L7

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
```

Insertion sort

- Insertion sort is an **in place** sorting method ie it rearranges the numbers within the input array, with at most a constant number of them stored outside the array at any time.
- The input array A contains the sorted sequence after the procedure is finished
- **Operations of insertion sort on an array A = 1, 4, -2, -3**
- After 1st iteration of *while* loop, with j = 2, i = 1: A = 1, 4, -2, -3
- After 1st iteration of *for* loop, with j = 2 : A = 1, 4, -2, -3

INSERTION SORT(A)

1. for j=2 to A.length
2. key = A[j];
3. //Insert A[j] into the sorted sequence A[1...j-1]
4. i = j-1
5. while i > 0 and A[i] > key
6. A[i+1]=A[i]
7. i=i-1
8. A[i+1]= key

Operation of Insertion Sort on A

- **A = 1, 4, -2, -3**
- After 1st iteration of *while* loop, with j = 3, i = 2: A = 1, 4, 4,-3
- After 2nd iteration of *while* loop, with j = 3, i = 1: A = 1, 1, 4,-3
- After 2nd iteration of *for* loop, with j = 3 : A = -2, 1, 4, -3
- After 1st iteration of *while* loop, with j = 4, i = 3: A = -2, 1, 4, 4
- After 2nd iteration of *while* loop, with j = 4, i = 2: A = -2, 1, 1, 4
- After 3rd iteration of *while* loop, with j = 4, i = 1: A = -2, -2, 1, 4
- After 3rd iteration of *for* loop, with j = 4 : A = -3, -2, 1, 4

```
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
```

Merge Sort

Algorithm

Design paradigms

- **Paradigm**
  - "In science and philosophy, a paradigm is a distinct set of concepts or thought patterns, including theories, research methods, postulates, and standards for what constitutes legitimate contributions to a field"- Wikipedia

Types of Design Paradigms

- Incremental Approach
- Divide and Conquer
- Greedy approach
- Dynamic Programming

Incremental approach

- Example: Insertion sort
  - In the so far sorted subarray, insert a new single element into its proper place, resulting in the new sorted subarray
  - Example:

[....] [.........]

A[1 .. j-1] *A [j .. n]*

Key = A [j]

Divide and Conquer

Our life is frittered away by detail. Simplify, simplify.

— Henry David Thoreau

The control of a large force is the same principle as the control of a few men: it is merely a question of dividing up their numbers.

— Sun Zi, *The Art of War* (c. 400 C.E.), translated by Lionel Giles (1910)

Nothing is particularly hard if you divide it into small jobs.

— Henry Ford

Henry Ford (July 30, 1863 - April 7, 1947) was an American captain of industry and a business magnate.

Founder of the Ford Motor company

Sponsor of the development of the **assembly line technique of mass production**.

Henry Ford’s Assembly line

Divide and Conquer

- **Three crucial steps**
  - **Divide** the problem into smaller sub problems
  - **Conquer** the smaller subproblems recursively.
  - **Combine** solutions of the subproblems to get the solution of the original problem

Divide and conquer - First step

- **Divide/Break** the problem into smaller sub problems
  - For example, Problem P is divided into subproblems P1 and P2.
  - Also, P1 and P2 resemble the original problem and their size is small

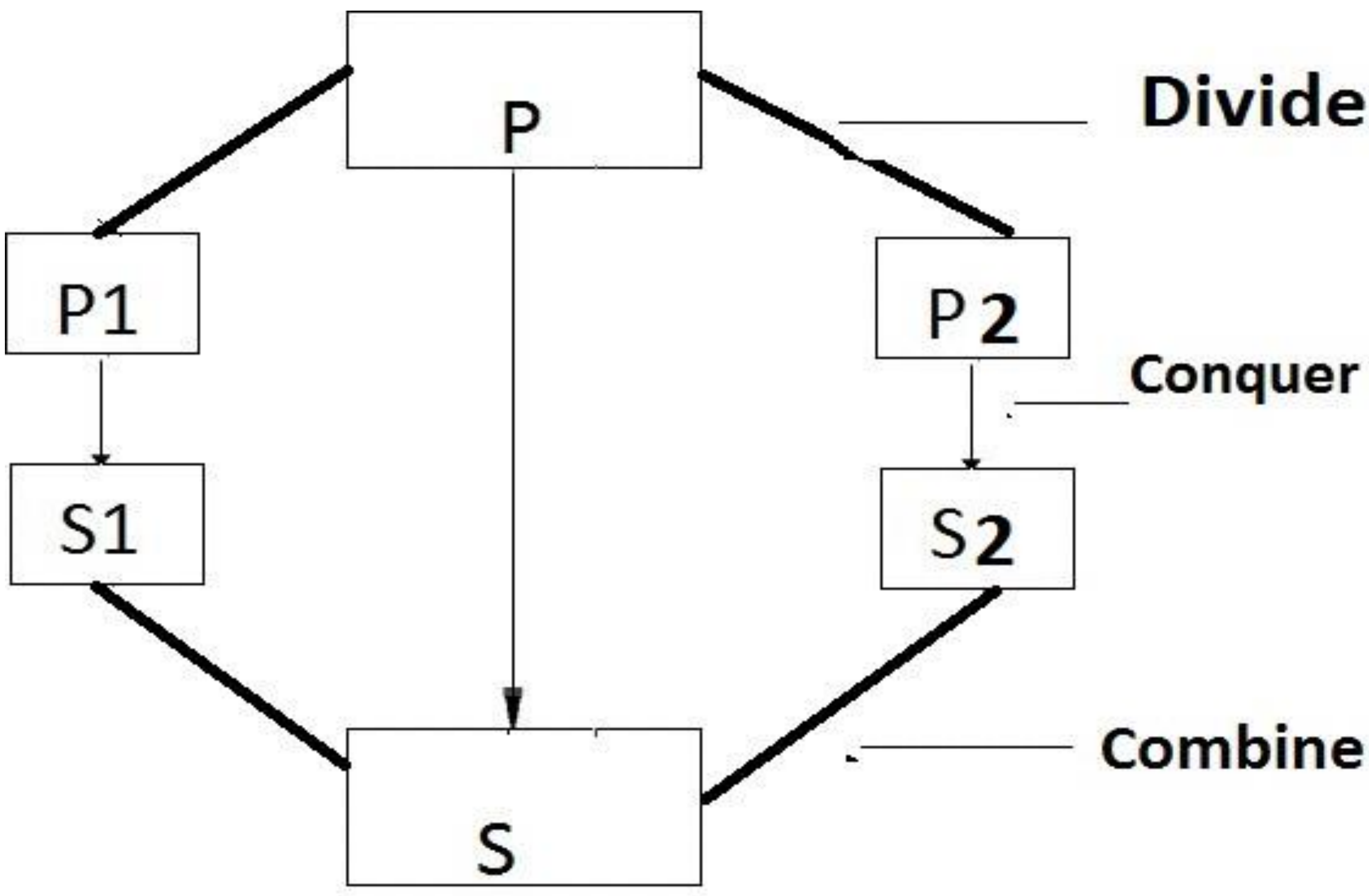
Divide and conquer - Second step

- Conquer/Solve the smaller subproblems recursively. If the subproblem size is small, solve them directly
  - P1 is solved to give S1, P2 is solved to give S2

Divide and conquer - Third step

- **Merge/Combine** these solutions to create a solution to the original problem
  - S1 and S2 are combined to give the solution S for the original problem P

Pictorial Representation: Divide and Conquer



Divide and Conquer (D & C)

- Most of the algorithms designed using D & C are recursive in nature
- **Recursive algorithms:** Call themselves recursively to solve the closely related subproblems
- **Examples**
  - Towers of Hanoi
  - Binary search
  - Merge Sort
  - Quick sort

Merge Sort

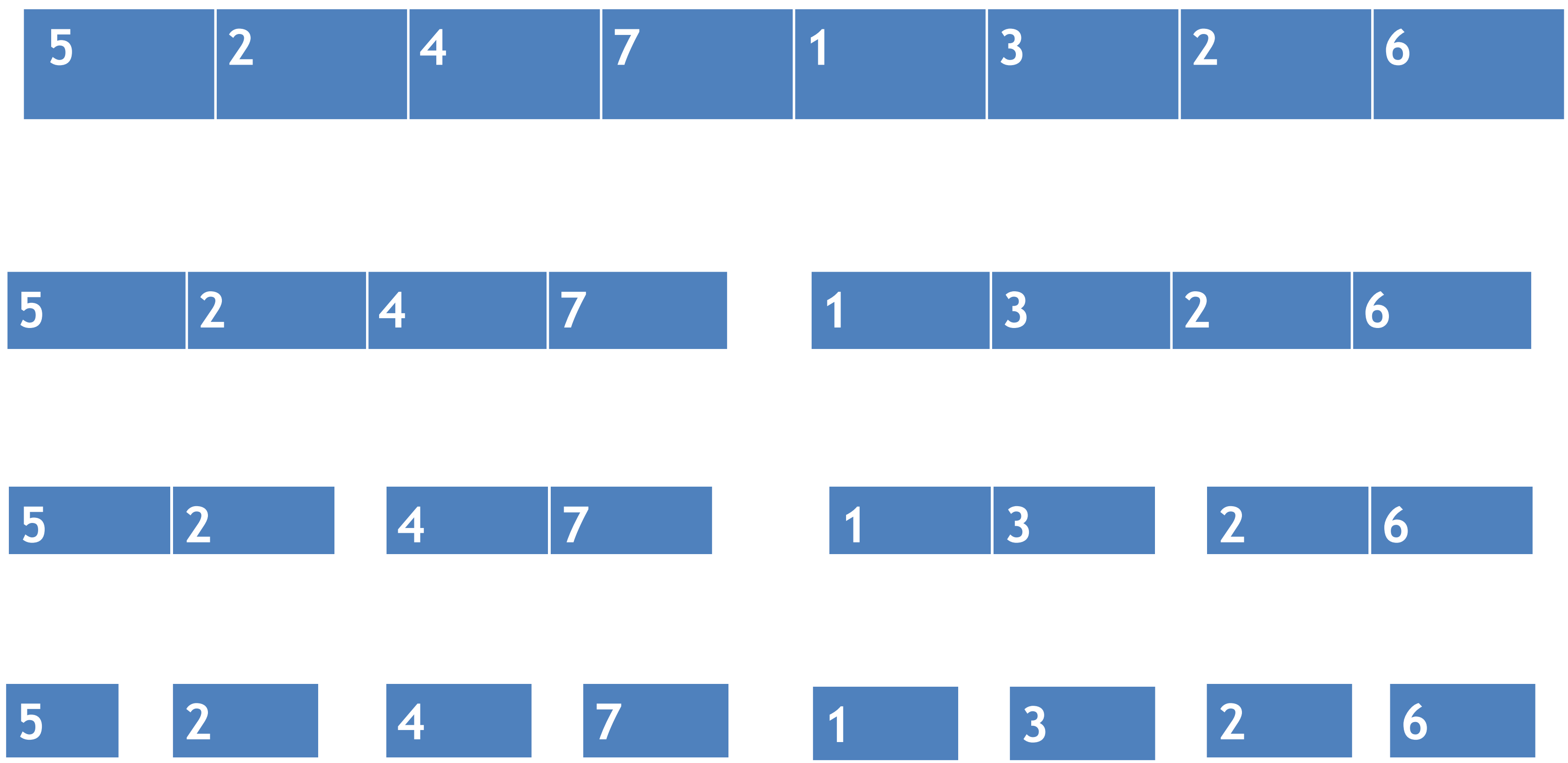
- Follows D & C paradigm

- Divide: Divide the n-element array into two subarrays of size n/2
- Conquer: Sort the two subarrays recursively
- Combine: Merge the two sorted subarrays to produce the sorted array

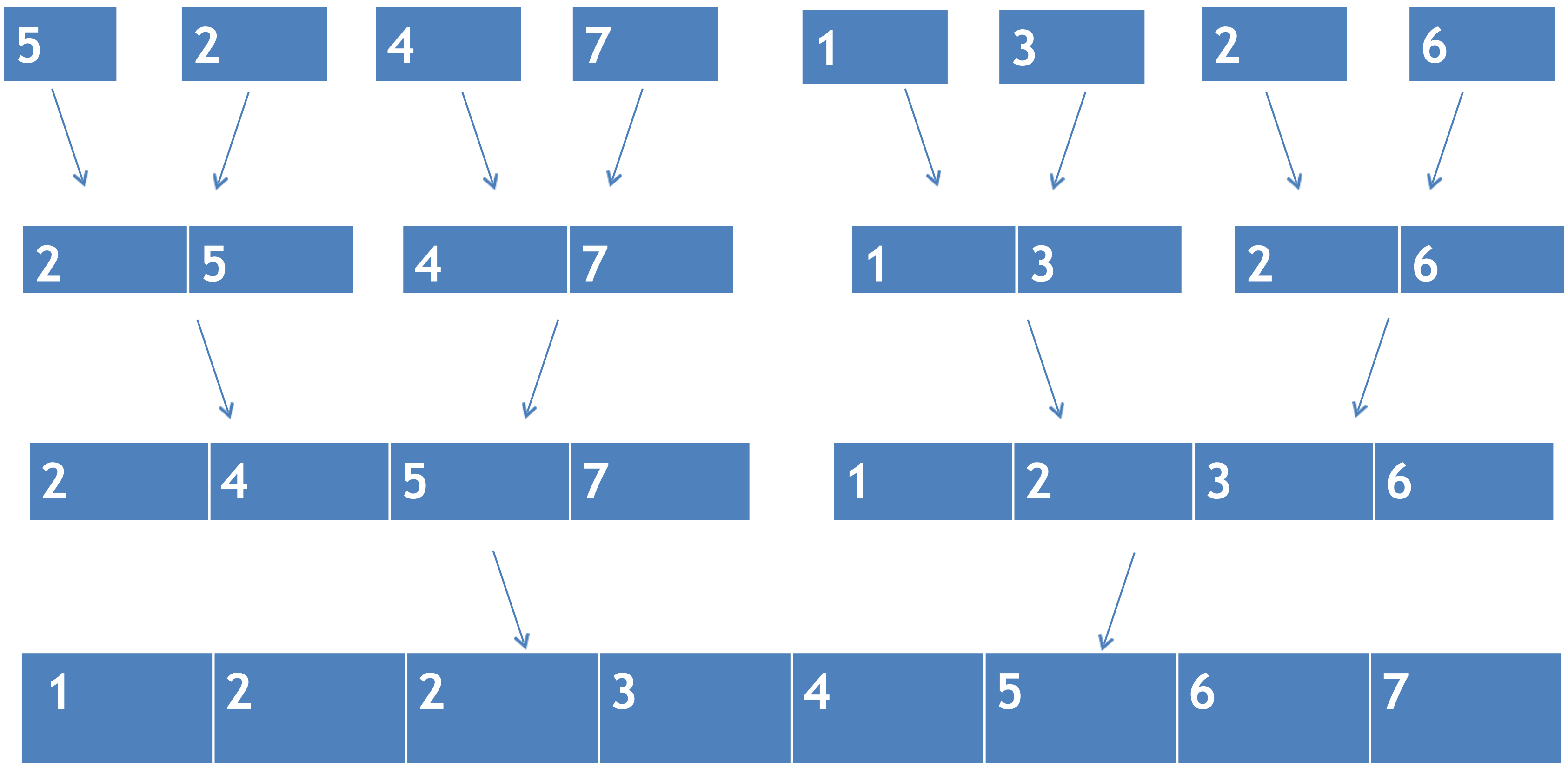
Merge Sort - Example

The operation of merge sort on the array

A= {5, 2, 4, 7, 1, 3, 2, 6}



Merging of sorted subarrays



Merge Sort – Recursive Algorithm

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

Ref: CLRS Book

MERGE-SORT

- If p >= r , the subarray has at most one element and is therefore already sorted.

- Otherwise, the divide step, computes an index q that partitions A[p...r] into two subarrays A[p…q] and A[q+1…r] containing (n/2) elements

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

Function call:

MERGE-SORT(A, 1, A.length)

Merge sort – Recursive algorithm

- **Base case:**
  - When the size of the subproblem is 1, we don't need to do any further
  - Its already sorted
- **Key operation:** Merging of two sorted arrays in the combine step
- Merge is done by calling another function Merge (A,p,q,r)

Thank You