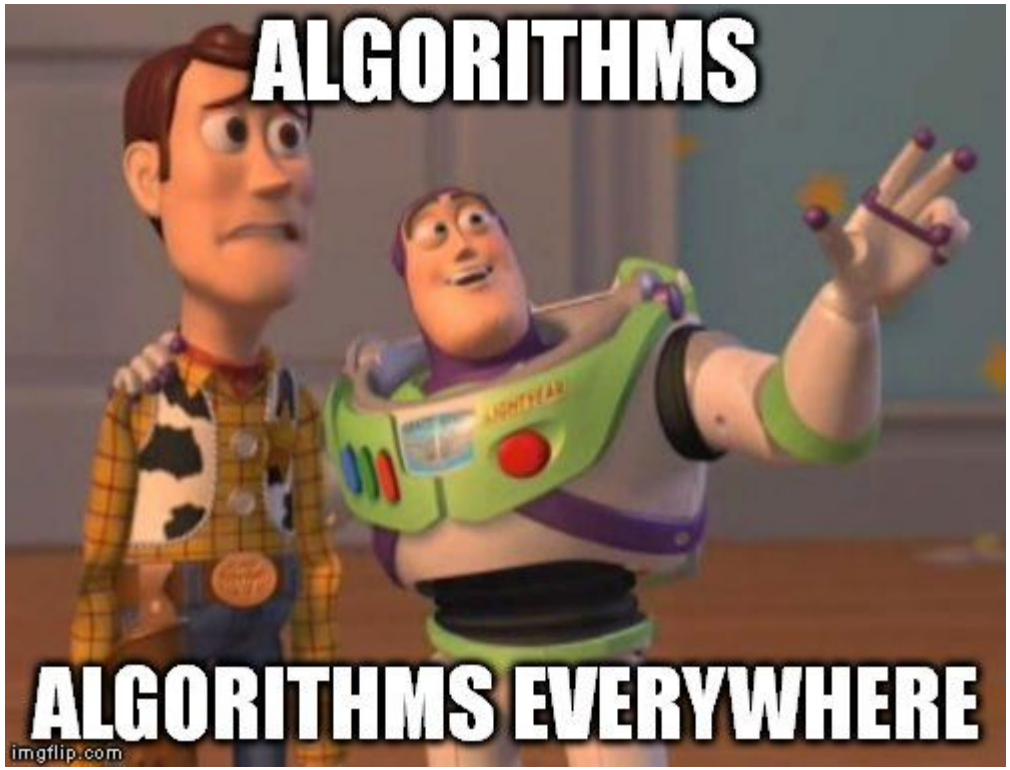
Основы программирования на C++

Лекция 5. Начало алгоритмов. Сортировки

Мем в начале



Трудоемкость: О-большое, затраты на время и память

Алгоритмы бывают разные, и все они различаются по следующим параметрам:

- **Скорость выполнения** – характеризует быстродействие алгоритма.
- **Затраты по памяти** – некоторые алгоритмы требуют ещё переменных или массивов
- **Устойчивость** - устойчивая сортировка не меняет взаимного расположения равных элементов

Порядок роста

Порядок роста описывает то, как сложность алгоритма растет с увеличением размера входных данных. Чаще всего он представлен в виде O-нотации (*от нем. «Ordnung» — порядок*): O(f(x)), где f(x) — формула, выражающая сложность алгоритма. В формуле может присутствовать переменная ***n***, представляющая размер входных данных.

Часто встречающиеся порядки

- **O(1)** – константный. Выполняется каждый раз одинаково. Например, алгоритм обмена значений.
- O(n) – линейный. Прямо зависит от размера входных данных. Например, отображение всех значений в массиве.
- $O(n^2)$ – квадратичный. Часто встречается в алгоритмах сортировок. Лучше не употреблять, так как на большом объеме данных долго будет работать.
- O(n3) – кубический. Это вообще п*&$@ц кромешный. Задумайтесь перед тем, как использовать алгоритм с таким порядком роста.
- O(log n) – логарифмический. Самые лучшие сортировки имеют такой порядок роста или O(n*log n). Используется в поиске в двоичном дереве. Крутая вещь :D

Как измеряем?

При измерении сложности алгоритмов и структур данных мы обычно говорим о двух вещах: количество операций, требуемых для завершения работы (вычислительная сложность), и объем ресурсов, в частности, памяти, который необходим алгоритму (пространственная сложность).

Алгоритм, который выполняется в десять раз быстрее, но использует в десять раз больше места, может вполне подходить для серверной машины с большим объемом памяти. Но на встроенных системах, где количество памяти ограничено, такой алгоритм использовать нельзя.

Операции, количество которых мы будем измерять, включают в себя:

- сравнения («больше», «меньше», «равно»);
- присваивания;
- выделение памяти.

Пузырьковая сортировка

Сортировка пузырьком — это самый простой алгоритм сортировки. Он проходит по массиву несколько раз, на каждом этапе перемещая самое большое значение из неотсортированных в конец массива.

Псевдокод:

```
function bubbleSort(a):
   for i = 0 to n - 2
     for j = i+1 to n - 1
       if a[j] > a[i]
         swap(a[j], a[i])
```

Сложность	Лучший случай	Средний случай	Худший случай
Время	O(n)	$O(n^2)$	$O(n^2)$
Память	O(1)	O(1)	O(1)

Сортировка вставками

Сортировка вставками работает, проходя по массиву и перемещая нужное значение в начало массива. После того, как обработана очередная позиция, мы знаем, что все позиции до нее отсортированы, а после нее — нет.

Важный момент: сортировка вставками обрабатывает элементы массива по порядку. Поскольку алгоритм проходит по элементам слева направо, мы знаем, что все, что слева от текущего индекса — уже отсортировано.

```
for j = 2 to A.length do
    key = A[j]
    i = j-1
    while (i > 0 and A[i] > key) do
        A[i + 1] = A[i]
        i = i - 1
    end while
    A[i+1] = key
end for[5]
```

Сложность	Наилучший случай	В среднем	Наихудший случай
Время	O(n)	$O(n^2)$	$O(n^2)$
Память	O(1)	O(1)	O(1)

Сортировка выбором

Сортировка выбором — это некий гибрид между пузырьковой и сортировкой вставками. Как и сортировка пузырьком, этот алгоритм проходит по массиву раз за разом, перемещая одно значение на правильную позицию. Однако, в отличие от пузырьковой сортировки, он выбирает наименьшее неотсортированное значение вместо наибольшего. Как и при сортировке вставками, упорядоченная часть массива расположена в начале, в то время как в пузырьковой сортировке она находится в конце.

Сложность	Наилучший случай	В среднем	Наихудший случай
Время	O(n)	$O(n^2)$	$O(n^2)$
Память	O(1)	O(1)	O(1)

Шейкер-сортировка

Шейкер-сортировка – это разновидность «пузырьковой сортировки», которая работает в две стороны: цикл доходит до правого конца массива и оставляет значение, потом сужает область сортировки в правой части и идёт до левой части. Потом перемещает самый малый элемент в левый конец и, дойдя до него, сужает область сортировки в левой части. Сортировка продолжается до тех пор, пока область не будет исчерпана.

Сложность	Наилучший случай	В среднем	Наихудший случай
Время	O(n)	$O(n^2)$	$O(n^2)$
Память	O(1)	O(1)	O(1)

Быстрая сортировка (quicksort)

Быстрая сортировка — это алгоритм типа «разделяй и властвуй». Он работает, рекурсивно повторяя следующие шаги:

- Выбрать ключевой индекс и разделить по нему массив на две части. Это можно делать разными способами
- Переместить все элементы больше ключевого в правую часть массива, а все элементы меньше ключевого — в левую. Теперь ключевой элемент находится в правильной позиции — он больше любого элемента слева и меньше любого элемента справа.
- Повторяем первые два шага, пока массив не будет полностью отсортирован.

Быстрая сортировка реализована в C++ в библиотеке **<algorithm>** методом **std::sort**.

Псевдокод остается в виде домашнего рассмотрения ☺️

Сложность	Наилучший случай	В среднем	Наихудший случай
Время	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n^2)$
Память	$O(1)$	$O(1)$	$O(1)$

Задача.

Все вместе разберите один из алгоритмов сортировки. Например, Шейкер-сортировка и сортировка выбором.

Мем в конце.