Семинар 2

Сортировки

1.Метод пузырька

- Bubble_Sort (A, n)
- Вход: массив A длины n
- Выход: массив с теми же числами, отсортированными от наименьшего до наибольшего
- **for** j:=1 **to** n-1
- sorted:=FALSE
- **for** i:=1 **to** n-j
- **if** A[i]>A[i+1] //инверсия
- *обмен* A[i] с A[i+1]
- sorted=TRUE
- **endfor**
- **i**f sorted=FALSE **then** EXIT // если массив уже отсортирован, выйти из
- // цикла
- **endfor**

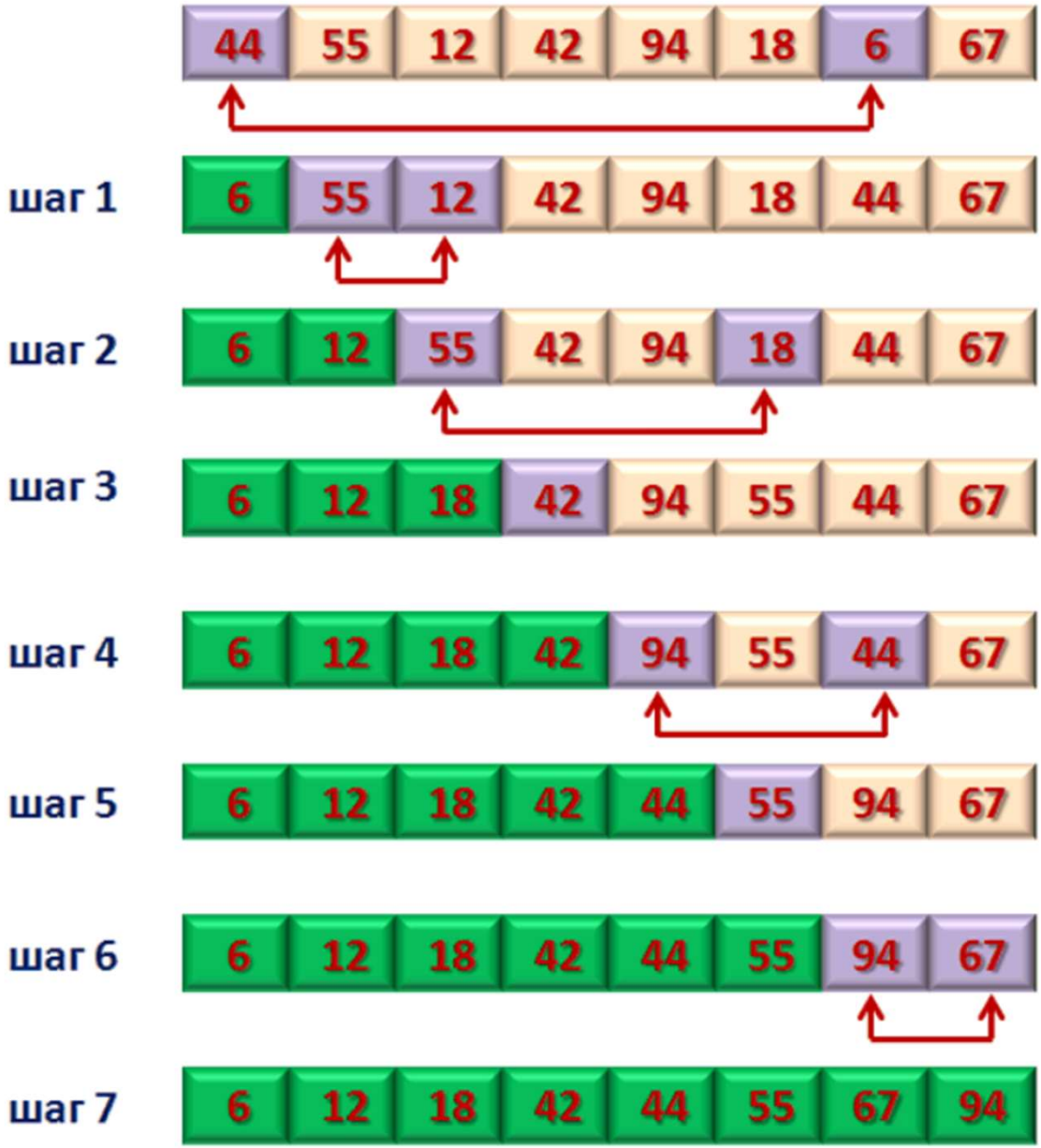
Сложность

Пример; [**10, 4, 14, 25, 77, 2**]

Корректность

Инвариант

2. Сортировка выбором (пример)



Сортировка выбором

```
Select_Sort (A, n)
  for i:=1 to n-1
     min:=i
     for j:= i+1 to n
        if A[i]>A[j] then
           min:=j
           обмен  A[i]  с A[min]
        endif
endfor
```

Разделяй и властвуй

MERGESORT

Вход: массив A из n разных целых чисел.

Выход: массив с теми же самыми целыми числами, отсортированными от наименьшего до наибольшего.

```
// базовые случаи проигнорированы
C := рекурсивно отсортировать первую половину A
D := рекурсивно отсортировать вторую половину A
вернуть Merge (C, D)
```

MERGE

Вход: отсортированные массивы C и D (длиной $n/2$ каждый).

Выход: отсортированный массив B (длиной n).

Упрощающее допущение: n — четное.

```
1 i := 1
2 j := 1
3 for k := 1 to n do
4     if C[i] < D[j] then
5         B[k] := C[i]          // заполнить выходной массив
6         i := i + 1                    // прирастить i
7     else                                // D[j] < C[i]
8         B[k] := D[j]
9         j := j + 1
```

ПАРАДИГМА «РАЗДЕЛЯЙ И ВЛАСТВУЙ»

1. *Разделить* входные данные на более мелкие подзадачи.
2. *Решить* подзадачи рекурсивным методом.
3. *Объединить* решения подзадач в решение исходной задачи.

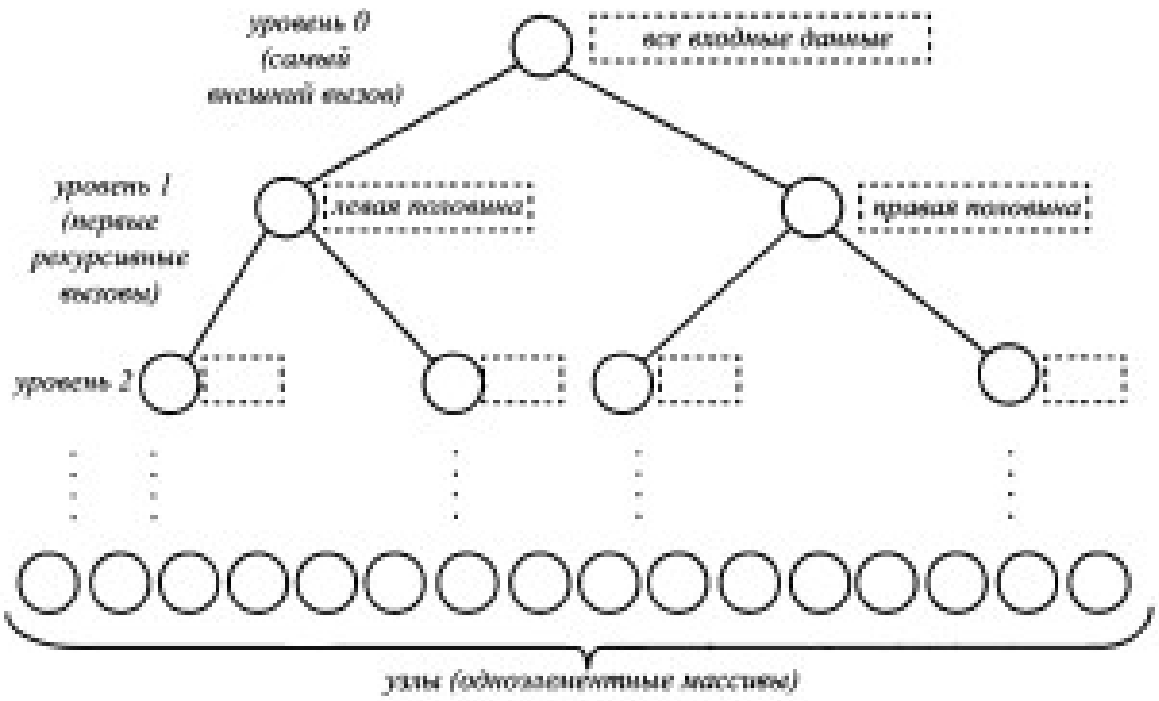


Рис. 1.5. Дерево рекурсии для алгоритма MergeSort. Узлы соответствуют рекурсивным вызовам. Уровень 0 соответствует самому первому вызову MergeSort, уровень 1 — следующим рекурсивным вызовам, и так далее

log n +1 уровней

На каждом уровне O(n) операций

Временная сложность O(n *log* n)

3. Подсчет количества инверсий (полный перебор)

ПОЛНЫЙ ПЕРЕБОР ДЛЯ ПОДСЧЕТА ИНВЕРСИЙ

Вход: массив A из n разных целых чисел.

Выход: количество инверсий массива A.

```
numInv := 0
for i := 1 to n - 1 do
    for j := i + 1 to n do
        if A[i] > A[j] then
            numInv := numInv + 1
return numInv
```

Подсчет количества инверсий

- Инверсия: при i<j A[i]>A[j]

Пример: [1, 3, 5, 2, 4, 6] Сколько инверсий?

Каково наибольшее количество инверсий может иметь массив и 6 элементов?

4. Подсчет количества инверсий с использованием стратегии «Разделяй и властвуй» (делим пополам)

Имеем 3 типа инверсий

1. Левая инверсия
2. Правая инверсия
3. Разделенная инверсия

COUNTINV

Вход: массив A из n разных целых чисел.

Выход: количество инверсий массива A.

```
if n = 0 or n = 1 then                          // базовые случаи
    return 0
else
    leftInv := CountInv(первая половина A)
    rightInv := CountInv(вторая половина A)
    splitInv := CountSplitInv(A)
    return leftInv + rightInv + splitInv
```

Будем подсчитывать инверсии одновременно с сортировкой массива: (A →B)

SORT-AND-COUNTINV

Вход: массив A из n разных целых чисел.

Выход: отсортированный массив B с теми же самыми целыми числами и количество инверсий массива A.

```
if n = 0 or n = 1 then                          // базовые случаи
   return (A,0)
else
    (C, leftInv) := Sort-and-CountInv(первая половина A)
    (D, rightInv) :=
      Sort-and-CountInv(вторая половина A)
    (B, splitInv) := Merge-and-CountSplitInv(C, D)
    return (B, leftInv + rightInv + splitInv)
```

ТЕСТОВОЕ ЗАДАНИЕ 3.2

Предположим, что входной массив A не имеет разделенных инверсий. Как соотносятся между собой отсортированные подмассивы C и D?

а) C содержит наименьший элемент массива A, D — второй наименьший, потом C — третий наименьший и так далее.

б) Все элементы массива C меньше всех элементов в D.

в) Все элементы массива C больше всех элементов в D.

г) Для ответа на этот вопрос недостаточно информации.

MERGE

Вход: отсортированные массивы C и D (длиной $n/2$ каждый).

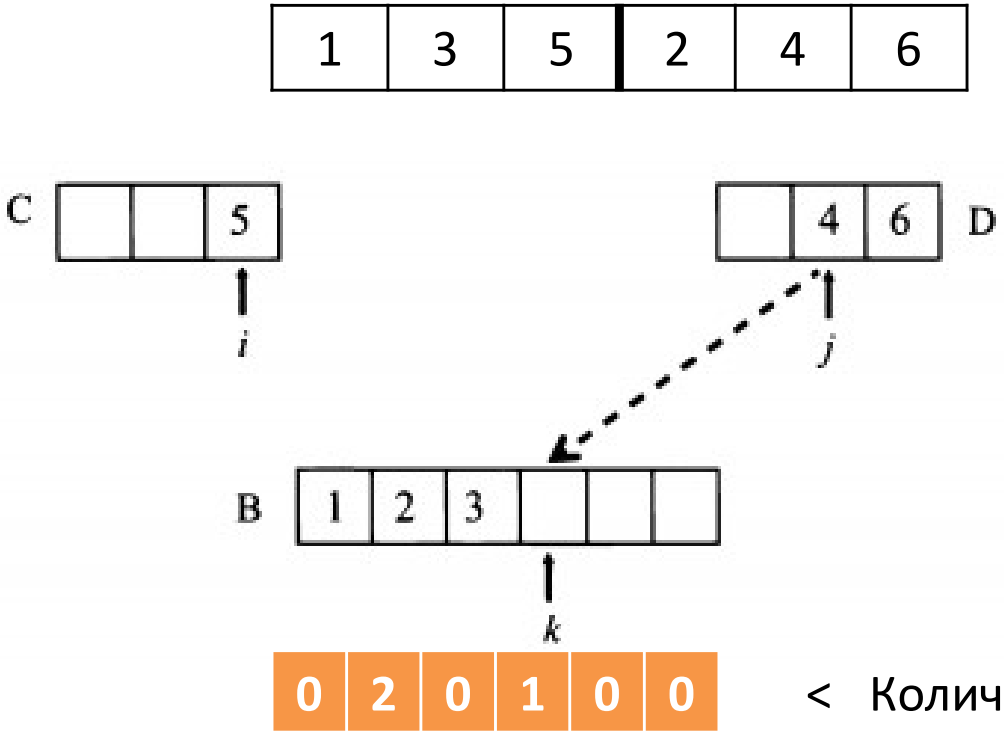
Выход: отсортированный массив B (длиной n).

Упрощающее допущение: n — четное.

```
1 i := 1
2 j := 1
3 for k := 1 to n do
4     if C[i] < D[j] then
5         B[k] := C[i]          // заполнить выходной массив
6         i := i + 1                        // прирастить i
7     else                                    // D[j] < C[i]
8         B[k] := D[j]
9         j := j + 1
```

Тогда подпрограмма **Merge** просто последовательно объединяет 2 массива

Есть разделенные инверсии



MERGE

Вход: отсортированные массивы C и D (длиной $n/2$ каждый).

Выход: отсортированный массив B (длиной n).

Упрощающее допущение: n — чётное.

```
1 i := 1
2 j := 1
3 for k := 1 to n do
4     if C[i] < D[j] then
5         B[k] := C[i]          // заполнить выходной массив
6         i := i + 1                    // прирастить i
7     else                                // D[j] < C[i]
8         B[k] := D[j]
9         j := j + 1
```

< Количество инверсий (сложить)

За сколько проходов подсчитали инверсии?

Подсчет разделенных инверсий с помощью Merge

MERGE-COUNTSPLITINV

Вход: отсортированные массивы C и D (длиной $n/2$ каждый).

Выход: отсортированный массив B (длиной n) и количество разделенных инверсий.

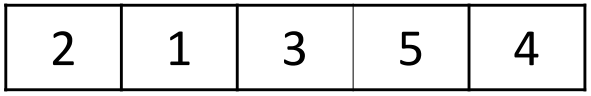
Упрощающее допущение: n — четное.

```
i := 1, j := 1, splitInv := 0
for k := 1 to n do
    if C[i] < D[j] then
        B[k] := C[i], i := i + 1
    else                                        // D[j] < C[i]
        B[k] := D[j], j := j + 1
        splitInv := splitInv + (n/2 - i + 1)
                                # оставшееся кол-во в C
return (B, splitInv)
```

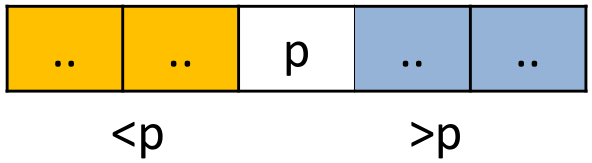
Корректность самостоятельно. Время? Курсовая работа

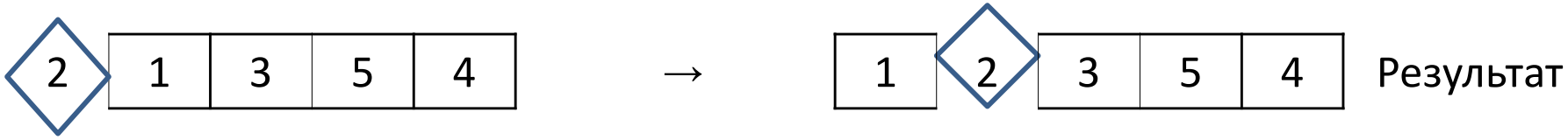
Быстрая сортировка Quick_Sort

Пример:



1) Выбираем опорный элемент p.
2) Перегруппировываем элементы так, чтобы
3) Делаем это рекурсивно, пока длина массива не станет ≤1





Разделить массив с разными опорными элементами

QUICKSORT (ВЫСОКОУРОВНЕВОЕ ОПИСАНИЕ)

Вход: массив A из n разных целых чисел.

Выход: элементы массива A отсортированы от наименьшего до наибольшего.

---

if $n \leq 1$ **then** // базовый случай – уже отсортирован
  return

Choosepivot(A) // выбрать опорный элемент

Partition (A) // разделить вокруг опорного элемента

QUICKSORT (левая часть <p)

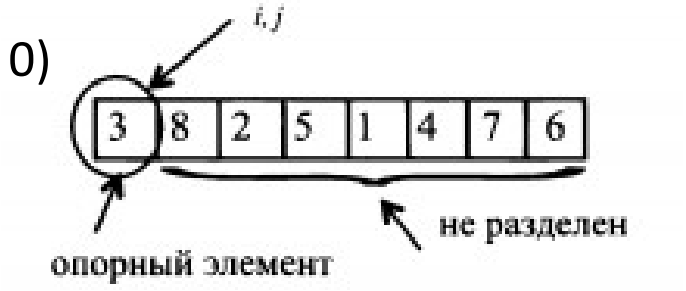
QUICKSORT (правая часть >p)

Пример (с первым элементом в качестве опроного)

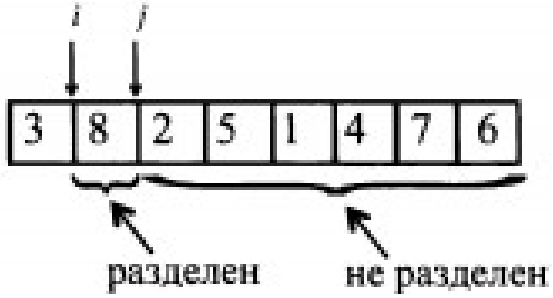
j - граница между неопорными элементами, которые уже рассмотрены и теми, которые еще не рассмотрены

i - граница между элементами меньше опорного и больше опорного (внутри группы рассматриваемых элементов)

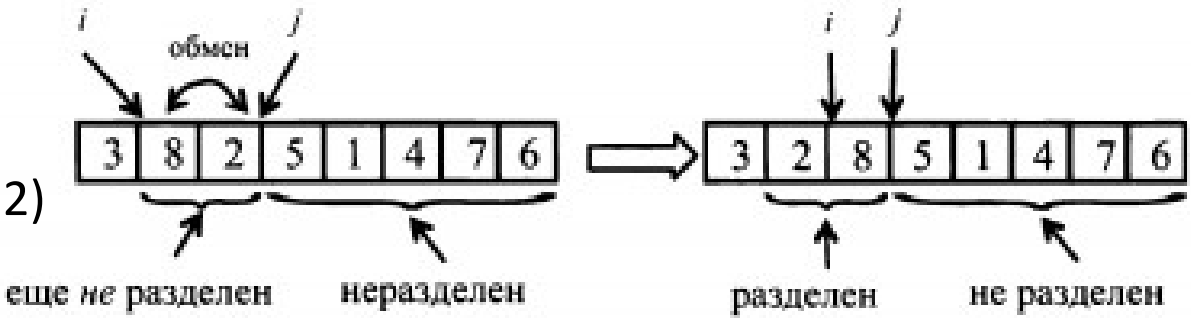
0)



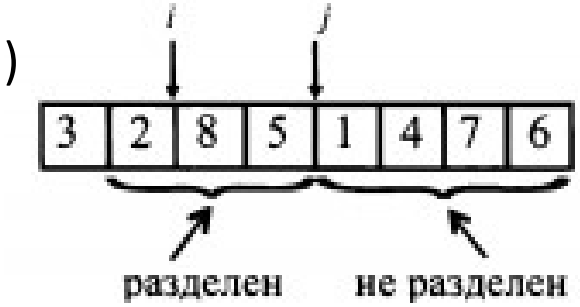
1)



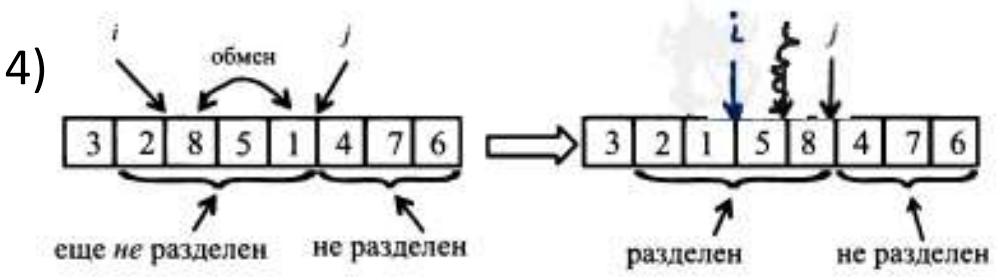
2)



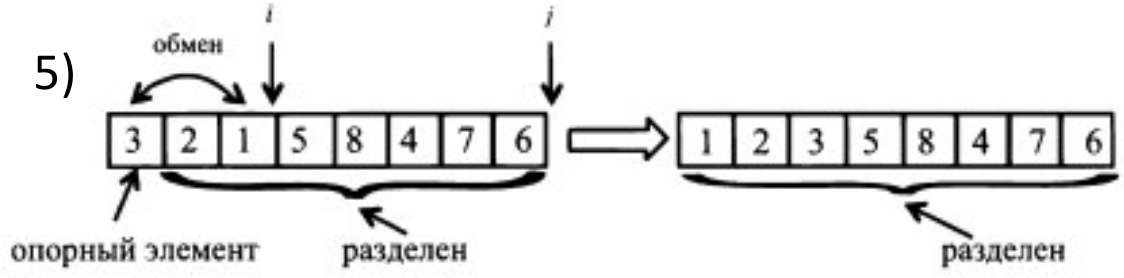
3)



4)



5)



Разное время обработки в зависимости от выбора p

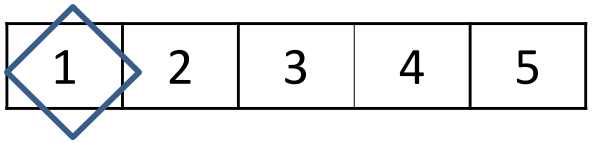
Пусть в качестве опорного всегда выбирается первый элемент. Каково время работы QUICKSORT, если n-элементный массив уже отсортирован?

а) $\Theta(n)$.

б) $\Theta(n \log n)$.

в) $\Theta(n^2)$.

г) $\Theta(n^3)$.



Пусть в качестве опорного всегда выбирается медиана. Каково время работы QUICKSORT, если n-элементный массив уже отсортирован?

а) $\Theta(n)$.

б) $\Theta(n \log n)$.

в) $\Theta(n^2)$.

г) $\Theta(n^3)$.