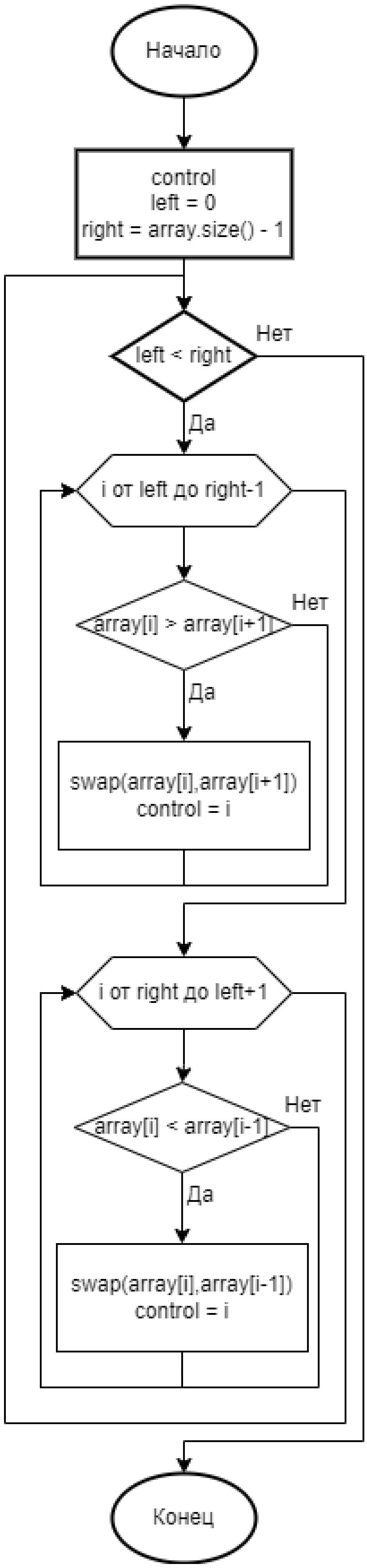
1.1.1 Словесное описание

Объявляем переменную control типа int, инициализируем переменные left и right значениями 0 и размерность массива - 1 соответственно. Пока левая граница меньше правой, то есть left < right, будет выполняться основной цикл, в который входят два цикла "для":

1) Для i = left и i < right: если элемент массива с индексом i больше элемента массива с индексом i+1, то поменять их местами и переменной control присвоить значение индекса i. Увеличение значения i на 1. Завершение цикла "для" и присваивание переменной right значение переменной control, то есть изменение правой границы.

2) Для i = right и i > left: если элемент массива с индексом i меньше элемента массива с индексом i-1, то поменять их местами и переменной control присвоить значение индекса i. Уменьшение значения i на 1. Завершение цикла "для" и присваивание переменной left значение переменной control, то есть изменение левой границы.

1.1.2 Блок-схема



1.1.3 Программный код

```
void Cocktail_Sort(int* arr, int size)
{
    int control;
    int left = 0, right = size - 1;
    while (left < right)
    {
        for (int i = left; i < right; i++)
        {
            if (arr[i] > arr[i + 1])
            {
                swap(arr[i], arr[i + 1]);
                control = i;
            }
        }
        right = control;
        for (int i = right; i > left; i--)
        {
            if (arr[i] < arr[i - 1])
            {
                swap(arr[i], arr[i - 1]);
                control = i;
            }
        }
        left = control;
    }
}
```

1.1.4 Тестирование

```
Array's size: 10


Unsorted array:
85 58 83 96 46 71 35 29 3 21

Sorted array:
3 21 29 35 46 58 71 83 85 96
```

1.1.5 Теоретическая сложность

В лучшем случае: $T(n)_{лучшее}=4*n+5=4n+5$.

В худшем случае: $T(n)_{худшее}=(3*n-1)*n+n/2+4=3n^2-\frac{n}{2}+4$.

В среднем случае: $T(n)_{лучшее} \leq T(n)_{среднее} \leq T(n)_{худшее}$.

1.1.6 Сводная таблица

n	Лучший случай		Средний случай		Худший случай	
	Кол-во операций	Время, мс	Кол-во операций	Время, мс	Кол-во операций	Время, мс
100	405	0	15128	0	29954	0
1000	4005	0	1678337	1	2999504	1
10000	40005	0	166925341	68	299995004	46
100000	400005	1	16521254853	7506	29999950004	4483
1000000	4000005	7	1654078268581	1036078	2999999500004	463752

1.1.7 Ёмкостная сложность

Составляет: $n+3$.

1.2 Quick sort:

1.2.1 Словесное описание

Если нижняя граница low меньше верхней границы high, то инициализируем 3 переменные целого типа: pivot, равной среднему элементу между low и high, i, равной low, и j, равной high. Пока i <= j:

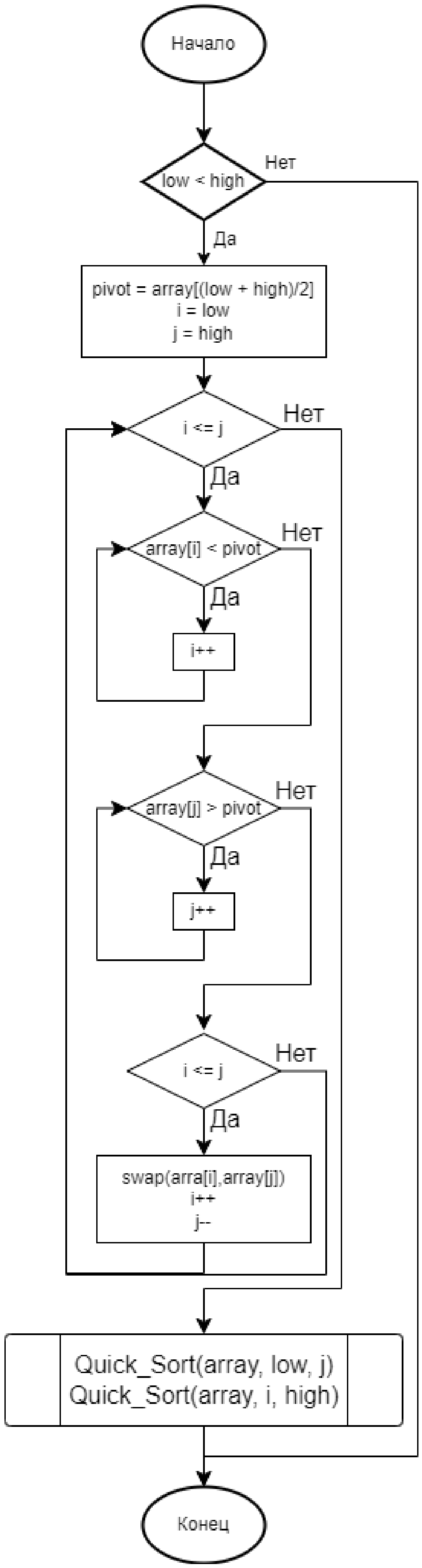
1) Пока array[i] < pivot: i++;

2) Пока array[j] > pivot: j--;

3) Если i <= j: поменять местами array[i] и array[j], i++, j--;

Вызвать Quick_Sort(array, low, j) и Quick_Sort(array, i, high). Завершение цикла пока.

1.2.2 Блок-схема



1.2.3 Программный код

```
void Quick_Sort(int* arr, int low, int high)
{
    if (low < high)
    {
        int pivot = arr[(low + high) / 2];
        int i = low;
        int j = high;
        while (i <= j)
        {
            while (arr[i] < pivot)
                i++;
            while (arr[j] > pivot)
                j--;
            if (i <= j)
            {
                swap(arr[i], arr[j]);
                i++;
                j--;
            }
        }
        Quick_Sort(arr, low, j);
        Quick_Sort(arr, i, high);
    }
}
```

1.2.4 Тестирование

```
Array's size: 10


Unsorted array:
69 0 99 19 21 24 27 61 88 73

Sorted array:
0 19 21 24 27 61 69 73 88 99
```

1.2.5 Теоретическая сложность

В лучшем случае: $T(n)_{лучшее} = n\log(n)$.

В худшем случае: $T(n)_{худшее} = n^2$.

В среднем случае: $T(n)_{лучшее} \leq T(n)_{среднее} \leq T(n)_{худшее}$.

1.2.6 Сводная таблица

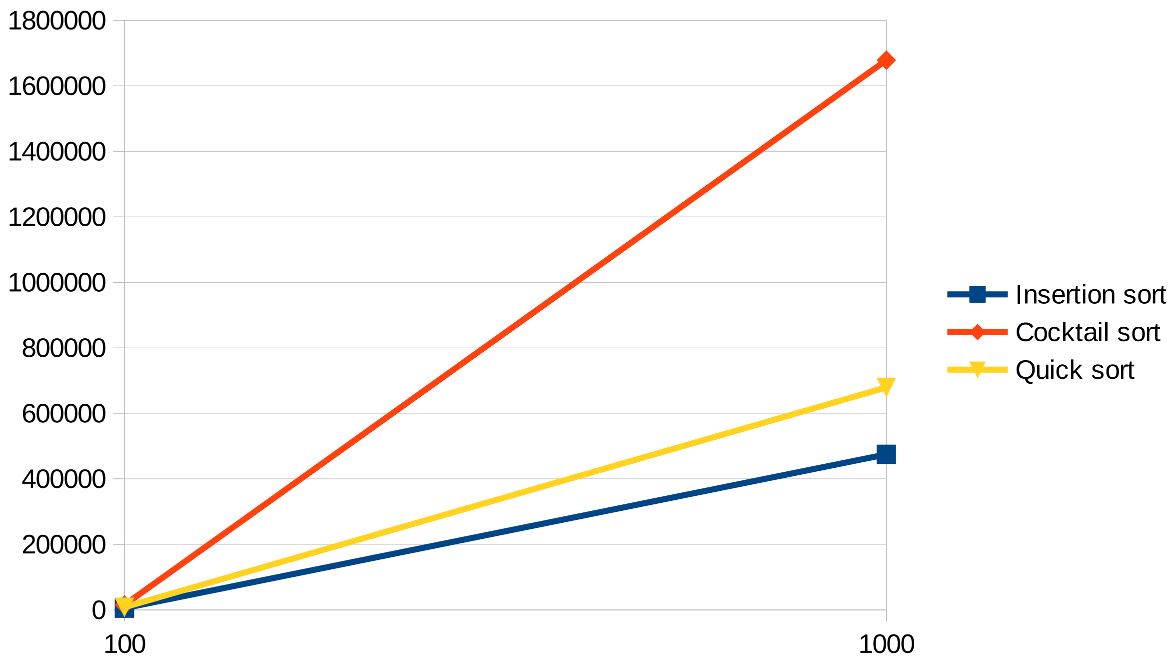
n	Лучший случай		Средний случай		Худший случай	
	Кол-во операций	Время, мс	Кол-во операций	Время, мс	Кол-во операций	Время, мс
100	200	0	7823	0	10000	0
1000	3000	0	679230	1	1000000	1
10000	40000	0	65341547	1	100000000	1
100000	500000	1	8529164123	3	10000000000	1
1000000	6000000	7	623872567145	33	1000000000000	8

1.2.7 Ёмкостная сложность

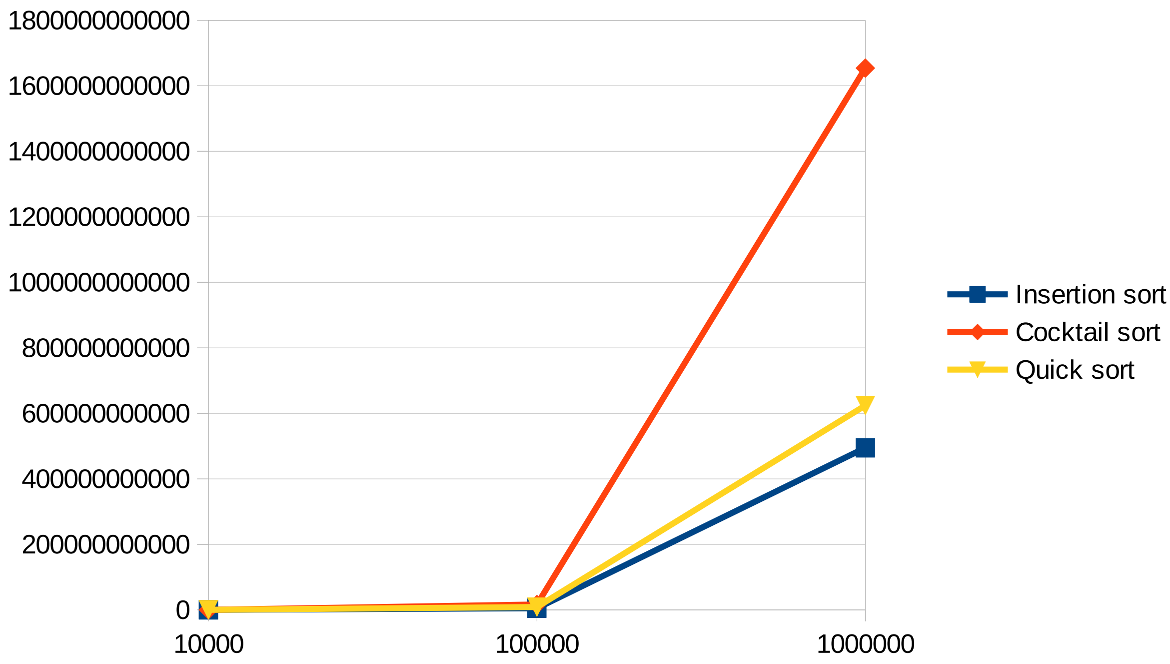
Составляет: $n+6$.

1.3.1 Графики

При $n \le 1000$



При n>1000



Вывод:

Алгоритмы сортировки зависят от упорядоченности и неупорядоченности массивов. Если, массив упорядочен, то элементы не сортируются, и происходит только сравнение их элементов. Если, массив неупорядочен, то производится и сравнение элементов и их перемещение. В любом случае Quick Sort действует быстрее других сортировок, представленных в практической работе.

2. Задание 2

Асимптотический анализ сложности алгоритмов

2.1 Insertion sort

В лучшем случае: $T(n)=6n-5$

В худшем случае: $T(n)=2n^2+3n-4$

В О-нотации (оценка сверху) для анализа худшего случая сортировки простого выбора: для T(n) подберем такую простую g(n) и константу C, так что C*g(n) превышает T(n), по мере того как n значительно растет. Получаем, что T(n) имеет порядок роста O(g(n)), если имеется константа с и счетчик n_0, такие что $0<T(n)\le C\cdot g(n)$, для $n\ge n_0$. В нашем случае $g(n)=n2$, $C=2$, а $n_0=2$. Следовательно, O(n^2).

В Ω-нотации (оценка снизу) для анализа лучшего случая сортировки обменом. Найдем такую константу с такую, что для бесконечного числа значений n> n_0 выполняется неравенство $T(n)\ge c\cdot k(n)$. Получим, что $k(n)=n^2$, $c=1$, а $n_0=1$. Следовательно: Ω(n^2)

Для данного алгоритма возможно получить асимптотически точную оценку вычислительной сложности алгоритма в нотации θ. Мы получим, что T(n) = θ(n^2). Докажем, что это действительно так. Для этого определим константы c_1, c_2 и n_0, для которых справедливо:

$c_1 * n^2 \le 2n^2 + 3n - 4 \le c_2 * n^2$ для всех $n \ge n_0$

Разделив неравенство на n^2, получим:

$$c_1 \le 1 + \frac{3}{n} - \frac{4}{n^2} \le c_2$$

Правая часть $1+\frac{3}{n}-\frac{4}{n^2}\le c_2$ выполнится для всех $n \ge 2$, если выбрать $c_2 \ge 3/2$ (при n→∞ 3/n→0 и 4/n^2).

Аналогично левая часть $c_1 \le 1+\frac{3}{n}-\frac{4}{n^2}$ выполнится для всех $n \ge 1$, если выбрать $c_1 \le 0$.

Тогда найдены $c_1 = 0$, $c_2 = 3/2$ и $n_0=4$, а, значит, по определению, T(n)= θ(n^2).

Алгоритм	Асимптотическая сложность алгоритма			
	Наихудший случай (сверху)	Наилучший случай (снизу)	Средний случай (точная оценка)	Ёмкостная сложность
Insertion sort	O (n^2)	Ω (n)	θ (n^2)	O (1)
Cocktail sort	O (n^2)	Ω (n)	θ(n^2)	O (1)
Quick sort	O (n^2)	Ω (n*log(n))	θ (n*log(n))	O (n)

Вывод:

Из трёх сортировок, исследуемых мной самой эффективной для большего количества элементов, является Quick sort.