



**WOLLO UNIVERSITY**  
**KOMBOLCHA INSTITUTES OF TECHNOLOGY**  
College of Informatics

# **Analysis of Algorithms**

---

## **Chapter 2**

### **Divide and Conquer**

Belachew N.  
nbelay2112@gmail.com

# Outline

- ✓ The General Method
- ✓ Binary Search
- ✓ Finding Maximum and Minimum
- ✓ Merge Sort
- ✓ Quick Sort
- ✓ Selection Sort

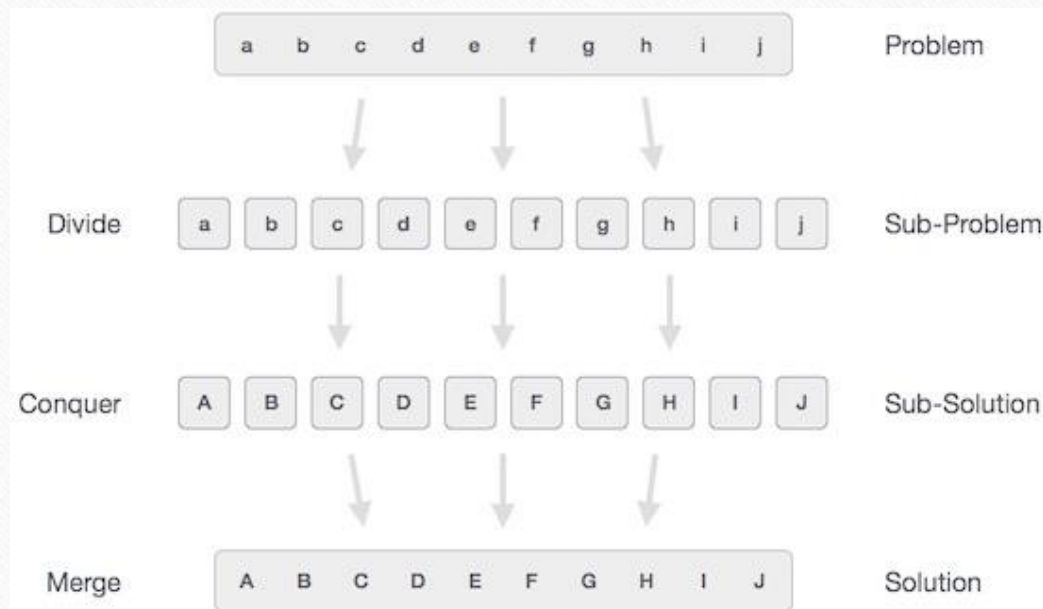


# Introduction

- ✓ In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently.
- ✓ When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where no more division is possible.
- ✓ Those "atomic" smallest possible sub-problem (fractions) are solved.
- ✓ The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.

## ...cont'd

- ✓ **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
- ✓ **Conquer** the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- ✓ **Combine** the solutions to the subproblems into the solution for the original problem.



## ...cont'd

- ✓ There are a number of computer algorithms based on divide-and-conquer programming approach:
  - Binary search
  - Finding maximum and minimum
  - Merge sort
  - Quick sort
  - Selection sort



# Binary Search

- ✓ Binary Search is applied on the sorted array or list of large size.
- ✓ The only limitation is that the array or list of elements must be sorted for the binary search algorithm to work on it.
- ✓ Binary Search is one of the fastest searching algorithms.
- ✓ It works on the principle of divide and conquer technique.

# How it works

- ✓ We basically ignore half of the elements just after one comparison.
- ✓ Step 1: Compare  $x$  with the middle element.
- ✓ Step 2: If  $x$  matches with middle element, we return the mid index.
- ✓ Step 3: Else If  $x$  is greater than the mid element, then  $x$  can only lie in right half subarray after the mid element. So we recur for right half and start with step 1
- ✓ Step 4: Else ( $x$  is smaller) then pick the elements to the left of the middle index, and start with Step 1.
- ✓ When a match is found, return the index of the element matched.
- ✓ If no match is found, then return -1

## ...cont'd

- ✓ Since each comparison binary search uses halves the search space, we can assert and easily prove that binary search will never use more than  $O(\log N)$  comparisons to find the target value.



# Finding Maximum and Minimum

- ✓ Let us consider another simple problem that can be solved by the divide and-conquer technique.
- ✓ The problem is to find the maximum and minimum items in a set of  $n$  elements.
- ✓ Let  $P = (n, a[i], \dots, a[j])$  denote an arbitrary instance of the problem.
- ✓ Let  $\text{small}(P)$  be true when  $n \leq 2$ .
  - ✓ In this case, the maximum and minimum are  $a[i]$  if  $n = 1$ .
  - ✓ If  $n = 2$ , the problem can be solved by making one comparison.

## ...cont'd

- ✓ If the list has more than two elements,  $P$  has to be divided into smaller instances.
- ✓ For example, we might divide  $P$  into the two instances  $P1 = (n/2, a[1], \dots, a[n/2])$  and  $P2 = (n - n/2, a[n/2 + 1], \dots, a[n])$ .
- ✓ After having divided  $P$  into two smaller sub problems, we can solve them by recursively invoking the same divide and conquer algorithm.
- ✓ If  $MAX(P)$  and  $MIN(P)$  are the maximum and minimum of the elements of  $P$ , then  $MAX(P)$  is the larger of  $MAX(P1)$  and  $MAX(P2)$  also  $MIN(P)$  is the smaller of  $MIN(P1)$  and  $MIN(P2)$ .

## ...cont'd

- ✓ The situation of set sizes one ( $i=j$ ) and two ( $i=j-1$ ) are handled separately.
- ✓ For sets containing more than two elements, the midpoint is determined and two new sub problems are generated.
- ✓ When the maxima and minima of this sub problems are determined, the two maxima are compared and the two minima are compared to achieve the solution for the entire set.



## ...cont'd

MaxMin(i, j, max, min)

// a[1:n] is a global array. Parameters i and j  
are integers,  $1 \leq i \leq j \leq n$ .

// The effect is to set max and min to the  
largest and //smallest values in a[i:j].

if (i=j) then max := min := a[i]; //Small(P)

else if (i=j-1) then // Another case of  
Small(P)

{

    if (a[i] < a[j]) then max := a[j]; min  
    := a[i];

    else max := a[i]; min := a[j];

else

    // if P is not small, divide P into sub-  
    problems.

    mid := ( i + j )/2;

    MaxMin( i, mid, max, min );

    MaxMin( mid+1, j, max1, min1 );

    // Combine the solutions.

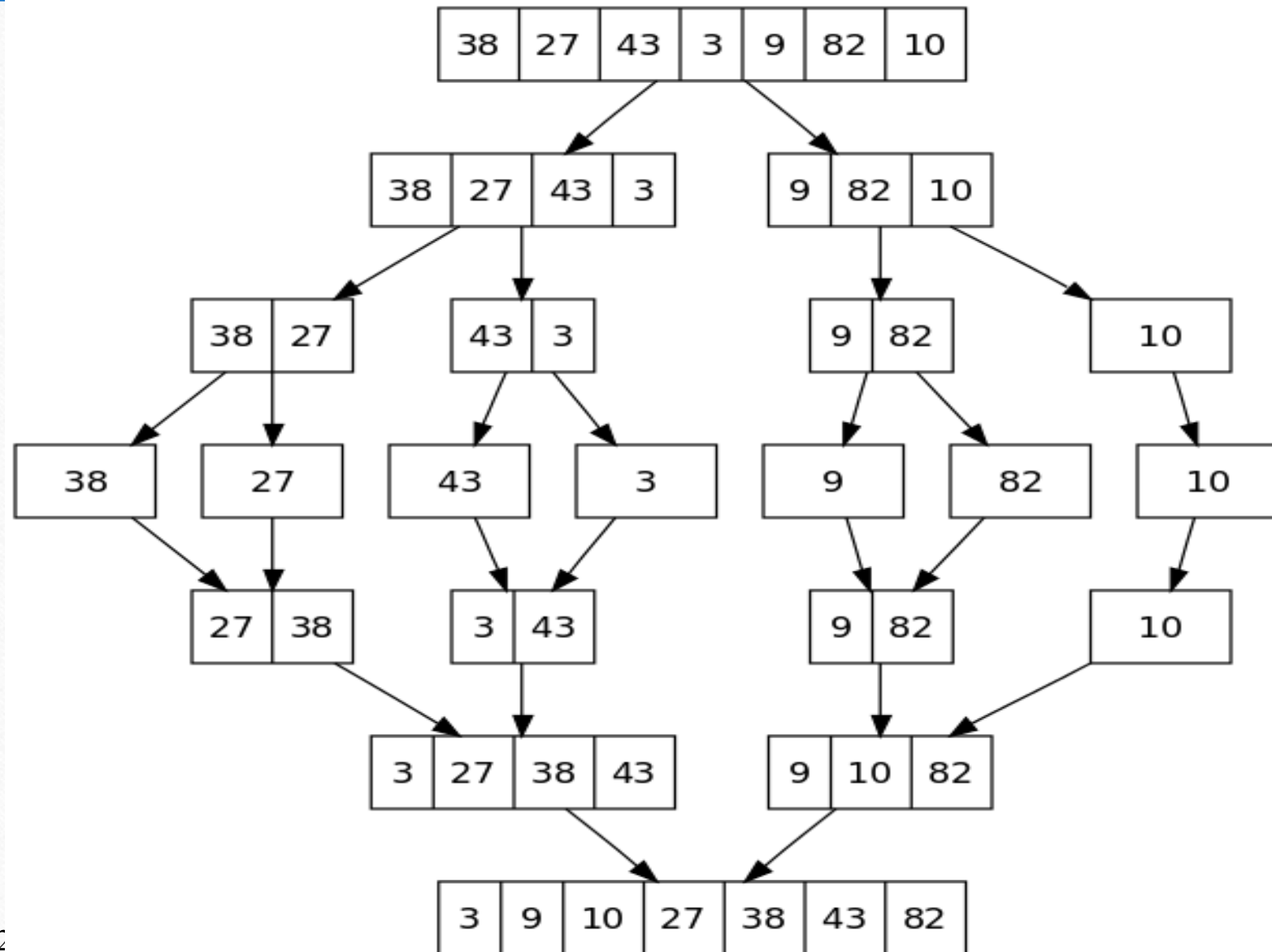
    if (max < max1) then max := max1;

    if (min > min1) then min := min1;

}

# Merge Sort

- ✓ Merge Sort is a divide and conquers algorithm in which original data is divided into a smaller set of data to sort the array.
- ✓ In merge sort the array is firstly divided into two halves, and then further sub-arrays are recursively divided into two halves till we get  $N$  sub-arrays, each containing 1 element.
- ✓ Then, the sub-arrays are repeatedly merged, to produce new array until there is one array remaining.
- ✓ This will be our sorted array at the end.





## ...cont'd

- ✓ So in the above picture we see that the size of the array is ( $n=7$ ).
- ✓ As we know  $\lceil 7/2 \rceil = 3$ , so while the first division of array, the size of one part will be 3 and another part will be ( $7-3=4$ ).
- ✓ Similarly, the array will recursively divide into two halves till we get all sub-array containing of exactly 1 element.
- ✓ Then each element is compared with the adjacent array to sort and merge the two adjacent arrays till the complete array is merged.
- ✓ Finally, all the elements are sorted and merged.
- ✓ Merge sort complexity is  $T(n) = O(n \log_2 n)$ .

# Quick Sort

- ✓ Quicksort is a fast sorting algorithm, which is used not only for educational purposes, but widely applied in practice.
- ✓ On the average, it has  $O(n \log n)$  complexity, making quicksort suitable for sorting big data volumes.
- ✓ The divide-and-conquer strategy is used in quicksort.

## ...cont'd

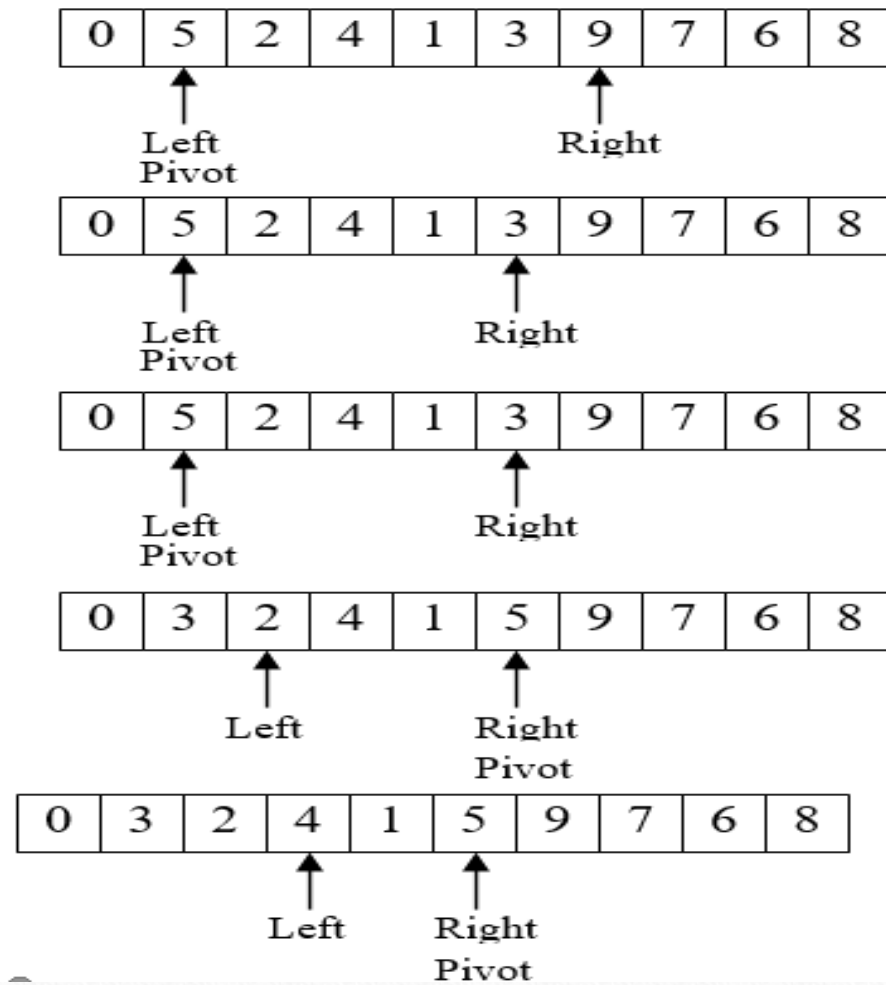
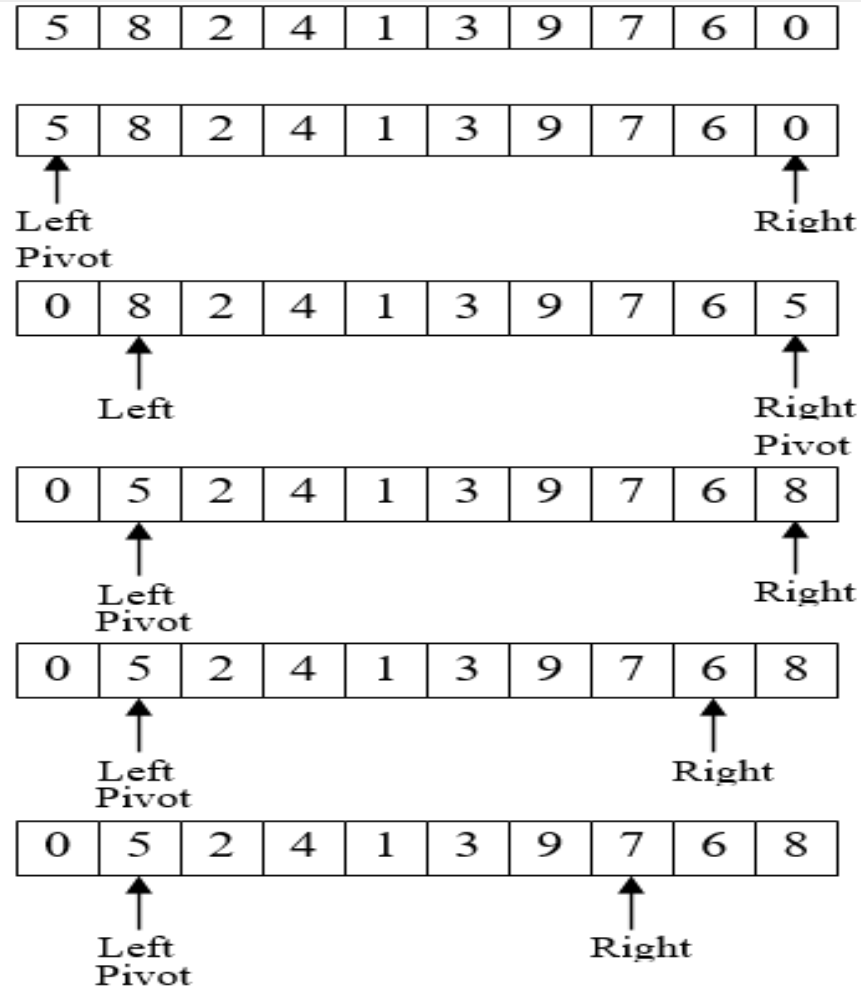
- ✓ The divide-and-conquer strategy is used in quick sort.
- ✓ **Choose a pivot value:** mostly the first element is taken as the pivot value.
- ✓ **Partition:** rearrange elements in such a way, that all elements which are lesser than the pivot go to the left part of the array and all elements greater than the pivot, go to the right part of the array.
- ✓ **Sort both parts:** apply quick sort algorithm recursively to the left and the right parts.



# Partition algorithm in detail

- ✓ There are two indices **i** and **j** and at the very beginning of the partition algorithm **i points to the first** element in the array and **j points to the last** one.
  - Then algorithm moves **i** forward, until an element with value greater or equal to the pivot is found.
  - Index **j** is moved backward, until an element with value lesser or equal to the pivot is found.
- ✓ If  $i \leq j$  then they are swapped and **i** steps to the next position (**i + 1**), **j** steps to the previous one (**j - 1**). Algorithm stops, when **i** becomes greater than **j**.
- ✓ After partition, all values before **i<sup>th</sup>** element are less or equal than the pivot and all values after **j<sup>th</sup>** element are greater or equal to the pivot.
  - *Example:* Sort {1, 12, 5, 26, 7, 14, 3, 7, 2} using quicksort.

# Example



0	3	2	4	1	5	9	7	6	8
---	---	---	---	---	---	---	---	---	---

↑   ↑  
 Left Right  
 Pivot

0	3	2	4	1	5	9	7	6	8
---	---	---	---	---	---	---	---	---	---

↑   ↑   ↑   ↑  
 Left Right Left Right  
 Pivot

0	3	2	4	1	5	8	7	6	9
---	---	---	---	---	---	---	---	---	---

↑   ↑   ↑   ↑  
 Left Right Left Right  
 Pivot

0	3	2	4	1	5	8	7	6	9
---	---	---	---	---	---	---	---	---	---

↑   ↑   ↑   ↑  
 Left Right Left Right  
 Pivot

0	3	2	4	1	5	8	7	6	9
---	---	---	---	---	---	---	---	---	---

↑   ↑   ↑   ↑  
 Left Right Left Right  
 Pivot

0	3	2	4	1	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

↑   ↑   ↑   ↑  
 Left Right Left Right  
 Pivot

5/18/2020

0	1	2	4	3	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

↑   ↑   ↑   ↑  
 Left Right Left Right  
 Pivot

0	1	2	4	3	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

↑   ↑  
 Left Right  
 Pivot

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

↑   ↑  
 Left Right  
 Pivot

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

↑   ↑  
 Left Right  
 Pivot

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



# Selection Sort

- Loop through the array from  $i=0$  to  $n-1$ .
- Select the smallest element in the array from  $i$  to  $n$
- Swap this value with value at position  $i$ .
- ✓ This algorithm is not suitable for large data.

# Example



# Review Questions

1. Explain divide and conquer strategy?
2. What is the difference between merge sort and quick sort?
3. Write the algorithm of selection sort and analysis its time complexity?



# End of Ch.2

---

Questions, Ambiguities, Doubts, ... ???