Алгоритмы сортировки

Denis Bakin

Оценка сложности алгоритма

Способы поиска нижней оценки: алгоритм противника

Необходимо предъявить стратегию противника, которая:

Оценка сложности алгоритма

Способы поиска нижней оценки: алгоритм противника

Необходимо предъявить стратегию противника, которая:

- заставит алгоритм выполнить не менее $f(n)$ операций

Оценка сложности алгоритма

Способы поиска нижней оценки: алгоритм противника

Необходимо предъявить стратегию противника, которая:

- заставит алгоритм выполнить не менее $f(n)$ операций
- задает непротиворечивую задачу (при этом неизвестные игроку данные могут быть любыми)

Оценка сложности алгоритма

Способы поиска нижней оценки: алгоритм противника

Необходимо предъявить стратегию противника, которая:

- заставит алгоритм выполнить не менее $f(n)$ операций
- задает непротиворечивую задачу (при этом неизвестные игроку данные могут быть любыми)
- пример: поиск максимума в массиве из n чисел требует не менее $n - 1$ сравнения

Оценка сложности алгоритма

Способы поиска нижней оценки: оценки в решающем дереве

- каждую задачу можно представить в виде дерева решений
- листы дерева — возможные ответы
- высота дерева — количество шагов (операций) в худшем случае
- оптимальная сложность сортировки?
- $\Omega(n \log n)$

Оценка сложности алгоритма

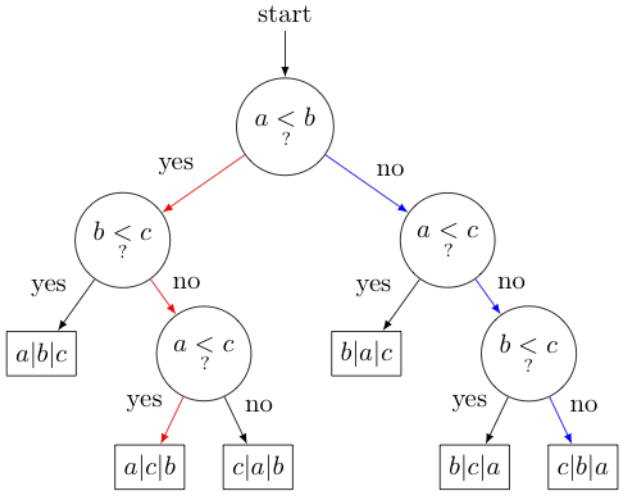

Figure 1: Пример решающего дерева

Оценка сложности алгоритма

Способы поиска нижней оценки: оценки в решающем дереве

Оценка сложности алгоритма

Способы поиска нижней оценки: оценки в решающем дереве

- n элементов можно упорядочить $n!$ способами

Оценка сложности алгоритма

Способы поиска нижней оценки: оценки в решающем дереве

- n элементов можно упорядочить $n!$ способами
- в решающем дереве должно быть не менее $n!$ листьев

Оценка сложности алгоритма

Способы поиска нижней оценки: оценки в решающем дереве

- n элементов можно упорядочить $n!$ способами
- в решающем дереве должно быть не менее $n!$ листьев
- дерево с h высотой имеет не более 2^h листьев

Оценка сложности алгоритма

Способы поиска нижней оценки: оценки в решающем дереве

- n элементов можно упорядочить $n!$ способами
- в решающем дереве должно быть не менее $n!$ листьев
- дерево с h высотой имеет не более 2^h листьев
- $2^h \geq n!$

Оценка сложности алгоритма

Способы поиска нижней оценки: оценки в решающем дереве

- n элементов можно упорядочить $n!$ способами
- в решающем дереве должно быть не менее $n!$ листьев
- дерево с h высотой имеет не более 2^h листьев
- $2^h \geq n!$
- $h \geq \log_2(n!)$\$

Оценка сложности алгоритма

Способы поиска нижней оценки: оценки в решающем дереве

- n элементов можно упорядочить $n!$ способами
- в решающем дереве должно быть не менее $n!$ листьев
- дерево с h высотой имеет не более 2^h листьев
- $2^h \geq n!$
- $h \geq \log_2(n!)$\$
- $h = \Omega(n \log n)$

Пространственной сложности

Дана последовательность натуральных чисел $x_1, x_2, \ldots, x_n$. Стандартным отклонением называется величина

$$\sigma = \sqrt{\frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \ldots + (x_n - \mu)^2}{n - 1}},$$

где μ — среднее арифметическое чисел $x_1, x_2, \ldots, x_n$.

Определите стандартное отклонение для данной последовательности чисел.

- какова оценка сложности по времени?
- какова оценка сложности по памяти?

Сортировка: постановка задачи

- Даны n объектов, для которых определён только оператор «меньше» (<)
- Нужно переставить элементы так, чтобы
  - $\forall i < j \ : \ a_i < a_j$
- Формально: найти такую *перестановку* индексов p, что
  - $a[p_1] < a[p_2] < ... < a[p_n]$

Квадратичные сортировки

Общие свойства

- $O(n^2)$
- Полезны на небольших данных или как часть других алгоритмов
- Основные представители:
  - сортировка пузырьком
  - сортировка выбором
  - сортировка вставками

Сортировка пузырьком

Идея алгоритма

- Проходим по массиву и сравниваем соседние элементы

Сортировка пузырьком

Идея алгоритма

- Проходим по массиву и сравниваем соседние элементы
- $a_i > a_{i+1} \Rightarrow \mathrm{swap}(a_i, a_{i+1})$

Сортировка пузырьком

Идея алгоритма

- Проходим по массиву и сравниваем соседние элементы
- $a_i > a_{i+1} \Rightarrow \mathrm{swap}(a_i, a_{i+1})$
- После каждого прохода **наибольший** элемент «всплывает» в конец

Сортировка пузырьком

Идея алгоритма

- Проходим по массиву и сравниваем соседние элементы
- $a_i > a_{i+1} \Rightarrow \mathrm{swap}(a_i, a_{i+1})$
- После каждого прохода **наибольший** элемент «всплывает» в конец
- После `k` проходов последние `k` элементов гарантированно на своих местах

Пузырёк: пример работы

Массив: $[5, 3, 8, 4, 2]$

Шаг	Действие	Результат
1	Сравнить 5 и $3 \to$ поменять	$[3, 5, 8, 4, 2]$
2	Сравнить 5 и $8 \to$ ок	$[3, 5, 8, 4, 2]$
3	Сравнить 8 и $4 \to$ поменять	$[3, 5, 4, 8, 2]$
4	Сравнить 8 и $2 \to$ поменять	$[3, 5, 4, 2, 8]$
...	Повторять, пока массив отсортирован	$[2, 3, 4, 5, 8]$

Пузырёк: псевдокод

```
function bubble_sort(arr):
    n = длина(arr)
    for i in 0 .. n-1:
        for j in 0 .. n-i-2:
            if arr[j] > arr[j+1]:
                swap(arr[j], arr[j+1])
    return arr
```

- Сложность: $O(n^2)$ сравнений и перестановок

Сортировка выбором

Идея алгоритма

- На каждой итерации ищем **минимальный** элемент в неотсортированной части

Сортировка выбором

Идея алгоритма

- На каждой итерации ищем **минимальный** элемент в неотсортированной части
- Меняем его местами с первым элементом этой части

Сортировка выбором

Идея алгоритма

- На каждой итерации ищем **минимальный** элемент в неотсортированной части
- Меняем его местами с первым элементом этой части
- После `k` итераций первые `k` элементов уже на своих местах

Выбор: пример работы

Массив: $[5, 3, 8, 4, 2]$

Шаг	Действие	Результат
1	Найти минимум (2), поставить в начало	$[2, 3, 8, 4, 5]$
2	Найти минимум в хвосте (3), поставить на позицию 2	$[2, 3, 8, 4, 5]$
3	Найти минимум в хвосте (4), поставить на позицию 3	$[2, 3, 4, 8, 5]$
4	Найти минимум в хвосте (5), поставить на позицию 4	$[2, 3, 4, 5, 8]$

Выбор: псевдокод

```
function selection_sort(arr):
    n = длина(arr)
    for i in 0 .. n-1:
        min_idx = i
        for j in i+1 .. n-1:
            if arr[j] < arr[min_idx]:
                min_idx = j
        swap(arr[i], arr[min_idx])
    return arr
```

- Сложность: $O(n^2)$ сравнений
- Число перестановок — всего $O(n)$ (по одной на итерацию)

Нижняя граница сортировки сравнениями

- Дерево решений для сортировки:
  - $n!$ листьев (все перестановки)
  - высота $\geq n \log_2 n$
- Нижняя граница:

$$\Omega(n \log n)$$

Сортировка слиянием

Операция merge

Вводим базовое преобразование – слияние двух отсортированных массивов:

- Линейный проход по обоим массивам
- На каждом шаге выбираем меньший из текущих элементов
- Время работы: $O(n)$

Сортировка слиянием

- Разбиваем массив на половины
- Рекурсивно сортируем
- Сливаем отсортированные половины

Сортировка слиянием

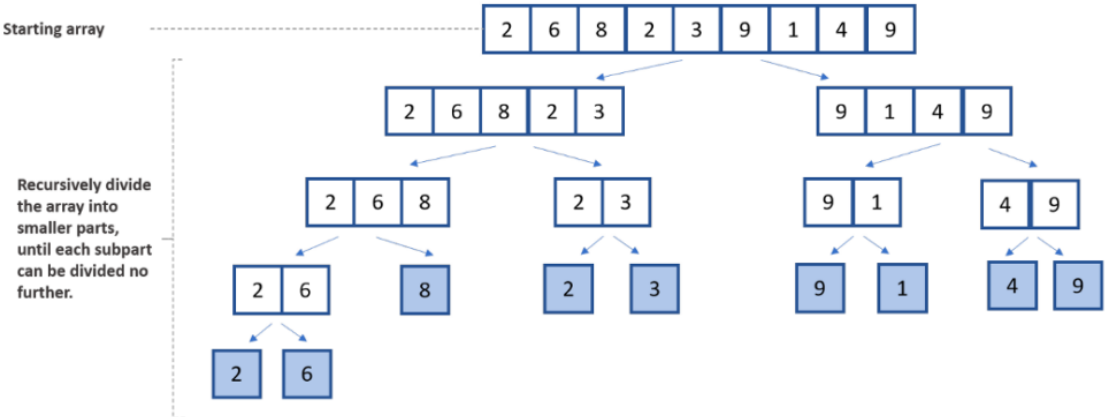

Figure 2: Разбиение

Сортировка слиянием

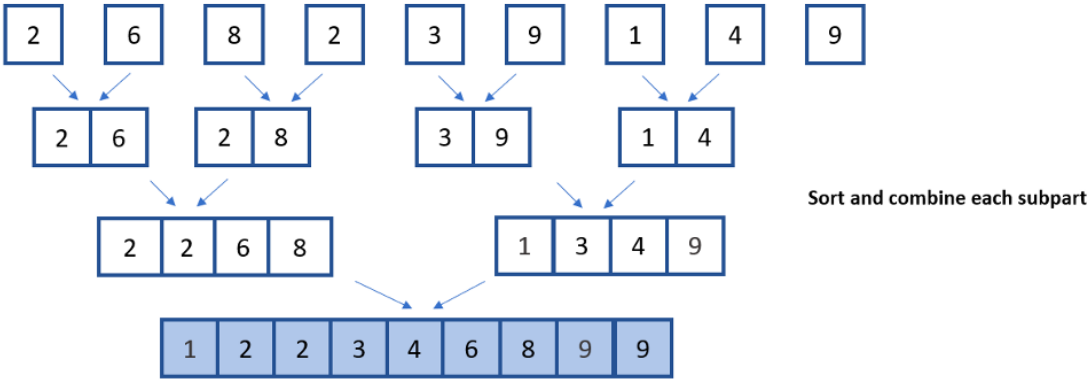

Figure 3: Слияние

Сложность merge sort

- Слияние — $O(n)$

Сложность merge sort

- Слияние — $O(n)$
- Глубина рекурсии — $\log n$, так как делим массив пополам

Сложность merge sort

- Слияние — $O(n)$
- Глубина рекурсии — $\log n$, так как делим массив пополам
- Итого: $O(n \log n)$

Сортировка подсчётом

- Подходит, если элементы из небольшого диапазона
- Считаем частоты каждого значения
- Выводим элементы в порядке возрастания
- Время работы: $O(n + m)$, где m — количество возможных значений

Counting sort illustration