Практическая работа №2

Тема: «Алгоритмы сортировки».

Цель работы: изучить алгоритмы сортировки.

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

Свойства и типы:

– Устойчивость — устойчивая сортировка не меняет взаимного расположения элементов с одинаковыми ключами.

– Естественность поведения — эффективность метода при обработке уже упорядоченных или частично упорядоченных данных. Алгоритм ведёт себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.

– Использование операции сравнения. Алгоритмы, использующие для сортировки сравнение элементов между собой, называются основанными на сравнениях. Минимальная трудоемкость худшего случая для этих алгоритмов составляет $O(n \log n)$, но они отличаются гибкостью применения. Для специальных случаев существуют более эффективные алгоритмы.

Ещё одним важным свойством алгоритма является его сфера применения. Здесь основных типов упорядочения два:

– Внутренняя сортировка оперирует массивами, целиком помещающимися в оперативной памяти с произвольным доступом к любой ячейке. Данные обычно упорядочиваются на том же месте без дополнительных затрат.

– В современных архитектурах персональных компьютеров широко применяется подкачка и кэширование памяти. Алгоритм сортировки должен хорошо сочетаться с применяемыми алгоритмами кэширования и подкачки.

– Внешняя сортировка оперирует запоминающими устройствами большого объёма, но не с произвольным доступом, а последовательным, то есть в данный момент виден только один элемент, а затраты на перемотку по сравнению с памятью неоправданно велики. Это накладывает некоторые дополнительные ограничения на алгоритм и приводит к специальным методам упорядочения, обычно использующим дополнительное дисковое пространство. Кроме того, доступ к данным во внешней памяти производится намного медленнее, чем операции с оперативной памятью.

– Доступ к носителю осуществляется последовательным образом: в каждый момент времени можно считать или записать только элемент, следующий за текущим.

– Объём данных не позволяет им разместиться в ОЗУ.

Также алгоритмы классифицируются по:

– потребности в дополнительной памяти или её отсутствию

– потребности в знаниях о структуре данных, выходящих за рамки операции сравнения, или отсутствию таковой

Сортировка простыми обменами, сортировка пузырьком — простой алгоритм сортировки. Для понимания и реализации этот алгоритм простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: $O(n^2)$.

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 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.

Исходный код на языке Python представлен на листинге 1.

Листинг 1. Пузырьковая сортировка на языке Python.

```
def bubble_sort(array):
    for i in range(len(array) - 1):
        for j in range(len(array) - i - 1):
            if array[j] > array[j + 1]:
                array[j], array[j + 1] = array[j + 1],
array[j]
    return array
```

Зависимость времени сортировки от длины массива для этой и других сортировок на рисунке показана следующим образом: красной линией показано максимальное время сортировки из 50 итераций, синей – минимальное, а зеленой среднее время сортировки за 50 итераций.

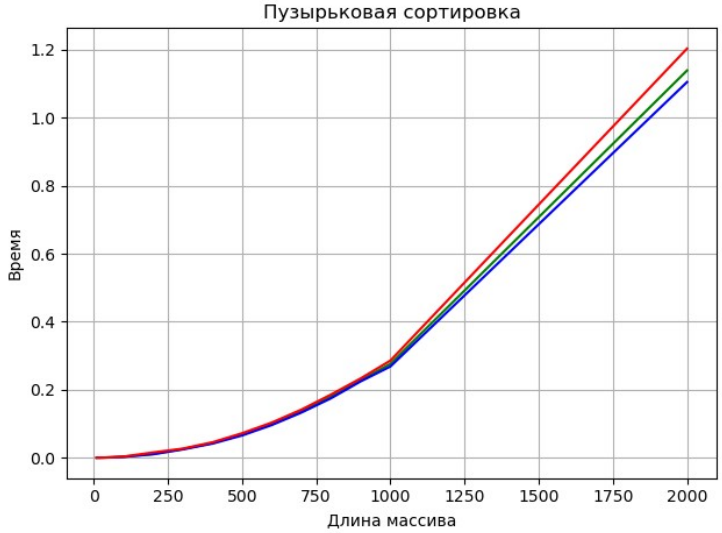

Рисунок 1. Результаты времени сортировки для пузырьковой сортировки.

Сортировка вставками — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Вычислительная сложность — $O(n^2)$.

На вход алгоритма подаётся последовательность n nчисел:$a_1, a_2, \ldots, a_n$. Сортируемые числа также называют ключами. Входная последовательность на практике представляется в виде массива с элементами. На выходе

алгоритм должен вернуть перестановку исходной последовательности $a_1^{'}, a_2^{'}, \ldots, a_n^{'}$, чтобы выполнялось следующее соотношение $a_1^{'} \leq a_2^{'} \leq \ldots \leq a_n^{'}$.

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей.

Исходный код на языке Python представлен на листинге 2.

Листинг 2. Сортировка вставками на языке Python.

```
def insertion_sort(array):
    for i in range(len(array)):
        j = i - 1
        key = array[i]
        while array[j] > key and j >= 0:
            array[j + 1] = array[j]
            j -= 1
        array[j + 1] = key
    return array
```

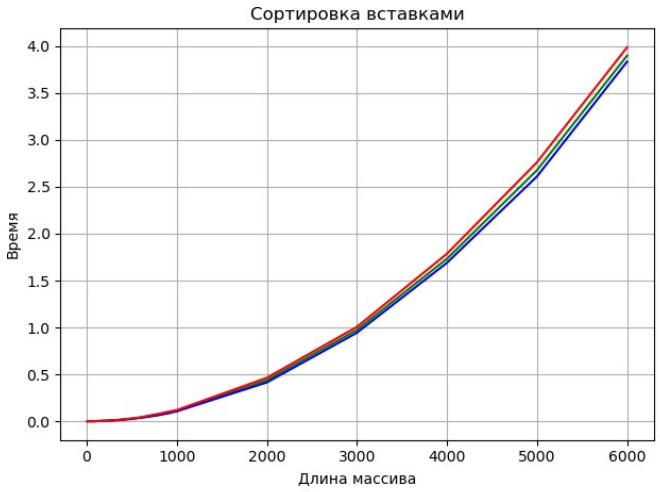

Рисунок 2. Результаты времени сортировки для сортировки вставками.

Сортировка слиянием — алгоритм сортировки, который упорядочивает списки в определённом порядке. Эта сортировка — хороший пример

использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения 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.

А) Соединение двух упорядоченных массивов в один.

Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по возрастанию подмассива. Тогда:

Б) 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.

В) «Прицепление» остатка.

Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

Исходный код на языке Python представлен на листинге 3.

Листинг 3. Сортировка слиянием на языке Python.

```
def merge_sort(array):
    if len(array) > 1:
        mid = len(array) // 2
        L = array[:mid]
        R = array[mid:]
        merge_sort(L)
        merge_sort(R)
        i = j = k = 0
```

```
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            array[k] = R[j]
            j += 1
            k += 1
    return array
```

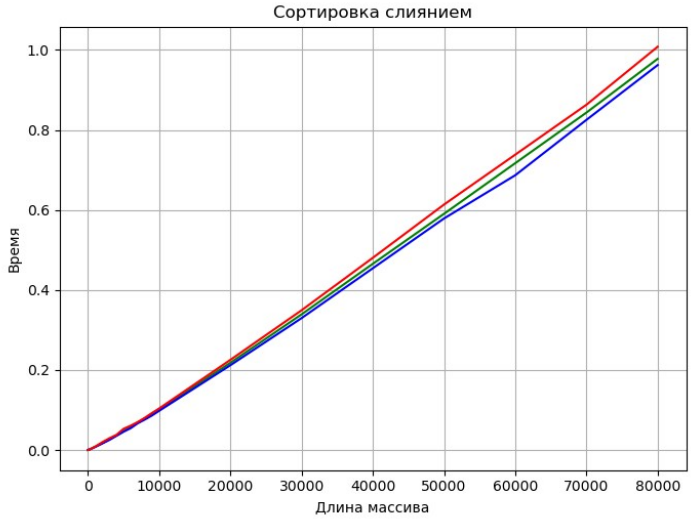

Рисунок 3. Результаты времени сортировки для сортировки слиянием.

Сортировка выбором — алгоритм сортировки. Может быть как устойчивый, так и неустойчивый. На массиве из n элементов имеет время выполнения в худшем, среднем и лучшем случае $\Theta(n^2)$, предполагая что сравнения делаются за постоянное время.

Шаги алгоритма:

1. Находим номер минимального значения в текущем списке.
2. Производим обмен этого значения со значением первой неотсортированной позиции.
3. Сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы.

Исходный код на языке Python представлен на листинге 4.

Листинг 4. Сортировка выбором на языке Python.

```
def selection_sort(array):
    for i in range(len(array) - 1):
        m = i
        j = i + 1
        while j < len(array):
            if array[j] < array[m]:
                m = j
            j += 1
        array[i], array[m] = array[m], array[i]
    return array
```

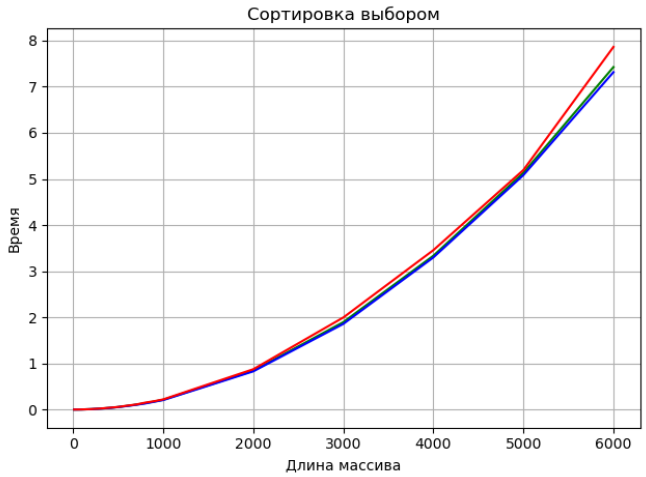

Рисунок 4. Результаты времени сортировки для сортировки выбором.

Вывод: при выполнении данной практической работы были изучены алгоритмы сортировки.