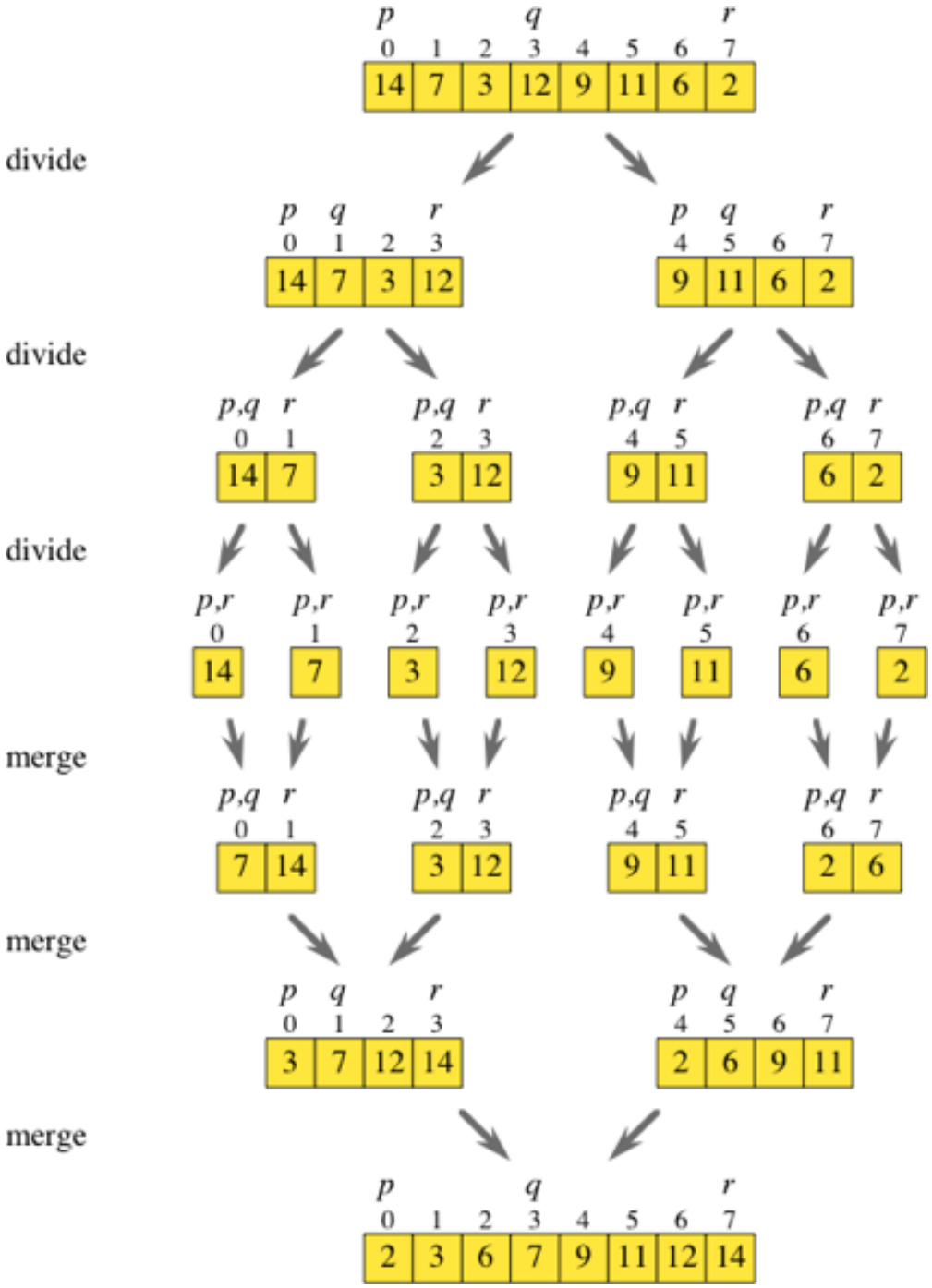
Divide and Conquer

- It is a fundamental algorithm design paradigm with three key steps:
  1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
  2. **Conquer** the subproblems by solving them recursively.
  3. **Combine** the solutions to the subproblems into the solution for the original problem.
- The base cases for the recursion are subproblems of constant size (trivial to solve).

P1 – Merge Sort

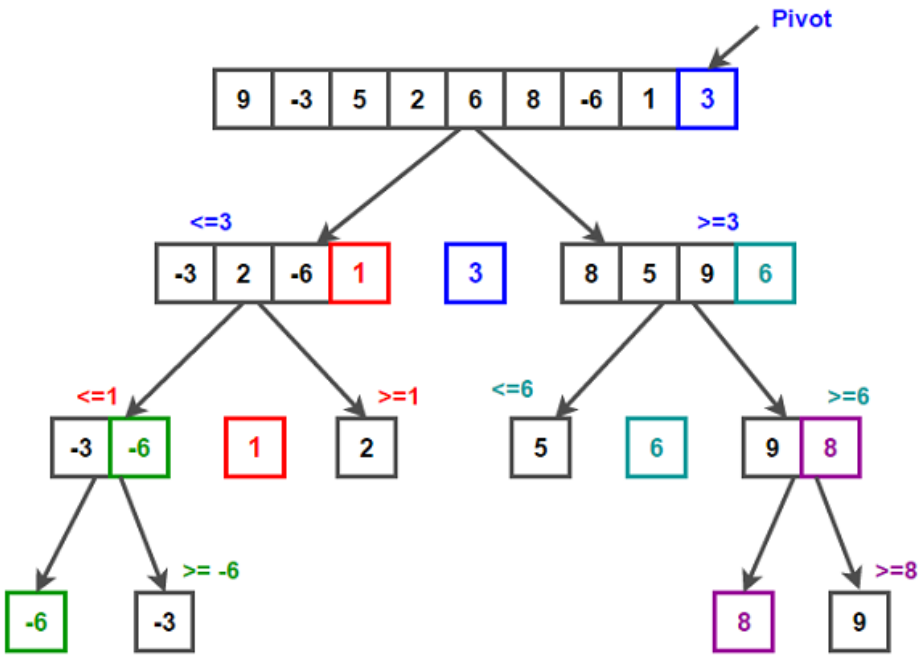


Algorithm:

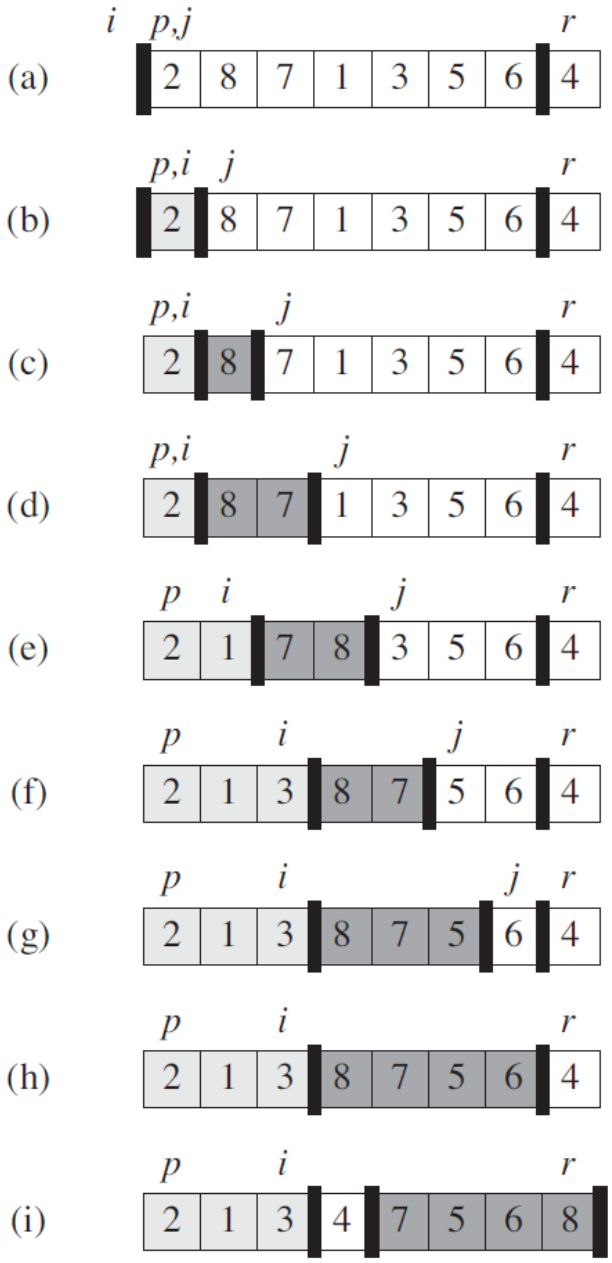
```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 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
```

P2 – Quick Sort



Partition Process:



Algorithm:

```
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)
```

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

Practices:

1. **Calculate pow(x,n)**

   Given, the value of x and n. You will calculate the value of x^n using the divide and conquer.

   Approach:

   If n is an even number, then your result is pow(x, n/2)*pow(x, n/2)
   If n is an odd number, then your result is pow(x,n/2)*pow(x,n/2)*x
   If n is 0, then your result is 1.

2. **Randomized Binary Search**

   Given a sorted array of **n** numbers and a search value **x**, you need to search for **x** from the sorted array.

   Normally, during binary search we always consider the middle element for our search, if not found then we decide to search the left subarray or the right subarray.

   But during the randomized binary search, we will randomly pick an element (instead of the middle element) for our search and if not found then we will decide to search in the left subarray or the right.

3. **Fast multiplication algorithm (Karatsuba Algorithm)**

   Old school method – $O(n^2)$

```
    1001
  x 101
  ------
    1001
   0000
+ 1001
  ------
  101101
```

   Karatsuba Algorithm- $O(n^{1.59})$

   Number 1: 12345 = 10^m * 12 + 345 = 10^m a + b
   Number 2: 56789 = 10^m * 56 + 789 = 10^m c + d
   here, m = ceil(number of digits/2) = 3

   Multiplication:

   12345 * 56789
   = (10^m * a + b) * (10^m * c + d)
   = 10^{2m} * ac + 10^m * (ad + bc) + bd
   = 10^{2m} * ac + 10^m * (ac + bd – (a-b)*(c-d)) + bd

4. **Count inversions in an array**

   Inversion count for an array indicates – how far the array is from being sorted. If the array is already sorted, then the inversion count is 0, but if the array is sorted in the reverse order, the inversion count is the maximum.

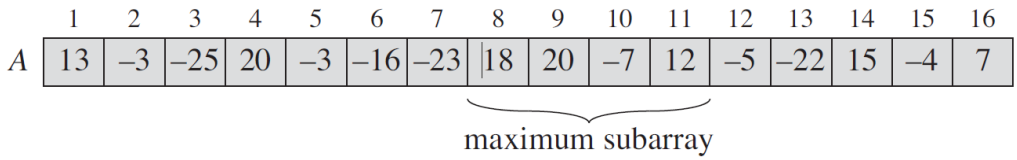
   For example,

   Input: arr[]={9, 5, 3, 1} **Output:** 6
   The inversions are: (9,5), (9,3), (9,1), (5,3), (5,1), (3,1)

5. Maximum subarray sum

Given a one-dimensional array that may contain both positive and negative integers, find the sum of continuous subarray of numbers which has the largest sum.

For example,



Approach:

1. Divide the array into two halves
2. Return the maximum of the following 3:
   a) Maximum subarray sum in left half (recursive call)
   b) Maximum subarray sum in right half (recursive call)
   c) Maximum subarray sum such that the subarray crosses the midpoint (process yourself)

6. Rotation count in rotated sorted array

Given a rotated sorted array of distinct integers, your task is to find out the total number of rotations.

For example,
{ 15, 17, 1, 2, 6, 11 } is rotated 2 times.
{ 7, 9, 11, 12, 5 } is rotated 4 times.

In short, the index of the smallest integer is the rotation count. Minimum element is the one whose previous element is greater than it. If there is no previous element, then there is no rotation. First check the middle element, if it is not the minimum element then decide whether to search the left subarray or the right.

7. Find a peak element

Given an array of integers, find a peak element in it. An array element is a peak if it is not smaller than its neighbors. For corner element, only consider one neighbor.

For example,
In { 5, 10, 15, 12, 20, 25 }, 12 is the peak element.

Similar to binary search, first check whether the middle element is the peak element or not. If the middle element is not the peak element, then check if the element on the right side is greater than the middle element then there is always a peak element on the right side. Similarly, if the element on the left side is greater than the middle element then there is always a peak element on the left side.