НАУЧНЫЙ ЖУРНАЛ
НАУКА И МИРОВОЗЗРЕНИЕ

СРАВНЕНИЕ АЛГОРИТМОВ СОРТИРОВКИ: АНАЛИЗ ЭФФЕКТИВНОСТИ

Гылычдурдыева Чынар

преподаватель кафедры Кибербезопасности, Международного университета нефти и газа имени Ягшыгелди Какаева, г. Ашхабад Туркменистан

Ёмудова Джахан

преподаватель кафедры Кибербезопасности, Международного университета нефти и газа имени Ягшыгелди Какаева, г. Ашхабад Туркменистан

Репова Аннагозел

преподаватель кафедры Кибербезопасности, Международного университета нефти и газа имени Ягшыгелди Какаева, г. Ашхабад Туркменистан

Мамметдурдыев Мустафа

студент, Международного университета нефти и газа имени Ягшыгелди Какаева, г. Ашхабад Туркменистан

Сортировка является одной из фундаментальных операций в области компьютерных алгоритмов. Она позволяет упорядочить набор данных в заданном порядке и является неотъемлемой частью множества приложений, начиная от базовых операций с данными до сложных алгоритмов машинного обучения и анализа данных.

Существует множество алгоритмов сортировки, каждый из которых имеет свои особенности и характеристики. Одной из важных задач при выборе алгоритма сортировки является анализ и сравнение их эффективности. Понимание производительности алгоритмов сортировки позволяет разработчикам искусно выбирать наиболее подходящий алгоритм для конкретных ситуаций, учитывая ограничения по времени и памяти.

Мы рассмотрим некоторые из наиболее часто используемых методов сортировки, применяя Python как язык программирования для иллюстрации:

1. Сортировка пузырьком (Bubble Sort) — это алгоритм сортировки, который сравнивает два соседних элемента массива и меняет их местами, пока они не будут в нужном порядке, имеет квадратичную сложность $O(n^2)$ в среднем и худшем случаях и $O(n)$ в лучшем случае, когда массив уже отсортирован [1]. Обычно используется для сортировки небольших массивов.
2. Алгоритм сортировки пузырьком работает следующим образом:

a. Начинаем с первого элемента массива;
b. Сравниваем первый элемент с вторым. Если первый элемент больше второго, меняем их местами;
c. Переходим к следующему элементу массива. Снова сравниваем его со следующим и меняем местами, если он больше;
d. Продолжаем этот процесс до конца массива. Таким образом, самый большой элемент перемещается на последнее место в первой итерации;
e. Повторяем те же шаги для оставшихся элементов массива, исключая уже отсортированные. После каждой итерации очередной наибольший элемент становится на свое место;
f. Алгоритм заканчивается, когда все элементы массива отсортированы.

Пример сортировки пузырьком на языке Python представлен на рисунке 1

```python
def bubbleSort(array) -> list:
    length = len(array)
    for i in range(length):
        for j in range(length - i - 1):
            if array[j] > array[j + 1]:
                temp = array[j]
                array[j] = array[j + 1]
                array[j + 1] = temp
    return array
```

Рисунок 1. Сортировка пузырьком на языке Python

2. Сортировка вставками (Insertion Sort) — это алгоритм сортировки, который постепенно строит отсортированную последовательность, проходя по списку и вставляя каждый элемент в правильное место. Он обладает линейной сложностью O(n) в лучшем случае (когда список уже отсортирован) и квадратичной сложностью $O(n^2)$ в среднем и худшем случаях [2].

a. Алгоритм сортировки вставками работает следующим образом;
b. Считаем первый элемент массива отсортированным;
c. Берем следующий элемент и сохраняем его в отдельной переменной (назовем ее ключ);
d. Сравниваем ключ с элементами в отсортированной части массива, начиная с последнего. Если ключ меньше текущего элемента, то сдвигаем текущий элемент на одну позицию вправо. Повторяем этот процесс, пока не найдем элемент, который меньше или равен ключу, или не дойдем до начала массива;

e. Вставляем ключ на найденное место или в начало массива, если он меньше всех элементов в отсортированной части;
f. Повторяем шаги 2-4 для оставшихся элементов в неотсортированной части массива, пока не отсортируем весь массив.

Пример сортировки вставками на языке Python представлен на рисунке 2

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

Рисунок 2. Сортировка вставками на языке Python

3. Сортировка выбором (Selection Sort): этот алгоритм сортировки на каждом шаге находит минимальный элемент из неотсортированной части списка и меняет его местами с первым неотсортированным элементом. Сложность алгоритма сортировки выбором составляет $O(n^2)$ в лучшем, среднем и худшем случаях, так как на каждой итерации требуется пройти по всему неотсортированному массиву [1]. Этот алгоритм не подходит для больших объемов данных, так как он работает очень медленно. Однако он прост в понимании и реализации и не требует дополнительной памяти.

Алгоритм сортировки выбором работает следующим образом:

a. Считаем первый элемент массива наименьшим;
b. Сравниваем наименьший элемент с остальными элементами массива. Если находим элемент меньше наименьшего, запоминаем его как новый наименьший;
c. После прохода по всему массиву меняем местами наименьший элемент с первым элементом неотсортированной части массива. Таким образом, наименьший элемент оказывается на первом месте отсортированной части массива;
d. Повторяем шаги 1-3 для оставшихся элементов неотсортированной части массива, пока не отсортируем весь массив.

Пример сортировки выбором на языке Python представлен на рисунке 3

```
def selectionSort(array) -> list:
    length = len(array)
    for i in range(0, length - 1):
        minIndex = i
        for j in range(i + 1, length):
            if array[j] < array[minIndex]:
                minIndex = j
        temp = array[i]
        array[i] = array[minIndex]
        array[minIndex] = temp
    return array
```

Рисунок 3. Сортировка выбором на языке Python

4. Сортировка слиянием (Merge Sort): это эффективный алгоритм сортировки, который основан на принципе "разделяй и властвуй". Он разбивает список на две половины, рекурсивно сортирует каждую половину, а затем объединяет их, чтобы получить отсортированный список. Сортировка слиянием имеет сложность O(n log n) во всех случаях [3]. Этот алгоритм эффективен для больших объемов данных, но требует дополнительной памяти для хранения временных массивов.

Алгоритм сортировки слиянием работает следующим образом:

a. Если массив состоит из одного элемента, то он уже отсортирован и возвращается как результат;
b. Если массив состоит из более чем одного элемента, то он делится на две равные или почти равные части;
c. Каждая часть сортируется рекурсивно с помощью сортировки слиянием;
d. Две отсортированные части объединяются в один отсортированный массив с помощью процедуры слияния, которая сравнивает элементы двух частей и копирует меньший элемент в результирующий массив, пока обе части не будут исчерпаны.

Пример сортировки слиянием на языке Python представлен на рисунке 4

```
def mergeSort(array) -> list:
    if len(array) > 1:
        mid = len(array)//2
        left = array[:mid]
        right = array[mid:]
        mergeSort(left)
        mergeSort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] <= right[j]:
                array[k] = left[i]
                i += 1
            else:
                array[k] = right[j]
                j += 1
            k += 1
        while i < len(left):
            array[k] = left[i]
            i += 1
            k += 1
        while j < len(right):
            array[k] = right[j]
            j += 1
            k += 1
    return array
```

Рисунок 4. Сортировка слиянием на языке Python

5. Быстрая сортировка (Quick Sort): это один из самых быстрых алгоритмов сортировки в среднем случае. Он также использует принцип "разделяй и властвуй", выбирая опорный элемент и разделяя список на две части: элементы, меньшие опорного, и элементы, большие опорного. Затем он рекурсивно сортирует каждую часть. Быстрая сортировка в среднем случае имеет сложность O(n log n), но в худшем случае может иметь квадратичную сложность $O(n^2)$ [4].

Быстрая сортировка работает следующим образом:

a. Если массив состоит из одного или нуля элементов, то он уже отсортирован и возвращается как результат;
b. Если массив состоит из более чем одного элемента, то выбирается один элемент в качестве опорного.

Это может быть любой элемент массива, но от выбора опорного зависит эффективность алгоритма. Существуют разные способы выбора опорного элемента, например, первый, последний, средний, случайный или медианный элемент массива;

c. Переставить элементы массива так, чтобы все элементы, меньшие опорного, оказались слева от него, а все элементы, большие или равные опорному, оказались справа от него. Это называется разбиением массива. Существуют разные способы выполнения разбиения, например, схема Ломуто, схема Хоара, схема Тарьяна и другие;
d. Рекурсивно применить быструю сортировку к левой и правой части массива, не включая опорный элемент;
e. Объединить отсортированные левую и правую части массива с опорным элементом в один отсортированный массив.

Пример быстрой сортировки используя схему разбиения Хоара на языке Python представлен на рисунке 5

```python
def quickSort(array, start, end) -> list:
    if end - start > 1:
        part = partition(array, start, end)
        quickSort(array, start, part)
        quickSort(array, part + 1, end)
    return array

def partition(array, start, end):
    pivot = array[start]
    i = start + 1
    j = end - 1
    while True:
        while (i <= j and array[i] <= pivot):
            i = i + 1
        while (i <= j and array[j] >= pivot):
            j = j - 1
        if i <= j:
            temp = array[i]
            array[i] = array[j]
            array[j] = temp
        else:
            temp = array[start]
            array[start] = array[j]
            array[j] = temp
            return j
```

Рисунок 5. Быстрая сортировка используя схему разбиения Хоара на языке Python

Для сравнения различных методов сортировки в Python мы воспользуемся библиотекой time. Эта библиотека позволяет замерять время выполнения кода, что полезно при оценке производительности алгоритмов.

У нас есть список из 441 числа, который мы будем сортировать. Для каждого метода сортировки выполним десять итераций и замерим среднее время выполнения, которое будет представлено в таблице 1.

Таблица 1.

Сравнение различных методов сортировки

Название метода	Среднее время выполнения, секунд
Сортировка пузырьком	0.005481553077697754
Сортировка вставками	0.002293300628662109
Сортировка выбором	0.0025913238525390623
Сортировка слиянием	0.000506448745727539
Быстрая сортировка	0.0002918243408203125

Таким образом, при выборе метода сортировки рекомендуется использовать быструю сортировку или сортировку слиянием для достижения наилучшей производительности.