**Московский Авиационный Институт**

(Национальный Исследовательский Университет)

Кафедра вычислительной математики и программирования

**Курсовая работа**

**по курсу «Языки и методы программирования»**

**II семестр**

**Задание №9**

**«Сортировка и поиск»**

Студент: Кондратьев Егор Алексеевич

Группа: М8О-106Б-19

Преподаватель: Дубинин Алексей Владимирович

Москва, 2020

**Содержание**

Оглавление

[Введение 3](#_Toc43965261)

[Сортировка Шелла 4](#_Toc43965262)

[Реализация алгоритма сортировки Шелла 4](#_Toc43965263)

[Бинарный поиск 5](#_Toc43965264)

[Оценка сложности 6](#_Toc43965265)

[Заключение 6](#_Toc43965266)

[Список литературы 6](#_Toc43965267)

# Введение

Сортировки - один из самых известных и необходимых программисту разделов алгоритмов. Различные сортировки подходят для разных задач и условий, некоторые лучше работают на больших данных, некоторые на почти отсортированных. Зачем нужны сортировки? Ответ прост - для удобства работы с данными. С упорядоченной коллекцией работать гораздо проще, а некоторые алгоритмы, как бинарный поиск, работают только на отсортированных структурах данных. Но недостаточно просто отсортировать коллекцию. Нужно сделать это быстро и эффективно. Отсюда и вытекают основные характеристики сортировки - сложность по времени и по памяти.

# Сортировка Шелла

Этот метод сортировки Д. Шелл предложил в 1959 г. Он использует минимум памяти и показывает высокие скорости при сортировке. По сути в методе Шелла применяются сравнения и перестановки элементов аналогичные [методу вставок](https://vscode.ru/prog-lessons/sortirovka-vstavkami-na-si.html), но при этом порядок сравниваемых элементов совершенно другой.

Идея **сортировки методом Шелла** состоит в том, чтобы сортировать элементы отстоящие друг от друга на некотором расстоянии step. Затем сортировка повторяется при меньших значениях step, и в конце процесс сортировки Шелла завершается при step = 1 (а именно обычной сортировкой вставками).

До сих пор продолжает обсуждаться вопрос выбора шага сортировки step. Шелл предложил такую последовательность: N/2, N/4, N/8 …, где N – количество элементов в сортируемом массиве.

# Реализация алгоритма сортировки Шелла

bool compare(Key l, Key r)

{

    if (l.f < r.f)

        return true;

    if (l.f == r.f && l.i8 < r.i8)

        return true;

    return false;

}

void ShellSort(Item \*Vector, int size)

{

    int step, i, j;

    Item tmp;

    for (step = size / 2; step > 0; step /= 2)

        for (i = step; i < size; i++)

            for (j = i - step; j >= 0 && compare(Vector[j].key, Vector[j + step].key); j -= step)

            {

                tmp = Vector[j];

                Vector[j] = Vector[j + step];

                Vector[j + step] = tmp;

            }

}

# Бинарный поиск

Бинарный поиск производится в упорядоченном массиве.

При бинарном поиске искомый ключ сравнивается с ключом среднего элемента в массиве. Если они равны, то поиск успешен. В противном случае поиск осуществляется аналогично в левой или правой частях массива.

Алгоритм может быть определен в рекурсивной и нерекурсивной формах.

Бинарный поиск также называют поиском методом деления отрезка пополам или дихотомии.

Количество шагов поиска определится как

log2n↑,

где n-количество элементов,  
↑ — округление в большую сторону до ближайшего целого числа.

На каждом шаге осуществляется поиск середины отрезка по формуле

m= (l + r)/2

Если искомый элемент равен элементу с индексом m, поиск завершается.

В случае если искомый элемент меньше элемента с индексом m, на место m перемещается правая граница рассматриваемого отрезка, в противном случае — левая граница.

bool compare(Key l, Key r)

{

    if (l.f < r.f)

        return true;

    if (l.f == r.f && l.i8 < r.i8)

        return true;

    return false;

}

int bin\_search(Item \*Vector, int n, Key k)

{

    int l = -1, r = n;

    while (r - l > 1)

    {

        int m = (l + r) / 2;

        if (compare(Vector[m].key, k))

            l = m;

        else

            r = m;

    }

    if (k.f == Vector[r].key.f && k.i8 == Vector[r].key.i8)

        return r;

    return -1;

}

# Оценка сложности

n- количество элементов во входном массиве

Алгоритм сортировки в среднем работает за 𝑂(𝑛 ⋅ log 𝑛)

В худшем случае деградирует до 𝑂(𝑛2

По памяти:

дополнительная память не требуется для массива, т. к. все замены происходят “на месте” без заведения дополнительных массивов.

Бинарный поиск работает за 𝑂(log 𝑛)

# Заключение

Сортировки является одним из самых необходимых и важных алгоритмов в программировании. Для их эффективного использования необходимо знать контекст применения каждой из сортировок, их особенности, затраты и способы модификации. Знание бинарного поиска позволит быстро и эффективно искать необходимые значения в отсортированной коллекции.

# Список литературы

1. ru.wikipedia.org/wiki/Сортировка\_Шелла