

Государственное образовательное учреждение высшего профессионального образования

«Московский Государственный Технический Университет имени Н. Э. Баумана»

ОТЧЕТ

По лабораторной работе №3

По курсу «Анализ алгоритмов»

Тема: **«Трудоёмкость сортировок»**

Студент: Кононенко С. Д.

Группа: ИУ7-51

Москва, 2017

**Алгоритм сортировки** — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

**Трудоемкость программы** (алгоритма) – это зависимость количества массовых операций (сравнения, обмены, сдвиги, повторения цикла и т.п.) от объема (размерностей) обрабатываемых данных.

**Теоретическая оценка трудоёмкости**

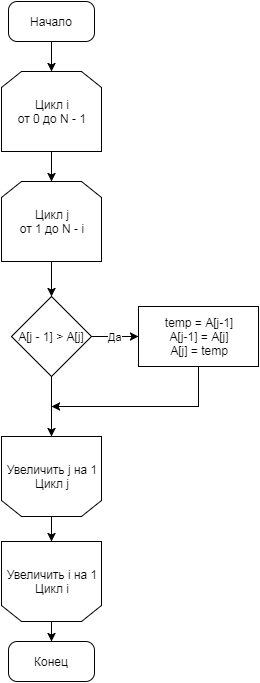
**Пузырьковая сортировка(bubble sort)**

Рисунок 1 Блок-схема Bubble Sort

2 + (N - 1)(2

+ 2 + (2

+ 4 +

)

)

= 2 + (N-1)(2 + (2 + 4 + )) = 2 + 2N - 2 + NN\*0.5 – N\*0.5 = 2N + NN

- N = N2 - N

**Сортировка вставками(selection sort)**

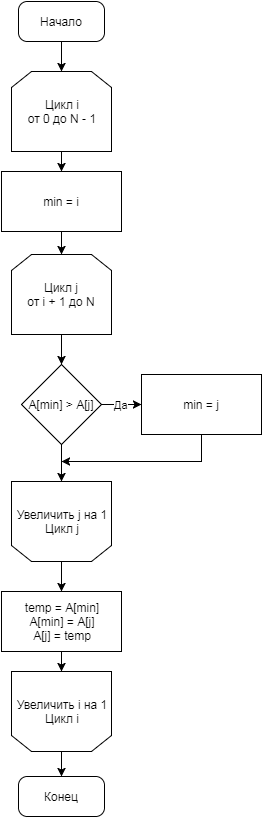
****

Рисунок 2 Блок-схема Selection Sort

2 + (N - 1)(2

+ 1

+ 2 + (2

+ 3 +

)

+ 7

)

= 2 + (N – 1)(2 + 1 + 2 + (2 + 3 + ) + 7) = 2 + 12N – 12 + NN\*0.5 – N\*0.5 =

= N2 + N -10

**Быстрая сортировка(Quick Sort)**

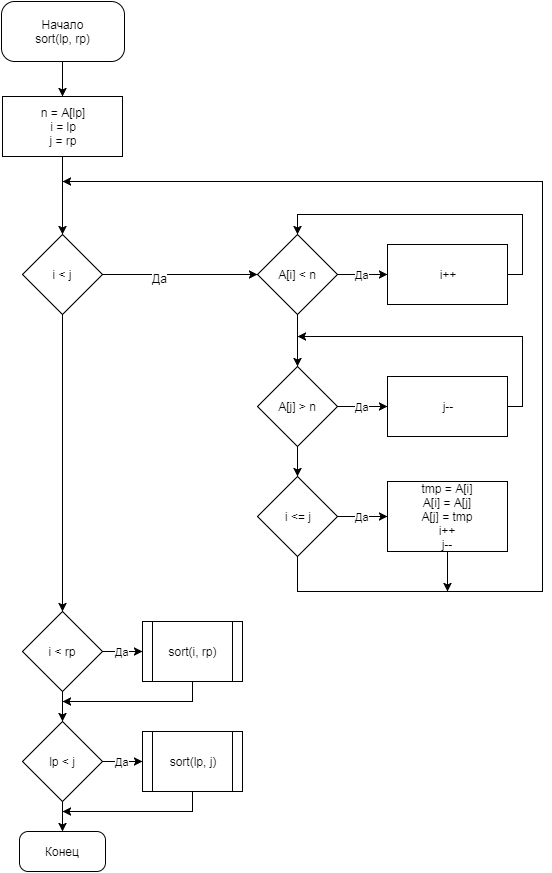


Рисунок 3 Блок-схема Qsort

*Трудоемкость*

Операция разделения массива относительно опорного элемента имеет сложность O(n), что будет верно для каждого уровня рекурсии

Глубина рекурсии в лучшем и среднем случаях будет равняться O(), в худшем случае - N

Следовательно общая сложность алгоритма будет O(n \* ) и O(n2) в лучшем\среднем и худшем случаях соответственно.

**Вывод:** Пузырьковая сортировка имеет высокую трудоёмкость и сильно зависит от случая. Сортировка выбором малочувствительна к случаю, а также имеет меньшую трудоемкость в среднем чем пузырьковая.

Быстрая сортировка имеет наименьшую трудоёмкость среди рассмотреных, а также, при грамотном выборе опорного элемента, может практически не иметь худших случаев.

**Bubble Sort**

void my\_bubble\_sort(void \*arr, size\_t count, size\_t size,

compare\_t compare)

{

char \*endp = (char \*)arr + count \* size;

for (register int i = 0; i < count - 1; i++)

for (char \* j = arr + size; j < endp - (i \* size); j+= size)

if (compare(j - size, j) > 0)

swap(j - size, j, size);

}

**Selection Sort**

void my\_selection\_sort(void \*arr, size\_t count, size\_t size,

compare\_t compare)

{

char \*endp = (char \*)arr + count \* size;

for (char \*i = arr; i < endp - size; i += size)

{

size\_t min = 0;

for (char \*j = i + size; j < endp; j += size)

if (compare(i + (min \* size), j) > 0)

min = (j - i) / size;

swap(i, i + (min \* size), size);

}

}

**Qsort**

static void sort(void \*const lp, void \*const rp, size\_t size,

compare\_t compare)

{

void \*n = malloc(size);

memcpy(n, lp, size);

void \*i = lp, \*j = rp;

while (i < j)

{

while (compare(i, n) < 0)

i = (char \*)i + size;

while (compare(j, n) > 0)

j = (char \*)j - size;

if (i <= j)

{

if (compare(i, j) != 0)

swap(i, j, size);

i = (char \*)i + size;

j = (char \*)j - size;

}

}

free(n);

if (i < rp)

sort(i, rp, size, compare);

if (lp < j)

sort(lp, j, size, compare);

}

void my\_qsort(void \*arr, size\_t count, size\_t size,

compare\_t compare)

{

void \*lp = arr;

void \*rp = (char \*)arr + size \* (count - 1);

sort(lp, rp, size, compare);

}

**Результаты**

На рис. 4 и 5 по оси OX расположены значения размерности сортуруемых массивов, а по оси OY время работы алгоритма в тиках \* -1e3

Рисунок 4 Гистограмма времени работы.

Рисунок 5 График времени работы

**Вывод:** результаты эксперимента подтвердили теоретическую оценку.

**Заключение**

Было рассмотрено 3 алгоритма сортировки: Пузырьковая, Сортировка вставками и Быстрая сортировка. Теоретически и экспериментально было выяснено, что:

- Пузырьковая сортировка являтеся плохим выбором в следствие высокой трудоёмкости, а также сильной зависимости от исходного состояния сортируемых данных.

- Сортировка выбором также является малоэфективной, но является устойчивым алгоритмом т.е. время её работы практически не зависит от входных данных

- Быстрая сортировка – наиболее эффективный алгоритм из рассмотреных, обладающий достаточно высокой устойчивостью.

- Время работы алгоритма может значительно ухудшаться при неподходящих входных данных, что следует учитывать при выборе алгоритма сортировки.