Алгоритмы сортировки

Задача

На вход алгоритма подаётся последовательность n элементов

$$a_1, a_2, \dots, a_n.$$

На выходе алгоритм должен вернуть перестановку исходной последовательности

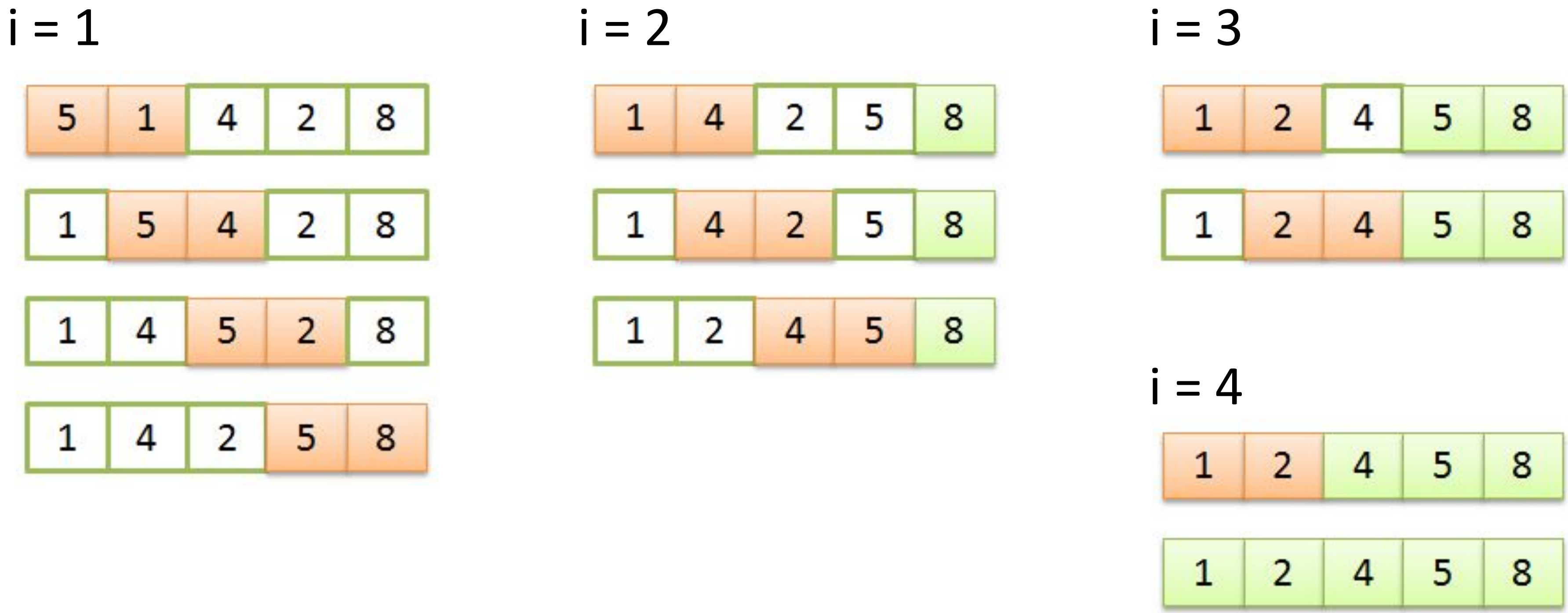
$$a'_1, a'_2, \dots, a'_n,$$

чтобы выполнялось следующее соотношение

$$a'_1 \le a'_2 \le \dots \le a'_n.$$

Сортировка пузырьком (bubble sort)

Пример


i = 1
5 1 4 2 8
1 5 4 2 8
1 4 5 2 8
1 4 2 5 8
i = 2
1 4 2 5 8
1 4 2 5 8
1 2 4 5 8
i = 3
1 2 4 5 8
1 2 4 5 8
i = 4
1 2 4 5 8
1 2 4 5 8

Алгоритм

```
for i = 1 to n-1
    for j = 1 to n-i
        if A[j] > A[j+1] then
            temp = A[j]
            A[j] = A[j+1]
            A[j+1] = temp
```

Сложность

```
for i = 1 to n-1
    for j = 1 to n-i
        if A[j] > A[j+1] then
            temp = A[j]
            A[j] = A[j+1]
            A[j+1] = temp
```

Количество операций

n

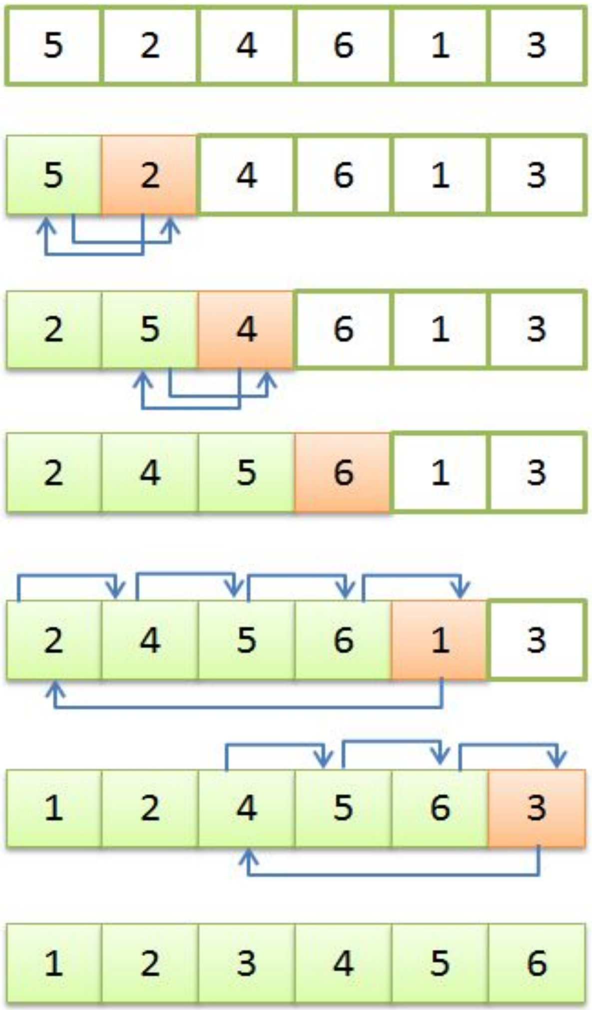
$$\sum_{1}^{n-1}(n-i+1)=\frac{(n+2)(n-1)}{2}=\frac{n^2+n-2}{2}$$

$$\sum_{1}^{n-1}(n-i)=\frac{(n-1+1)(n-1)}{2}=\frac{n^2-n}{2}$$

=> $T(n)=O(n^2).$

Сортировка вставками
(insertion sort)

Пример



Алгоритм

for j = 2 to n

key = A[j]

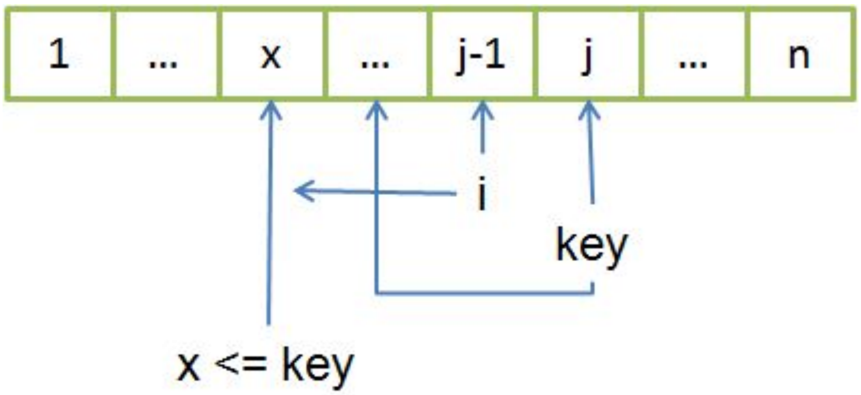
i = j – 1

while i > 0 and A[i] > key

A[i+1] = A[i]

i = i – 1

A[i+1] = key



Сложность

	Количество операций
for j = 2 to n	n
key = A[j]	n-1
i = j – 1	n-1
while i > 0 and A[i] > key	$\sum_{2}^{n} t_j$
A[i+1] = A[i]	$\sum_{2}^{n} (t_j - 1)$
i = i – 1	$\sum_{2}^{n} (t_j - 1)$
A[i+1] = key	n-1

Сложность

Лучший случай: массив отсортирован по возрастанию, тогда $t_j = 1,$

$$\sum_{2}^{n} t_j = n-1, \sum_{2}^{n} (t_j - 1) = 0.$$

$$T(n) = O(n).$$

Худший случай: массив отсортирован по убыванию, тогда $t_j = j,$

$$\sum_{2}^{n} t_j = \sum_{2}^{n} j = \frac{(2+n)(n-1)}{2} = \frac{n^2+n-2}{2},$$

$$\sum_{2}^{n} (t_j - 1) = \sum_{2}^{n} (j-1) = \frac{(1+n-1)(n-1)}{2} = \frac{n^2-n}{2}.$$

$$T(n) = O(n^2).$$

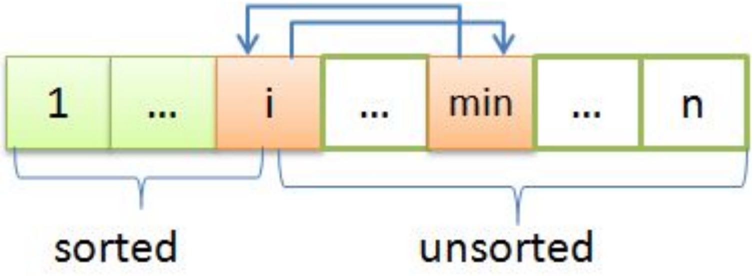
Сортировка выбором (selection sort)

Пример

i
min
4 9 7 6 2 3
i
min
2 9 7 6 4 3
i
min
2 3 7 6 4 9
i=min
2 3 4 6 7 9
i=min
2 3 4 6 7 9
2 3 4 6 7 9

Алгоритм

```
for i = 1 to n-1 do
    min = i
    for j = i+1 to n do
        if A[min] > A[j] then
            min = j
    if min<>i then
        temp = a[i]
        a[i] = a[min]
        a[min] = temp
```



Сложность

```
for i = 1 to n-1 do
    min = i
    for j = i+1 to n do
        if A[min] > A[j] then
            min = j
    if min<>i then
        temp = a[i]
        a[i] = a[min]
        a[min] = temp
```

Количество операций

n

n-1

$$\sum_{i=1}^{n-1}(n-i+1)=\frac{(n+2)(n-1)}{2}=\frac{n^2+n-2}{2}$$

$$\sum_{i=1}^{n-1}(n-i)=\frac{(n-1+1)(n-1)}{2}=\frac{n^2-n}{2}$$

n-1

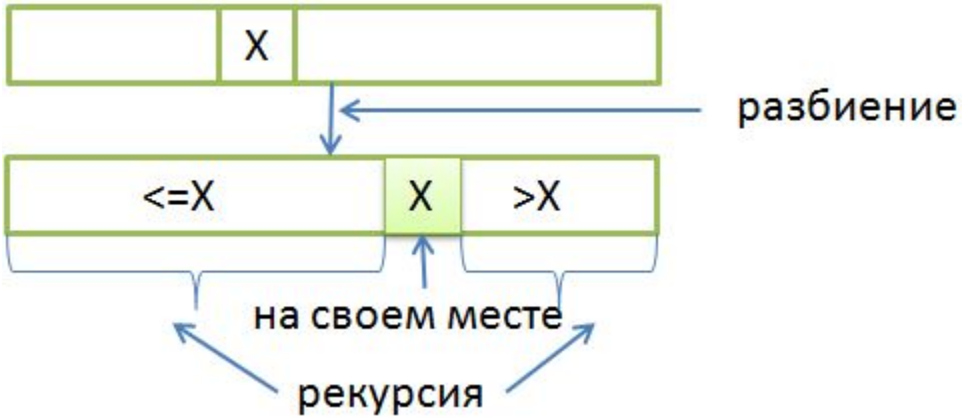
=> $T(n)=O(n^2).$

Быстрая сортировка (Хоара) (*QuickSort*)

Идея

Вход: массив A, индексы l и r, которые определяют подмассив для сортировки A[l...r].
Выход: отсортированный подмассив A[l...r].

1) Разбить массив A на 2 части: A[l...q-1] (где все элементы <= A[q]) и A[q+1...r] (где все элементы > A[q]). Элемент X=A[q] - опорный.
2) Рекурсивное решение задачи для A[l...q-1] и A[q+1...r].



Алгоритм

```
QuickSort(A,l,r)
If l<r then
    q = Partition(A,l,r)
    QuickSort(A,l,q-1)
    QuickSort(A, q+1,r)
```

```
Partition(A,l,r)
X = A[r]
i = l - 1
for j = l to r-1
    if A[j]<=X then
        i = i + 1
        A[i] <--> A[j]
A[r] <--> A[i+1]
return i+1
```

Алгоритм (случайный выбор опорного элемента)

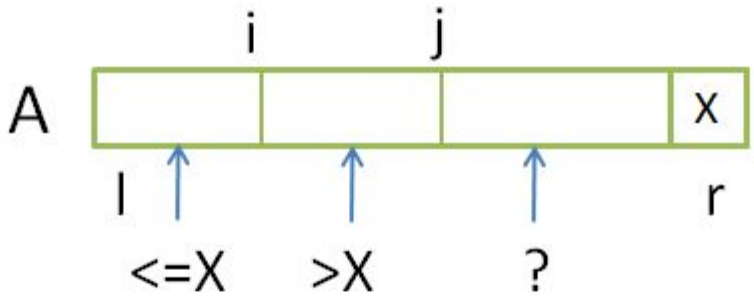
RandomQuickSort(A,l,r)

```
If l<r then
      q = RandomPartition(A,l,r)
      RandomQuickSort(A,l,q-1)
      RandomQuickSort(A, q+1,r)
```

RandomPartition(A,l,r)

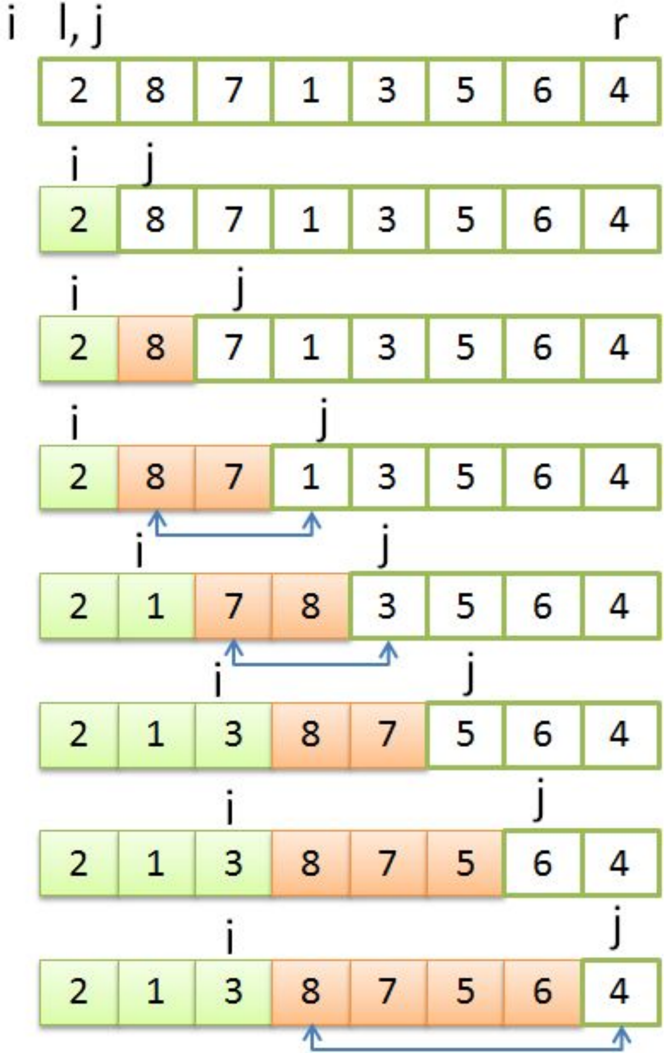
```
i=random(l,r)
A[i] <--> A[r]
Partition(A,l,r)
```

Процедура разбиения



1) X = A[r] - опорный элемент (разделитель)
2) A[l...i] <= X
3) A[i+1...j-1] >X
4) A[j...r-1] - элементы, которые еще не рассмотрены

Сложность T(n) = O(n)



A[j]=2 < X=4 => i=i+1, j=j+1

A[j]=8 > X=4 => j=j+1

A[j]=7 > X=4 => j=j+1

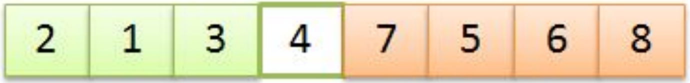
A[j]=1 < X=4 => i=i+1, j=j+1

A[j]=3 < X=4 => i=i+1, j=j+1

A[j]=5 > X=4 => j=j+1

A[j]=6 > X=4 => j=j+1

A[r] <--> A[i+1]



Сложность

Лучший случай. В наиболее сбалансированном варианте при каждой операции разделения массив делится на две почти одинаковые части. В результате общая сложность алгоритма $O(n*\log_2 n)$.

Средний случай. В среднем глубина рекурсии не превысит $2\log_{3/4}n$, что равно $O(\log n)$. А поскольку на каждом уровне рекурсии по-прежнему выполняется не более $O(n)$ операций, средняя сложность составит $O(n*\log n)$.

Худший случай. В самом несбалансированном варианте (в качестве опорного выбран максимальный или минимальный элемент) каждое разделение даёт два подмассива размерами 1 и n-1. Т.о. сложность $O(n^2)$.

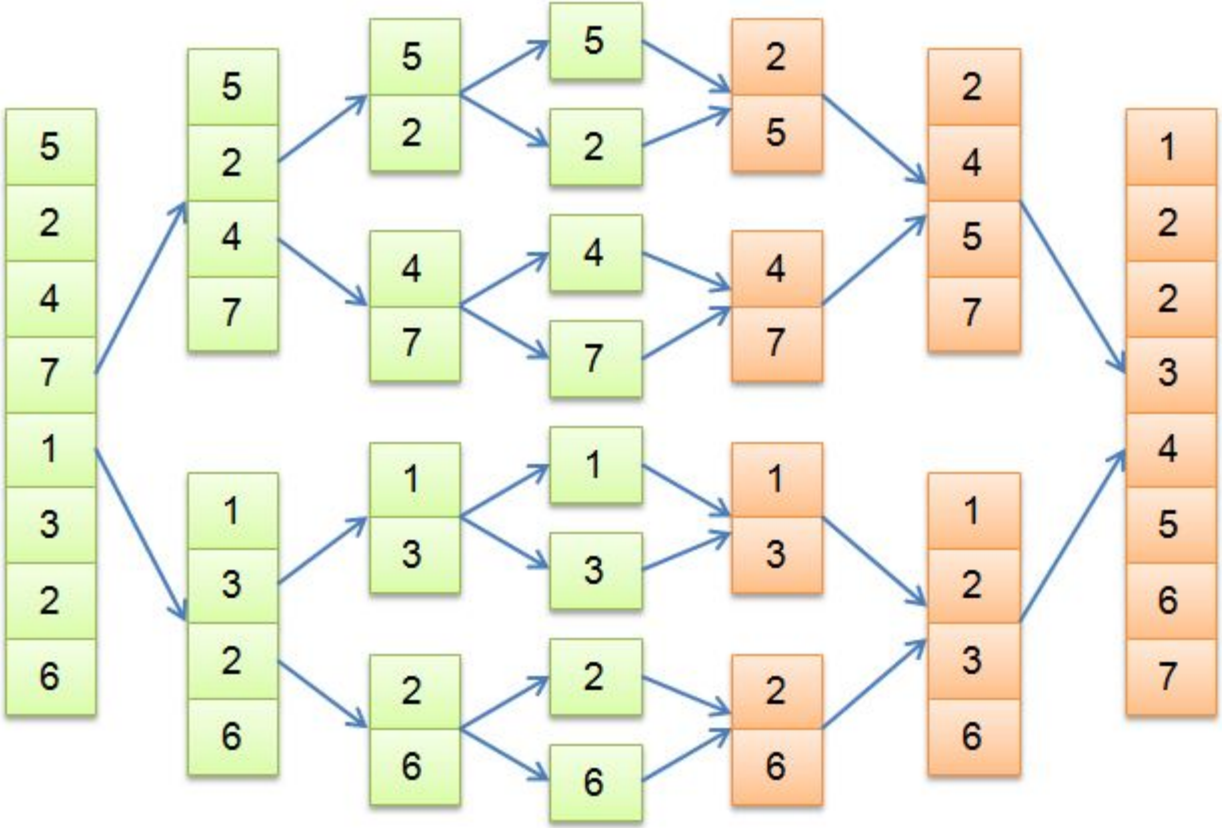
Сортировка слиянием
(*merge sort*)

Идея

Вход: массив A, индексы l и r, которые определяют подмассив для сортировки A[l...r].
Выход: отсортированный подмассив A[l...r].

1) Сортируемый массив разбивается на две части примерно одинакового размера
2) Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом
3) Два упорядочённых массива половинного размера сливаются в один.

Пример


5 2 4 7 1 3 2 6
5 2 4 7
1 3 2 6
5 2
4 7
1 3
2 6
5
2
4
7
1
3
2
6
2 5
4 7
1 3
2 6
2 4 5 7
1 2 3 6
1 2 2 3 4 5 6 7

Алгоритм *MergeSort*

MergeSort(A,l,r)

If l<r then

    q = [(l + r)/2]
    MergeSort(A,l,q)
    MergeSort(A,q+1,r)
    Merge(A,l,q,r)

Сложность алгоритма

$T(n) = O(n * \log n)$

$V(n) = O(n)$

Алгоритм *Merge*

```
Merge(A,l,q,r)
n1= q-l+1
n2= r-q
Создать массивы
      L[1...n1+1] и R[1...n2+1]
for i = 1 to n1
      L[i]=A[l+i-1]
for j = 1 to n2
      R[j]=A[q+j]
L[n1+1]= ∞
R[n2+1]= ∞
i= 1
j= 1
for k = l to r
      if L[i]<=R[j] then
      A[k]=L[i]
      i=i+1
      else
      A[k]=R[j]
      j=j+1
```