L9

Merge

Algorithm & Illustration

Merge Sort – Recursive Algorithm

MERGE-SORT(A, p, r)

if $p < r$
    $q = \lfloor (p + r)/2 \rfloor$
    MERGE-SORT(A, p, q)
    MERGE-SORT$(A, q + 1, r)$
    MERGE(A, p, q, r)

Ref: CLRS Book

Merge function

- Merge is done by calling another function **Merge (A,p,q,r)**

A- Array, p,q,r are indices s.t p <= q < r

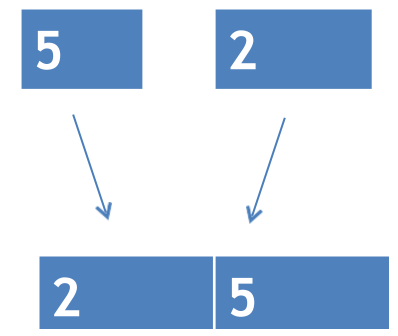
- **Assumption:** A[p…q] and A[q+1… r] are in sorted order
- **Input:** Array A, indices p, q, and r
- **Output:** Merges A[p…q] and A[q+1 … r] and produce a single sorted subarray A[p…r]

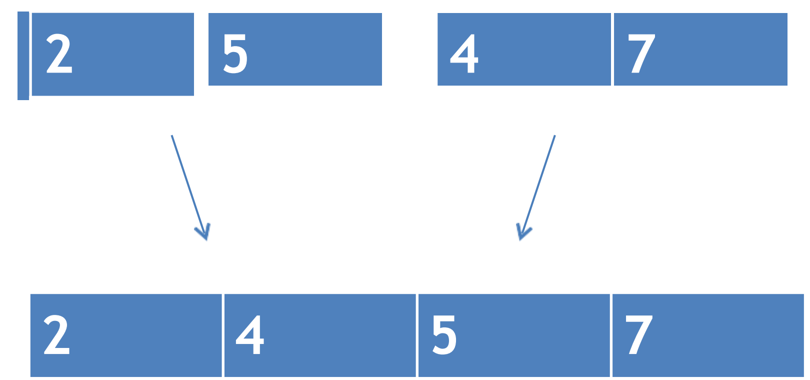
Merge function

MERGE(A, p, q, r)

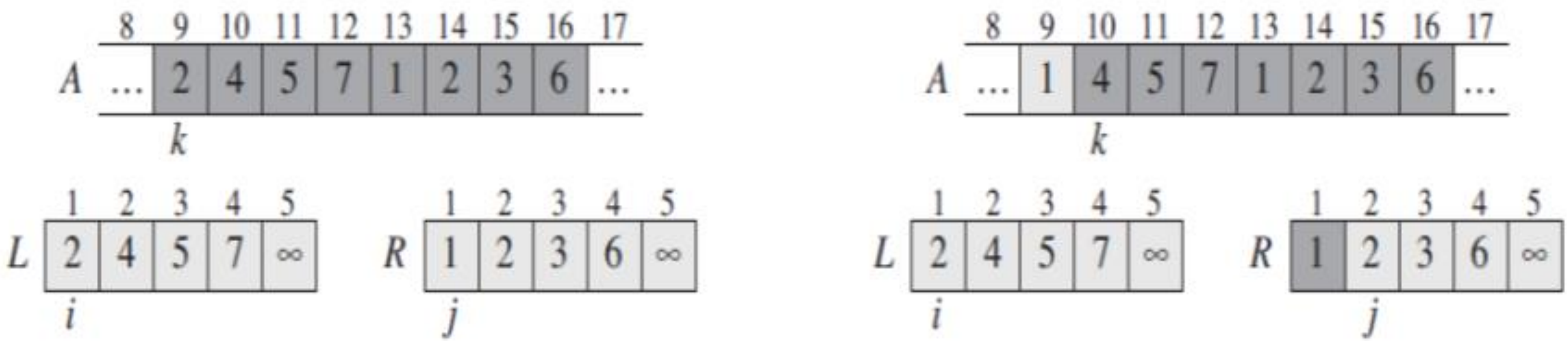
```
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
        L[i] = A[p + i - 1]
for j = 1 to n2
        R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
        if L[i] ≤ R[j]
                A[k] = L[i]
                i = i + 1
        else A[k] = R[j]
                j = j + 1
```

Sentinel - a special value





Ref: CLRS Book

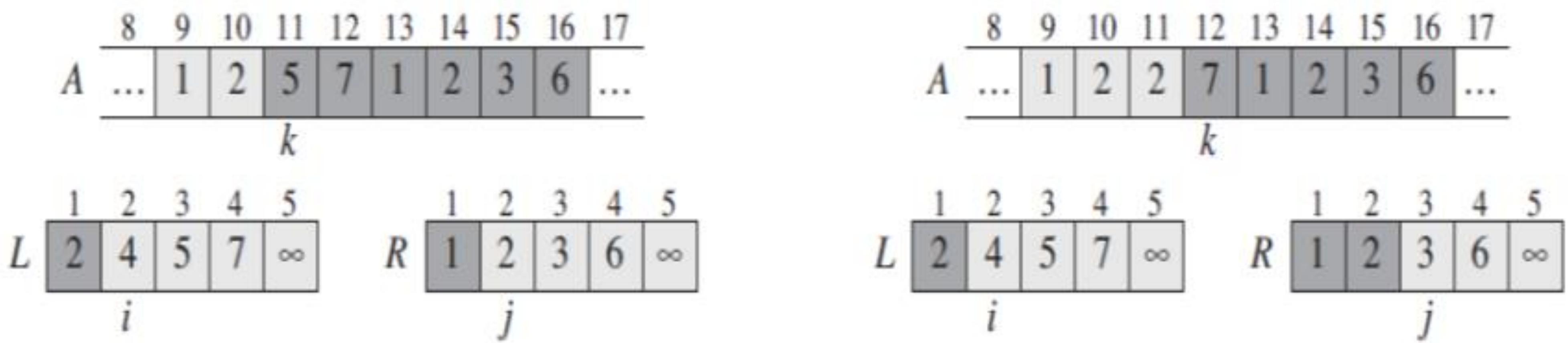


(a) (b)

MERGE(A, p, q, r)

```
n₁ = q − p + 1
n₂ = r − q
let L[1 .. n₁ + 1] and R[1 .. n₂ + 1] be new arrays
for i = 1 to n₁
    L[i] = A[p + i − 1]
for j = 1 to n₂
    R[j] = A[q + j]
L[n₁ + 1] = ∞
R[n₂ + 1] = ∞
```

```
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```



(c) (d)

Correctness of Insertion Sort

Insertion Sort

```
INSERTION SORT(A)
for j=2 to A.length
        key = A[ j ];
        //Insert A[ j ] into the sorted sequence A[1...j-1]
        i = j-1
        while i > 0 and A[ i ] > key
                A[ i+1 ]=A[ i ]
                i=i-1
        A[ i+1 ]= key
```

How do we prove that INSERTION SORT(A) is correct?

- We observe the algorithm critically and try to understand
- We observe that :
    - The index *j* indicates:
        - The current number/card being inserted
    - At the beginning of each iteration of *for* loop indexed by *j:*

      A [1 .. j-1] is sorted and *A [j+1..n]* is remaining
- **Elements A [1.. j-1] are the elements originally in positions 1 through j-1, but now in sorted order**
- We state these properties of A[1 .. j-1] formally as a **loop invariant**

INSERTION SORT(A)

```
for j=2 to A.length
        key = A[ j ];
//Insert A[ j ] into the sorted sequence A[1...j-1]
        i = j-1
        while i > 0 and A[ i ] > key
                A[ i+1 ]=A[ i ]
                i=i-1
        A[ i+1 ]= key
```

Loop Invariant of INSERTION SORT(A)

- **At the start of each iteration of the for loop of lines 1-8, the subarray A [1 ... j-1] consists of the elements originally in A [1 ... j-1], but in sorted order.**
- We use loop invariant (a property of the algorithm) to prove that the algorithm is correct
- We must show three things about a loop invariant**:**
  - Initialization
  - Maintenance
  - Termination

Correctness of Insertion Sort

- We must show three things about a loop invariant **:**
  - **Initialization:** The loop invariant is true prior to the first iteration of the loop
  - **Maintenance:** If the loop invariant is true before an iteration of the loop, it remains true before the next iteration
  - **Termination:** When the loop terminates, the loop invariant gives us a useful property that helps us to show that the algorithm is correct
- When the first two properties hold, the loop invariant is true prior to every iteration of the loop.

Loop invariant Vs mathematical induction

- Invariant holds before the first iteration ~ base case of mathematical induction
- Invariant holds from iteration to iteration ~ inductive step
- Third property ie. The termination property differs from math induction
  - In math induction, we use the inductive step infinitely, whereas here we stop the induction when the loop terminates

Proof – Correctness of Insertion sort

- **Initialization**: Loop invariant trivially holds before the first loop iteration. A [1] is sorted by itself.
- **Maintenance:**
  - *for* loop works by moving A[j-1], A[j-2], A[j-3] and so on by one position to the right and finds the proper position for A[j] and inserts the value of A[j].
  - Hence, A[1..j-1] consists of the elements originally in A [1..j-1], but in sorted order.
  - Incrementing j for the next iteration preserves the loop invariant

Proof: Correctness of Insertion sort -contd.

- **Termination:**
  - *for* loop terminates when $j > A.length = n$ ie. $j = n+1$
  - Substituting n+1 in the wording of loop invariant, the subarray A[1..n] consists of originally in A[1..n], but in sorted order
- Observe that A[1..n] is the entire array and since the entire array is sorted, the algorithm is correct.
- After proving Insertion sort is correct, we have to prove insertion sort is efficient.
- For that we analyse insertion sort

Analyzing the algorithms

- **What do we mean by “analyzing the algorithms”**
- Analyzing the algorithms means predicting the resources the algorithm uses
- What are the resources?
- **Computational time and Memory for storage**
- **Why do we analyze algorithms?**
- Analyzing several algorithms for a particular problem results in the most efficient algorithm in terms of computational time/memory

Thank You