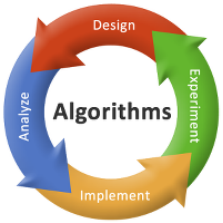
Sorting – Insertion Sort & Merge Sort

Course: Algorithms

Faculty: Dr. Rajendra Prasath



Autumn 2018

Sorting – Insertion Sort and Merge Sort

This lecture covers the two variations of sorting algorithms for sorting a set of n elements. The first one is the insertion sort and the second one is the merge sort. We present the algorithms and their complexity analysis. We also discuss the suitable data structures for each of the algorithm.

Recap: Sorting Algorithms

- Suggest a simple algorithm for Sorting n elements
  - Correctness: First Test whether will the algorithm work for a small set of the input? Then apply on a bigger set
  - Choose a suitable data structure
  - Perform complexity analysis
    - How much space and time required in the worst case?
  - Is the solution adaptable?
    - With the growing size of the input n
- How to get the tight bound of the sorting algorithm in terms of the running time?

Recap: Bubble Sort - Analysis

- Complexity Analysis
  - Worst Case
    - $O(n^2)$ comparisons and swaps
  - Average Case
    - $O(n^2)$ comparisons and swaps
  - Best Case
    - O(n) comparisons and swaps – but requires at least two passes through the input data

- Is this the Stable Sorting Algorithm?
  - Maintain relative (numerical or lexicographical) order?

  - Yes, bubble sort does …

- O(1) extra space

- Adaptive: O(n) when nearly sorted

Insertion Sort

- How many of you are good in "Playing Cards"?

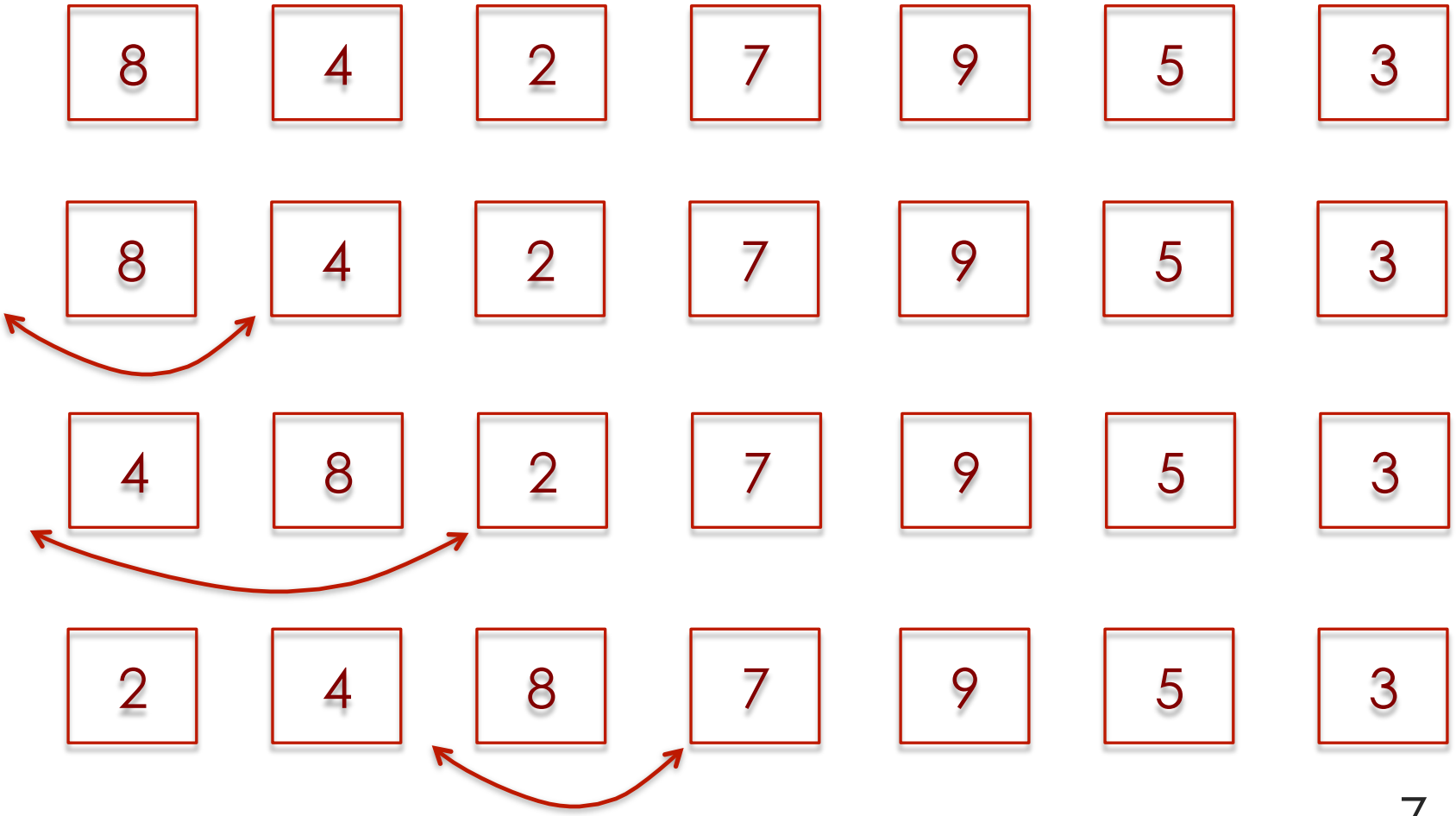
Insertion Sort

- Consider the following input of size n (= 7)

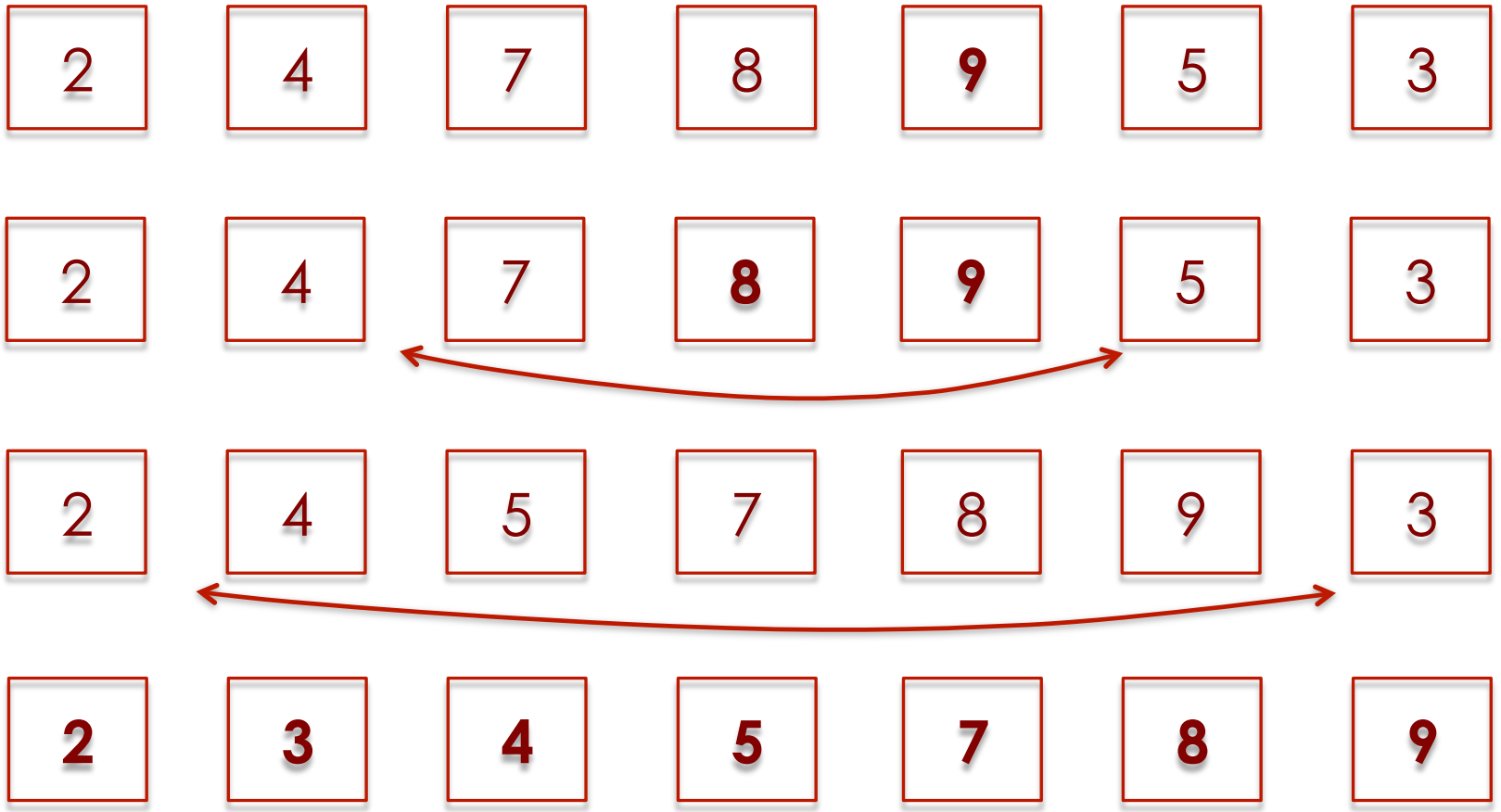
- Decide Data Structure
  - Array or list or any other data structures
    - Why do you consider that data structure?

- It is same as how to quickly sort the cards in hand
- Pick a card (any??) and insert it in place

Insertion Sort – An Illustration

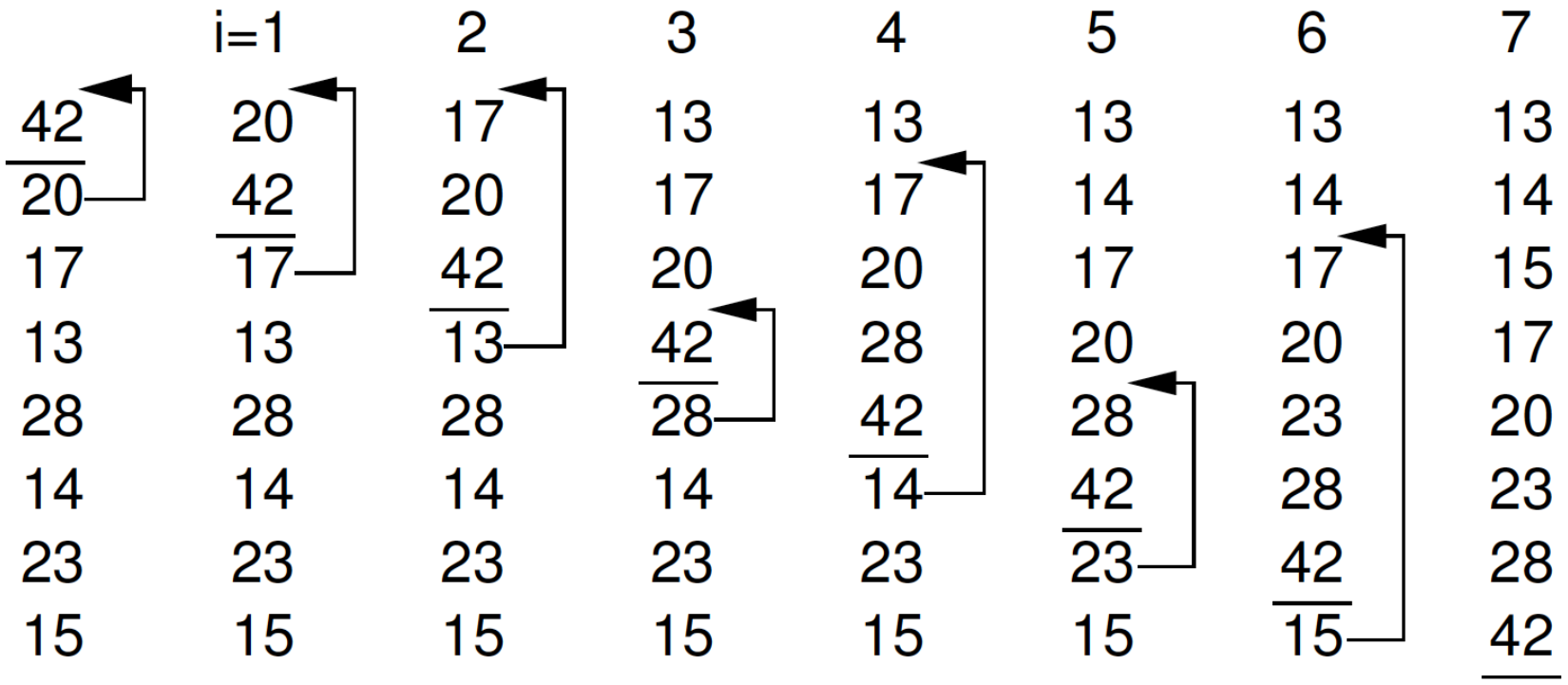


Insertion Sort – An Illustration



- Final Sorted Sequence – Compute Complexity

Insertion Sort – Example 2



Insertion Sort – Algorithm

- Task: Sort an array $A[]$ of size n

  procedure **InsertionSort()**

  input: An array A of n elements

  output: The Sorted Sequence A

  begin

  for each i **from** 1 **to** $n-1$

  Pick an element $A[i]$

  insert $A[i]$ into sorted sequence $A[0...I - 1]$

  endfor

  end

- Outcome: The Sorted array A of size n

Insertion Sort – Characteristics

- Auxiliary Space: O(1)

- Boundary Cases:
  - Maximum time is taken for reversely sorted sequence
  - Minimum time is taken for already sorted sequence

- Paradigm: Incremental Approach

- Sorting the elements **In-Place**
- This is a **Stable Sorting** algorithm

- Computational Complexity ?
  - Running time in terms of the input size n

Insertion Sort – Analysis

- Computational Complexity of Insertion Sort

- Worst Case
  - $O(n^2)$ – How?
- Average Case
  - $O(n^2)$ – How?
- Best Case
  - $O(n)$ – How?

- Can you improve insertion sort in anyway?

Binary Insertion Sort

- Can we reduce the number of Comparisons?
  - Use binary search to reduce the number of comparisons

- **Binary Insertion Sort**
  - Using binary search, find the proper location to insert the selected item at each iteration
  - In normal insertion, sort it takes O(k) (at k^{th} iteration) in the worst case
  - This can be reduced to O(logk) by using binary search

  - Still the worst case running time is $O(n^2)$
    - The series of swaps required for each insertion

Merge Sort - Introduction

- Consider the data generated online?
  - Volume of the data
    - Big Data Problem
- Can we take all n values at the same time and sort them in subsequent steps?
  - Do we need to keep all data in memory at the same time?
  - Do we really have computational power to perform such task (assume the limited computational power – like your Laptop / Desktop)
- How to handle the sorting problem
  - Divide and Conquer ??

Merge Sort – Basic Idea

- Basic Idea:
  - Divide the input into smaller instances
  - Sort each smaller instances
  - Merge these sorted instances to get the overall sorted sequence

- Issues:
  - How to divide them in an effective way?
  - Can we always divide the input into near equal halves?

  - Can we adopt the results of Master's Theorem?

Merge Sort – Steps

- Divide and Conquer Approach
- Step – 1:
  - If only one element is in the list (already sorted) then return the element
- Step – 2:
  - Divide the list recursively into two halves until it can not be divided anymore
- Step – 3:
  - Merge the smaller lists into the new list in sorted order

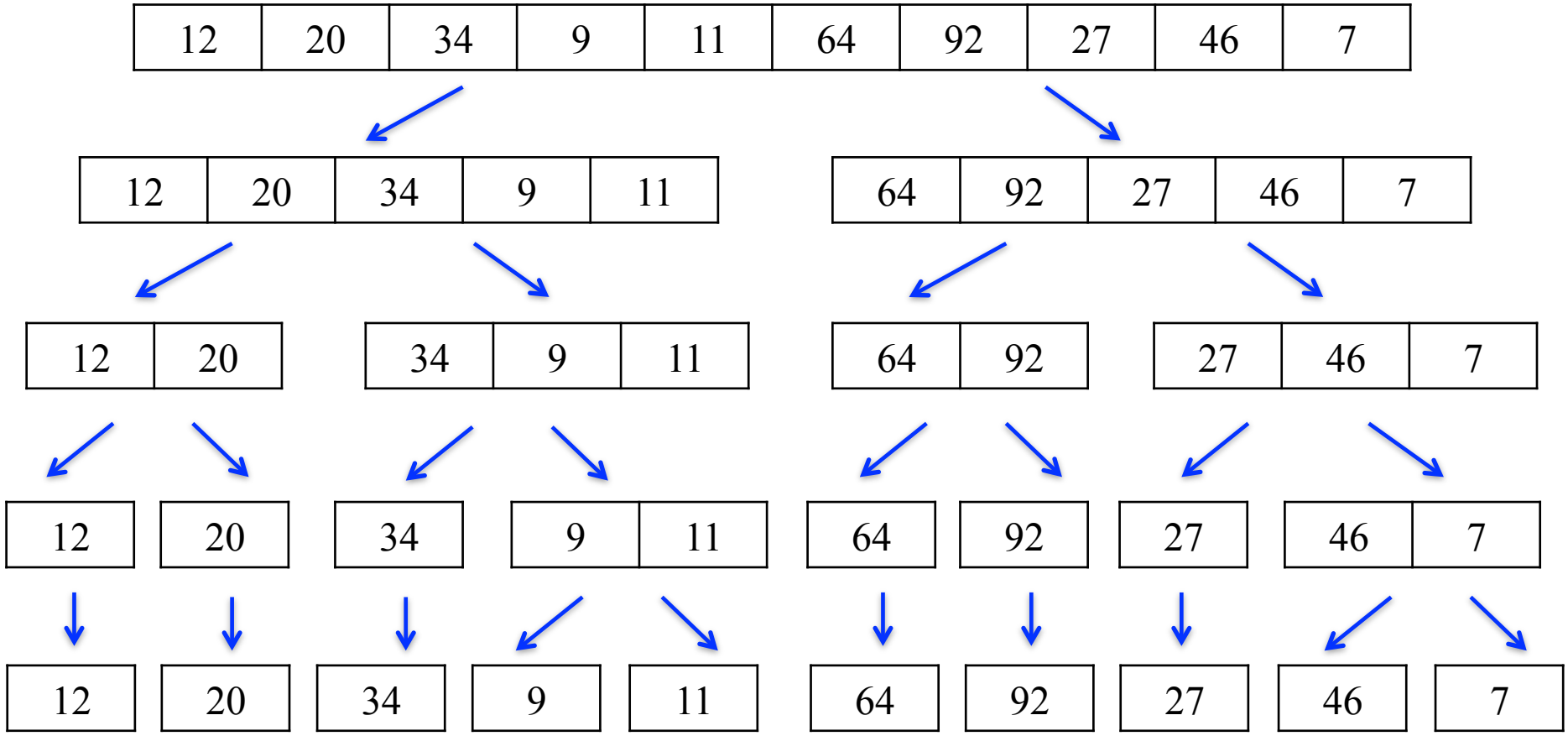
Merge Sort – Algorithm

- Divide and Conquer Approach – A Rough Skeleton

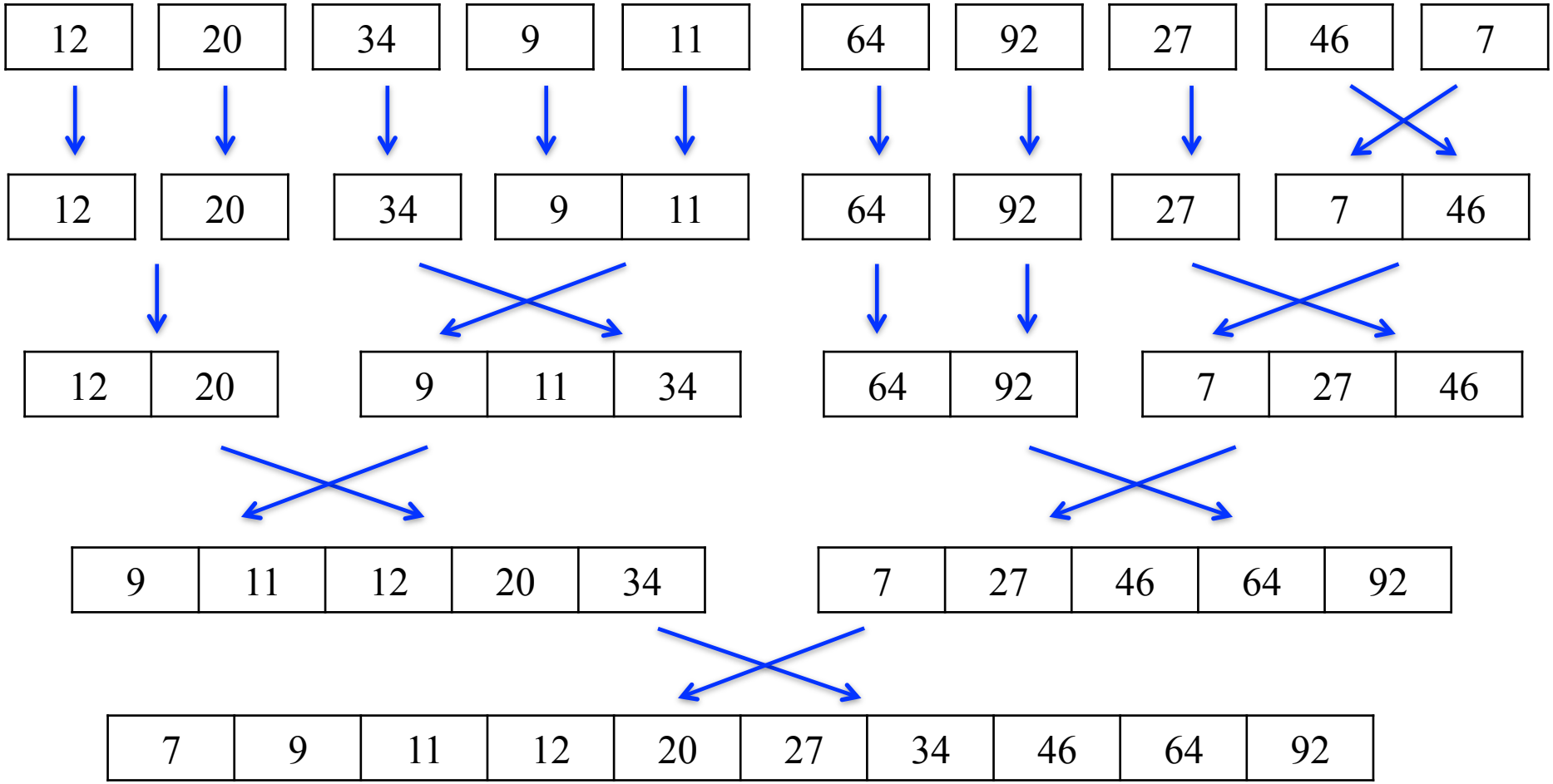
```
procedure MergeSort(A, n) {
    input: An array A of n elements
    output: The sorted array A of n elements

    begin
        if n <= 1 return A
        A1 = First half of the items in A
        A2 = Second half of the items in A
        return merge( MergeSort(A1),
                      MergeSort(A2));
    end
```

Merge Sort – An Illustration

- Divide and Conquer Approach



Merge Sort – An Illustration



Merge Sort – Complexity

- Can we apply Master's Theorem?
- The Recurrence Relation is:

$$t(n)=2*t(n/2)+O(n)$$

where t(n/2) is the time to solve a problem of size n/2

=> $t(n)=2*[2*t(n/4)+O(n/2)]+O(n)$

$t(n)=2*[2*[2*t(n/8)+O(n/4)]+O(n/2)]+O(n)$

This equation reduces to $t(n)=2^3*t(n/2^3)+O(3*n)$

…

In general, $t(n)=2^k*t(n/2^k)+O(k*n)$

Base case: problem size is 1 → t(1) is a constant

Thus we get the closed-form formula for

$$t(n)=n*c+O(n*\log(n))$$

➔ Complexity = O(nlogn)

Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

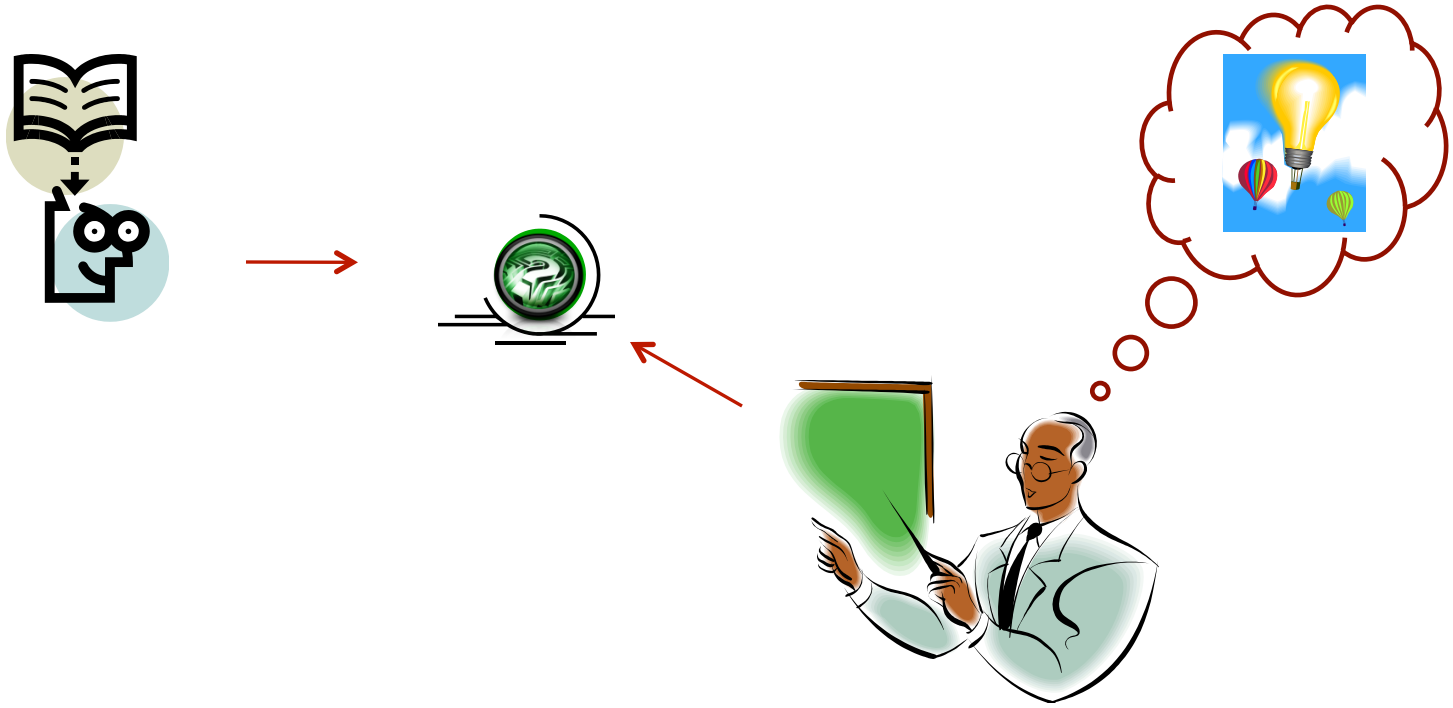
- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Assistance

- You may post your questions to me at any time
- You may meet me in person on available time or with an appointment
- TA s would assist you to clear your doubts.
- You may leave me an email any time (email is the best way to reach me faster)

Thanks …



… Questions ???