**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**Московский государственный технический университет**

**им. Н.Э. Баумана**

**(МГТУ им. Н.Э. Баумана)**

**Факультет Информатика и системы управления**

**Кафедра Программное обеспечение ЭВМ и информационные технологии (ИУ7)**

Лабораторная работа №3

По дисциплине: «Анализ алгоритмов»

Тема: «Трудоемкость алгоритмов сортировки»

Выполнил: Набережный А.А.

Группа: ИУ7-52

Преподаватель: Волкова Л.Л.

Москва 2017 г.

Оглавление

[1. Постановка задачи 3](#_Toc499767368)

[2.Описание алгоритмов 4](#_Toc499767369)

[3.Реализация алгоритмов 5](#_Toc499767370)

[4. Оценка трудоёмкости 8](#_Toc499767371)

[5. Примеры работы 8](#_Toc499767372)

[6. Сравнение 9](#_Toc499767373)

[7. Выводы 11](#_Toc499767375)

[8. Заключение 13](#_Toc499767376)

# 1. Постановка задачи

Реализовать три алгоритма сортировки. В данной работе будут рассмотрены такие сортировки как: сортировка простого обмена(пузырёк), сортировка вставками и сортировка слиянием. Дать теоретическую оценку одной из выше приведённых и пояснить откуда берётся оценка остальных двух. провести замеры процессорного времени выполнения для лучшего, худшего и произвольного случаев. Сравнить эффективность алгоритмов между собой.

**2.Описание алгоритмов**

**Пузырёк**

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 {\displaystyle N-1}раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде, отсюда и название алгоритма).

**Вставками**

Этот алгоритм использует метод последовательного включения нового элемента в уже отсортированную часть массива. Изначально рассматривается массив, состоящий из одного элемента, к которому последовательно добавляются новые элементы из еще не отсортированной части исходного массива. При этом одновременно с поиском места для вставки нового элемента уже отсортированная часть массива справа сдвигается на одну позицию, освобождая место для вставки этого элемента. Таким образом, алгоритм работает «по месту», т.е. сортирует массив, не требуя дополнительного массива для размещения результата.

**Слиянием**

Эта сортировка — хороший пример использования принципа «[разделяй и властвуй](https://ru.wikipedia.org/wiki/Разделяй_и_властвуй_(программирование))». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью [рекурсивного вызова](https://ru.wikipedia.org/wiki/Рекурсия) или непосредственно, если их размер достаточно мал. Наконец, их [решения](https://ru.wikipedia.org/wiki/Решение_задач) комбинируются, и получается решение исходной задачи.

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.  
Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по неубыванию подмассива. Тогда:  
3.2. Слияние двух подмассивов в третий результирующий массив.  
На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.  
3.3. «Прицепление» остатка.  
Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

# 3.Реализация алгоритмов

В качестве языка программирования выбран с++

**Пузырёк**

bool bubble\_sort(const int \*const a1, int \*a2, const int n)

{

if (n < 1)

return false;

memcpy(a2, a1, n\*sizeof(int));

int t;

for(int i = 1; i < n; i++)

for(int j = 0; j < n-i; j++)

{

if(a2[j+1] < a2[j])

{

t = a2[j];

a2[j] = a2[j+1];

a2[j+1] = t;

}

}

return true;

}

**Вставками**

bool insertion\_sort(const int \*a1, int \*a2, const int n)

{

if (n < 1)

return false;

memcpy(a2, a1, n\*sizeof(int));

int t;

for(int i = 1; i < n; i++)

for(int j = i; j > 0; j--)

{

if(a2[j] < a2[j-1])

{

t = a2[j];

a2[j] = a2[j-1];

a2[j-1] = t;

}

}

return true;

}

**Слиянием**

void merge(int \*arr, int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int \*L = new int[n1];

int \*R = new int[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

delete[] L;

delete[] R;

}

void mergeSort(int \*arr, int l, int r)

{

if (l < r)

{

int m = l+(r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

bool marge\_sort(const int \*a1, int \*a2, const int n)

{

if (n < 1)

return false;

memcpy(a2, a1, n\*sizeof(int));

mergeSort(a2, 0, n-1);

return true;

}

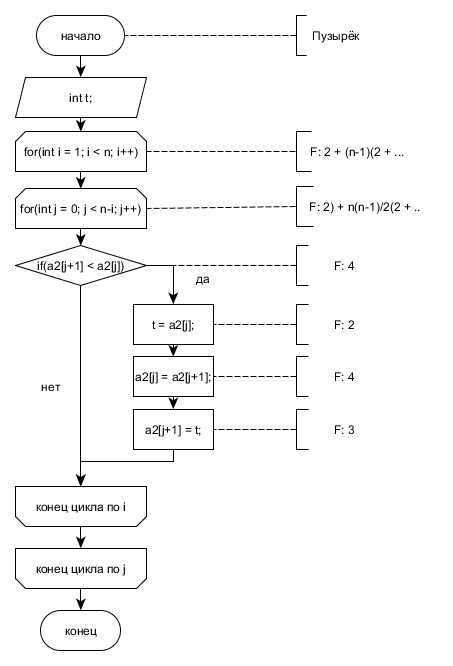
# Оценка трудоёмкости

Определим операции единичной трудоемкости: + , - , \*, / , % , < , <= , >= , > , == , != , = , [ ] , ++ , -- , += , -=.

Для оценки трудоемкости будем использовать C++ - подобную модель.

Ниже представлена схема алгоритма. В комментариях указана трудоемкость каждой отдельной операции.

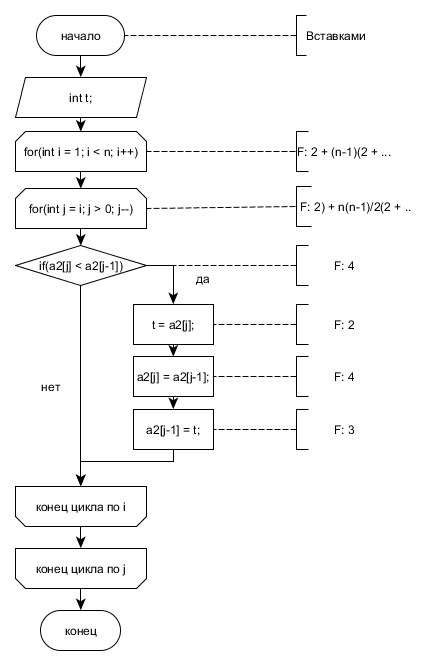
**Пузырёк**



Fл.сл = 4N - 2 + N(N-1)/2(2 + 4) = 3N^2 + N - 2

Fх.сл. = 4N -2 + N(N-1)/2(2 + 4 +9) = 7.5N^2 – 3.5N - 2

**Вставками**



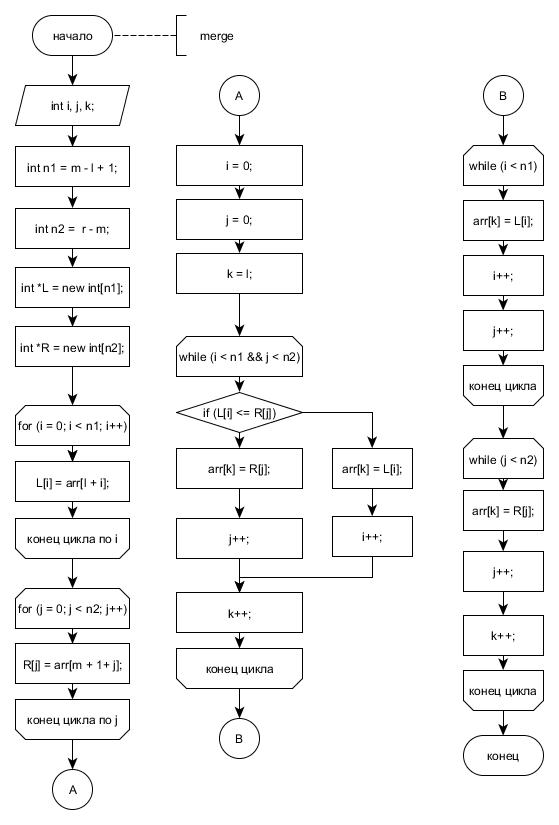
Fл.сл = 4N - 2 + N(N-1)/2(2 + 4) = 3N^2 + N - 2

Fх.сл. = 4N -2 + N(N-1)/2(2 + 4 +9) = 7.5N^2 – 3.5N - 2

**Слиянием**

Чтобы оценить время работы этого алгоритма, составим рекуррентное соотношение. Пускай T(n) - время сортировки массива длины n, тогда для сортировки слиянием справедливо T(n) = 2T(n/2) + O(n), где O(n) - время необходимое на то, чтобы слить два массива длины n. Распишем это соотношение:

T(n) = 2T(n/2) + O(n) = 4T(n/4) + 2O(n) = … = T(1) + log2(n)O(n) = O(n\*log2(n))



# 5. Примеры работы

Для неупорядоченного массива представлены замеры процессорного времени в секундах см. Таблицу1.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Пузырёк | Вставками | Слиянием |
| 100 | 0.000070 | 0.000071 | 0.000228 |
| 200 | 0.000323 | 0.000270 | 0.000712 |
| 300 | 0.000743 | 0.000594 | 0.000712 |
| 400 | 0.001285 | 0.001055 | 0.000980 |
| 500 | 0.001977 | 0.001680 | 0.001225 |
| 600 | 0.002792 | 0.002378 | 0.001507 |
| 700 | 0.003757 | 0.003250 | 0.001753 |
| 800 | 0.004796 | 0.004424 | 0.002671 |
| 900 | 0.006256 | 0.005482 | 0.002528 |
| 1000 | 0.008247 | 0.009122 | 0.002538 |

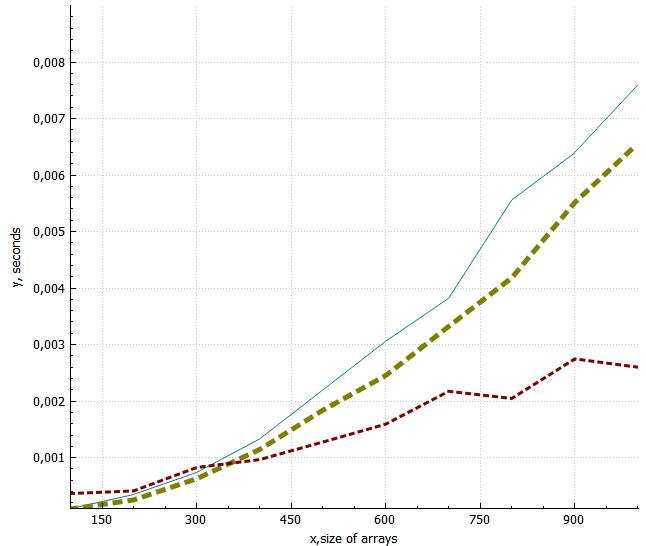
Таблица 1

# 6. Сравнение

Элементы массива лежат в диапазоне от 0 до 10000.

Сравнение реализованных алгоритмов проводилось на трёх, различным образом отсортированных, массивах см. графики 1, 2, 3.

**Массивы не отсортированы.**



**График 1**

Сплошная голубая линия соответствует сортировке Пузырьком.

Толстая штриховая жёлтая - сортировке Вставками.

Тонкая штриховая красная - сортировке Слиянием.

**Массивы отсортированы по возрастанию.**

# отсорт

**График 2**

**Массивы отсортированы по убыванию.**

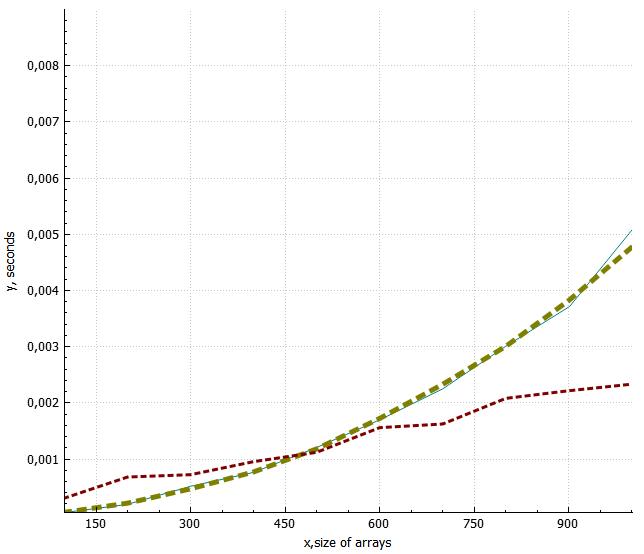


График 3

Алгоритм простого обмена и вставками наиболее хорошо работают уже с отсортированными массивами, что не скажешь про сортировку слиянием, которая не зависит от качества сортировки исходного массива..

# 7. Выводы

У каждого алгоритма есть свои достоинства и недостатки.

Сортировка пузырьком и сортировка вставками.

Достоинства:

* простота реализации
* нет дополнительных расходов по памяти

Недостатки:

* время выполнения пропорциональна квадрату количества элементов

Сортировка слиянием.

Достоинства:

* время выполнения пропорционально О(n\*log2(n))

Недостатки:

* на «почти отсортированных» массивах работает так же, как на хаотичных.
* требует дополнительной памяти по размеру исходного массива.

# 8. Заключение

В ходе выполнения лабораторной работы были реализованы три алгоритма сортировки массива: сортировка простым выбором (пузырьком), сортировка вставками и слиянием. Определены теоретически сложности работы этих алгоритмов; они составили N2 для сортировки пузырьком, N2 для сортировки вставками и N\*log2(N) для сортировки слиянием. Выполнены замеры времени работы алгоритмов для лучшего, худшего и произвольного случаев. А также сделаны выводы о достоинствах и недостатках реализованных алгоритмов.