МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"
РТУ МИРЭА

Институт информационных технологий
Кафедра математического обеспечения и стандартизации
информационных технологий

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 3

Тема. Стек. Очередь. Применение стека и очереди: польская запись выражений и вычисления значения выражения.

по дисциплине

«Дисциплина структуры и алгоритмы обработки данных**»**

Выполнил студент группы ИНБО-15-20 Ло Ван Хунг

Принял ассистент кафедры МОСИТ Коваленко М.А.

Практическая работа выполнена «__»_____2021 г.

«Зачтено» «__»_____2021 г.

Москва 2021

Задания

1.1 Задание 1. Оценить зависимость времени выполнения алгоритма простой сортировки на массиве, заполненном случайными числами

1. Составить программу сортировки (функцию) одномерного целочисленного массива A[n], используя алгоритм согласно варианту, индивидуального задания - Алгоритм задания 1. Провести тестирование программы на исходном массиве, сформированном вводом с клавиатуры. Рабочий массив A сформировать с использованием генератора псевдослучайных чисел.

2. Провести контрольные прогоны программы для размеров массива n = 100, 1000, 10000, 100000 и 1000000 элементов с вычислением времени выполнения T(n) – (в миллисекундах/секундах). Полученные результаты свести в сводную таблицу.

3. Построить график зависимости времени выполнения программы от размера массива.

4. Провести эмпирическую (практическую) оценку вычислительной сложности алгоритма, для чего предусмотреть в программе подсчет фактического количества операций сравнения Сф и количества операций перемещения Мф. Полученные результаты Сф+Мф вставить в сводную таблицу. Сводная таблица результатов n T(n) Тт=f(C+M) Тп=Сф+Мф 100 1000 10000 100000 1000000

5. Построить в одной координатной плоскости графики зависимости теоретической Тт=f(C+M)=O(f(n)) и практической Тп=(Сф+Мф) вычислительной сложности алгоритма от размера n массива.

6. Определить емкостную сложность алгоритма от n.

7. Провести анализ полученных результатов. Сделать выводы о проделанной работе, основанные на полученных результатах.

1.2 Задание 2. Оценить вычислительную сложность алгоритма простой сортировки в наихудшем и наилучшем случаях.

1. Провести дополнительные прогоны программы на массивах, отсортированных:

А) строго в убывающем порядке значений элементов, результаты представить в сводной таблице по формату Таблица 1;

Б) строго в возрастающем порядке значений элементов, результаты представить в сводной таблице по формату Таблица 1;

В) Провести анализ зависимости (или независимости) алгоритма сортировки от исходной упорядоченности массива.

1.3 Задание 3. Оценить эффективность алгоритмов простых сортировок

1. Выполнить разработку алгоритма и программную реализацию Алгоритма задания 3 варианта.
2. Сформировать таблицу в соответствии с форматом Таблица 1 на тех же массивах, что и в задании 1.
3. Выполнить сравнительный анализом полученных результатов контрольных прогонов и построением соответствующих графиков. 4. Определить емкостную сложность алгоритма от n.

1. Выполнить разработку алгоритма и программную реализацию Алгоритма задания 3 варианта.

2. Сформировать таблицу в соответствии с форматом Таблица 1 на тех же массивах, что и в задании 1.

3. Выполнить сравнительный анализом полученных результатов контрольных прогонов и построением соответствующих графиков.

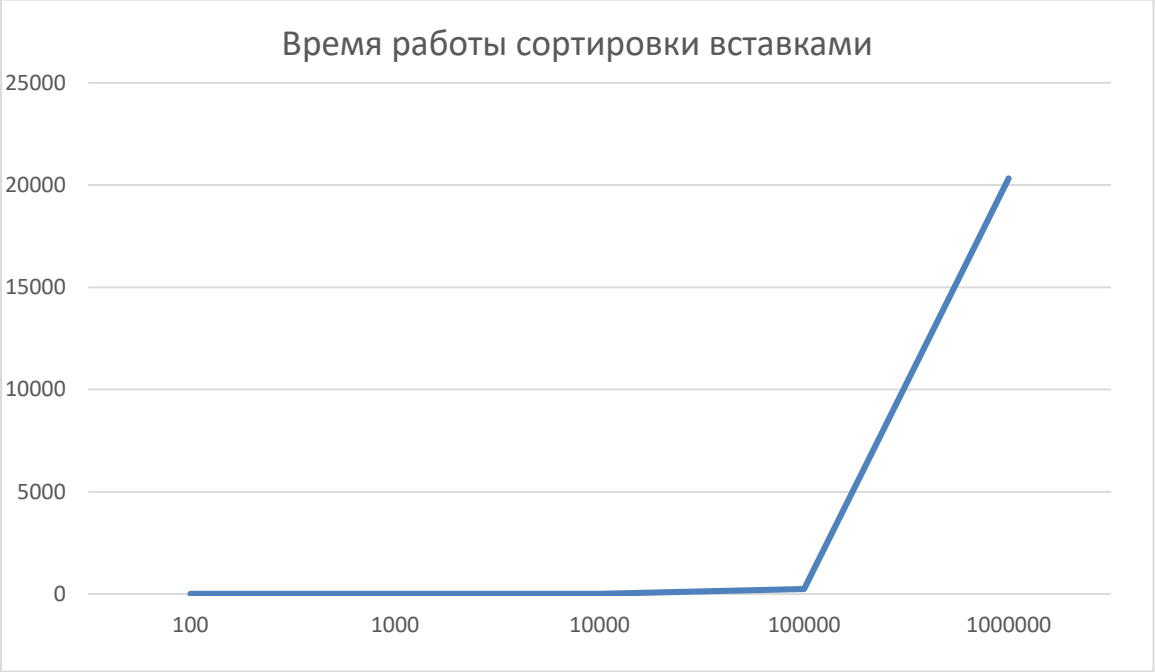
4. Определить емкостную сложность алгоритма от n.

Отчет по заданию 1. Зависимость вычислительной сложности алгоритма сортировки массива от размера массива n

Сортировка Вставками

Сводная таблица результатов при сортировке массива, заполненного случайными значениями :

n	T	f(C+M)	Сф+Мф
100	2	10000	7452
1000	2	1000000	746925
10000	5	100000000	74875077
100000	247	10000000000	7475185318
1000000	20328	1000000000000	762917413555



Код программы:

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;
```

```
int main()
{
    setlocale(LC_ALL, "ru");
    srand(time(0));
    int r, sr = 0, per=0;
    cin >> r;
    int* x = new int[r];
    for (int i = 0; i < r; i++) {
        x[i] = rand() % 100;
    }
    cout << "Сортировка всавками" << endl;
    int k, t;
    for (int i = 1; i < r; i++)
        for (int j = i; j > 0; j--) {
            if (x[j - 1] > x[j]) {
                swap(x[j - 1], x[j]);
                per++;
            }
            sr++;
        }
    for (int i = 0; i < r; i++)
        cout << x[i] << " ";
    cout << "t = " << clock()/CLOCKS_PER_SEC << " ";
    cout << "per = " << per << " ";
    cout << "sr = " << sr << " ";
    return 0;
}
```

Данная сортировка при работе с массивом большого размера, заполненного случайными значениями, работает значительно дольше.

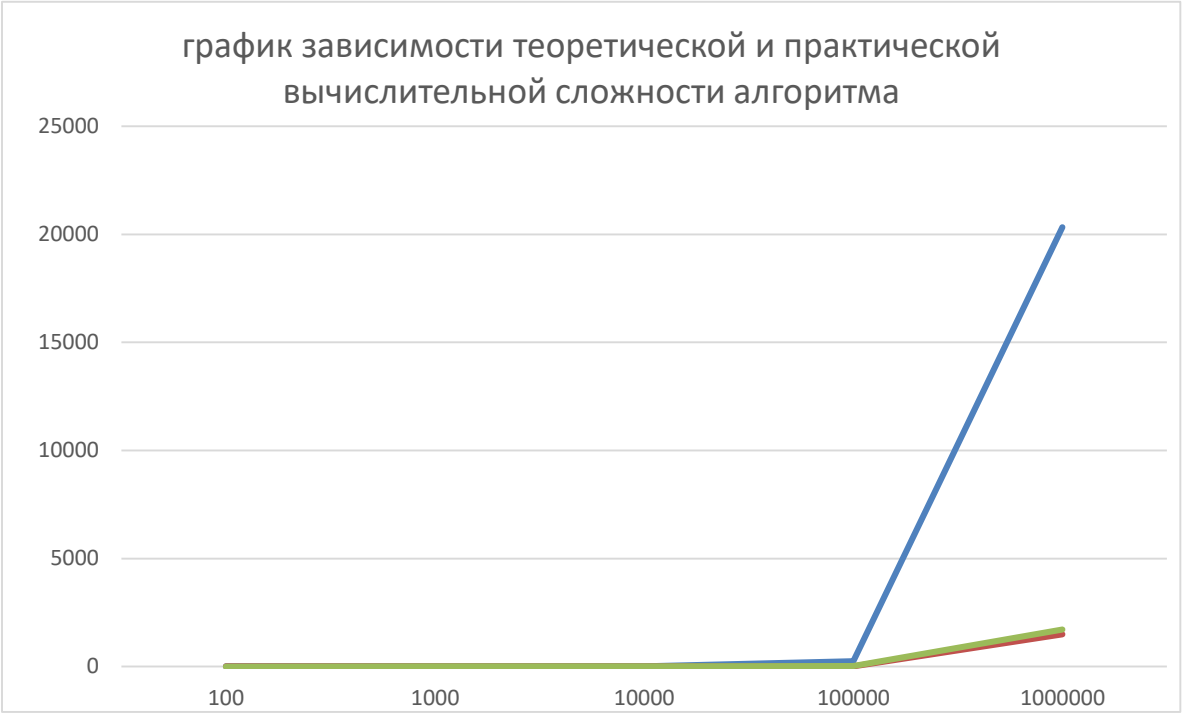
2.2 Отчет по заданию 2. Зависимость сложности алгоритма сортировки от упорядоченности массива (худший и лучший случаи)

Сводная таблица результатов при сортировке массива с упорядоченными значениями

n	T	f(C+M)	Сф+Мф
100	2	10000	100
1000	2	1000000	1000
10000	3	100000000	10000
100000	15	10000000000	100000
1000000	1487	1000000000000	1000000

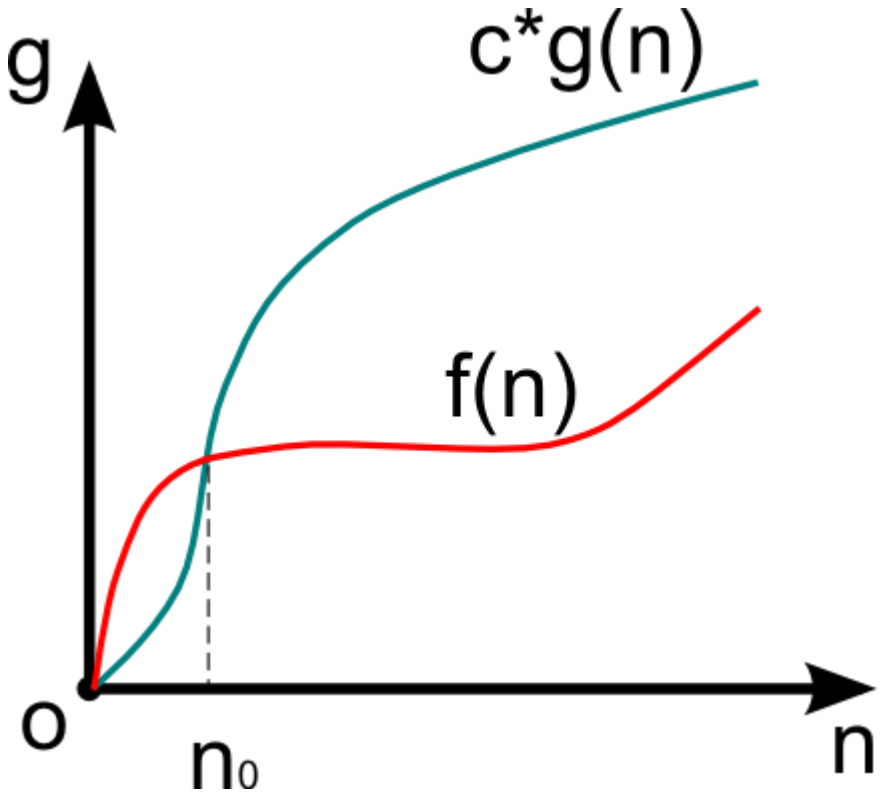
Сводная таблица результатов при сортировке массива с убывающими значениями

n	T	f(C+M)	Сф+Мф
100	1	10000	2550
1000	2	1000000	250500
10000	3	100000000	25005000
100000	19	10000000000	2500050000
1000000	1710	1000000000000	250000500000



Определить емкостную сложность алгоритма от n:

Виды асимптотических оценок
O – оценка для худшего случая
Рассмотрим сложность $f(n) > 0$, функцию того же порядка $g(n) > 0$, размер входа $n > 0$.
Если $f(n) = O(g(n))$ и существуют константы $c > 0$, $n_0 > 0$, то
$0 < f(n) < c*g(n)$,
для $n > n_0$.

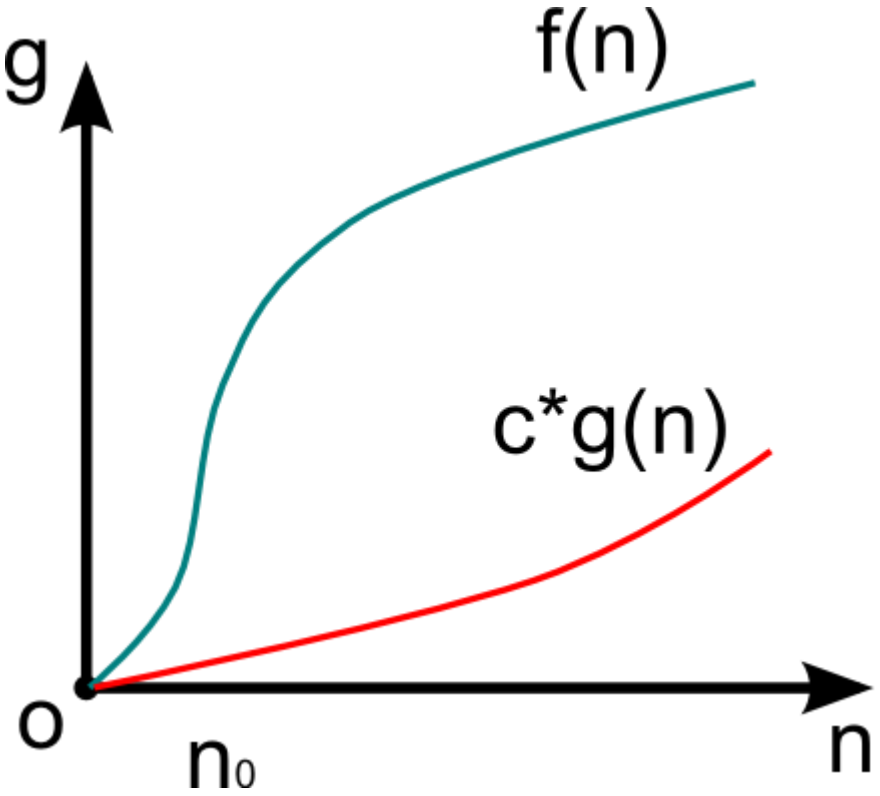


Функция g(n) в данном случае асимптотически-точная оценка f(n). Если f(n) – функция сложности алгоритма, то порядок сложности определяется как f(n) – O(g(n)).

Данное выражение определяет класс функций, которые растут не быстрее, чем g(n) с точностью до константного множителя.

Ω – оценка для лучшего случая

Определение схоже с определением оценки для худшего случая, однако
$f(n) = \Omega(g(n))$, если
$0 < c*g(n) < f(n)$



$\Omega(g(n))$ определяет класс функций, которые растут не медленнее, чем функция $g(n)$ с точностью до константного множителя.

Θ – оценка для среднего случая
Стоит лишь упомянуть, что в данном случае функция $f(n)$ при $n > n_0$ всюду находится между $c_1*g(n)$ и $c_2*g(n)$, где c – константный множитель.
Например, при $f(n) = n^2 + n$; $g(n) = n^2$.

При наличии в массиве некоторой последовательности, сортировка завершает свою работу намного быстрее.

2.3 Отчет по заданию 3 .Оценка эффективности алгоритмов простых сортировок

Сортировка Выбором

Сводная таблица результатов при сортировке массива со случайными значениями

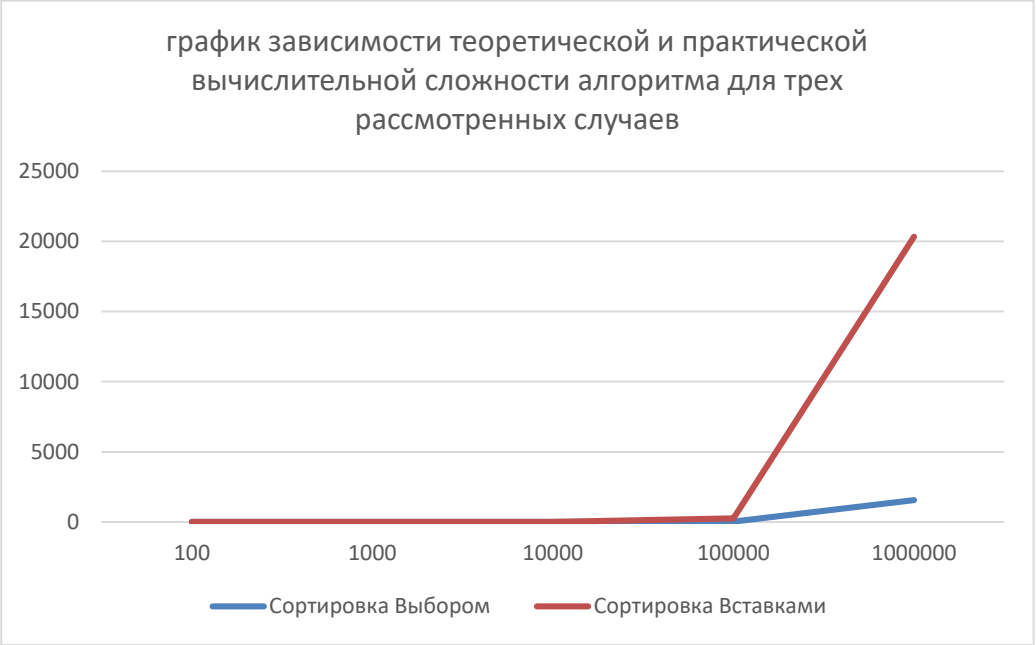
n	T	f(C+M)	Сф+Мф
100	3	10000	425
1000	3	1000000	5195
10000	6	100000000	51575
100000	28	10000000000	519267
1000000	1554	1000000000000	5186458

Код программы:

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main()
{
    setlocale(LC_ALL, "ru");
    srand(time(0));
    int r, sr = 0, per=0;
    cin >> r;
    int* x = new int[r];
    for (int i = 0; i < r; i++) {
        x[i] = rand() % 100;
    }
    cout << "Сортировка выбором" << endl;
    int max = 0, k, t;
    for (int i = 0; i < r; i++) {
        t = i;
        k = x[i];
        for (int j = i + 1; j < r; j++)
            if (x[j] < k) {
                t = j;
                k = x[j];
                sr++;
            }
        per++;
        x[t] = x[i];
        x[i] = k;
    }
    cout << "t = " << clock()/CLOCKS_PER_SEC << " ";
    cout << "per = " << per << " ";
    cout << "sr = " << sr << " ";
    return 0;
}
```



Эффективность алгоритма — это свойство алгоритма, которое связано с вычислительными ресурсами, используемыми алгоритмом. Алгоритм должен быть проанализирован с целью определения необходимых алгоритму ресурсов.

Сортировка выбором завершает работу приблизительно за равные промежутки времени при равных по размеру массивах, независимо от значений.

Исходя из практических результатов трёх сортировок на трёх разных массивах, можно сказать, что на массивах малых размеров алгоритмы заканчивают свою работу за примерно равные промежутки времени, но на значительно больших по размерности массивах сортировка выбором быстрее даёт результат.

3 Выводы

Вопросы по теме

1. Какие сортировки называют простыми?

   Простые сортировки

   К *простым внутренним сортировкам* относят методы, сложность которых пропорциональна квадрату размерности входных данных. Иными словами, при *сортировке массива*, состоящего из N *компонент*, такие алгоритмы будут выполнять $C*N^2$ действий, где C - некоторая константа.

2. Что означает понятие «внутренняя сортировка»?

   Внутренняя сортировка (англ. internal sort) — разновидность алгоритмов сортировки или их реализаций, при которой объема оперативной памяти достаточно для помещения в неё сортируемого массива данных с произвольным доступом к любой ячейке и, собственно, для выполнения

алгоритма.

3. Какие операции считаются основными при оценке сложности алгоритма сортировки?

- Операции, непосредственно влияющие на сложность алгоритма; операции, составляющие «накладные» расходы при выполнении алгоритма (например, выделение памяти для хранения промежуточных данных). Непосредственный подсчет или оценка количества выполняемых операций позволяет оценить T(n). Для оценки порядка сложности можно использовать анализ программного кода, реализующего алгоритм.

4. Какие характеристики сложности алгоритма используются при оценке эффективности алгоритма?

Сложность алгоритма – количественная характеристика, которая говорит о том, сколько времени он работает (временна́я сложность) либо о том, какой объем памяти требуется для его работы (емкостная сложность). Емкостная сложность алгоритма определяется числом ячеек памяти, используемых в процессе его выполнения.

5. Какая вычислительная и емкостная сложность алгоритма: Простого обмена, Простой вставки, Простого выбора?

Емкостная сложность — асимптотическая оценка числа одновременно существующих скалярных величин при выполнении алгоритма на входных данных длиною n. Структурная сложность — характеристика количества управляющих структур в алгоритме и специфики их взаиморасположения.

6. Какую роль в сортировке играет условие Айверсона?

Условие Айверсона: если в ходе сортировки при сравнении элементов не было сделано ни одной перестановки, то множество считается упорядоченным (условие Айверсона выполняется только при шаге d= 1).

7. Примените условие Айверсона к сортировке Простого обмена, заполните столбцы сводной таблицы для этого алгоритма с применением условия Айверсона. Сравните результаты с данными для сортировки Простого обмена без условия Айверсона. Сформулируйте выводы.

Условия Айверсона

Помоему всем понятно, что на каком-то этапе, может даже на начальном, массив окажется полностью отсортированным, и для того, чтобы зря не тратить время на сортировку существует условие Айверсона.

Условие:Сортировку можно прекратить досрочно, если на каком-то этапе в ходе сравнения не будет сделано ни одной перестановки.

8. Определите, каким алгоритмом, рассмотренным в этом задании, сортировался массив. Дан массив 5 6 1 2 3.Шаги выполнения сортировки: 1) 1 5 6 2 3 2) 1 2 5 6 3 3) 1 2 3 5 6

1. Простого обмена

2 .Простой вставки

3 .Простого выбора

9. Определите вычислительную теоретическую сложность алгоритма сортировки, рассмотренную в вопросе 8.

Поэтому традиционной мерой трудоемкости сортировок являются средние количества необходимых сравнений ключей (С) и число пересылок или перестановок элементов (Р), рассматриваемые как функции числа *n* сортируемых элементов. Хорошие алгоритмы сортировки требуют порядка $n \bullet {}^{\wedge}{}_{2}(я)$ сравнений, а простые — порядка n^2 сравнений и эффективны только при малых значениях длины входа.