Merge Sort

Divide and Conquer Approach

Three Steps:

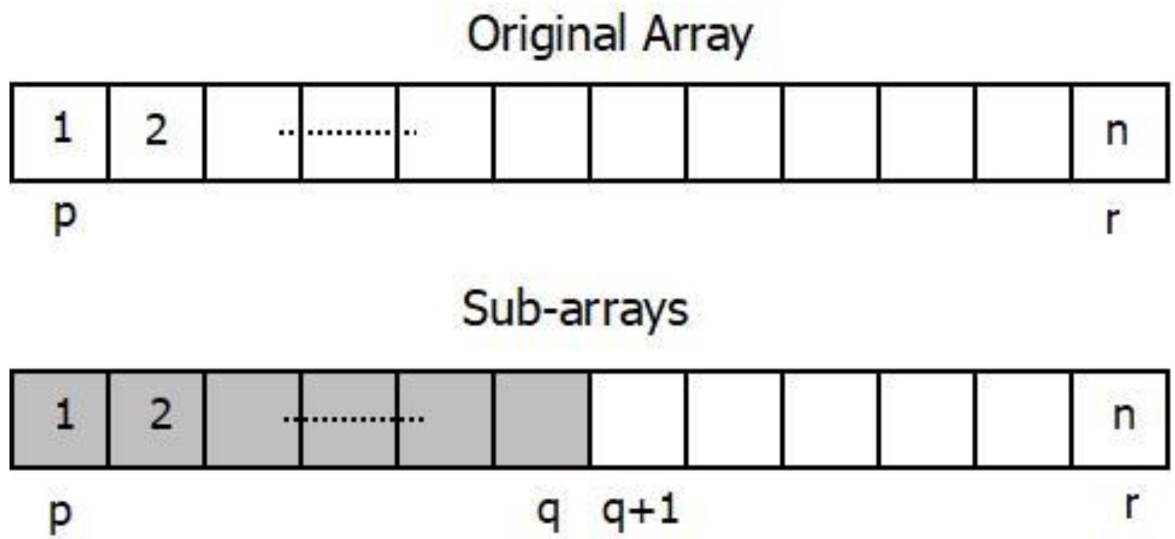
- **Divide** the problem into a number of sub-problems (smaller in size)
- **Conquer** the sub-problems. Solve them recursively
- **Combine** the solutions to the sub-problems into the solution for the original problem

Merge Sort

- **Divide:** n-element sequence to be sorted → Two sub-sequences of $n/2$ elements each
- **Conquer:** Sort the two sub-sequences recursively using merge sort
- **Combine:** Merge the two sorted sub-sequences to produce the sorted answer

The recursion “bottoms out”
when the sub-sequence is of length 1→ Already sorted

Merge Sort Algorithm



MERGE-SORT(A, p, r)

if $p < r$

$q = \lfloor (p + r)/2 \rfloor$

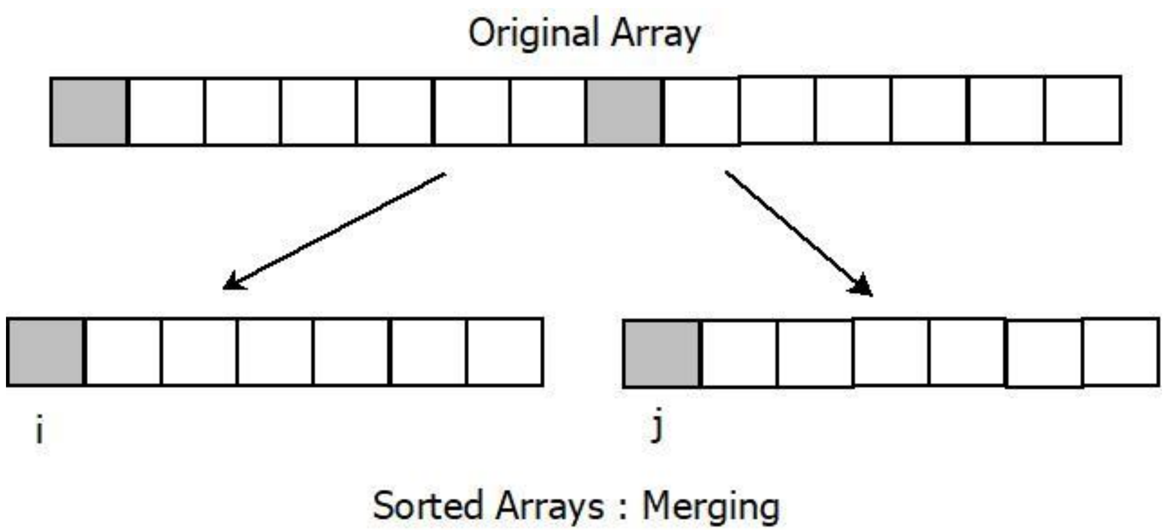
MERGE-SORT (A, p, q)

MERGE-SORT $(A, q + 1, r)$

MERGE (A, p, q, r)

MERGE Procedure

- The **key** operation of the merge sort
  - "Combine" Step: Merging of two sorted sequences into one
  - Subarrays $A[p..q]$ and $A[q+1..r]$ are in sorted order
  - MERGE(P, q, r) merges these two sorted array into one



- MERGE procedure takes time $\Theta(n)$ where $n = r - p + 1$

MERGE Algorithm

- ***Sentinel*** element ∞: Avoid additional checks and simplify the code

```
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
```

Loop Invariant

- **Loop Invariant:**
  - At the start of each iteration of the **for** loop of lines 12–17, the subarray $A[p..k-1]$ contains the $k-p$ smallest elements of $L[1..n_1+1]$ and $R[1..n_2+1]$, in sorted order
  - Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A
- Let’s show the following for the loop invariant statement
  - Initialization
  - Maintenance
  - Termination

Loop Invariant

- **Initialization:**
  - Prior to the first iteration of the loop, we have $k = p$, so that the subarray $A[p..k-1]$ is empty
  - This empty subarray contains the $k - p = 0$ smallest elements of L and R
  - Since $i = j = 1$, both $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A

Loop Invariant

- **Maintenance:**
  - Assuming $L[i] \le R[j]$ then $L[i]$ is the smallest element not yet copied back into A
  - $A[p..k-1]$ contains the $k-p$ smallest elements. After $L[i]$ is copied into $A[k]$, the subarray $A[p..k]$ will contain the $k-p+1$ smallest elements
  - Incrementing k (in the **for** loop update) and i (in line 15) re-establishes the loop invariant for the next iteration
  - If instead $L[i] > R[j]$, then lines 16–17 perform the appropriate action

Loop Invariant

- **Termination:**
  - At termination, $k = r + 1$
  - The subarray $A[p..k-1]$, which is $A[p..r]$, contains the $k - p = r - p + 1$ smallest elements of $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$, in sorted order
  - The arrays L and R together contain $n_1 + n_2 + 2 = r - p + 3$ elements. All but the two largest have been copied back into A, and these two largest elements are the sentinels

Analyzing Merge Sort

- Depth: $\log n$, Cost of each level: cn