



# Алгоритмы и структуры данных

Лекция 2. Алгоритмы сортировки.

(с) Глухих Михаил Игоревич, [glukhikh@mail.ru](mailto:glukhikh@mail.ru)

# Корректность алгоритма

- $\text{Output} = \text{Alg}(\text{Input})$
- Математически
  - ЕСЛИ  $\text{Precondition}(\text{Input})$ , ТО  $\text{Postcondition}(\text{Alg}(\text{Input}))$
  - ЕСЛИ  $\text{Invariant}(\text{Input})$ , ТО  $\text{Invariant}(\text{Alg}(\text{Input}))$
- Как доказать корректность?

# Пример: Алгоритмы сортировки

## ▶ Простые

- ▶ вставками (включениями)
- ▶ выбором
- ▶ обменом (пузырьком)

## ▶ Сложные

- ▶ слияниями
- ▶ Хоара, или быстрая сортировка
- ▶ двоичной кучей, или пирамидальная сортировка

# Простые сортировки

- ▶ **Сортировка вставками**

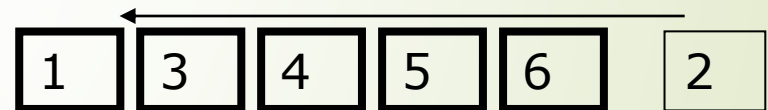
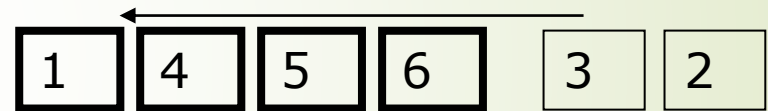
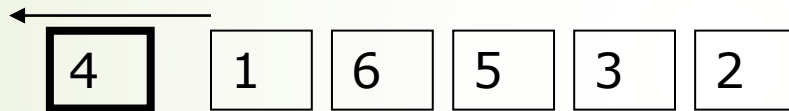
# Пример: сортировка вставками

## ► Псевдокод

```
for j = 1 to list.size - 1:  
    key = list[j]  
    i = j - 1  
    while i >= 0 && list[i] > key:  
        list[i+1] = list[i]  
        i--  
    list[i+1] = key
```

## Сортировка вставками

- На каждой итерации вставляем очередной элемент в отсортированную часть массива



# Сортировка вставками: принцип рекуррентности

- ▶ Рекуррентный (индуктивный) алгоритм
  - ▶ База: 1 элемент всегда упорядочен

# Сортировка вставками: принцип рекуррентности

- ▶ Рекуррентный (индуктивный) алгоритм
  - ▶ База: 1 элемент всегда упорядочен
  - ▶ Переход: от  $j-1$  упорядочены к  $j$  упорядочены
    - ▶ Пусть  $j-1$  элементов уже упорядочено
    - ▶ Берём элемент номер  $j$
    - ▶ Находим среди  $j-1$  элементов такую пару  $A[i]$  и  $A[i+1]$ , что  $A[i] \leq A[j] \leq A[i+1]$ ;  
либо, если её нет, то одно из двух:  $A[j] \leq A[0]$  или  $A[j-1] \leq A[j]$
    - ▶ Вставляем элемент номер  $j$  между этой парой



# Сортировка вставками: принцип рекуррентности

- ▶ Рекуррентный (индуктивный) алгоритм
  - ▶ База: 1 элемент всегда упорядочен
  - ▶ Переход: от  $j-1$  упорядочены к  $j$  упорядочены
    - ▶ Пусть  $j-1$  элементов уже упорядочено
    - ▶ Берём элемент номер  $j$
    - ▶ Находим среди  $j-1$  элементов такую пару  $A[i]$  и  $A[i+1]$ , что  $A[i] \leq A[j] \leq A[i+1]$ ;  
либо, если её нет, то одно из двух:  $A[j] \leq A[0]$  или  $A[j-1] \leq A[j]$
    - ▶ Вставляем элемент номер  $j$  между этой парой
  - ▶ Результат: все  $N$  элементов упорядочены

## Сортировка вставками: $O(?)$

- ▶ Пусть размер списка  $N$   
for  $j = 1$  to  $list.size - 1$ :  
     $key = list[j]$   
     $i = j - 1$   
    while  $i \geq 0 \ \&\& \ list[i] > key$ :  
         $list[i+1] = list[i]$   
         $i--$   
     $list[i+1] = key$

# Интересные вопросы про $O(f(n))$

- ▶ Что это такое?

# Интересные вопросы про $O(f(n))$

- Что это такое?
- Верно ли, что:
  - $O(n)+O(n)=2*O(n)$
  - $O(n)+O(n)=O(2*n)$
  - $O(n)+O(n)=O(n)$

# Интересные вопросы про $O(f(n))$

- Что это такое?
- Верно ли, что:
  - $O(n)+O(n)=2*O(n)$
  - $O(n)+O(n)=O(2*n)$
  - $O(n)+O(n)=O(n)$
  - $O(n)*O(n)=O^2(n)$
  - $O(n)*O(n)=O(n^2)$
  - $O(n)*O(n)=O(n)$

# Математика: варианты оценки затрат

- $f(n) = O(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) < Cg(n)$ 
  - ОЦЕНКА СВЕРХУ ~ РАСТЁТ МЕДЛЕННЕЕ

# Математика: варианты оценки затрат

- $f(n) = O(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) < Cg(n)$ 
  - ОЦЕНКА СВЕРХУ ~ РАСТЁТ МЕДЛЕННЕЕ
- $f(n) = \Omega(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) > Cg(n)$ 
  - ОЦЕНКА СНИЗУ ~ РАСТЁТ БЫСТРЕЕ

## Математика: варианты оценки затрат

- $f(n) = O(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) < Cg(n)$ 
  - ОЦЕНКА СВЕРХУ ~ РАСТЁТ МЕДЛЕННЕЕ
- $f(n) = \Omega(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) > Cg(n)$ 
  - ОЦЕНКА СНИЗУ ~ РАСТЁТ БЫСТРЕЕ
- $f(n) = \Theta(g(n))$ : одновременно  $O$  и  $\Omega$ 
  - ОЦЕНКА С ДВУХ СТОРОН ~ РАСТЁТ С ТОЙ ЖЕ СКОРОСТЬЮ



# Математика: варианты оценки затрат

- $f(n) = O(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) < Cg(n)$ 
  - ОЦЕНКА СВЕРХУ ~ РАСТЁТ МЕДЛЕННЕЕ
- $f(n) = \Omega(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) > Cg(n)$ 
  - ОЦЕНКА СНИЗУ ~ РАСТЁТ БЫСТРЕЕ
- $f(n) = \Theta(g(n))$ : одновременно  $O$  и  $\Omega$ 
  - ОЦЕНКА С ДВУХ СТОРОН ~ РАСТЁТ С ТОЙ ЖЕ СКОРОСТЬЮ
- $f(n) = o(g(n))$ : для любого (сколь угодно малого) коэффициента  $C$  существует  $n_0$ : при  $n > n_0$ ...
  - РАСТЁТ ЗНАЧИТЕЛЬНО МЕДЛЕННЕЕ

# Математика: варианты оценки затрат

- ▶ Программисты, как правило, используют  $O$ 
  - ▶ Действительно, важна в первую очередь верхняя граница
  - ▶ Но найти её важно точно
    - ▶ и поэтому часто имеется в виду оценка с двух сторон ( $\Theta$ )

## Сортировка вставками: $O(?)$

- ▶ Пусть размер списка  $N$   
for  $j = 1$  to  $list.size - 1$ :  
     $key = list[j]$   
     $i = j - 1$   
    while  $i \geq 0$  &&  $list[i] > key$ :  
         $list[i+1] = list[i]$   
         $i--$   
     $list[i+1] = key$

## Сортировка вставками: $O(?)$

- ▶ Пусть размер списка  $N$

```
for j = 1 to list.size - 1:           // N-1
    key = list[j]                     // N-1
    i = j - 1                         // N-1
    while i >= 0 && list[i] > key:    // N-1
        list[i+1] = list[i]         // 0..N(N-1)/2
        i--
    list[i+1] = key                   // N-1
```

## Сортировка вставками: $O(?)$

- ▶ Пусть размер списка  $N$

```
for j = 1 to list.size - 1:           // N-1
    key = list[j]                     // N-1
    i = j - 1                         // N-1
    while i >= 0 && list[i] > key:     // N-1
        list[i+1] = list[i]          // 0..N(N-1)/2
        i--
    list[i+1] = key                   // N-1
```

- ▶  $5(N-1) \dots 5(N-1) + N(N-1)/2$

# Затраты времени

- ▶ В наихудшем случае (обычно важнее всего)

## Затраты времени

- В наихудшем случае (обычно важнее всего)
- В среднем случае (важно, если наихудший случай происходит крайне редко)

## Затраты времени

- В наихудшем случае (обычно важнее всего)
- В среднем случае (важно, если наихудший случай происходит крайне редко)
- В наилучшем случае (как правило, неважно)



## Затраты времени

- В наихудшем случае (обычно важнее всего)
- В среднем случае (важно, если наихудший случай происходит крайне редко)
- В наилучшем случае (как правило, неважно)
- Каковы наилучший и наихудший случай для сортировки вставками?

## Затраты времени

- В наихудшем случае (обычно важнее всего)
- В среднем случае (важно, если наихудший случай происходит крайне редко)
- В наилучшем случае (как правило, неважно)
- Каковы наилучший и наихудший случай для сортировки вставками?
- Как разработать алгоритм с минимальными затратами в наилучшем случае?

## Основные характеристики алгоритмов сортировки

- ▶ Трудоёмкость  $T=O(\dots)$  – в среднем и худшем случае
- ▶ Ресурсоёмкость  $R=O(\dots)$ 
  - ▶ Для  $R=O(1)$  говорят «сортировка на месте»

## Основные характеристики алгоритмов сортировки

- ▶ Трудоёмкость  $T=O(\dots)$  – в среднем и худшем случае
- ▶ Ресурсоёмкость  $R=O(\dots)$ 
  - ▶ Для  $R=O(1)$  говорят «сортировка на месте»
- ▶ Устойчивость – изменяется ли порядок равных элементов в списке
  - ▶ Бывает важна в некоторых случаях, например, при индексации баз данных
  - ▶ Точно неважна, если элемент – это только ключ сравнения
- ▶ Операции сравнения – используются или нет

# Алгоритмы сортировки

- ▶ Простые

- ▶  $O(N^2)$ , где  $N$  – число элементов

- ▶ Сложные

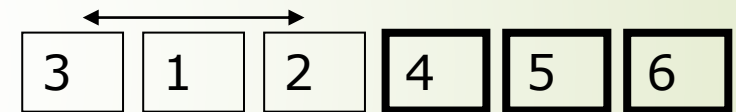
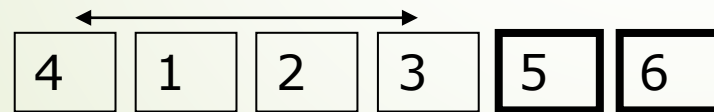
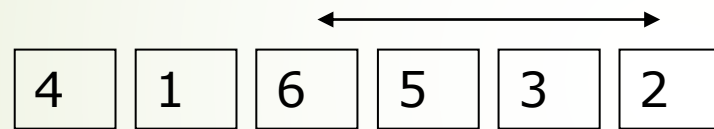
- ▶  $O(N \log_2 N)$ , где  $N$  – число элементов; для больших значений  $N$  существенно быстрее простых методов

# Простые сортировки

- Сортировка вставками
- **Сортировка выбором**

# Простые алгоритмы сортировки

- Сортировка **выбором** – на каждой итерации находим максимальный элемент и меняем с последним



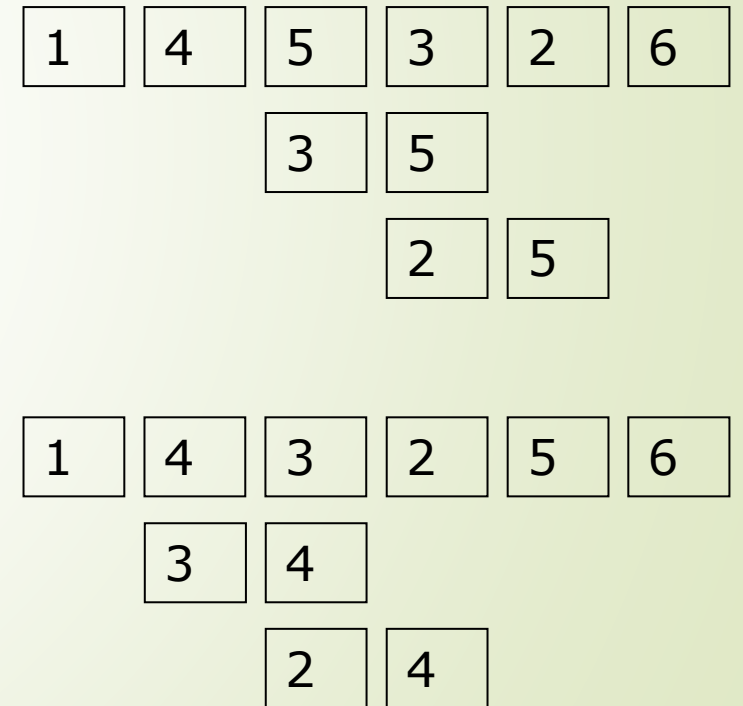
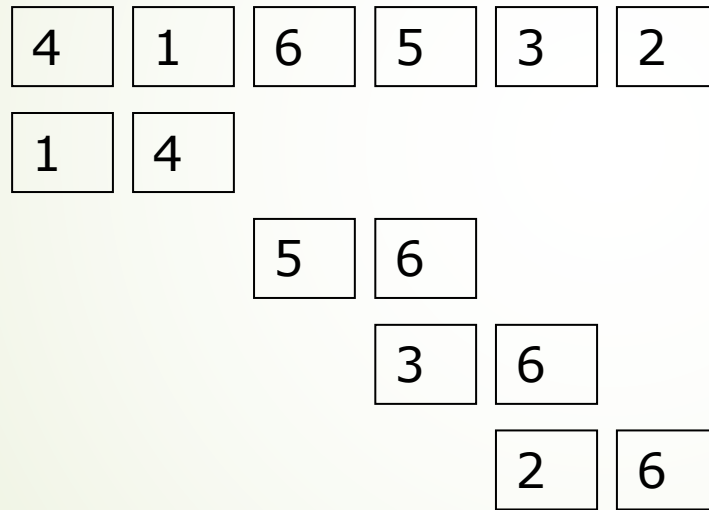
# Простые сортировки

- Сортировка вставками
- Сортировка выбором
- **Сортировка обменом (пузырьком)**



# Простые алгоритмы сортировки

- Сортировка **обменом** – последовательно меняем местами элементы в паре, если их порядок неверен



## Устойчивые сортировки

- ▶ Пузырьком  $T=O(N^2)$ ,  $R=O(1)$
- ▶ Вставками  $T=O(N^2)$ ,  $R=O(1)$
- ▶ Слиянием  $T=O(N \log N)$ ,  $R=O(N)$

# Неустойчивые сортировки

- ▶ Выбором  $T=O(N^2)$ ,  $R=O(1)$ 
  - ▶ Этот алгоритм уже на первом шаге меняет местами минимальный элемент в списке с первым, что и приводит к неустойчивости
  - ▶ Пример:  $(2A, 2B, 1) \rightarrow (1, 2B, 2A)$  (считаем  $2A == 2B$ )

# Сложные сортировки

- ▶ **Сортировка слиянием**

# Принцип декомпозиции

- Или “Разделяй и властвуй”
  - Рекурсивный подход
- Разделение
  - Задача делится на  $k$  меньших частей (часто на две)
- Властвование
  - Каждая из частей решается отдельно (если требуется, таким же образом, как и первая)
- Комбинирование
  - Затем результаты объединяются

# Сортировка слиянием: принцип

- **Разделение**
  - Делим  $n$  элементов на левую и правую половину
- **Властвование**
  - Упорядочиваем каждую половину отдельно
- **Комбинирование**
  - Составляем две половины вместе

## Пример: сортировка слиянием

- ▶ Псевдокод комбинирования (left, right → list)

```
li = 0
ri = 0
for i = 0 to list.size - 1:
    if left[li] <= right[ri]: // ~~~ IOBE
        list[i] = left[li++]
    else:
        list[i] = right[ri++]
```

# Сортировка слиянием: $O(?)$

► Псевдокод

```
sort(list, p = 0, q = list.size): // S(N)
  m = (p + q) / 2                // 1
  sort(list, p, m)                // S(N/2)
  sort(list, m, q)                // S(N/2)
  merge(list, p, m, q)           // 5N
```



# Сортировка слиянием: $O(?)$

► Псевдокод

```
sort(list, p = 0, q = list.size): // S(N)
  m = (p + q) / 2                // 1
  sort(list, p, m)               // S(N/2)
  sort(list, m, q)               // S(N/2)
  merge(list, p, m, q)           // 5N
```

# Сортировка слиянием: $O(?)$

► Псевдокод

```
sort(list, p = 0, q = list.size): // S(N)
  m = (p + q) / 2                // 1
  sort(list, p, m)               // S(N/2)
  sort(list, m, q)               // S(N/2)
  merge(list, p, m, q)           // 5N
```

►  $S(N) = 2S(N/2) + 5N + 1$

► Верно ли:  $O(N) = 2O(N/2) + 5N + 1 = O(N) + O(N) = O(N)$  ?

# Сортировка слиянием: $O(?)$

► Псевдокод

```
sort(list, p = 0, q = list.size): // S(N)
    m = (p + q) / 2                // 1
    sort(list, p, m)               // S(N/2)
    sort(list, m, q)              // S(N/2)
    merge(list, p, m, q)          // 5N
```

►  $S(N) = 2S(N/2) + 5N + 1$

► Верно ли:  $O(N) = 2O(N/2) + 5N + 1 = O(N) + O(N) = O(N)$  ?

► А так:  $S(N) = 2S(N/2) + O(N)$  ?

## Сортировка слиянием: $O(?)$

➤ Псевдокод

```
sort(list, p = 0, q = list.size): // S(N)
    m = (p + q) / 2                // 1
    sort(list, p, m)               // S(N/2)
    sort(list, m, q)               // S(N/2)
    merge(list, p, m, q)           // 5N
```

➤  $S(N) = 2S(N/2) + 5N + 1$

➤ Верно ли:  $O(N) = 2O(N/2) + 5N + 1 = O(N) + O(N) = O(N)$  ?

➤ А так:  $S(N) = 2S(N/2) + O(N)$  ?

➤ Как насчёт наилучшего и наихудшего варианта?

# Как решать (проверять): принцип

- ▶ Рекуррентное соотношение
  - ▶  $S(N) = 2S(N/2) + O(N)$

## Как решать (проверять): принцип

- Рекуррентное соотношение

- $S(N) = 2S(N/2) + O(N)$

- Предположим:  $S(N) = O(N) \sim C * N$

- $C * N = 2C * N/2 + D * N = C * N + D * N$

- НЕВЕРНО

## Как решать (проверять): принцип

- Рекуррентное соотношение

- $S(N) = 2S(N/2) + O(N)$

- Предположим:  $S(N) = O(N) \sim C * N$

- $C * N = 2C * N/2 + D * N = C * N + D * N$

- НЕВЕРНО

- Предположим:  $S(N) = O(N^2) \sim C * N^2$

- $C * N^2 = 2C * N^2/4 + D * N = C * N^2/2 + D * N$

- НЕВЕРНО

# Как решать (проверять): принцип

- ▶ Рекуррентное соотношение
  - ▶  $S(N) = 2S(N/2) + O(N)$
- ▶ Предположим:  $S(N) = O(N) \sim C * N$ 
  - ▶  $C * N = 2C * N/2 + D * N = C * N + D * N$
  - ▶ НЕВЕРНО
- ▶ Предположим:  $S(N) = O(N^2) \sim C * N^2$ 
  - ▶  $C * N^2 = 2C * N^2/4 + D * N = C * N^2/2 + D * N$
  - ▶ НЕВЕРНО
- ▶ Предположим:  $S(N) = O(N \lg N) \sim C * N \lg N$ 
  - ▶  $C * N \lg N = 2C * N \lg(N/2) / 2 + D * N = C * N \lg N - C * N \lg 2 + D * N$
  - ▶ ВЕРНО



# Сложные сортировки

- Сортировка слиянием
- **Быстрая сортировка (сортировка Хоара)**

# Быстрая сортировка

- ▶ Используется принцип декомпозиции «Разделяй и Властвуй»
- ▶ На каждом шаге массив  $A[\text{Min} \dots \text{Max}]$  путём перестановки элементов разбивается на два подмассива  $A[\text{Min} \dots R]$  и  $A[R+1 \dots \text{Max}]$ , таких, что
  - ▶  $A[J] \leq A[R]$  для  $J \leq R$
  - ▶  $A[J] \geq A[R]$  для  $J > R$
- ▶ Каждый из подмассивов сортируется рекурсивно

# Сортировка Хоара

- ▶ На каждом шаге алгоритма выбирается **разделяющий элемент** массива (в идеале – его **медиана**), после чего элементы переставляются так, чтобы меньшие оказались слева, а большие – справа

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

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

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

# Сортировка Хоара

- ▶ На каждом шаге алгоритма выбирается **разделяющий элемент** массива (в идеале – его **медиана**), после чего элементы переставляются так, чтобы меньшие оказались слева, а большие – справа

<b>4</b>	1	6	5	7	3	2
----------	---	---	---	---	---	---

2	1	<b>6</b>	5	7	<b>3</b>	<b>4</b>
---	---	----------	---	---	----------	----------

2	1	<b>3</b>		<b>5</b>	7	6	<b>4</b>
---	---	----------	--	----------	---	---	----------

# Псевдокод: разбиение

```
PARTITION(A, Min, Max):  
  X = A[RANDOM(Min, Max)]  
  L=Min, R=Max  
  while L <= R:  
    while A[L] < X:  
      L++  
    while A[R] > X:  
      R--  
    if (L <= R):  
      Swap A[L] with A[R]  
      L++, R--
```

- **Инвариант:** в начале каждой итерации элементы #Min...#L-1  $\leq X$ , #R+1...#Max  $\geq X$

# Псевдокод: быстрая сортировка

```
QUICK-SORT (A, Min, Max) :  
  if (Min < Max) :  
    R = PARTITION (A, Min, Max)  
    QUICK-SORT (A, Min, R)  
    QUICK-SORT (A, R+1, MAX)
```

## Производительность быстрой сортировки

- ▶ Наихудший случай:  $R == \text{Min}$  или  $R == \text{Max}$ 
  - ▶  $T(N) = T(N-1) + O(N)$
- ▶ Наилучший случай:  $R == (\text{Min} + \text{Max}) / 2$ 
  - ▶  $T(N) = 2T(N/2) + O(N)$
- ▶ 20 / 80:  $R = 0.2\text{Min} + 0.8\text{Max}$ 
  - ▶  $T(N) = T(0.8N) + T(0.2N) + O(N)$
- ▶ «Средний» ~ чередование наилучшего и наихудшего

## Сложные сортировки

- Сортировка слиянием
- Быстрая сортировка (сортировка Хоара)
- **Пирамидальная сортировка**



# Итоги

- Рассмотрели наиболее распространённые виды сортировок
- Далее
  - Прочие виды сортировок
  - Задачи на построение алгоритмов