Sorting Algorithms - II

Quick Sort

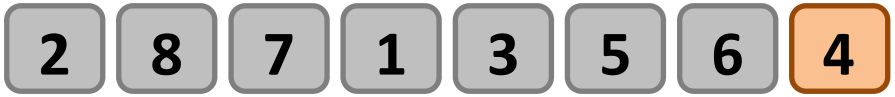
Merge Sort

Quick Sort

Quick Sort

- Divide and Conquer algorithm.
- Picks an element as pivot and partitions the given array around the picked pivot, such that
  - The pivot is placed at its correct position
  - All elements smaller than the pivot are placed before the pivot.
  - All elements greater than the pivot are placed after the pivot.
- Several ways to pick a pivot.
  - The first element.
  - The last element.
  - Any random element.
  - The median.

Quick Sort



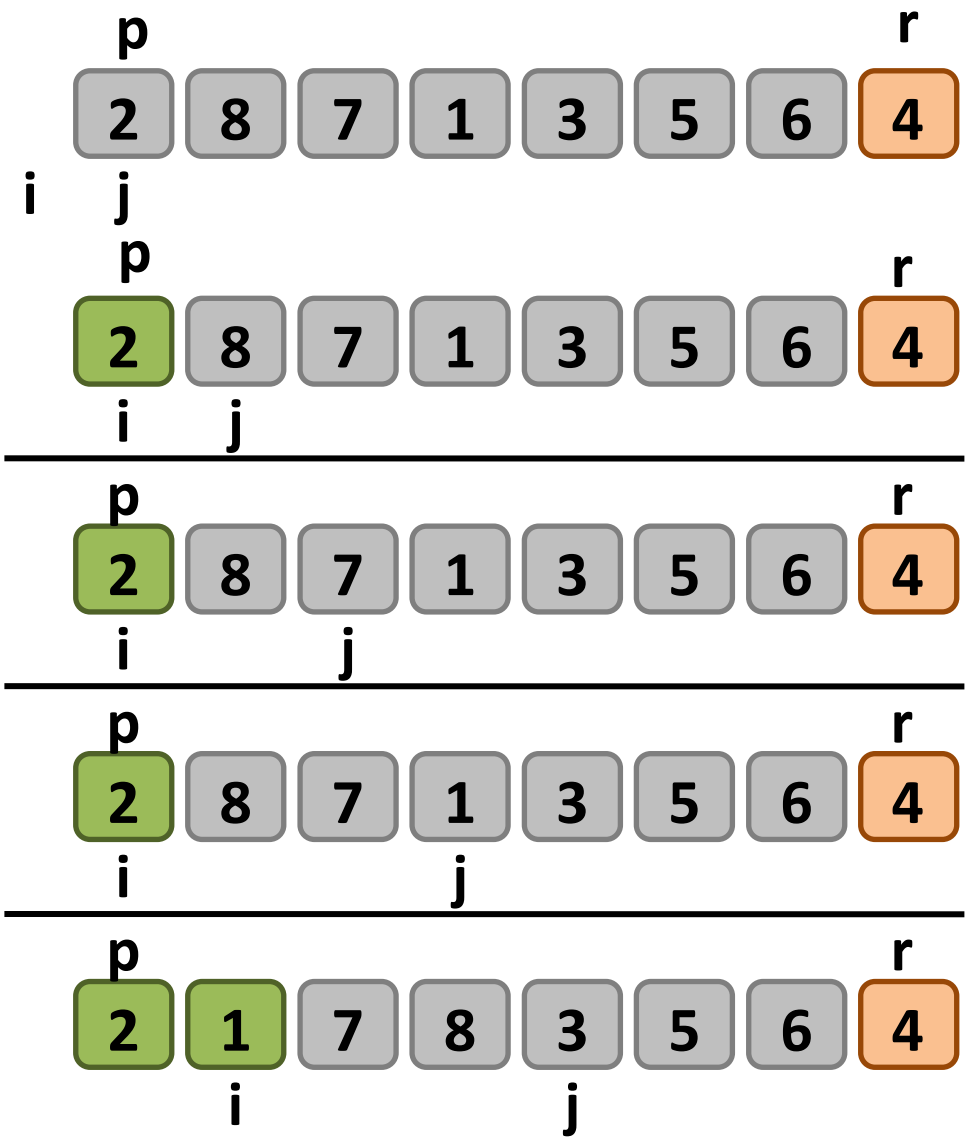
- QUICKSORT(A, p, r)

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

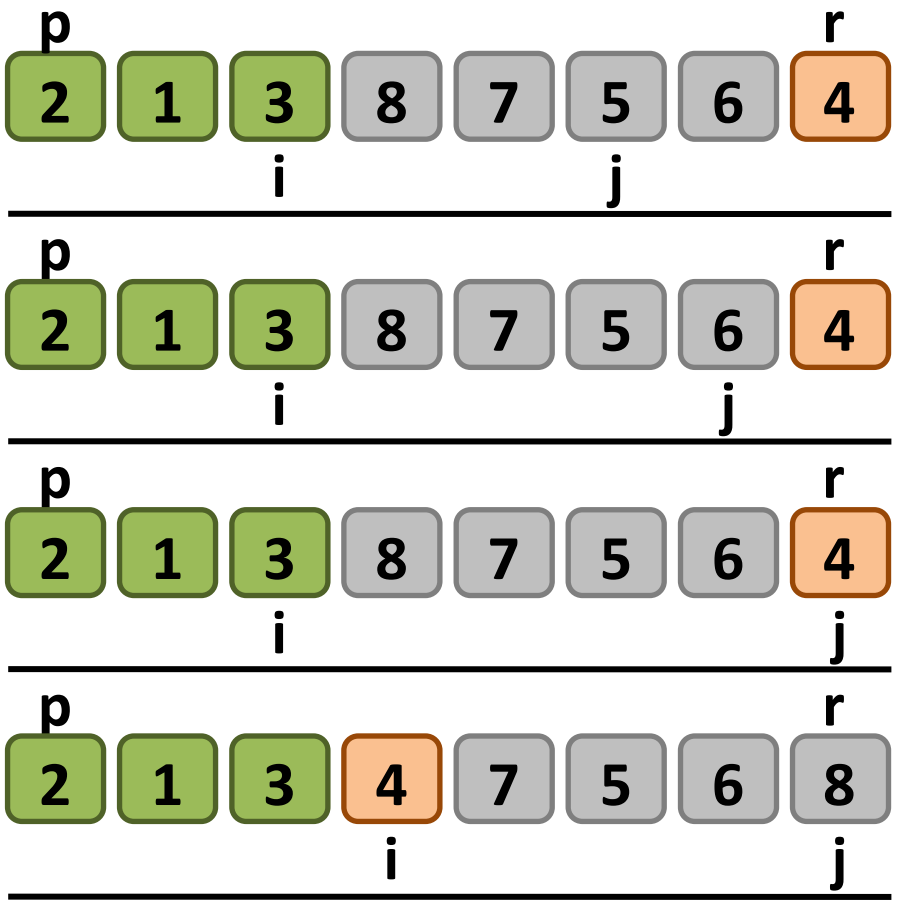
To sort an array A with n elements, the first call to QUICKSORT is made with p = 0 and r = n – 1.

Algorithm

1. PARTITION(A, p, r)
2. x = A[r]
3. i = p – 1
4. for j = p to r – 1
5. if A[j] ≤ x
6. i = i + 1
7. Exchange A[i] with A[j]
8. Exchange A[i + 1] with A[r]
9. return i + 1



Algorithm




```
1.  PARTITION(A, p, r)
2.  x = A[r]
3.  i = p – 1
4.  for j = p to r – 1
5.      if A[j] ≤ x
6.          i = i + 1
7.          Exchange A[i] with A[j]
8.  Exchange A[i + 1] with A[r]
9.  return i + 1
```

Call sequence for QUICKSORT

QUICKSORT (A, 0, 7) Partition point at index 3

QUICKSORT (A, 0, 2) Partition point at index 2

QUICKSORT (A, 0, 1) Partition point at index 0

QUICKSORT (A, 0, -1)

QUICKSORT (A, 1, 1)

QUICKSORT (A, 3, 2)

QUICKSORT (A, 4, 7) Partition point at index 7

QUICKSORT (A, 4, 6) Partition point at index 5

QUICKSORT (A, 4, 4)

QUICKSORT (A, 6, 6)

QUICKSORT (A, 8, 7)

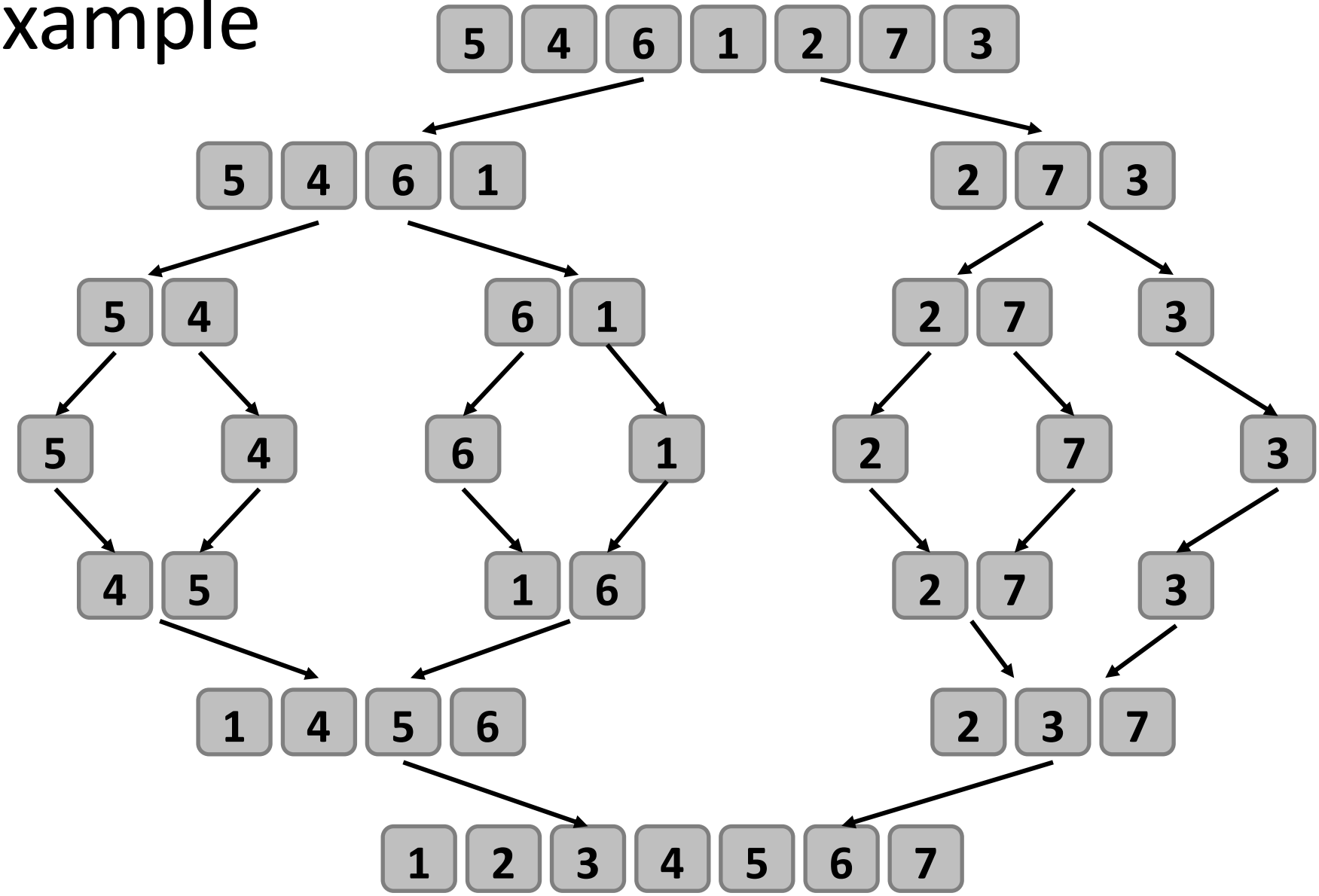
Merge Sort

Merge Sort

- Based on the divide-and-conquer paradigm.
- To sort an array A[p .. r], (initially p = 0 and r = n-1)

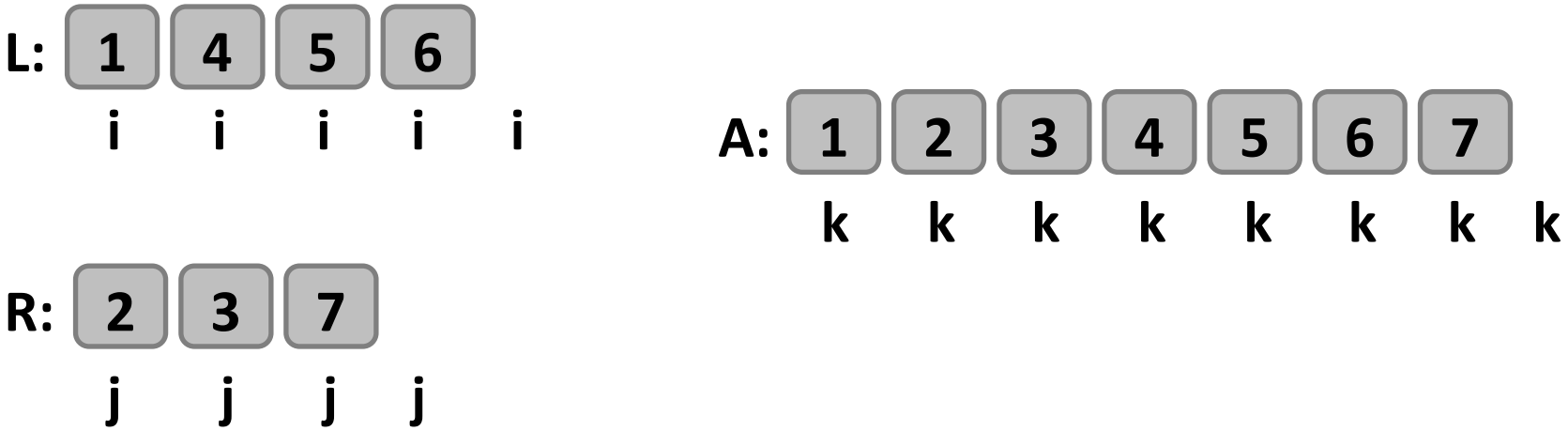
1. Divide Step
   - If a given array A has zero or one element, then return as it is already sorted.
   - Otherwise, split A[p…r] into two subarrays A[p…q] and A[q + 1... r], each containing about half of the elements of A[p…r]. That is, q is the halfway point of A[p...r].
2. Conquer Step
   - Recursively sort the two subarrays A[p…q] and A[q + 1…r].
3. Combine Step
   - Combine the elements back in A[p…r] by merging the two sorted subarrays A[p…q] and A[q + 1…r] into a sorted sequence.

Example



Merge Two Sorted Arrays

n1 - #Elements in L
n2 - #Elements in R




```
8.  i = 0, j = 0, and k = p.
9.  while i < n1 and j < n2
10.         if L[i] ≤ R[j]
11.             A[k] = L[i]
12.             i = i + 1
13.         else
14.             A[k] = R[j]
15.             j = j + 1
16.         k++
17. while i < n1
18.         A[k] = L[i]
19.         i++
20.         k++
21. while j < n2
22.         A[k] = R[j]
23.         j++
24.         k++
```

Algorithm

- MERGE-SORT (A, p, r)

1. if p < r
2. q = FLOOR[(p + r)/2]
3. MERGE-SORT(A, p, q)
4. MERGE-SORT(A, q + 1, r)
5. MERGE (A, p, q, r)

- To sort an array A with n elements, the first call to MERGE-SORT is made with p = 0 and r = n – 1.

Contd…

- Algorithm MERGE (A, p, q, r)
- Input: Array A and indices p, q, r such that $p \leq q \leq r$. Subarrays A[p…q] and A[q + 1…r] are sorted.
- Output: The two subarrays are merged into a single sorted subarray in A[p .. r].

1. n1 = q − p + 1
2. n2 = r − q
3. Create arrays L[n1] and R[n2]
4. for i = 0 to n1 – 1
5. L[i] = A[p + i]
6. for j = 0 to n2 – 1
7. R[j] = A[q + 1 + j]

Contd…

```
8.  i = 0, j = 0, and k = p.
9.  while i < n1 and j < n2
10.         if L[i] ≤ R[j]
11.                 A[k] = L[i]
12.                 i = i + 1
13.         else
14.                 A[k] = R[j]
15.                 j = j + 1
16.         k++
17. while i < n1
18.         A[k] = L[i]
19.         i++
20.         k++
21. while j < n2
22.         A[k] = R[j];
23.         j++;
24.         k++;
```

Call sequence for an array with size 8

- MERGE-SORT (A, 0, 7)
  - MERGE-SORT (A, 0, 3)
    - MERGE-SORT (A, 0, 1)
      - MERGE-SORT (A, 0, 0)
      - MERGE-SORT (A, 1, 1)
      - **MERGE (A, 0, 0, 1)**
    - MERGE-SORT (A, 2, 3)
      - MERGE-SORT (A, 2, 2)
      - MERGE-SORT (A, 3, 3)
      - **MERGE (A, 2, 2, 3)**
    - **MERGE (A, 0, 1, 3)**
  - MERGE-SORT (A, 4, 7)
    - MERGE-SORT (A, 4, 5)
      - MERGE-SORT (A, 4, 4)
      - MERGE-SORT (A, 5, 5)
      - **MERGE (A, 4, 4, 5)**
    - MERGE-SORT (A, 6, 7)
      - MERGE-SORT (A, 6, 6)
      - MERGE-SORT (A, 7, 7)
      - **MERGE (A, 6, 6, 7)**
    - **MERGE (A, 4, 5, 7)**
  - **MERGE (A, 0, 3, 7)**

5	4	6	1	2	7	3	8
5	4	6	1	2	7	3	8
5	4	6	1	2	7	3	8
5	4	6	1	2	7	3	8
5	4	6	1	2	7	3	8
4	5	6	1	2	7	3	8
4	5	6	1	2	7	3	8
4	5	6	1	2	7	3	8
4	5	6	1	2	7	3	8
4	5	1	6	2	7	3	8
1	4	5	6	2	7	3	8
1	4	5	6	2	7	3	8
1	4	5	6	2	7	3	8
1	4	5	6	2	7	3	8
1	4	5	6	2	7	3	8
1	4	5	6	2	7	3	8
1	4	5	6	2	7	3	8
1	4	5	6	2	7	3	8
1	4	5	6	2	7	3	8
1	4	5	6	2	7	3	8
1	4	5	6	2	3	7	8
1	2	3	4	5	6	7	8

Thank You