Сортировки

Понятие

- Сортировать - распределять, разбирать по сортам, качеству, размерам, по сходным признакам. (толковый словарь Ожегова)
- синонимы: классификация, систематизация.
- перегруппировка элементов в некотором определенном порядке (упорядочивание, ранжирование).

Классы сортировок

- сортировка массивов
- сортировка (последовательных) файлов.

или

- внутренняя сортировка
- внешняя сортировка

Определение

Частичным порядком на множестве S называется такое бинарное отношение R, что для любых a, b и c из S

1) aRa (R рефлексивно),
2) aRb и bRc => aRc (R транзитивно),
3) aRb и bRa => a=b (R антисимметрично).

Определение

Линейным, или полным, **порядком** на множестве S называется такой частичный порядок R на S, что для любых двух элементов a, b выполняется либо aRb, либо bRa (другими словами, элементы a, b сравнимы)

Задача сортировки

Пусть дана последовательность из n элементов a_1 , a_2 , … , a_n , выбранных из множества, на котором задан линейный порядок.

- элемент a_i назовем записью,
- линейный порядок будем обозначать ≤

Каждая запись a_i имеет ключ k_i , который управляет процессом сортировки.

помимо ключа, запись может иметь некоторую дополнительную информацию, которая не влияет на процесс сортировки, но всегда присутствует в этой записи.

Задача сортировки

- Требуется найти такую перестановку π = (π(1), π(2),…, π(n)) этих n записей, после которой ключи расположились бы в неубывающем порядке:

$$k_{\pi(1)} \le k_{\pi(2)} \le \ldots \le k_{\pi(n)}$$

Определение

Алгоритм сортировки называется **устойчивым**, если в процессе сортировки относительное расположение элементов одинаковыми ключами не изменяется предполагается, что элементы уже были отсортированы по некоторому вторичному ключу)

$\pi(i) < \pi(j)$, если $k_{\pi(i)} \leq k_{\pi(j)}$ и $i < j$.

Сортировки

Все алгоритмы сортировки можно разбить на три группы:

1. сортировка с помощью включения,
2. сортировка выбором,
3. сортировка с помощью обменов

Сортировка с помощью включения

- Пусть элементы a_1 , a_2 , … , a_{i-1} , ; $1 < i \leq n$ уже упорядочены на предыдущих этапах данным алгоритмом.
- На очередном этапе необходимо взять элемент a_i , и включить в нужное место уже упорядоченной последовательности a_1 , a_2 , … , a_{i-1} .

Сортировка с помощью включения

В зависимости от того, как происходит процесс включения элемента, различаю прямое включение и двоичное включение.

Алгоритм сортировки с помощью прямого включения

Алгоритм insertion (a, n);
Input: массив A, содержащий n чисел (n≥1).
Output: упорядоченный массив A.

```
for i ← 2 to n {
    x ← a[i]; a[0] ← x; j ← i;
    while (x.key < a[j-1].key) {
        a[j] ← a[j-1]; j ← j-1
        }
    a[j] ← x;
    }
```

Пример

2	8	2	4	9	3	6
4	2	8	4	9	3	6
9	2	4	8	9	3	6
3	2	4	8	9	3	6
6	2	3	4	8	9	6
Конец	2	3	4	6	8	9

Оценка трудоемкости алгоритма

В худшем случае:

$T(n)=T(n-1)+Cn$, $T(1)=0$; $T(n)=\Theta(n^2)$

Число сравнений на i-ом шаге – $\Theta(i)$

Число пересылок на i-ом шаге – $\Theta(i)$

$$T(n) = \sum_{i=2}^{n} \Theta(i) = \Theta(n^2)$$

В среднем:

$$T(n) = \sum_{i=2}^{n} \Theta(i/2) = \Theta(n^2)$$

Алгоритм двоичного включения

```
Алгоритмinsertion2(a, n);
for i ← 2to n{
    x ← a[i]; L ←1; R ← i;
    while (L < R) {
        m=[(L+R)/2];
        if (a[m].key< x.key) {L ←m + 1}
        else R ←m;
        }
    for j ←i downto (R + 1){
        a[j] ←a[j -1];}
    a[R] ← x;
    }
```

Алгоритм двоичного включения

В худшем случае:

$T(n)=T(n-1)+Cn$, $T(1)=0$; $T(n)=\Theta(n^2)$

Число сравнений на i-ом шаге – $\Theta(\log_2 i)$

Число пересылок на i-ом шаге – $\Theta(i)$

$$T(n) = \sum_{i=2}^{n} \Theta(\log_2 i) + \Theta(i) = \Theta(n^2)$$

В среднем:

$$T(n) = \sum_{i=2}^{n} \Theta(i/2) = \Theta(n^2)$$

Сортировка выбором

Идея сортировки заключается в следующем:

1.Выбрать элемент с наименьшим ключом и поменять его с первым элементом. Теперь первый элемент стоит на своем месте.

2.Повторить действия с оставшимися n-1 элементами.

3.Процесс заканчивается, когда n -1 элементов будут помещены на свои места.

Сортировка выбором

```
Алгоритм selection (a, n);
    for i ← 1to n-1{
        k ← i;
        for j ← i+1to n{
            if a[j].key< a[k].key {
                k ← j;
            }
        a[i]↔a[k];
        }
```

Оценка трудоемкости алгоритма

В худшем случае:

$T(n)=Cn+T(n-1)$, $T(1)=0$;

$T(n)=\Theta(n^2)$

Сортировка с помощью обменов

1. Сортировки помощью обменов основываются на сравнении двух элементов.
2. Если порядок элементов не соответствует упорядоченности, то происходит их обмен.
3. Процесс повторяется до тех пор, пока элементы не будут упорядочены.

Сортировка с помощью обменов

- Пузырьковая сортировка
- Шейкерная сортировка

Пузырьковая сортировка

- Просматриваем исходную последовательность справа налево и на каждом шаге меньший из двух соседних элементов перемещается в левую позицию.
- В результате первого просмотра самый маленький элемент будет находиться в крайней левой позиции.
- После чего повторяем описанный выше процесс, рассматривая в качестве исходной последовательности массив, начиная со 2-ой позиции и т.д.

Пузырьковая сортировка

Алгоритм bubble_sort (a, n);

```
for i ← 2to n {
    for j ← n downto i {
        if (a[j-1].key> a[j].key){
            a[j-1]↔a[j];
            }
        }
    }
```

Шейкерная сортировка

Анализ алгоритма пузырьковой сортировки приводит к следующим наблюдениям:

- Если при некотором из проходов нет перестановок, то алгоритм можно завершить.
- Если зафиксировать индекс k последнего обмена (все пары левее этого индекса уж упорядочены), то просмотр можно завершить на этом индексе, а не идти до нижнего предела для индекса i.
- Чередование направлений для просмотра (всплываетсамый легкий,тонет самый тяжелый).

Шейкерная сортировка

```
Алгоритм shaker_sort (a, n);
L ← 2; R ← n; k ← n;
repeat
      for j ← Rdown to L
            if (a[j-1].key> a[j].key){
                  a[j-1]↔a[j]; k ← j;
                  }
      L ← k+1;
      for j ← Lto R
            if (a[j-1].key> a[j].key){
                  a[j-1]↔a[j]; k ← j;
                  }
      R ← k-1;
until (L>R);
```

Сортировка слиянием

Сортировка слиянием заключается в следующем:

1.Делим последовательность элементов на две части;

2.Сортируем отдельно каждую из частей;

3.Производим слияние отсортированных частей последовательности:

a)при слиянии сравниваем наименьшие элементы и меньший из них отправляем в список вывода;

b)повторяем описанные действия до тех пор, пока не исчерпается одна из частей;

c)все оставшиеся элементы другой части пересылаем в список вывода.

Процедура слияния отсортированных частей последовательности

```
Алгоритм merge (a, L, Z, R);
i ← L; j ← Z+1; k ← 1;
while (i ≤ Z) &(j ≤R){
        if (a[i].key< a[j].key){
                c[k]←a[i]; i++; k++;}
        else {
                c[k]← a[j]; j++; k++;}
        }
while (i ≤ Z) {
        c[k]←a[i]; i++; k++;}
while (j ≤R){
        c[k]←a[j]; j++; k++;}
k ← 0
for i ← L to R {
        k++; a[i] ← c[k];}
```

Сортировка слиянием

Алгоритм merge_sort (L, R);

```
if (L ≠ R ){
    k ← (L+R) div 2;
    merge_sort(L, k);
    merge_sort(k+1, R);
    merge (a, L, k, R);
    }
```

Быстрая сортировка (Хоара)

Суть алгоритма состоит в следующем:

1.Выбрать некоторый элемент x для сравнения (это может быть средний, первый или последний элемент)

2.Используя обмены, выполнить процедуру разделения, суть которой заключается следующем: разбить массив на две части: левую с ключами ≤x и правую с ключами ≥x.

Быстрая сортировка

Данные действия могу быть выполнены, например, следующим алгоритмом:

- просматриваем массив слева, пока не встретим элемент a[i]> x
- просматриваем массив справа, пока не встретим элемент a[j] < x
- меняем местами эти два элемента
- продолжаем просмотр обмен до тех пор, пока не буду просмотрены все элементы массива (i> j).
- Повторяем процедуру разделения к получившимся двум частям, затем частям частей так далее, пока каждая из частей не будет состоять из одного единственного элемента

Быстрая сортировка

```
Алгоритм quick_sort (L, R);
x ← a[(L+R) div 2]; i ← L; j ← R;
repeat{
    while (a[i].key < x.key) {i++;}
    while (a[j].key> x.key) {j--;}
    if (i ≤ j) {
        a[i] ↔ a[j]; i++; j--;}
    } until (i ≥ j)
if (L < j) {quick_sort(L, j);}
if (i < R) {quick_sort(i, R);}
```

Оценка сложности алгоритма

- Процедура разделения n элементов требует Cn операций, так как каждый элемент последовательности необходимо сравнить с выбранным элементом, поэтом рекуррентное уравнение будет иметь вид:

$$T(n) = Cn + T(|A1|) + T(|A2|),$$

- где |A1|, |A2| -размеры левой и правой части массива после процедуры разделения.
- Если предположить, что разделение в среднем разбивает часть пополам то

$$T(n) = Cn + 2T(n/2),\ T(n) = \Theta(n\log n).$$

Оценка сложности алгоритма

- Худшим случаем является ситуация, когда в качестве сравниваемого элемента выбирается наибольший из всех элементов рассматриваемой части. В этом случае после процедуры разделения $|A1| = n-1$, $|A2| = 1$.
- Рекуррентное уравнение будет иметь вид

  $T(n) = Cn + T(n-1) + T(1)$, $T(n) = \Theta(n^2)$
- Сам Ч. Хоар предполагает, что x надо выбирать случайно, а для небольших выборок останавливаться на медиане.

Пример

0	1	2	3	4	5	6	7	8
9	3	4	8	7	2	11	6	5

Фиксируем элемент x = 7 (средний)

0	1	2	3	4	5	6	7	8
9	3	4	8	7	2	11	6	5

0	1	2	3	4	5	6	7	8
5	3	4	8	7	2	11	6	9

Пример

0	1	2	3	4	5	6	7	8
5	3	4	8	7	2	11	6	9

0	1	2	3	4	5	6	7	8
5	3	4	6	7	2	11	8	9

0	1	2	3	4	5	6	7	8
5	3	4	6	7	2	11	8	9

0	1	2	3	4	5	6	7	8
5	3	4	6	2	7	11	8	9

Пример

0	1	2	3	4	5	6	7	8
5	3	4	6	2	7	11	8	9

0	1	2	3	4
5	3	4	6	2

5	6	7	8
7	11	8	9

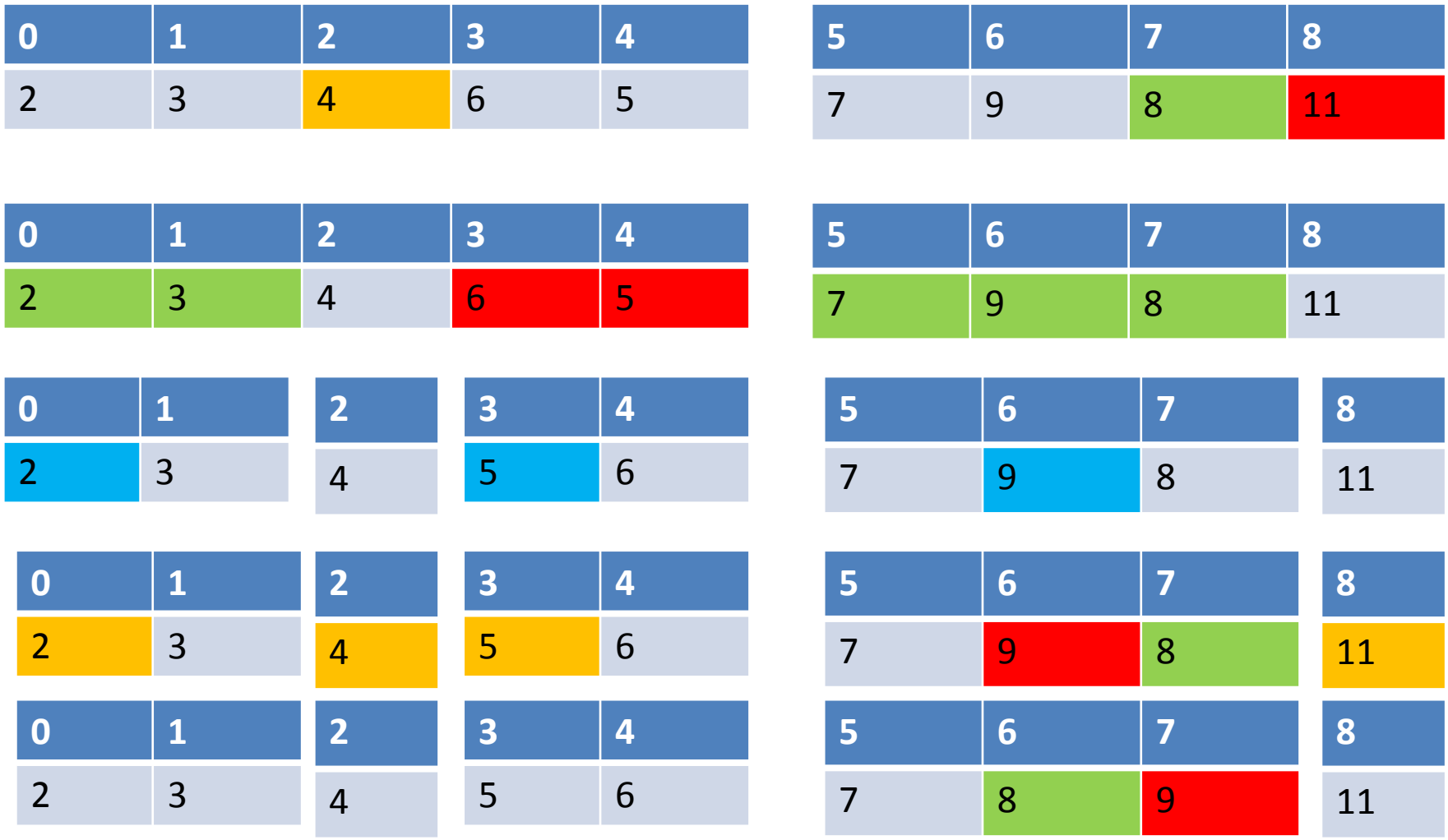
0	1	2	3	4
5	3	4	6	2

5	6	7	8
7	11	8	9

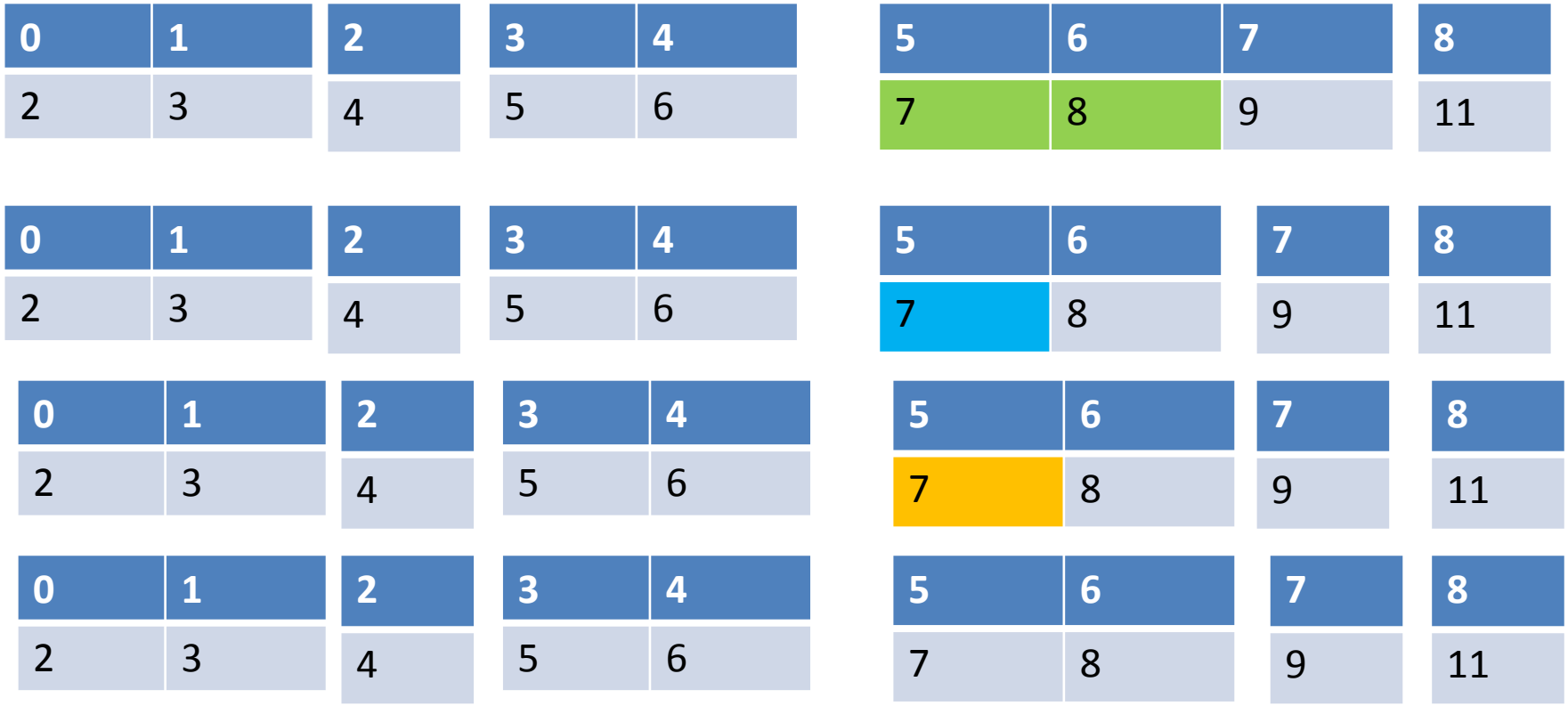
0	1	2	3	4
2	3	4	6	5

5	6	7	8
7	9	8	11

Пример


0 1 2 3 4
2 3 4 6 5
5 6 7 8
7 9 8 11
0 1 2 3 4
2 3 4 6 5
5 6 7 8
7 9 8 11
0 1 2 3 4
2 3 4 5 6
5 6 7 8
7 9 8 11
0 1 2 3 4
2 3 4 5 6
5 6 7 8
7 9 8 11
0 1 2 3 4
2 3 4 5 6
5 6 7 8
7 8 9 11

Пример


0 1 2 3 4 5 6 7 8
2 3 4 5 6 7 8 9 11
0 1 2 3 4 5 6 7 8
2 3 4 5 6 7 8 9 11
0 1 2 3 4 5 6 7 8
2 3 4 5 6 7 8 9 11
0 1 2 3 4 5 6 7 8
2 3 4 5 6 7 8 9 11

Порядковые статистики

- Задача: дано множество из n чисел. Найти тот его элемент, который будет k-м по счёту, если расположить элементы множества в порядке возрастания.
- В англоязычной литературе такой элемент называется k-й порядковой статистикой (order statistic).
- Например, минимум (minimum) - это порядковая статистика номер 1, а максимум (maximum) - порядковая статистика номер п.

Порядковые статистики

- Медианой (median) называется элемент множества, находящийся (по счёту) посередине между минимумом и максимумом. Точнее говоря, если n нечётно, то медиана - это порядковая статистика номер i = (n + 1)/2, а если n четно, то медиан даже две: с номерами i = n/2 и i = n/2 + 1.
- Можно ещё сказать, что, независимо от чётности n, медианы имеют номер i = [(n+1)/2]
- В дальнейшем мы будем называть медианой меньшую из двух (если их две).

Порядковые статистики

Пример: 18 24 12 27 19

Медиана =19

Поиск медианы является частным случаем более общей задачи:

нахождение k-го наименьшего элемента из n элементов

Выбор элемента с данным номером

- Дано: Множество A из n различных элементов и целое число k, $1 < k < n$.
- Найти: Элемент x из A, для которого ровно k-1 элементов множества A меньше x.
- Эту задачу можно решить за время $\Theta(n \lg n)$: отсортировать числа, после чего взять k-й элемент в полученном массиве.
- Однако, есть и более быстрые алгоритмы.
- Рассмотрим алгоритм для нахождения k-го наименьшего элемента из n элементов, предложенный Ч. Хоаром

Выбор элемента с k номером (A1)

Выполняется операция разделения на отрезке [L, R], где первоначально L = 1, R = n , а в качестве разделителя берется x = a[k]. В результате разделения получаются индексы i, j такие, что

a[h] < x, h < i; a[h] > x, h > j; i > j;

a) если j ≤ k ≤ i, то элемент a[k] разделяет массив на две части в нужной пропорции; алгоритм заканчивает вою работу;

b) если i < k, то выбранное значение x было слишком мало, поэтом процесс разделения необходимо выполнить на отрезке [i, R]

c) если k < j , то значение x было слишком велико, поэтому процесс разделения необходимо выполнить на отрезке [L, j]

2. Процесс разделения повторять до тех пор, пока не возникнет ситуация a) - значение x соответствует значению k-ого наименьшего элемента.

Реализация (A1)

```
L ← 1; R ← n;
while (L< k) {
    x ←A[k];
    Разделение a[L] … a[R]
     if (i < k) {L ← i;}
     if (k < j) {R ← j;}
    if (j ≤ k ≤ i) {return x ; break;}
}
```

Оценка сложности алгоритма (A1)

- Если предположить, что разделение **в среднем** разбивает часть, где находится требуем элемент пополам, то рекуррентное уравнение будет иметь вид $T(n) = Cn + T(n/2)$
- По теореме о решении рекуррентного уравнения трудоемкость алгоритма в среднем есть $\Theta(n)$.
- Таким образом, получаем явное преимущество по сравнению с прямыми методами, где сначала сортируется все множество, а затем выбирается k-ый элемент.
- В худшем случае, когда в качестве разделителя берется максимальный (минимальный элемент) рассматриваемой подпоследовательности, рекуррентное уравнение будет иметь вид

$$T(n) = Cn + T(n - 1)$$

трудоемкость алгоритма в этом случае $\Theta(n^2)$

Выбор элемента с k номером (A2)

1. Разбиваем исходную последовательность A на n/5 подпоследовательностей по пять элементов в каждой. В каждой такой подпоследовательности находим медиану. Это потребует $C_1 n$ операций.
2. Из найденных на первом шаге медиан строим последовательность M и рекурсивно находим ее медиану x. Так как длина рассматриваемой последовательности M равна n/5, то трудоемкость нахождения медианы для этой последовательности равна T (n/5).
3. Для полученного элемента x выполним процесс разделения, который потребует $C_2 n$ операций. В результате вся рассматриваемая последовательность A будет разбита на части: A1 , где элементы не больше x; A2 , где элемент не меньше x. Одна из частей может быть отброшена, причем не сложно показать, что количество элементов каждой из частей не меньше n/4.
4. Решаем задачу нахождения k-ого наименьшего элемента оставшихся 3n/4 элементов, что потребует времени T(3n/4).

Оценка сложности алгоритма (A2)

- Таким образом, рекуррентное уравнение для описанного выше алгоритма имеет вид

$$T(n) = C1n + T(n/5) + C2n + T(3n/4)$$

- Решая данное уравнение методом подстановки, при $g(n) = 20Cn$ или методом рекурсивны деревьев, получаем, что трудоемкость приведенного выше алгоритма есть $\Theta(n)$.
- Если исходную последовательность разбивать на семерки, то рекуррентное уравнение будет иметь вид

$$T(n) = C3n + T(n/7)+T(3n/4)<(28/3)C_3n$$

Сортировка вычерпыванием

Алгоритм сортировки вычёрпыванием (bucket sort) работает за линейное (среднее) время. Эта сортировка годится не для любых исходных данных: говоря о линейном среднем времени, мы предполагаем, что на вход подаётся последовательность независимых случайных чисел, равномерно распределённых на промежутке [0; 1).

Заметим, что этот алгоритм – детерминированный (не использует генератора случайных чисел); понятие случайности возникает лишь при анализе времени его работы.

Сортировка вычерпыванием

1. промежуток [0; 1) делится на n равных частей, после чего для чисел из каждой части выделяется свой ящик-черпак (bucket), и n подлежащих сортировке чисел раскладываются по этим ящикам.
2. Поскольку числа равномерно распределены на отрезке [0;1), следует ожидать, что в каждом ящике их будет немного.
3. Теперь отсортируем числа в каждом ящике по отдельности и пройдёмся по ящикам в порядке возрастания, выписывая попавшие в каждый из них числа также в порядке возрастания.
4. Будем считать, что на вход подается n-элементный массив A, причем $0 \le A[i] < 1$ для всех i.
5. Используется также вспомогательный массив B[0. .n - 1], состоящий из списков, соответствующих ящикам.

Алгоритм

Алгоритм bucket_sort(A)

n ← length[A]

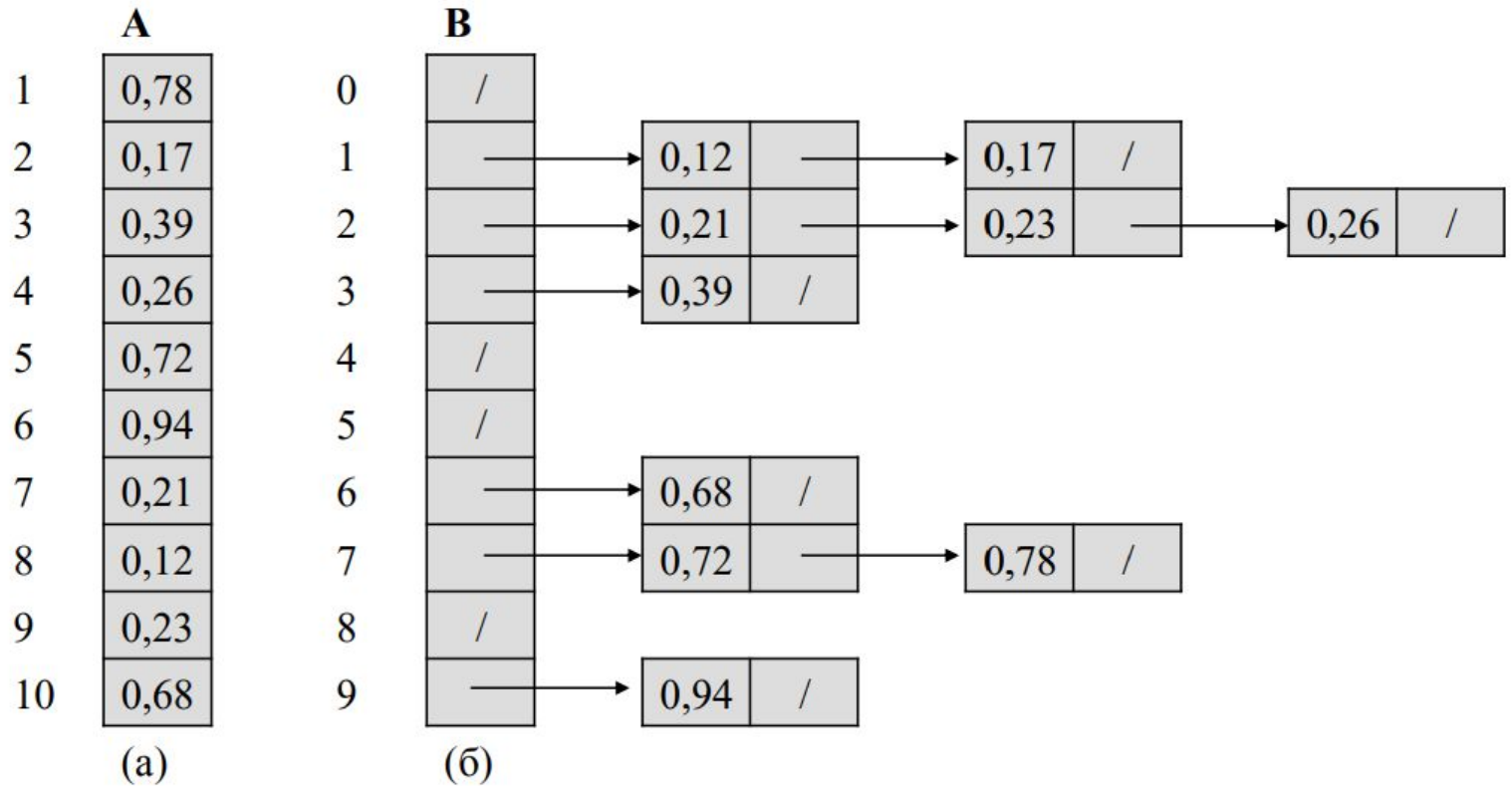
for i ← 1 to n

{ добавить A[i] к списку B[[nA[i]]] }

for i ← 0 to n-1

{отсортировать список B[i] сортировкой вставками} соединить списки B[0], B[1], …B[n - 1]

Пример


A
1 0,78
2 0,17
3 0,39
4 0,26
5 0,72
6 0,94
7 0,21
8 0,12
9 0,23
10 0,68
(а)
B
0 /
1 0,12 0,17 /
2 0,21 0,23 0,26 /
3 0,39 /
4 /
5 /
6 0,68 /
7 0,72 0,78 /
8 /
9 0,94 /
(б)

Время работы алгоритма

- Операции во всех строках, кроме пятой, требуют (общего) времени $\Theta(n)$. Просмотр всех ящиков также занимает время $\Theta(n)$. Таким образом, нам остаётся только оценить время сортировки вставками внутри ящиков.
- Пусть в ящик $B[i]$ попало n_i чисел (n_i – случайная величина).
- Поскольку сортировка вставками работает за квадратичное время, МО длительности сортировки чисел в ящике номер i есть $\Theta(M[n_i^2])$, а МО суммарного времени сортировки по всех ящиках есть $O(n)$
- Т.е. МО времени работы алгоритма сортировки вычёрпыванием в самом деле линейно зависит от количества чисел.

Время работы алгоритма

Т.к. числа распределены равномерно, а величины всех отрезков равны, вероятность того, что данное число попадет в ящик номер n, равна $1/n$.

Это аналогично примеру с шарами и урнами: у нас n шаров-чисел, n урн-ящиков, и вероятность попадания данного шара в данную урну равна $p=1/n$.

Поэтому числа n_i распределены биномиально:

$P(n_i = k) = C_n^k p^k (1-p)^{n-k}$,

$M[n_i] = np = 1$, и

$D[n_i] = np(1-p) = 1 - 1/n$.

Лексикографическая сортировка

Пусть S некоторое множество на котором задан

‹– линейный порядок.

Лексикографическим порядком на множестве S называется такое продолжение отношения ‹ на кортежи (списки, слова) элементов из S при котором

$$(s_1 , s_2 , \ldots, s_p) \lessdot (t_1 , t_2 , \ldots, t_q)$$

означает выполнение одного из условий:

существует целое j, что $s_j \lessdot t_j$ и для всех $i < j$ справедливо $s_i = t_i$.

$p \le q$ и $s_i = t_i$ при $1 \le i \le p$.

Лексикографическая сортировка

Очевидно, что любое целое число можно считать k-членным кортежем цифр от 0 до n – 1, где n - основание системы счисления, в которой рассматриваются цифры.

Рассмотрим сначала сортировку k-членных кортежей, элементы которых заключены в интервале от ‘a’ до ‘z’.

Сортировка вычёрпыванием

- Создадим исходную очередь A из n элементов, в которую занесем все рассматриваемые кортежи длины k.
- Организуем количество очередей, равное количеству маленьких латинских букв в алфавите. Для этого используем вспомогательный массив B[‘a’..’z’], состоящий из списков, соответствующих ящикам-черпакам.
- Количество итераций равно длине кортежей.
  - на i -ой итерации идет сортировка по k–i + 1 компоненте кортежей, т.е. некоторый кортеж A[j] удаляется из исходной очереди и добавляется в очередь B[A[j][k–i + 1]].
  - после выполнения i-ой итерации, в исходной очереди находится последовательность кортежей, полученная в результате “переписывания” (удаления и добавления) элементов всех непустых очередей, начиная с очереди, адрес начала которой находится в переменной B[‘a’], и заканчивая – B[‘z’].

Рекуррентное уравнение для алгоритма

- Пусть m — количество организованных очередей, тогда трудоемкость сортировки кортежей по некоторой компоненте есть Θ (n+m),
- т.к. количество кортежей – n и их можно распределить по очередям за время O(n); для сцепления m очередей требуется время O(m).
- Тогда рекуррентное уравнение будет иметь вид:

$$T(k) = (n + m) + T(k - 1),\ T(k) = \Theta(k\,(n + m)).$$

Алгоритм

```
Алгоритм lexicographical_sort
k ← length(A[1]);
for c ← ‘a’ to ‘z’
    { B[c].tail ← null; B[c].head ← null; }
for i ← 1 to k {
    for j ← 1 to n { B[A[j][k – i + 1]].moveToQueue(A[j]); }
    s ← 1;
    for c ← ‘a’ to ‘z’
    while B[c].head ≠ null
        { A[s] ← B[c].removeFromQueue; s++; }
}
```

Алгоритм сортировки кортежей разной длины

- Сначала сортируемые кортежи располагаются в порядке убывания длины.
- Пусть l_{max} - длина самого длинного кортежа, тогда сортировка вычёрпыванием производится l_{max} раз.
- На начальном этапе в исходную очередь A для сортировки помещаются только кортежи длины l_{max} и на первом этапе для сортировки используется только компонента l_{max} .
- После этого в исходную очередь A добавляются все кортежи длины $l_{max} – 1$, и для сортировки используется только компонента $l_{max} – 1$.
- На следующих этапах происходит сортировка соответствующей компоненты $l_{max} – 2, \ldots 1$, аналогичным образом.

Алгоритм сортировки кортежей разной длины

- Пусть l_i – длина i-ого кортежа,

m — количество организованных очередей.

Тогда описанный алгоритм упорядочивает кортежи разной длина за время $\Theta(l^* + m)$,

где $l^* = \sum_{i=1}^{n} l_i$

Оценка сложности

- Пусть l_i–длина i-ого кортежа, m —количество организованных очередей.

Тогда описанный алгоритм упорядочивает кортежи разной длина за время Θ(l*+ m), где $l^* = \sum_{i=1}^{m} l_i$

Пример

Дан массив:
Один, Два, Три, Четыре, Пять, Шесть,
Семь, Восемь, Девять, Десять

Упорядочиваем по длине:
6 – Четыре, Восемь, Девять, Десять

5 – Шесть

4 – Один, Пять, Семь

3 – Два, Три

1-я итерация

Вспомогательный массив ***B:***

е: Четыре

Ь: Восемь, Девять, Десять

2-я итерация

Вспомогательный массив ***В:***

м: Восемь

р: Четыре

т: Девять, Десять

ь: Шесть

3-я итерация

Вспомогательный массив ***В:***

е: Восемь

н: Один

т: Шесть

ь: Пять, Семь

ы: Четыре

я: Девять, Десять

4-я итерация

Вспомогательный массив ***в:***

а: Два

в: Девять

и: Один, Три

м: Семь

с: Восемь, Шесть, Десять

т: Пять, Четыре

5-я итерация

Вспомогательный массив ***в:***

в: Два

д: Один

е: Девять, Семь, Шесть, Десять, Четыре

о: Восемь

р: Три

я: Пять

6-я итерация

Вспомогательный массив ***в:***

в: Восемь

д: Два, Девять, Десять

о: Один

п: Пять

с: Семь

т: Три

ч: Четыре

ш: Шесть

Пирамидальная сортировка

Сортировка пирамидой использует бинарное сортирующее дерево.

Сортирующее дерево — это такое дерево, у которого выполнены условия:

1. Каждый лист имеет глубину либо d либо d−1, d — максимальная глубина дерева.
2. Значение в любой вершине не меньше (другой вариант — не больше) значения её потомков.

Пирамидальная сортировка

Этап 1. Выстраиваем элементы массива в виде сортирующего дерева

$A[i] \geq A[2i+1]$

$A[i] \geq A[2i+2]$

$0 \leq i < [n/2]$

Этот шаг требует $O(n)$ операций.

Пирамидальная сортировка

Этап 2. Будем удалять элементы из корня по одному за раз и перестраивать дерево, т.е. обменивать A[0] и A[n-1]. В результате обмена A[n-1] будет хранить максимальный элемент массива. Далее уменьшаем размер массива на 1(исключаем последний элемент) и переходим к этапу 1.

Продолжаем до тех пор, пока в дереве не останется 1 элемент.

Этот шаг требует O(nlogn) операций.

Достоинства сортировки

- Имеет доказанную оценку худшего случая O(nlogn).
- Сортирует на месте, то есть требует всего O(1) дополнительной памяти.

Недостатки сортировки

- Неустойчив — для обеспечения устойчивости нужно расширять ключ.
- На почти отсортированных массивах работает столь же долго, как и на хаотических данных.
- Из-за сложности алгоритма выигрыш получается только на больших n.

Пример

0	1	2	3	4	5	6
18	24	12	27	19	4	13
18	24	13	27	19	4	12
18	24	13	27	19	4	12
18	27	13	24	19	4	12
18	27	13	24	19	4	12
27	18	13	24	19	4	12
27	18	13	24	19	4	12
12	18	13	24	19	4	27

Пример

0	1	2	3	4	5	6
12	18	13	24	19	4	27
12	18	13	24	19	4	27
12	24	13	18	19	4	27
12	24	13	18	19	4	27
24	12	13	18	19	4	27
24	12	13	18	19	4	27
4	12	13	18	19	24	27

Пример

0	1	2	3	4	5	6
4	12	13	18	19	24	27
4	19	13	18	12	24	27
4	19	13	18	12	24	27
19	4	13	18	12	24	27
19	4	13	18	12	24	27
12	4	13	18	19	24	27
12	4	13	18	19	24	27
12	18	13	4	19	24	27
12	18	13	4	19	24	27
18	12	13	4	19	24	27
18	12	13	4	19	24	27
4	12	13	18	19	24	27

Пример

0	1	2	3	4	5	6
4	12	13	18	19	24	27
13	12	4	18	19	24	27
13	12	4	18	19	24	27
4	12	13	18	19	24	27
4	12	13	18	19	24	27
12	4	13	18	19	24	27
12	4	13	18	19	24	27
4	12	13	18	19	24	27
4	12	13	18	19	24	27
4	12	13	18	19	24	27