MERGE(A, p, q, r)

```
 1  n_1 = q - p + 1
 2  n_2 = r - q
 3  Пусть L[1..n_1 + 1] и R[1..n_2 + 1] — новые массивы
 4  for i = 1 to n_1
 5      L[i] = A[p + i - 1]
 6  for j = 1 to n_2
 7      R[j] = A[q + j]
 8  L[n_1 + 1] = ∞
 9  R[n_2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

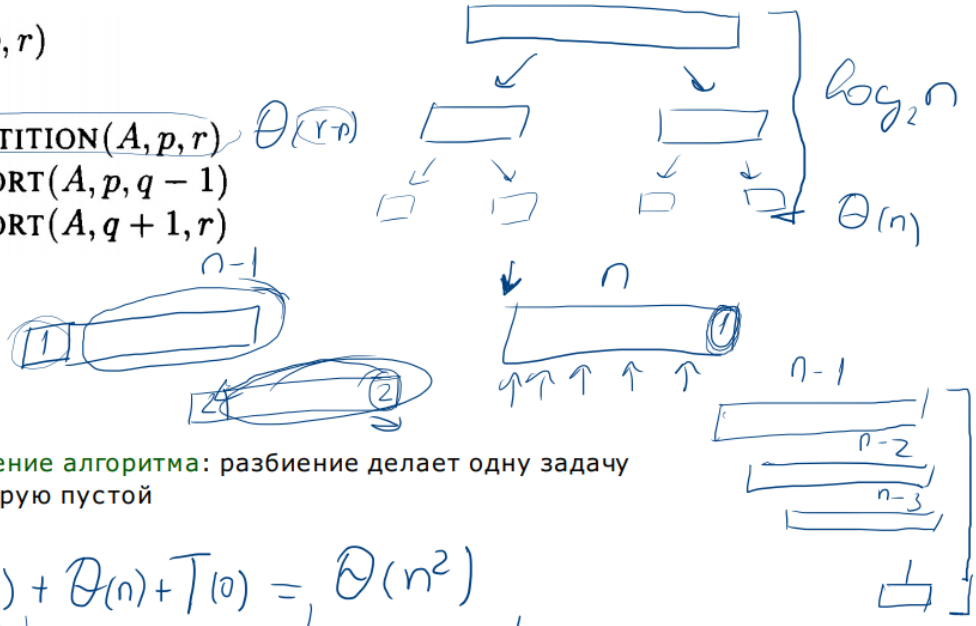
Инициализация. Перед первой итерацией цикла $k = p$, так что подмассив $A[p..k-1]$ пуст. Он содержит $k - p = 0$ наименьших элементов массивов L и R, а поскольку $i = j = 1$, элементы $L[i]$ и $R[j]$ — наименьшие элементы массивов L и R, не скопированные обратно в массив A.

Сохранение. Чтобы убедиться, что инвариант цикла сохраняется после каждой итерации, сначала предположим, что $L[i] \leq R[j]$. Тогда $L[i]$ — наименьший элемент, пока еще не скопированный в массив A. Поскольку в подмассиве $A[p..k-1]$ содержится $k - p$ наименьших элементов, после копирования в строке 14 $L[i]$ в $A[k]$ в подмассиве $A[p..k]$ будет содержаться $k - p + 1$ наименьших элементов. В результате увеличения параметра k цикла **for** и значения переменной i (строка 15), инвариант цикла восстанавливается перед следующей итерацией. Если же выполняется неравенство $L[i] > R[j]$, то в строках 16 и 17 выполняются соответствующие действия, в ходе которых также сохраняется инвариант цикла.

Завершение. Алгоритм завершается, когда $k = r + 1$. В соответствии с инвариантом цикла подмассив $A[p..k-1]$ (т.е. подмассив $A[p..r]$) содержит $k - p = r - p + 1$ наименьших элементов массивов $L[1..n_1+1]$ и $R[1..n_2+1]$ в отсортированном порядке. Суммарное количество элементов в массивах L и R равно $n_1 + n_2 + 2 = r - p + 3$. Все они, кроме двух самых больших, скопированы обратно в массив A, а два оставшихся элемента являются сигнальными.

Анализ алгоритма быстрой сортировки

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```



Наихудшее поведение алгоритма: разбиение делает одну задачу размера n-1, а вторую пустой

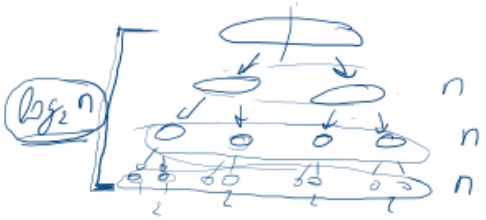
$$T(n) = T(n-1) + \Theta(n) + T(0) = \Theta(n^2)$$

арифм. прогр.

Наилучшее поведение алгоритма: разбиение делает две подзадачи половинного размера

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n \log n)$$

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  Обменять A[r] и A[i]
3  return PARTITION(A, p, r)
```



В среднем время сортировки: $\Theta(n \log n)$

$\Theta(n \log n)$

Merge sort

$\Omega(n \log n)$