Эмпирический анализ алгоритмов сортировки.

1. Постановка задачи.

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

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

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

1.3 Построить график зависимости времени выполнения программы от размера массива.

1.4 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. 1.5 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.

1.6 Определить емкостную сложность алгоритма от n.

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

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

2.1 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.

Задание 3. Оценить эффективность алгоритмов простых сортировок.

3.1 Выполнить разработку алгоритма и программную реализацию Selection Sort.

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

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

3.4 Определить емкостную сложность алгоритма от n.

2. Выполнение

Задание 1.

1.1 Ход выполнения работы задания 1 будет продемонстрирован с помощью алгоритма сортировки пузырьком (Bubble sort) (рис. 1).

```
void BubbleSort(std::vector <int>& values) //Сортировка пузырьком
{
    int if_count = 0;
    int swap_count = 0;
    for (int i = 0; i + 1 < values.size(); ++i)
    {
        for (int j = 0; j + 1 < values.size() - i; ++j)
        {
            if_count++;
            if (values[j + i] < values[j])
            {
                swap_count++;
                std::swap(values[j], values[j + 1]);
            }
        }
    }
    std::cout << "Количество сравнений: " << if_count << "\nКоличество смещений: " << swap_count << '\n';
}
```

Рисунок 1. Реализация сортировки пузырьком.

Заполнение контейнера осуществляется с помощью генератора псевдослучайных чисел rand() (рис. 2).

```
void VectorFill(std::vector <int>& values) //Функция заполнения вектора
{
    for (int i = 0; i < values.size(); i++)
    {
        values[i] = rand();
    }
}
```

Рисунок 2. Функция заполнения вектора.

```cpp
int main()
{
    srand(time(NULL));
    setlocale(LC_ALL, "ru");
    std::vector <int> mas(100000);
    std::cout << "Количество элементов" << mas.size() << '\n';
    VectorFill(mas);
    unsigned int start - clock();
    BubbleSort(mas);
    unsigned int end = clock();
    std::cout << (end - start) / (double)CLOCKS_PER_SEC << '\n';
    return 0;
}
```

Рисунок 3. Обработка данных в основной функции main().

1.2 Выполнение алгоритма с разным количеством случайных элементов.

Рисунок 4. Выполнение алгоритма с 100 элементами.

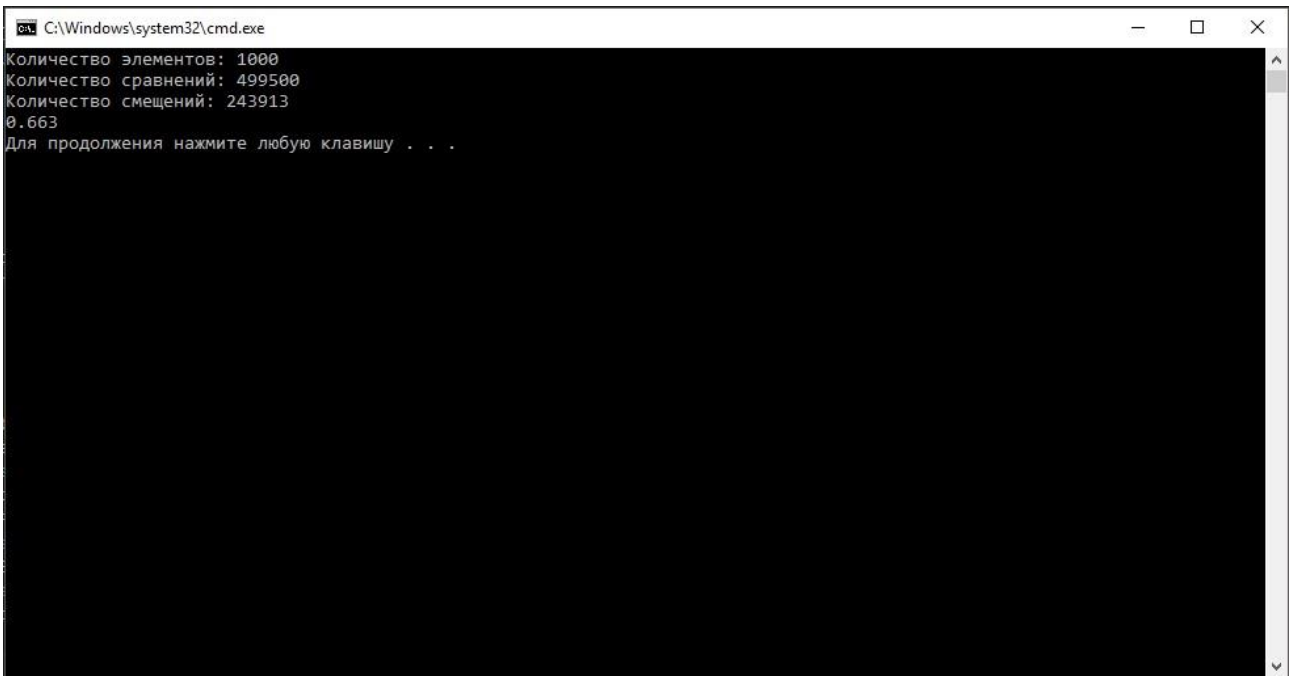


Рисунок 5. Выполнение алгоритма с 1 000 элементами.

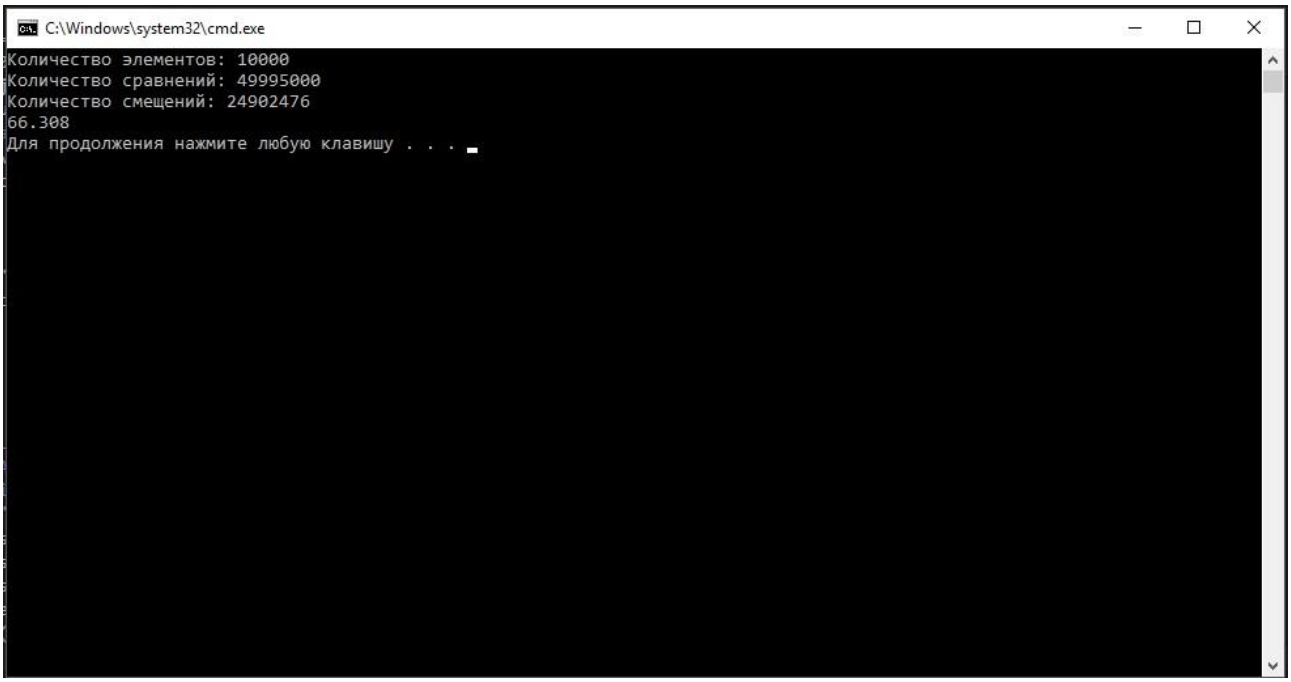


Рисунок 6. Выполнение алгоритма с 10 000 элементами.

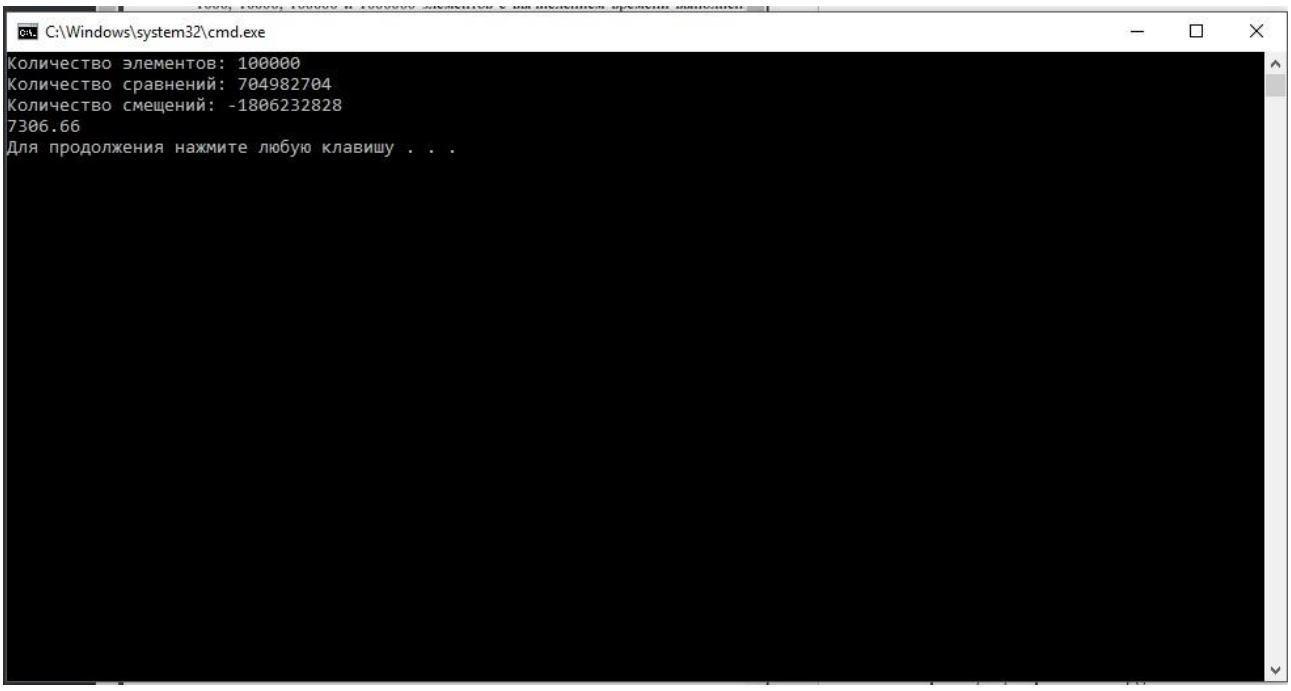


Рисунок 7. Выполнение алгоритма с 100 000 элементами.

При получении количества смещений 100 000 элементов произошел StackOverFlow. По моим подсчетам количество смещений равно 2488734467.

1.3 График зависимости времени выполнения от количества элементов вектора продемонстрирован ниже на рисунке 8.

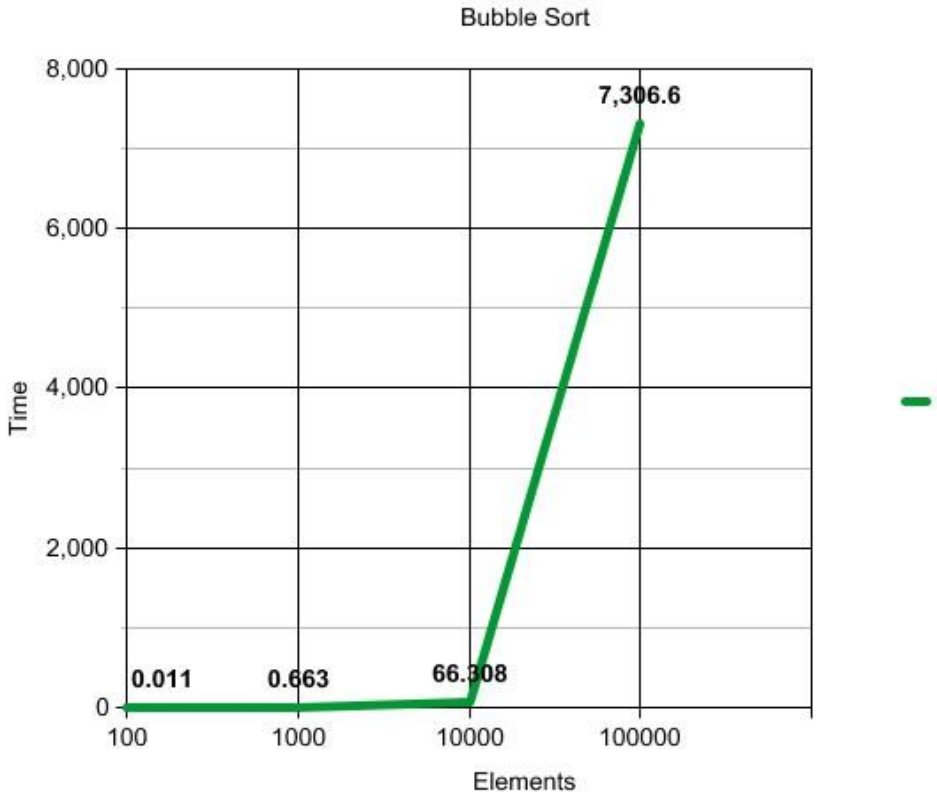


Рисунок 8. График зависимости времени выполнения от количества.

1.4 В программе предусмотрен подсчет количества операций сравнения Сф и количество перемещений Мф. Таблица 1. Сводная таблица.

n	T(n) в сек.	Тт = O(f(n))	Тп = Сф + Мф
100	0.011	10000	7544
1000	0.663	100000	743413
10000	66.308	1000000	74897476
100000	7306.6	10000000	3193717171

1.5 Построение графиков зависимости Теоретической Тт и практической Тт вычислительной сложности алгоритмов от размера контейнера. Теоретическое время выполнение высчитывается по формуле O(n^2).

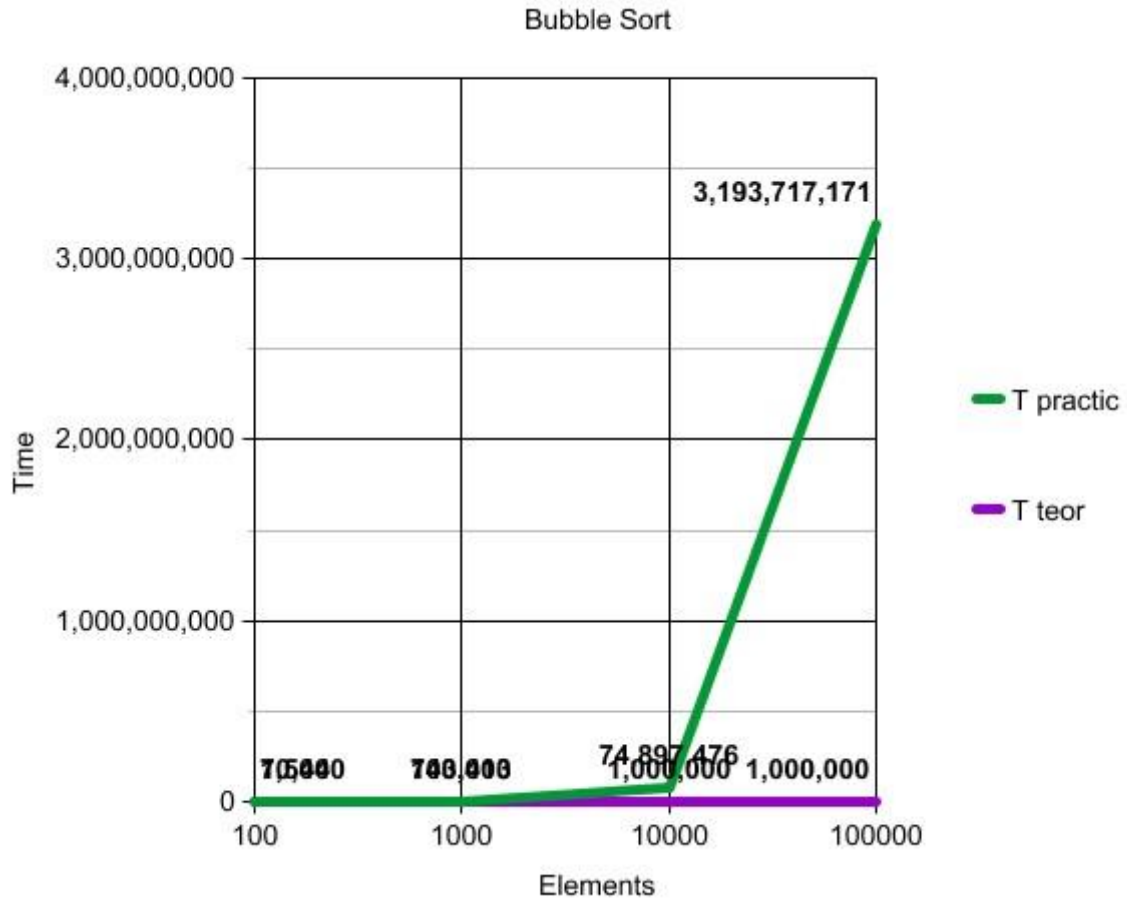


Рисунок 9. График

1.6 Емкостная сложность алгоритма (Space Complexity) – O(1). В функции не используются дополнительные переменные.

1.7 Мы можем наблюдать стремительный рост временных затрат при увеличении количества элементов в массиве. Алгоритм имеет сложность в

среднем Θ = n^2, что является плохим показателем. Следует вывод, что сортировку пузырьком следует использовать при гарантированном низком количестве элементов.

Задание 2.

Для экономии времени беру количество элементов не выше 10 000. Как известно, скорость выполнения в лучшем случае равна Ω(n), в худшем – O(n).

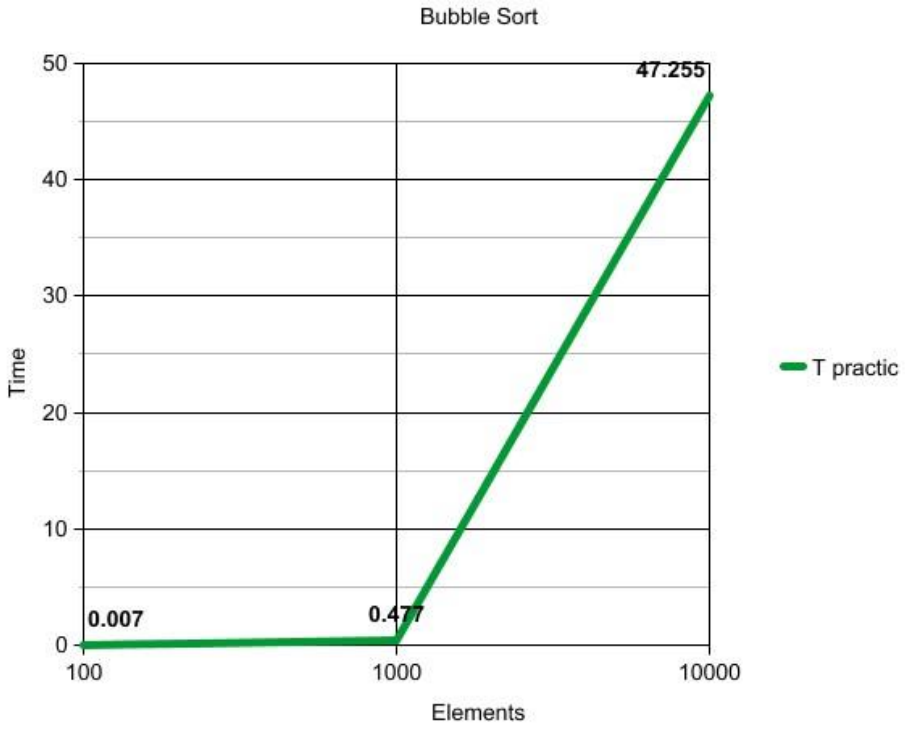
а) Контейнер заполнен числами от 1 до n с помощью функции FillVectorA();

```
void FillVectorA(std::vector <int>& values) //FillVectorA
{
    for (int i = 0; i < values.size(); i++)
    {
        values[i] = i;
    }
}
```

Рисунок 10. FillVectorA.

Таблица 2. Расчет в лучшем случае.

n	T(n) в сек.	Тт = O(f(n))	Тп = Сф + Мф
100	0.007	100	4950
1000	0.477	1000	499500
10000	47.255	10000	49995000



б) Контейнер заполнен числами от n до 0 с помощью функции FillVectorB();

```
void FillVectorB(std::vector <int>& values) //FillVectorB
{
    int j = 0;
    for (int i = values.size(); i > 0; i++)
    {
        values[j] = i;
        j++;
    }
}
```

Рисунок 10. FillVectorB.

Таблица 2. Расчет в худшем случае.

n	T(n) в сек.	Тт = O(f(n))	Тп = Сф + Мф
100	0.016	10000	9900
1000	0.871	100000	999000
10000	91.617	1000000	99990000

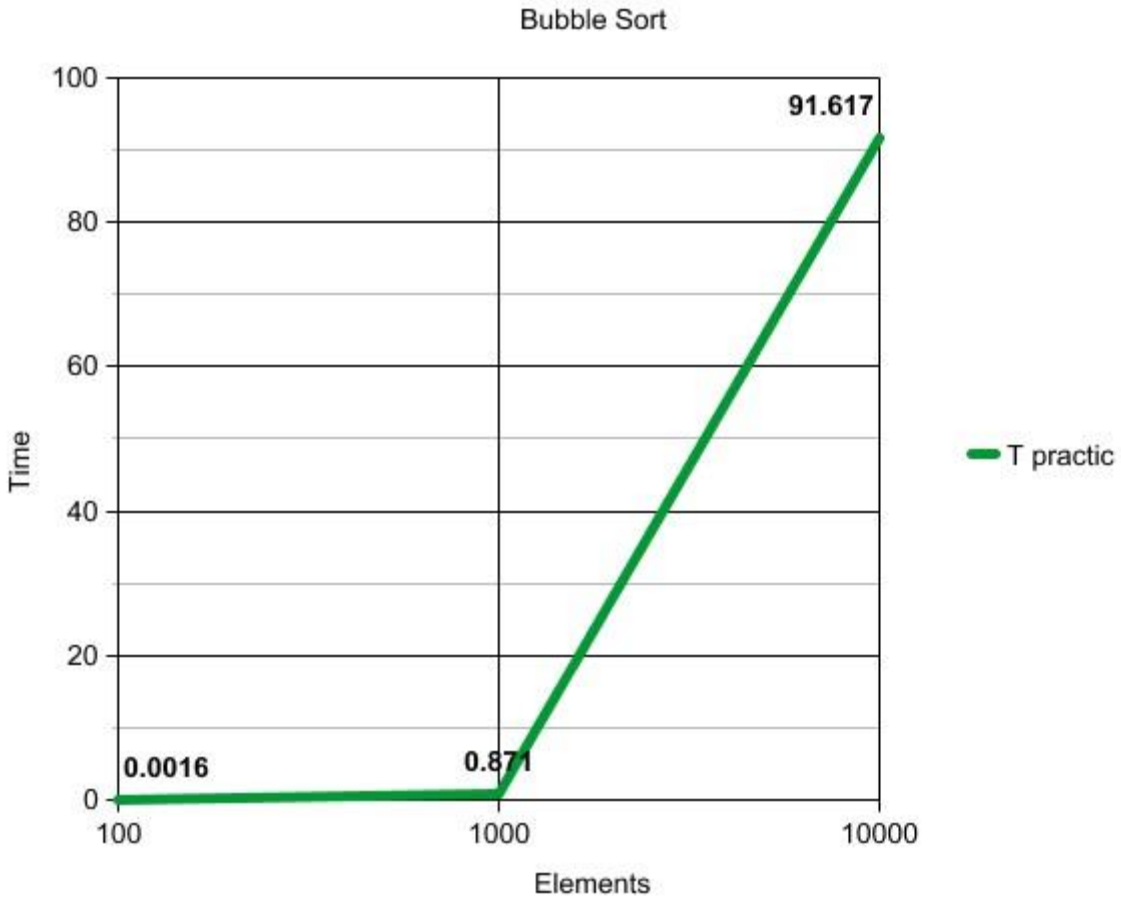


Рисунок 11. График зависимости времени от количества элементов.

В) Как мы можем заметить, сортировка пузырьком зависит от

сложности заполнения контейнера. При чем, в худшем случае время выполнения увеличится в 1.5 раза при 10 000 элементов.

Задание 3.

3.1 Разработка алгоритма сортировки выбором.

```
void SelectiomSort(int* A, int n) //Сортировка выбором
{
    int if_count = 0;
    int swap_count = 0;
    int count;
    int key;
    for (i = 0; i < n; i++)
    {
        count = A[i];
        key = i;
        for (j = i + 1; j < n; j++)
        {
            if_count++;
            if (A[j] < A[key])
            {
                swap_count++;
                key = j;
            }
        }
        if_count++;
        if (key != i)
        {
            swap_count++;
            A[i] = A[key];
            A[key] = count;
        }
    }
    std::cout << "Количество сравнений: " << if_count << "\nКоличество смещений: " << swap_count << '\n';
}
```

Рисунок 12. Алгоритм сортировки выбором.

3.2 Ниже представлена сводная таблица по алгоритму сортировки выбором. Для упрощения тестирования было взято количество элементов не выше 10 000.

Таблица 4. Сводные данные по алгоритму

n	**T(n) в сек.**	**Тт = O(f(n))**	**Тп = Сф + Мф**
100	0.006	10000	5489
1000	0.007	100000	506903
10000	0.237	1000000	50092642
100000	24.146	10000000	706121882

3.3 Графики зависимостей предоставлены ниже на рисунках 13 и 14.

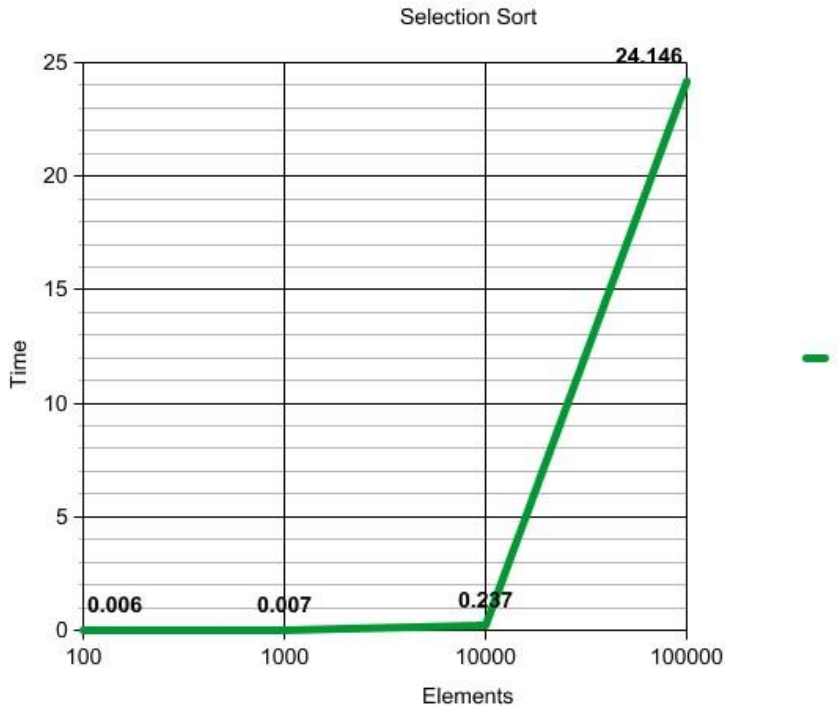


Рисунок 13. График зависимости времени и элементов.

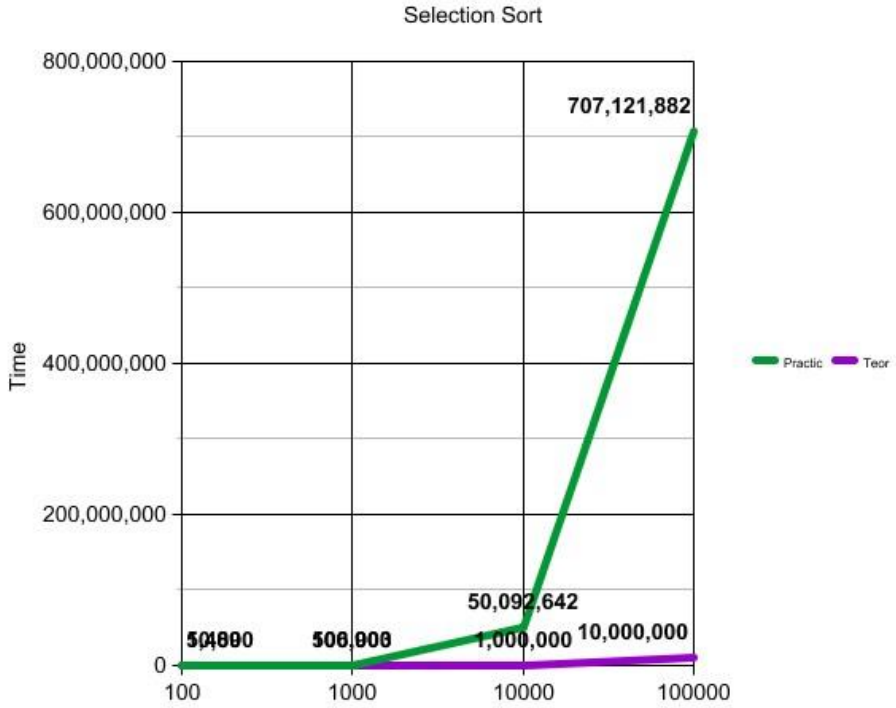


Рисунок 14. График зависимости теоретической вычислительной сложности и практической в.с. от количества элементов.

3.4 Емкостная сложность алгоритма от n – O(n) основной и O(1) вспомогательной. При этом сложность алгоритма оценивается в O(n^2) во всех трех возможных состояниях.

Мы можем заметить, что сортировка выбором во много раз эффективнее сортировки пузырьком. При этом их оценочная сложность равна.