Адрианов Н.М.
Иванов А.Б.

Алгоритмы и структуры данных

Сортировки

Quicksort и Timsort

Нижняя оценка сложности сортировки

Теорема. Любой детерминированный алгоритм сортировки ***сравнением*** имеет сложность в ***худшем*** случае $\Omega\,(n \log n)$.

Количество перестановок в массиве из n элементов $= n!$

Работу алгоритма на различных входных данных можно представить в виде бинарного дерева. Каждое ветвление – сравнение элементов массива. Если при любых входных данных количество сравнений не больше S, то глубина дерева не больше S, а количество конечных узлов – не больше 2^S.

$$2^S \geq n! \qquad \Rightarrow \qquad S = \Omega(n \log n)$$

Формула Стирлинга: $\quad n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

или грубая оценка $\quad n! > \left[\frac{n}{2}\right]^{\left[\frac{n}{2}\right]}$

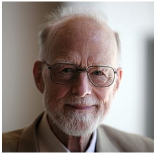
Свойства алгоритмов сортировки

- Сложность
- Необходимая память
- Устойчивость

Сортировка слиянием:

- $O(n \log n)$
- $\geq N$
- Устойчивый

Быстрая сортировка (Quicksort)



Tony Hoare в 1959 году в СССР

Интерес Сэра Тони Хоара к компьютерным вычислениям проснулся в начале пятидесятых годов, когда он изучал философию (наряду с латинским и греческим) в Оксфордском университете, под руководством Джона Лукаса. Во время своей службы в Королевском военно-морском флоте изучал русский язык. В 1959 году, будучи аспирантом Московского государственного университета, он изучал машинный перевод языков и теорию вероятностей, в школе А.Н. Колмогорова. Для эффективного поиска слов в словаре, он разработал известный алгоритм «быстрой сортировки».

`http://theoryandpractice.ru/presenters/16008-toni-khoar`

Быстрая сортировка (Quicksort)

3	7	2	6	4	8	1	5

```
procedure QuickSort(a[], left, right):
  // a[] - массив от 0 до n-1
  pivot = a[left]
  i = left + 1
  for j in [left + 1 .. right]:
    if a[j] < pivot:
      переставить a[j] и a[i]
      i++
    end
  end
  переставить a[left] и a[i-1]

  QuickSort(a, left, i - 2)
  QuickSort(a, i, right)
end
```

Быстрая сортировка (корректность)

Быстрая сортировка (Quicksort)

Сложность? Выбор опорного элемента?

- В худшем случае: $O(n^2)$
- В лучшем случае (опорный элемент - медиана):

$$T(n) = 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n) \implies T(n) = O(n \log n)$$

- В «плохом» случае - опорный элемент делит массив в соотношении $99/100$: $O(n \log n)$

Быстрая сортировка (Quicksort)

Теорема. Сложность быстрой сортировки «в среднем» – $O(n \log n)$ (при случайном выборе опорного элемента).

- Пространство элементарных событий $\Omega = \{$все возможные последовательности опорных элементов$\}$
- Случайная величина $X(\omega) =$"количество выполняемых сравнений при последовательности опорных элементов ω"
- Наша цель:

$$M[X] = O(n \log n)$$

Быстрая сортировка (Quicksort)

- z_i - i-ая порядковая статистика
- $X_{ij}(\omega)$ - количество сравнений z_i и z_j
- По определению

$$X(\omega) = \sum_{i<j} X_{ij}(\omega)$$

- Из линейности математического ожидания

$$M[X] = \sum_{i<j} M[X_{ij}]$$

- Или

$$M[X] = \sum_{i<j} P[X_{ij} = 1]$$

Быстрая сортировка (Quicksort)

$$P[X_{ij} = 1] = \frac{2}{j - i + 1}$$

Быстрая сортировка (Quicksort)

$$P[X_{ij} = 1] = \frac{2}{j - i + 1}$$

$$M[X] = \sum_{i<j} P[X_{ij} = 1]$$

$$\Downarrow$$

$$M[X] = \sum_{i<j} P[X_{ij} = 1] = \sum_{i<j} \frac{2}{j - i + 1} =$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} \leq 2n \sum_{k=2}^{n} \frac{1}{k} \leq 2n \ln n$$

Quicksort: анализ (второй метод)

Partitioning: $n+1$ сравнение.

Пусть C_n – мат.ожидание количества сравнений при сортировке массива из n элементов.

$$C_n = n + 1 + \sum_{k=1}^{n} \frac{1}{n}(C_{k-1} + C_{n-k}), \qquad C_0 = C_1 = 0$$

Quicksort: анализ (второй метод)

$$C_n = n + 1 + \sum_{k=1}^{n} \frac{1}{n}(C_{k-1} + C_{n-k})$$

$$C_n = n + 1 + \frac{2}{n} \sum_{k=1}^{n} C_{k-1}$$

$$nC_n = n(n+1) + 2 \sum_{k=1}^{n} C_{k-1}$$

$$(n-1)C_{n-1} = (n-1)n + 2 \sum_{k=1}^{n-1} C_{k-1}$$

$$nC_n - (n-1)C_{n-1} = 2n + 2C_{n-1}$$

$$nC_n = (n+1)C_{n-1} + 2n$$

Quicksort: анализ (второй метод)

$$nC_n = (n+1)C_{n-1} + 2n$$

$$\begin{aligned}
\frac{C_n}{n+1} &= \frac{C_{n-1}}{n} + \frac{2}{n+1} = \\
&= \frac{C_{n-2}}{n-1} + \frac{2}{n} + \frac{2}{n+1} = \\
&= \frac{C_1}{2} + \frac{2}{3} + \ldots + \frac{2}{n} + \frac{2}{n+1}
\end{aligned}$$

$$C_n \sim 2(n+1)\sum_{k=1}^{n+1}\frac{1}{k} - C \cdot n \sim 2n \ln n$$

Быстрая сортировка: зачем?

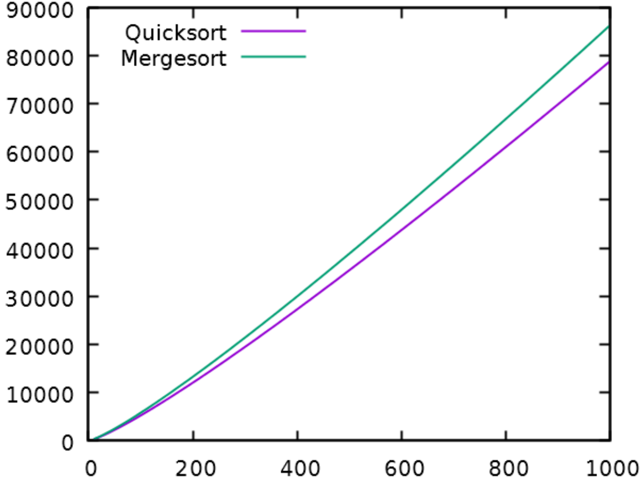
- Сложность
- Необходимая память
- Устойчивость

- $O(n \log n)$
- $\approx \log(n)$
- не является устойчивым

Быстрая сортировка: зачем?

Сортировка слиянием: $12.5\, x \log(x)$

Быстрая сортировка: $11.67\, x \log(x) - 1.74\, x$



3-WAY QUICKSORT

Какое время работы алгоритма на массиве, состоящем из одинаковых элементов?

Стандартный quicksort: $\Theta(n^2)$.

Решение: 3-way quicksort.

Разбиваем массив на 3 части:

- $a[i] < p$ для $0 \leq i \leq r$
- $a[i] = p$ для $r + 1 \leq i \leq s$
- $a[i] > p$ для $s + 1 \leq i \leq n - 1$

3-WAY QUICKSORT

Задача о голландском национальном флаге (Edsger Wybe Dijkstra)

Отсортировать массив, состоящий из 0, 1 и 2

3-WAY QUICKSORT

Инвариант цикла:

$a[j] < p$
при $j \in [low+1, lt)$

$a[j] = p$
при $j \in [lt, i)$

$a[j] > p$
при $j \in (gt, high]$

```
procedure QuickSort3Way(a[]):
  // a[] - массив от 0 до n-1
  перемешать (shuffle) массив a
  QuickSort3Way(a, 0, n-1)
end

procedure QuickSort3Way(a[], low, high):
  p = a[low]
  i = low + 1
  lt = low + 1
  gt = high
  while i <= gt:
    if a[i] < p:
      Exch(a, lt++, i++)
    else if a[i] > p:
      Exch(a, i, gt--)
    else
      i++
  end
  Exch(a, low, --lt)

  QuickSort3Way(a, low, lt - 1)
  QuickSort3Way(a, gt + 1, high)
end
```

QuickSort: переполнение стека

В самом худшем случае будет n рекурсивных вызовов, что может привести к переполнению стека.

Решение: вместо двух рекурсивных вызовов делать только один (для части меньшего размера). Вторую (большую) часть обрабатывать, не используя рекурсивный вызов.

Практические улучшения

На массивах небольшого размера выгоднее использовать сортировку вставками.

Выбор опорного элемента: медиана из трех (median-of-3). Выбираем 3 случайных элемента и в качестве опорного берем второй по порядку (медиану).

Tukey's ninther: выбираем 3 тройки случайных элементов, в каждой тройке находим медиану и в качестве опорного берем второй медиану медиан.

Сортировка в Java и .NET

.NET 4.0: QuickSort

.NET 4.5: QuickSort + переключение

- маленькие части $n \leq 16$ – InsertionSort
- если количество Partition превышает $2 \log N$ – HeapSort

Java:

- primitive types:
  $n < 47$ – InsertionSort
  $n < 286$ – QuickSort
  $n \geq 286$ – возможно (!) переключение на MergeSort
  QuickSort dual-pivot: 5 элементов, 2й и 4й – опорные
- objects: TimSort

Go:

- QuickSort с использованием Tukey's Ninther

Методы сортировок: характеристики

Алгоритм	stable	in-place	Время	Доп. память
Selection	−	+	N^2	1
Insertion	+	+	между N и N^2	1
Shell	−	+	$N \log N$? $N^{6/5}$?	1
Quicksort	−	+	$N \log N$	$\log N$
3-way quicksort	−	+	между N и $N \log N$	$\log N$
Mergesort	+	−	$N \log N$	N
Timsort	+	−	между N и $N \log N$	между 1 и N
Heapsort	−	+	$N \log N$	1
???	+	+	$N \log N$	1

Timsort

- Tim Peters (2002)
  `http://bugs.python.org/file4451/timsort.txt`
- Python 2.3, Java SE 7, Android 1.5
- Идея: в реальных ситуациях массивы данных часто содержат в себе упорядоченные подмассивы
- Небольшие подмассивы сортируются вставками
- Модифицированная процедура слияния
- Хочется сливать массивы близкого размера

Февраль 2015: обнаружена ошибка (сразу исправлена)

`http://www.envisage-project.eu/`

`proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it/`

Май 2018: в Java исправление было некорректно

`https://bugs.openjdk.java.net/browse/JDK-8203864`
`https://arxiv.org/pdf/1805.08612.pdf`

Timsort: «run»

$$a_1 \le a_2 \le a_3 \le \ldots \le a_r$$

$$a_1 > a_2 > a_3 > \ldots > a_r$$

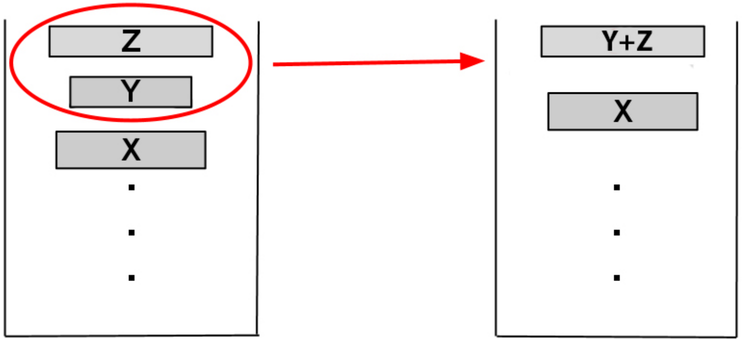
Выбираем число `minrun` $\in [32, 64)$ так, чтобы $N/$`minrun` оказалось степенью двойки или немного меньше.

Пример: $N = 2112$
`minrun` $= 32$ – плохой $(2112 = 66 \times 32)$,
`minrun` $= 33$ – хороший $(2112 = 64 \times 33)$.

Timsort: общая схема

1. Определяем `minrun`, текущая позиция $= 1$
2. Находим run, начинающийся с текущей позиции
3. Если run убывающий – переворачиваем
4. Если run короче `minrun` – дополняем и сортируем вставками
5. Добавляем в стек (начало подмассива, длина)
6. Проверяем условия баланса стека (след. слайд)
7. Если не дошли до конца массива – переходим к 2
8. Сливаем подмассивы, хранящиеся в стеке

Timsort: стек



- $X > Y + Z$
- $Y > Z$

Если размер стека не меньше 3 и $X \leq Y + Z$ – сливаем Y с минимальным из X и Z. Иначе если $Y \leq Z$ – сливаем Y с Z.

Timsort: слияние

Слияние всегда выполняется для соседних подмассивов.

Создаем временный массив, размер которого равен размеру меньшего из сливаемых подмассивов. Меньший из подмассивов копируем во временный массив.

Возможен переход в режима «галопа».

Сортировка

Теорема. Любой детерминированный алгоритм сортировки ***сравнением*** имеет сложность в ***худшем*** случае $\Omega(n \log n)$.

Можно ли быстрее?

Сортировка подсчетом

Задача: отсортировать массив из N чисел от 0 до $R-1$

Сортировка подсчетом

Задача: Ω – пространство объектов, $f : \Omega \to [0..R-1]$.
Отсортировать массив из N объектов по возрастанию $f(x)$.

`count[i+1]` – на какой позиции нужно поставить число
`aux` – вспомогательный массив

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

Сортировка подсчетом

- Сложность $O(N+R)$
- Дополнительная память $O(N+R)$
- Стабильная
- Необязательно числа / буквы

LSD-сортировка (least significant digit)

Задача: отсортировать массив из N строк одинаковой длины.

Применим сортировку подсчетом: $\Omega =$ строки длины W, $f_d(x) = d$-й символ в строке x.

Сортируем сначала по последнему символу (используя f_W), потом по предпоследнему (используя f_{W-1}) и т.д. После сортировки по первому символу массив строк окажется отсортированным в лексикографическом порядке (следствие стабильности сортировки подсчетом!).

LSD-сортировка (least significant digit)

```
public static void sort(String[] a, int W)
{
    int R = 256;
    int N = a.length;
    String[] aux = new String[N];

    for (int d = W-1; d >= 0; d--)
    {
        int[] count = new int[R+1];
        for (int i = 0; i < N; i++)
            count[a[i].charAt(d) + 1]++;
        for (int r = 0; r < R; r++)
            count[r+1] += count[r];
        for (int i = 0; i < N; i++)
            aux[count[a[i].charAt(d)]++] = a[i];
        for (int i = 0; i < N; i++)
            a[i] = aux[i];
    }
}
```

MSD-сортировка (most significant digit)

Задача: отсортировать массив из N строк.

Сортируем сначала по первому символу (используя f_1), затем диапазон для каждой буквы на первом месте сортируем рекурсивно по второй букве и т.д.

MSD-сортировка (most significant digit)

```
public static void sort(String[] a)
{
    aux = new String[a.length];
    sort(a, aux, 0, a.length, 0);
}
private static void sort(String[] a, String[] aux, int lo, int hi, int
     d)
{
    if (hi <= lo) return;
    int[] count = new int[R+2];
    for (int i = lo; i <= hi; i++)
        count[charAt(a[i], d) + 2]++;
    for (int r = 0; r < R+1; r++)
        count[r+1] += count[r];
    for (int i = lo; i <= hi; i++)
        aux[count[charAt(a[i], d) + 1]++] = a[i];
    for (int i = lo; i <= hi; i++)
        a[i] = aux[i - lo];
    for (int r = 0; r < R; r++)
        sort(a, aux, lo + count[r], lo + count[r+1] - 1, d+1);
}
```

LSD/MSD = поразрядные сортировки

Возможно применение для строк в различных алфавитах:

- Десятичные цифры (0123456789)
- Шестнадцатеричные цифры (0123456789ABCDEF)
- ДНК-последовательности (ACGT)
- Английские буквы (a..z, $R = 26$)
- ASCII ($R = 128/256$)
- Unicode ($R = 65536$)?

Вопрос: какой алгоритм выбрать для сортировки миллиона 32-битных целых чисел?

QuickSelect

Дано: массив размера n, $1 \le k \le n$
Найти: k-ю порядковую статистику

Пусть $C(n,k)$ – матожидание количества сравнений

$$C(n,\cdot) \le n - 1 + \frac{1}{n}\sum_{i=1}^{n} C(\max\{i-1, n-i\}, \cdot))$$

$$C(n,\cdot) \le n - 1 + \frac{2}{n}\sum_{i=1}^{n/2} C(n-i, \cdot))$$

Индукция: $C(m,\cdot) \le cm$ для $m < n \Rightarrow$

$$C(n,\cdot) \le \left(1 + \frac{3}{4}c\right) n \le cn$$

Детерминированный Select

`Select(массив размера n, k)`

- разбиваем на пятерки ($m = \lceil n/5 \rceil$), сортируем каждую, выбираем медиану в каждой пятерке
- находим медиану p: `Select(массив размера m, m/2)`
- выполняем `Partition` с опорным элементом p
- определяем в какой из частей находится k-я статистика, вызываем `Select` рекурсивно

Сравнение реализаций Select