

# АЛГОРИТМЫ СОРТИРОВКИ И ПОНЯТИЕ АНАЛИЗА АЛГОРИТМОВ



# АНАЛИЗ АЛГОРИТМОВ

- Анализ алгоритмов заключается в определении требуемых для его выполнения ресурсов.
- Примеры ресурсов: память, необходимое аппаратное обеспечение, пропускная способность сети, время выполнения.

# ВРЕМЯ РАБОТЫ АЛГОРИТМА

- Время работы программы обычно представляется в виде функции от размера входных данных.
- Время работы алгоритма для некоторых входных данных – это количество элементарных операций или шагов, которые необходимо выполнить.

# СПОСОБЫ ИЗМЕРЕНИЯ РАЗМЕРА ВХОДНЫХ ДАННЫХ

- Размер входных данных измеряется по-разному в зависимости от задачи:
  - В сортировках массива – это число элементов массива,
  - В алгоритмах на графах – это может быть несколько чисел (число вершин и число ребер графа),
  - В перемножении целых чисел – число битов, необходимых для представления входных данных в двоичных обозначениях.

# СОРТИРОВКА ВСТАВКАМИ (1)

- Вход алгоритма – массив из  $n$  чисел  $A[1..n]$

Insertion\_sort( $A$ )

  for  $j := 2$  to length[ $A$ ]

    do key :=  $A[j]$

      //вставка элемента  $A[j]$  в отсортированную послед-ть  $A[1..j-1]$ :

$i := j - 1$

      while  $i > 0$  и  $A[i] > \text{key}$

        do  $A[i+1] := A[i]$

$i := i - 1$

$A[i+1] := \text{key}$

# СОРТИРОВКА ВСТАВКАМИ (2)

- 

Insertion\_sort(A)

```
for j := 2 to length[A]
```

```
  do key := A[j]
```

```
    //вставка элемента A[j] в отсортированную послед-ть A[1..j-1]:
```

```
    i := j - 1
```

```
    while i > 0 и A[i] > key
```

```
      do A[i+1] := A[i]
```

```
      i := i - 1
```

```
    A[i+1] := key
```

число инструкций:  $n$

время 1 инстр.:  $c_1$

число инструкций:  $n-1$

время 1 инстр.:  $c_2$

число инструкций:  $n-1$

время 1 инстр.:  $c_3$

число инструкций:  $\sum_{j=2}^n t_j$

время 1 инстр.:  $c_4$

число инструкций:  $\sum_{j=2}^n (t_j - 1)$

время 1 инстр.:  $c_5$

число инструкций:  $\sum_{j=2}^n (t_j - 1)$

время 1 инстр.:  $c_6$

число инструкций:  $n-1$

время 1 инстр.:  $c_2$

время работы алгоритма  $T(n) = ?$

# СОРТИРОВКА ВСТАВКАМИ (3)

- 

Insertion\_sort(A)

```
for j:= 2 to length[A]
```

```
do key := A[j]
```

```
//вставка элемента A[j] в отсортированную послед-ть A[1..j-1]:
```

```
i := j - 1
```

```
while i > 0 и A[i] > key
```

```
do A[i+1] := A[i]
```

```
i := i - 1
```

```
A[i+1] := key
```

число инструкций:  $n$                       время 1 инстр.:  $c_1$

число инструкций:  $n-1$                       время 1 инстр.:  $c_2$

число инструкций:  $n-1$                       время 1 инстр.:  $c_3$

число инструкций:  $\sum_{j=2}^n t_j$                       время 1 инстр.:  $c_4$

число инструкций:  $\sum_{j=2}^n (t_j - 1)$                       время 1 инстр.:  $c_5$

число инструкций:  $\sum_{j=2}^n (t_j - 1)$                       время 1 инстр.:  $c_6$

число инструкций:  $n-1$                       время 1 инстр.:  $c_2$

время работы алгоритма (в худшем и среднем случаях)  $T(n) = an^2 + bn + c = O(n^2)$  //  $O$  - временная сложность алгоритм  
в благоприятном случае  $T(n) = O(n)$

# МЕТОД ДЕКОМПОЗИЦИИ “РАЗБИВАЙ И ВЛАСТВУЙ”

- **1 этап: Разделение.** Сложная задача разбивается на несколько более простых (которые подобны исходной задаче, но имеют меньший объем).
- **2 этап. Властвование.** Рекурсивное решение вспомогательных задач. Когда объем подзадачи достаточно мал, то она решается непосредственно.
- **3 этап. Комбинирование.** Решение исходной задачи из решений вспомогательных задач.



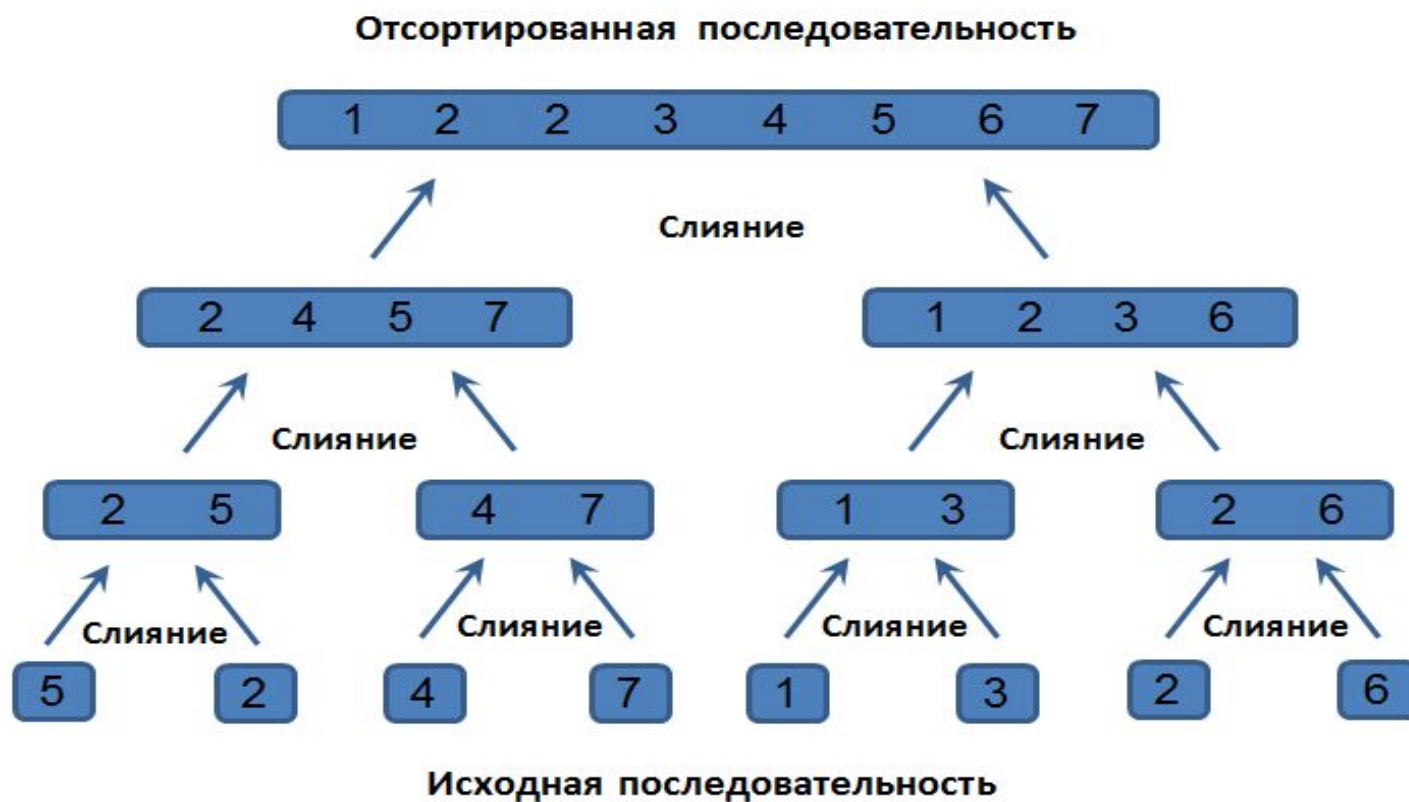
# СОРТИРОВКА СЛИЯНИЕМ (1)

- **Разделение:** сортируемая последовательность, состоящая из  $n$  элементов, разбивается на две меньшие последовательности, каждая из которых содержит  $n/2$  элементов.
- **Властвование:** сортировка обеих вспомогательных последовательностей методом слияния.
- **Комбинирование:** слияние двух отсортированных последовательностей для получения окончательного результата.

# СОРТИРОВКА СЛИЯНИЕМ (2)

- Merge\_sort(A, p, r) // сортировка элементов в подмассиве A[p, q]
  - if (p < r)
  - then q := [(p+r) / 2]
  - Merge\_sort(A, p, q)
  - Merge\_sort(A, q+1, r)
  - Merge(A, p, q, r) // слияние двух отсортированных подмассивов

# СОРТИРОВКА СЛИЯНИЕМ (3)



# СОРТИРОВКА СЛИЯНИЕМ (4)

Merge\_sort (A, p, q, r) //Слияние двух упорядоченных массивов

- Создаются два временных массива L и R, в которые копируются последовательности  $A[1..q-p+1]$  и  $A[r-q, r]$  соответственно
- Указатели текущей позиции ставятся на первые элементы обоих массивов
- В итоговом массиве A указатель ставится на позицию  $j := p$ .
- Пока один из массивов не закончится:
  - Сравниваются значения текущих элементов, берется меньший из них и копируется в  $A[j]$ .
  - В массиве, из которого бы взят элемент, сдвигается указатель текущей позиции
  - В итоговом массиве указатель сдвигается на следующую позицию:  $j = j + 1$
- Все элементы оставшегося массива копируются в  $A[j..r]$

# СОРТИРОВКА СЛИЯНИЕМ (5)

- Сложность алгоритма:  $T(n) = O(n \lg n)$