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»

ОТЧЁТ ПО АНАЛИЗУ АЛГОРИТМОВ

к лабораторной работе №4 на тему:

Алгоритмы сортировки. Рассчет вычислительной сложности алгоритма.

Студент: Квасников А.В. ИУ7-56

Москва 2018

Введение

Постановка задачи.

В данной работе требуется оценить трудоемкость двух алгоритмов сортировок, привести оценку трудоемкости третьего алгоритма сортировки из справочника, определить лучшие и худшие случаи входных данных для каждого из алгоритмов. Необходимо провести замеры времени работы данных алгоритмов и сравнить результаты с их теоретической оценкой трудоемкости.

Цели лабораторной работы.

Цели данной лабораторной работы - дать теоретическую оценку трудоемкости алгоритмов сортировок и сравнить результаты теоретической оценки трудоемкости с полученными в ходе проведения эксперимента результатами.

1 Аналитическая часть

1.1 Описание алгоритмов

1.1.1 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 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).

1.1.2 Сортировка выбором.

Идея метода состоит в том, чтобы создавать отсортированную последовательность путем присоединения к ней одного элемента за другим в правильном порядке. Будем строить готовую последовательность, начиная с левого конца массива. Алгоритм состоит из n последовательных шагов, начиная от нулевого и заканчивая (n-1). Таким образом, так как число обменов всегда будет меньше числа сравнений, время сортировки растет квадратично относительно количества элементов. Алгоритм не использует дополнительной памяти: все операции происходят "на месте".

1.1.3 Сортировка Шелла.

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочивающем элементов при d = 1. Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места.

1.2 Модель вычислений

Стоимость каждой операции из следующего множества будем считать 1: {+, -, *, /, %, <, <=, !=, >=, >, == (равенство), = (присваивание), &, ||, [] (обращение по индексу)}.

Стоимость условного перехода if-else возьмем за 0 и будем учитывать лишь стоимость выражения внутри.

Теперь определим стоимость циклов. Пусть цикл имеет вид for (i = 0; i < N; i++), где N - кол-во итераций. Тогда i = 0, i < N - операции, выполняемые перед началом выполнения тела цикла (их общая стоимость равна 2), i++, i < N - операции, выполняемые после тела цикла (стоимость равна 2).

Таким образом, суммарная стоимость цикла будет составлять f_цикла = 2 + N * (f_тела + 2), или в общем виде f_цикла = f_нач_операции + N *(f_тела + f_операций_после_тела_цикла).

2. Конструкторская часть

2.1 Разработка алгоритмов

2.1.1 Сортировка пузырьком

```
sz = ДЛИНА МАССИВА
ЕСЛИ sz <= 1 ТО
     ЗАВЕРШИТЬ
B = TRUE
ПОКА B = TRUE
     B = FALSE
     ПОКА (I; I + 1 < sz; УВЕЛИЧИТЬ I)
          ЕСЛИ A[I] > A[I+1] ТО
               ПОМЕНЯТЬ МЕСТАМИ A[I] И A[I+1]
               B = TRUE
     УМЕНЬШИТЬ ДЛИНУ МАССИВА
```

2.1.2 Сортировка выбором

```
ПОКА I = 0; I < ДЛИНА МАССИВА; УВЕЛИЧИТЬ I)
     MIN = A[I]
     ПОКА J = I + 1; J < ДЛИНА МАССИВА; УВЕЛИЧИТЬ J)
          ЕСЛИ A[J] < MIN ТО
               MIN = A[J]
               INDEX = J
     ПОМЕНЯТЬ МЕСТАМИ A[I] И A[INDEX]
```

2.1.3 Сортировка Шелла

```
sz = ДЛИНА МАССИВА
STEP = sz/2
ПОКА STEP >= 1
     ПОКА I = 0 + STEP; I < ДЛИНА МАССИВА; УВЕЛИЧИТЬ I)
          J = I
          DIFF = J - STEP
          ПОКА A[DIFF] >= A[0] И DIFF > J) {
                    ПОМЕНЯТЬ МЕСТАМИ A[DIFF] И A[J]
                    J = DIFF;
                    DIFF = J - STEP;
          STEP /= 2;
```

2.2 Оценка трудоемкости алгоритмов

2.1.1 Сортировка пузырьком

Сложность сортировки пузырьком = $O(N^2)$ [1].

1

2.1.2 Сортировка выбором

```
for (int *i = l; i < r; i++) {
     int minz = *i, *ind = i;
     for (int *j = i + 1; j < r; j++) {
          if (*j < minz) minz = *j, ind = j;
     }
     swap(*i, *ind);
}
```

Сложность сортировки выбором = 2 + n(2 + (1 + 1 + 2 + m(2 + (1 + 1 + 1)) + 1 + 1 + 1)) = 2 + 5nm + 9n

$O(N^2)$ в обоих случаях

2.1.3 Сортировка Шелла

Сложность сортировки Шелла[2].:
Худшее время = $O(N^2)$
Лучшее время = $O(N * log^2 N)$

3. Технологическая часть

3.1 Требования к программному обеспечению

Программа разрабатывалась в среде XCode, macOS.

3.2 Листинг кода

3.2.1 Алгоритм сортировки пузырьком.

```
void bubblesort(int* l, int* r) {
     int sz = r - l;
     if (sz <= 1) return;
     bool b = true;
     while (b) {
          b = false;
          for (int* i = l; i + 1 < r; i++) {
               if (*i > *(i + 1)) {
                    swap(*i, *(i + 1));
                    b = true;
               }
          }
          r--;
     }
}
```

3.2.2 Алгоритм сортировки выбором.

```
void selectionsort(int* l, int* r) {
     for (int *i = l; i < r; i++) {
          int minz = *i, *ind = i;
          for (int *j = i + 1; j < r; j++) {
               if (*j < minz) minz = *j, ind = j;
          }
          swap(*i, *ind);
     }
}
```

3.2.3 Алгоритм сортировки Шелла

```
void shellsort(int* l, int* r) {
     int sz = r - l;
     int step = sz / 2;
     while (step >= 1) {
          for (int *i = l + step; i < r; i++) {
               int *j = i;
               int *diff = j - step;
               while (diff >= l && *diff > *j) {
                    swap(*diff, *j);
                    j = diff;
                    diff = j - step;
               }
          }
          step /= 2;
     }
}
```

4 Экспериментальная часть

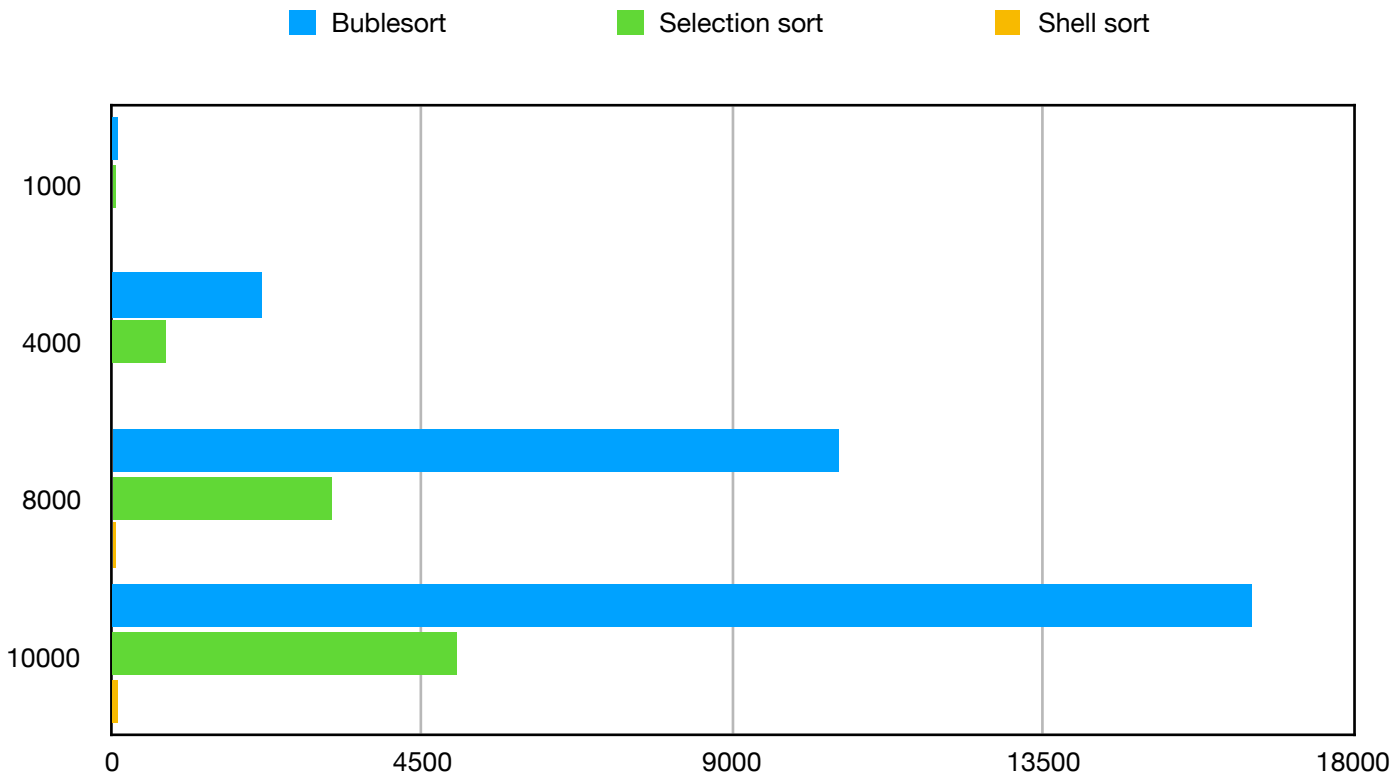
4.1 Результаты работы алгоритмов

Для исследования скоростных характеристик был использован компьютер на базе macOS. Так же, для наглядного анализа эффективности алгоритмов по временной характеристике построены графики по значениям таблицы.

Таблица 4.1 — Таблица для сравнения результатов работы алгоритмов для отсортированного по убыванию массива в миллисекундах.

N	1000	4000	8000	10000
Bublesort	105	2173	10529	16511
Selection sort	58	799	3193	5021
Shell sort	7	28	61	87

Рисунок 4.1 — График скорости работы алгоритмов.



Лучшие и худшие случаи сортировок.

Рисунок 4.2 — График сортировки Шелла в двух случаях.

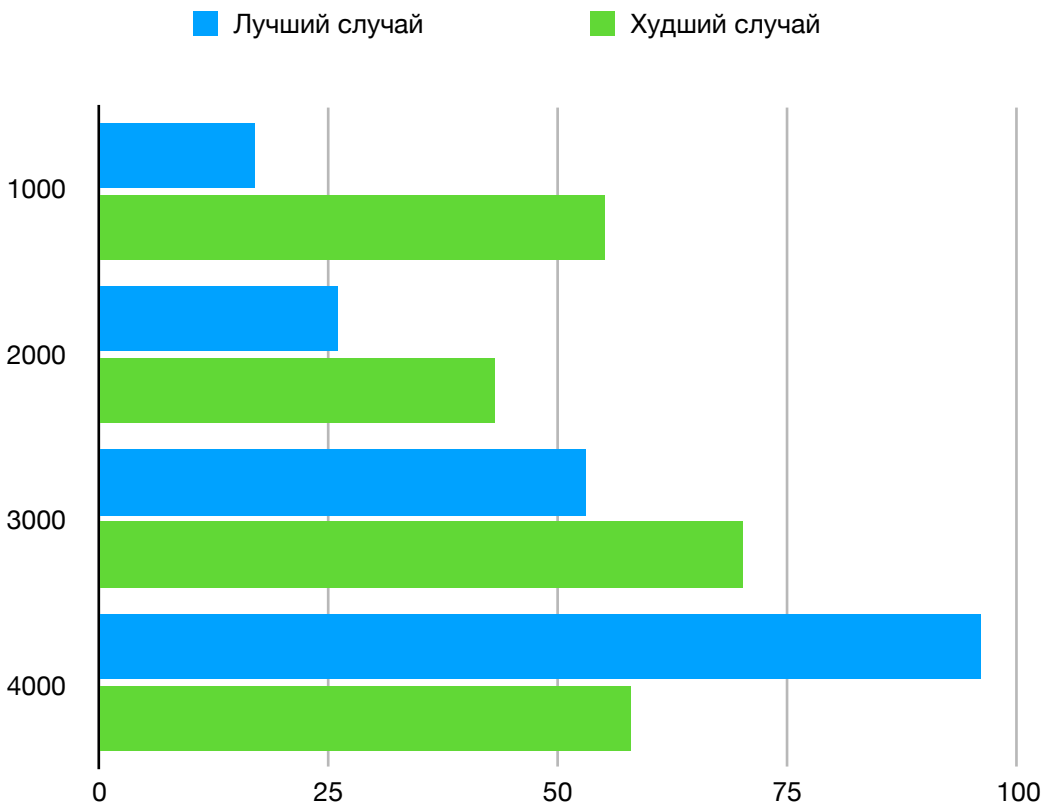


Рисунок 4.3 — График сортировки пузырьком в двух случаях.

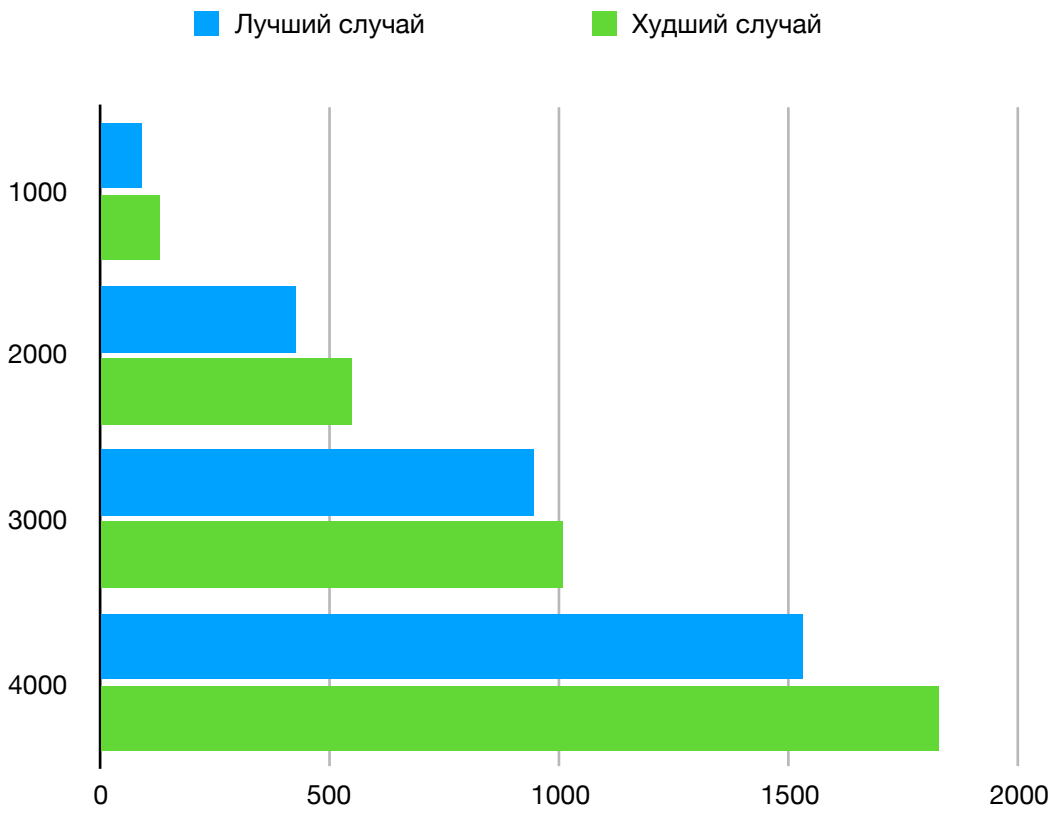
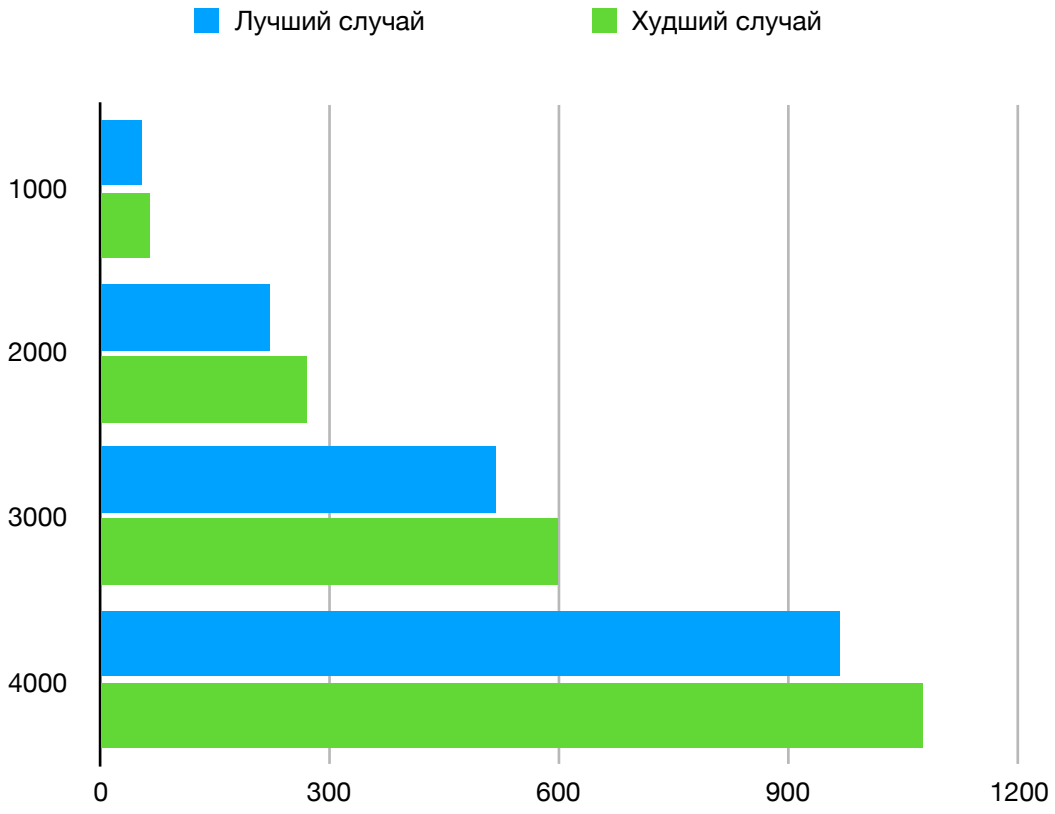


Рисунок 4.4 — График сортировки выбором в двух случаях.



Заключение

В результате работы было сделано следующее:
а) определена теоретическая оценка выбранных алгоритмов сортировки;
б) изучены теоретические оценки алгоритмов из справочных материалов;

Список использованных ресурсов

[1]. https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BF%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%BC

[2]. https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%A8%D0%B5%D0%BB%D0%BB%D0%B0

1