Course: Advanced Algorithm and Problem Solving

WEEK 2 - Divide and Conquer Algorithms

Lemlem Kassa (Ph.D.)

Addis Ababa Science and Technology University, Ethiopia

April, 2025

Week 2: Divide and Conquer Algorithms

Content

1. Introduction to Divide and conquer algorithms
2. Introduction to Recursive Algorithm
3. Divide and Conquer Applications
   - Merge Sort
   - Quick Sort
   - Binary Search

Lecture Learning Outcome

➢By the end of this lecture students are expected to understand :-

- Divide and conquer algorithm
- Recursive Algorithm Approach
- Algorithms which apply divide and conquer methods
- Merge sort, Quick Sort, and Binary Search algorithms, their time complexities and application areas.

1. Introduction to Divide and conquer algorithm

Divide and conquer algorithm

- Strategies of solving a large problem :-
  - breaking the problem into smaller sub-problems
  - solving the sub-problems, and
  - combining them to get the desired output.
- To use the divide and conquer algorithm, **recursion** is used.

[1]. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, MIT Press, 2009. Page-39

1. Introduction to Divide and conquer algorithm

Steps involved in **Divide and Conquer Algorithms**

1. **Divide**: Divide the given problem into sub-problems using recursion.
2. **Conquer**: Solve the smaller sub-problems recursively. If the subproblem is small enough, then solve it directly.
3. **Combine:** Combine the solutions of the sub-problems that are part of the recursive process to solve the actual problem.

Advantages of Divide and Conquer Algorithm

- This approach is suitable for multiprocessing systems.
- It makes efficient use of memory caches.

[1]. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, MIT Press, 2009. Page-77

2. Introduction to Recursive Algorithm

Recursive Algorithm

- An algorithm that uses recursion to solve a problem.
- Recursive algorithms typically have two parts:
  - *Base case:* Which is a condition that stops the recursion.
  - *Recursive case:* Which is a call to the function itself with a smaller version of the problem.

[2] Geeks for Geeks, Recursive Algorithms, Kartik, 2024. https://www.geeksforgeeks.org/recursion-algorithms/

2. Introduction to Recursive Algorithm …(cont'd)

Types of Recursion

- *Direct recursion*: This is typified by the factorial implementation where the methods call itself.
- *In-Direct recursion*: This happens where one method, say method A, calls another method B, which then calls method A. This involves two or more methods that eventually create a **circular call sequence**.
  - *Head recursion*: The recursive call is made at the beginning of the method.
  - *Tail recursion*: The recursive call is the last statement

[2] Geeks for Geeks, Recursive Algorithms, Kartik, 2024. https://www.geeksforgeeks.org/recursion-algorithms/

2. Introduction to Recursive Algorithm …(cont'd)

When to Use Recursion?

- Recursion is a powerful technique that can be used to solve a wide variety of problems.
- However, it is important to use recursion carefully, as it can lead to stack overflows if not used properly.
- **Recursion should be used when:**
  - The problem can be broken down into smaller subproblems that can be solved recursively.
  - The base case is easy to identify.
  - The recursive calls are tail recursive.

[2] Geeks for Geeks, Recursive Algorithms, Kartik, 2024.
https://www.geeksforgeeks.org/recursion-algorithms/

2. Introduction to Recursive Algorithm …(cont'd)

Examples : Some common examples of recursion:

- **Example 1: Factorial:** The factorial of a number n is the product of all the integers from 1 to n. The factorial of n can be defined recursively as:

```
factorial(n) = n * factorial(n-1)
```

- **Example 2: Fibonacci sequence:** The Fibonacci sequence is a sequence of numbers where each number is the sum of the two preceding numbers.
  - The Fibonacci sequence can be defined recursively as:

```
fib(n) = fib(n-1) + fib(n-2)
```

[2] Geeks for Geeks, Recursive Algorithms, Kartik, 2024.
https://www.geeksforgeeks.org/recursion-algorithms/

2. Introduction to Recursive Algorithm …(cont'd)

Applications of Recursion Algorithms

- **Tree and Graph Traversal**: Depth-first search (DFS) and breadth-first search (BFS)
- **Dynamic Programming**: Solving optimization problems by breaking them into smaller subproblems
- **Divide-and-Conquer:** Solving problems by dividing them into smaller parts, solving each part recursively, and combining the results
- **Backtracking:** Exploring all possible solutions to a problem by recursively trying different options, etc.

[2] Geeks for Geeks, Recursive Algorithms, Kartik, 2024.
https://www.geeksforgeeks.org/recursion-algorithms/

3. Divide and Conquer Applications …(cont'd)

a) Binary Search

- A searching algorithm for finding an element's position in a sorted array.
- In this approach, the element is always searched in the middle of a portion of an array.

Note

- Binary search can be implemented **only on a sorted list of items**.
- If the elements are not sorted already, we need to sort them first.

[3]. Programiz,Binary Search (With Code). https://www.programiz.com/dsa/binary-search

3. Divide and Conquer Applications …(cont'd)

How does Binary Search Works?

- Binary Search Algorithm can be implemented in two ways:-
  - Iterative Method
  - Recursive Method: - follows the divide and conquer approach.

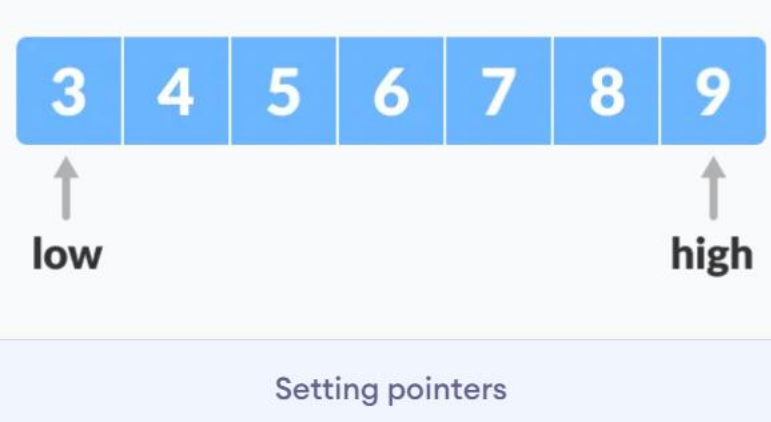
Example:-

- Consider the following array to understand the approach
- Let **x = 4** be the element to be searched

[3]. Programiz,Binary Search (With Code). https://www.programiz.com/dsa/binary-search

3. Divide and Conquer Applications …(cont'd)

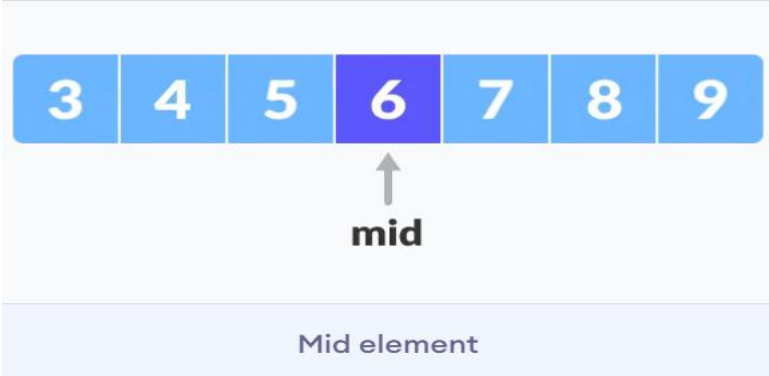
- **Step-1** : Set two pointers low and high at the lowest and the highest positions respectively



- **Step-2:** Find the middle position mid of the array ie.

  mid = (low + high)/2,

  and arr[mid] = 6



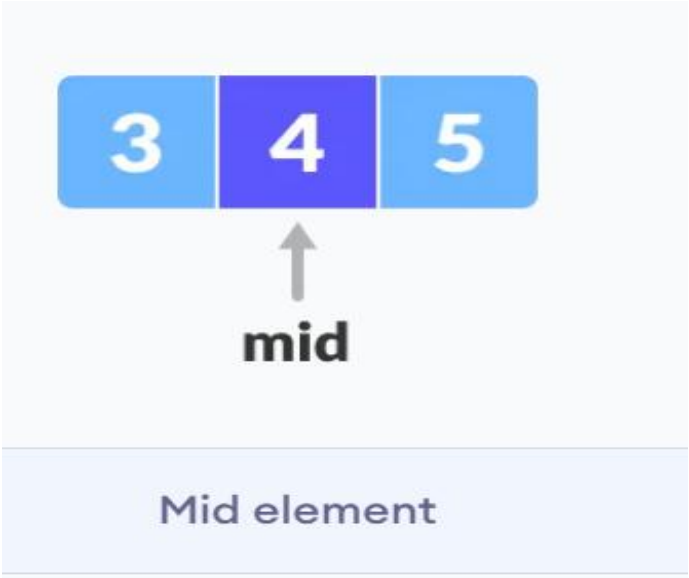
3. Divide and Conquer Applications …(cont'd)

- **Step-3**:- **If x == arr[mid],** then **return mid**. **Else**, compare the element to be searched with **arr[mid].**
- **Step-4**:- If **x > arr[mid],** compare x with the middle element of the elements on the right side of arr[mid]. This is done by setting low to **low = mid + 1.**
- **Step-5**:- **Else,** compare x with the middle element of the elements on the left side of arr[mid]. This is done by setting high to **high = mid - 1**.

[3]. Programiz,Binary Search. https://www.programiz.com/dsa/binary-search

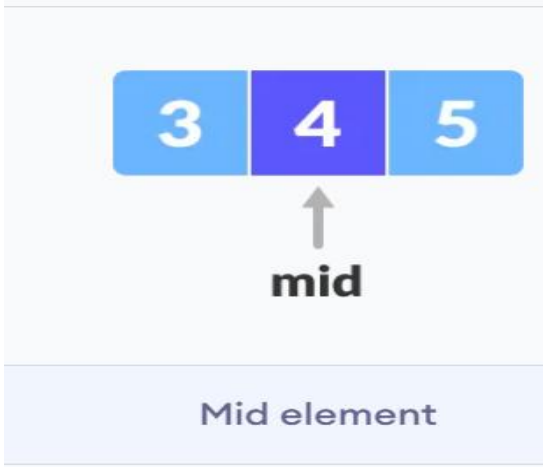
3. Divide and Conquer Applications …(cont'd)

Step-6: Repeat steps 2 to 5 until low meets high



Step-7: Repeat steps 3 to 6 until low meets high

x = 4 is found



[3]. Programiz,Binary Search (With Code). https://www.programiz.com/dsa/binary-search

3. Divide and Conquer Applications …(cont'd)

Binary Search Algorithm

- **Recursive Method**

```
binarySearch(arr, x, low, high)
    if low > high
        return False
    else
        mid = (low + high) / 2
        if x == arr[mid]
            return mid
        else if x > arr[mid]      // x is on the right side
            return binarySearch(arr, x, mid + 1, high)
        else                      // x is on the left side
            return binarySearch(arr, x, low, mid - 1)
```

Iteration Method

```
do until the pointers low and high meet each
other.
    mid = (low + high)/2
    if (x == arr[mid])
        return mid
   else if (x > arr[mid]) // x is on the right side
        low = mid + 1
    else                        // x is on the left side
        high = mid -1
```

[3]. Programiz,Binary Search (With Code). https://www.programiz.com/dsa/binary-search

3. Divide and Conquer Applications …(cont'd)

Binary Search Applications

- In libraries of Java, .Net, C++ , the binary search is used to pinpoint the place where the error happens.

Complexities of Binary Search

Time Complexities

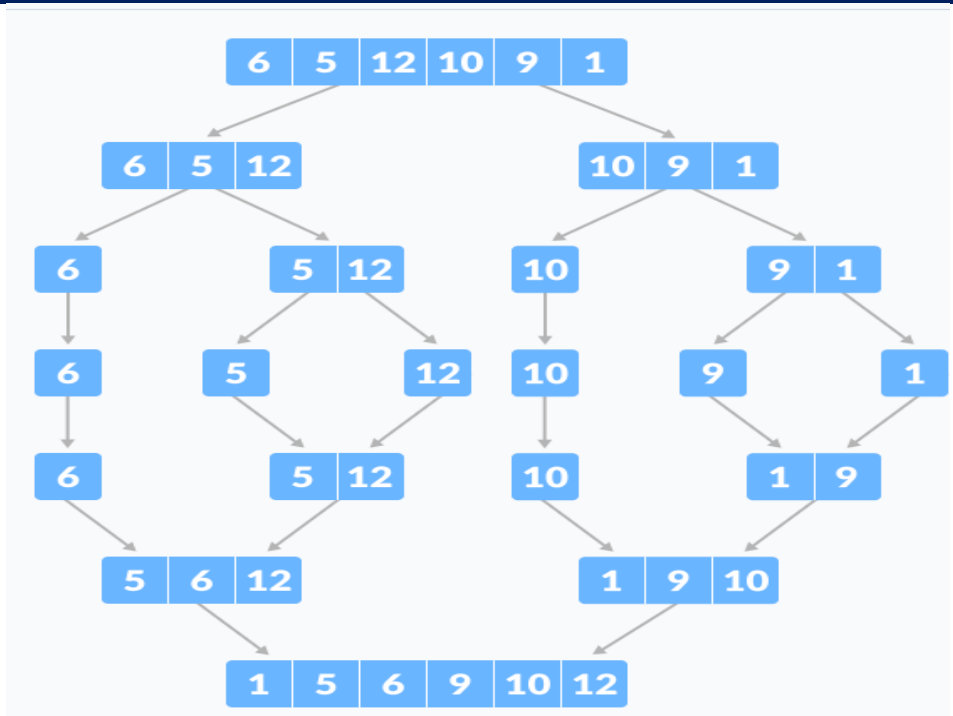
- Best case complexity: O(1)
- Average case complexity: O(log n)
- Worst case complexity: O(log n)

[3]. Programiz,Binary Search (With Code). https://www.programiz.com/dsa/binary-search

3. Divide and Conquer Applications …(cont'd)

b) Merge Sort

- One of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm.
- Here, a problem is divided into multiple sub-problems. Each sub-problem is solved individually.
- Finally, sub-problems are combined to form the final solution

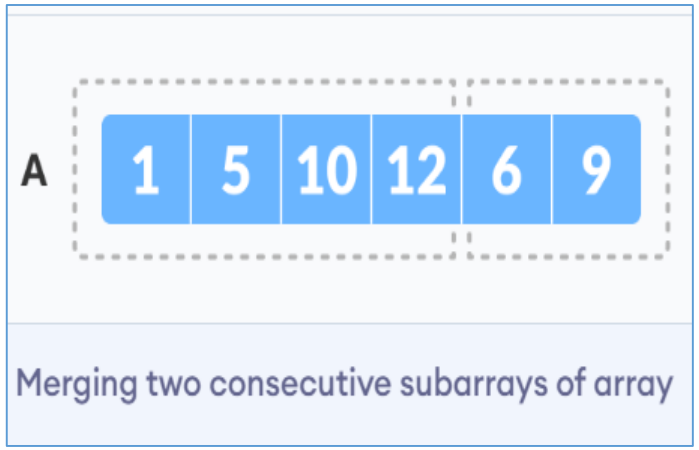


3. Divide and Conquer Applications …(cont'd)

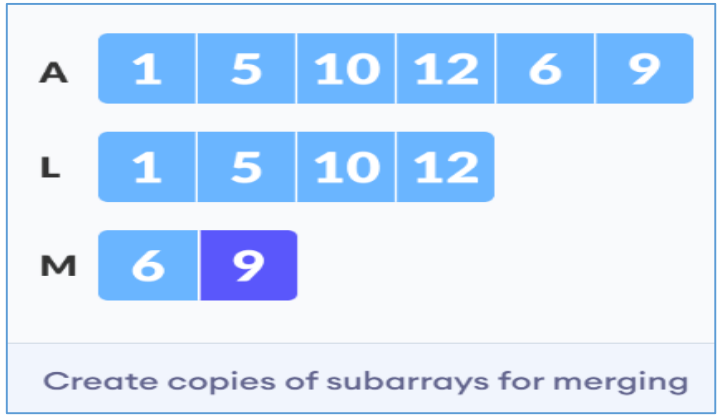
- Merge Sort is a divide and conquers algorithm, it divides the given array into equal parts and then merges the 2 sorted parts.
- **There are 2 main functions**:
  - **merge():** This function is used to merge the 2 halves of the array. It assumes that both parts of the array are sorted and merges both of them.
  - **mergeSort():** This function divides the array into 2 parts. ***low*** *to* ***mid*** and ***mid+1 to high*** where,
    - low = leftmost index of the array
    - high = rightmost index of the array
    - mid = Middle index of the array

- Recursively split the array, and go from top-down until all sub-arrays size becomes 1.

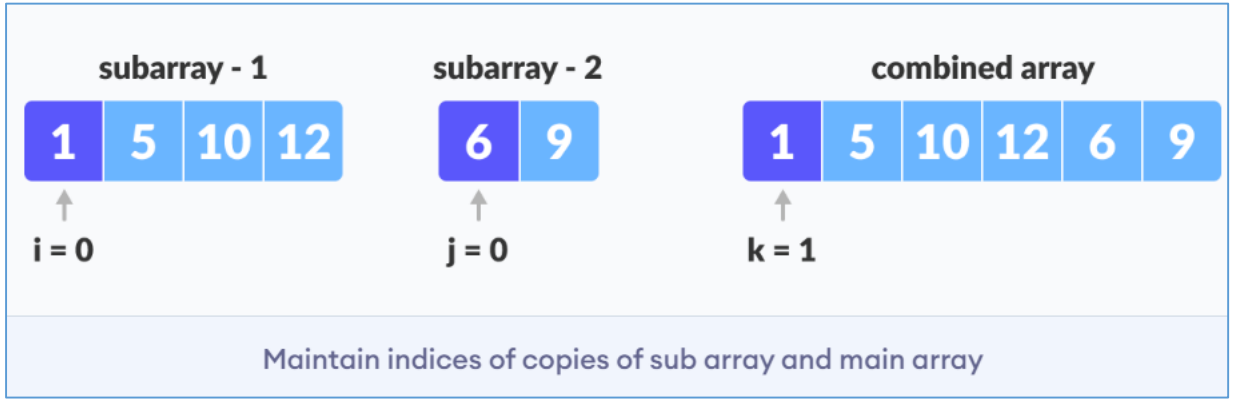
3. Divide and Conquer Applications …(cont'd)



Step 1: Create duplicate copies of sub-arrays to be sorted



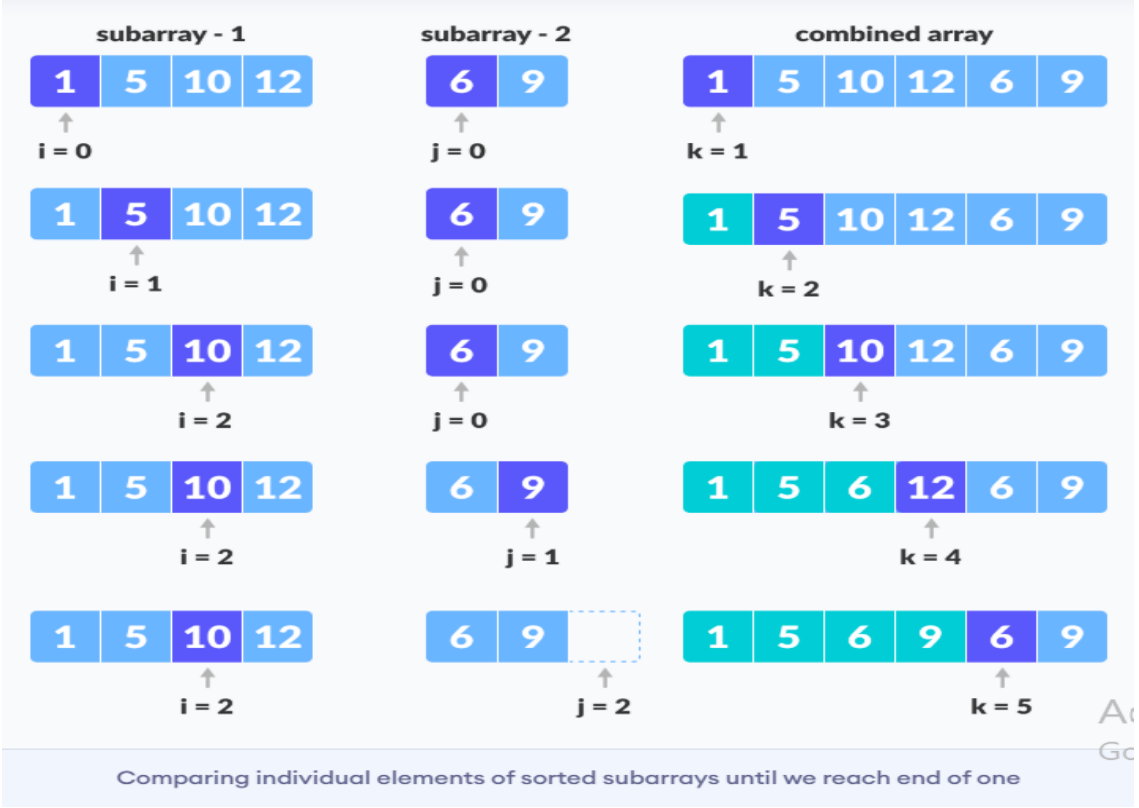
Step 2: Maintain current index of sub-arrays and main array



[3]. Programiz, merge-sort. https://www.programiz.com/dsa/merge-sort

3. Divide and Conquer Applications …(cont'd)

- **Step 3:** Until we reach the end of either L or M, pick larger among elements L and M and place them in the correct position at A[p..r]

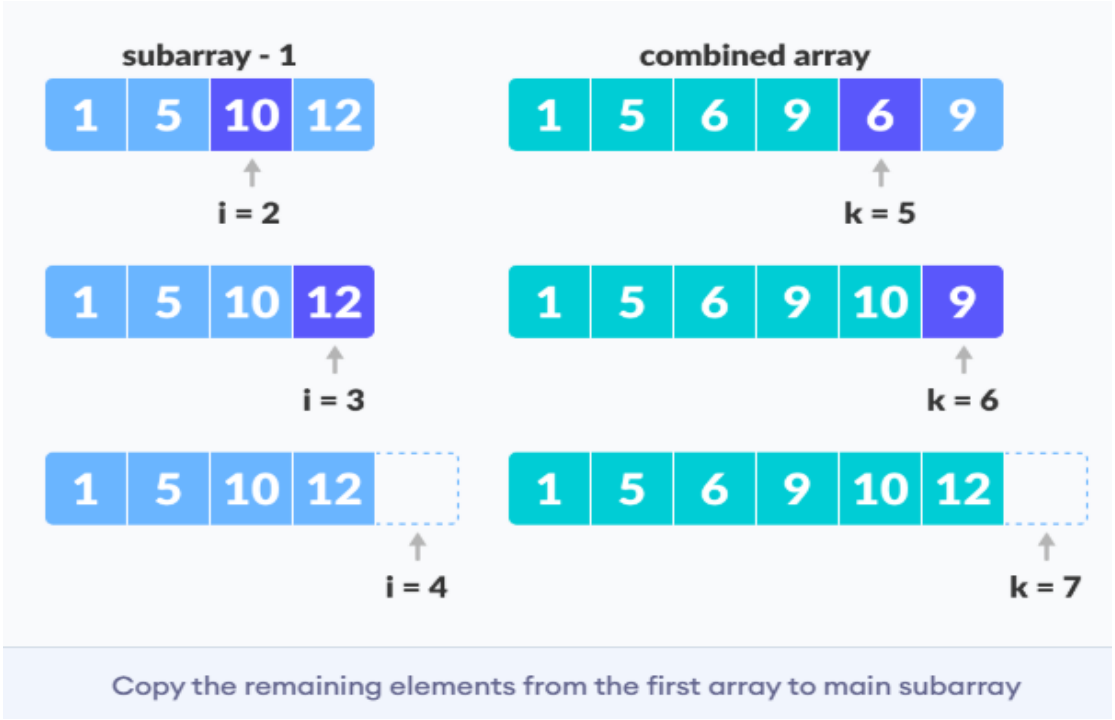


[3]. Programiz, merge-sort. https://www.programiz.com/dsa/merge-sort

3. Divide and Conquer Applications …(cont'd)

- **Step 4:** When we run out of elements in either L or M, pick up the remaining elements and put in A[p..r]

  - At the end of the merge function, the subarray A[p..r] is sorted.



[3]. Programiz, merge-sort. https://www.programiz.com/dsa/merge-sort

3. Divide and Conquer Applications …(cont'd)

Merge Sort Complexity

Time Complexity

- Best----- O(n*log n)
- Worst----O(n*log n)
- Average--O(n*log n)

Time complexity: O(nlogn)

- **Reason:** At each step, we divide the whole array, for that **logn** and we assume **n steps** are taken to get sorted array, so overall time complexity will be **nlogn**.

Merge Sort Applications

- E-commerce applications

[3]. Programiz, merge-sort. https://www.programiz.com/dsa/merge-sort

3. Divide and Conquer Applications …(cont'd)

c) Quicksort Algorithm

It is based on the **divide and conquer approach** where

1. An array is divided into subarrays by selecting a **pivot element** (element selected from the array).

- While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.

2. The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.

How Quicksort Algorithm works?

1. Select the Pivot Element

- There are different variations of quicksort where the pivot element is selected from different positions.
- Here, we will be selecting the rightmost element of the array as the pivot element.

[3]. Programiz,, QuickSort https://www.programiz.com/dsa/quick-sort

3. Divide and Conquer Applications …(cont'd)

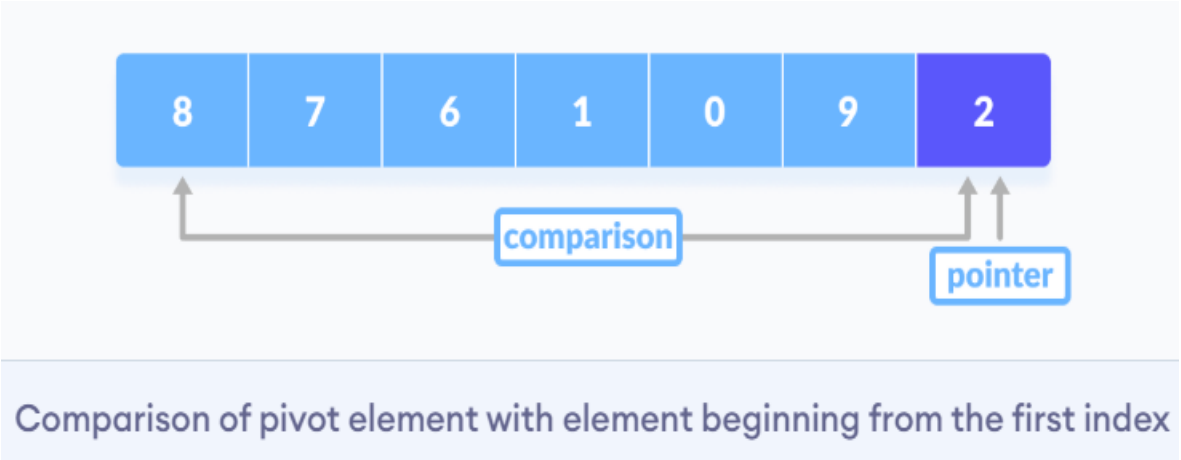
2. Rearrange the Array

- Now the elements of the array are rearranged so that elements that are smaller than the pivot are put on the left and the elements greater than the pivot are put on the right.

[3]. Programiz,, QuickSort https://www.programiz.com/dsa/quick-sort

3. Divide and Conquer Applications …(cont'd)

- A pointer is fixed at the pivot element.
- The pivot element is compared with the elements beginning from the first index.

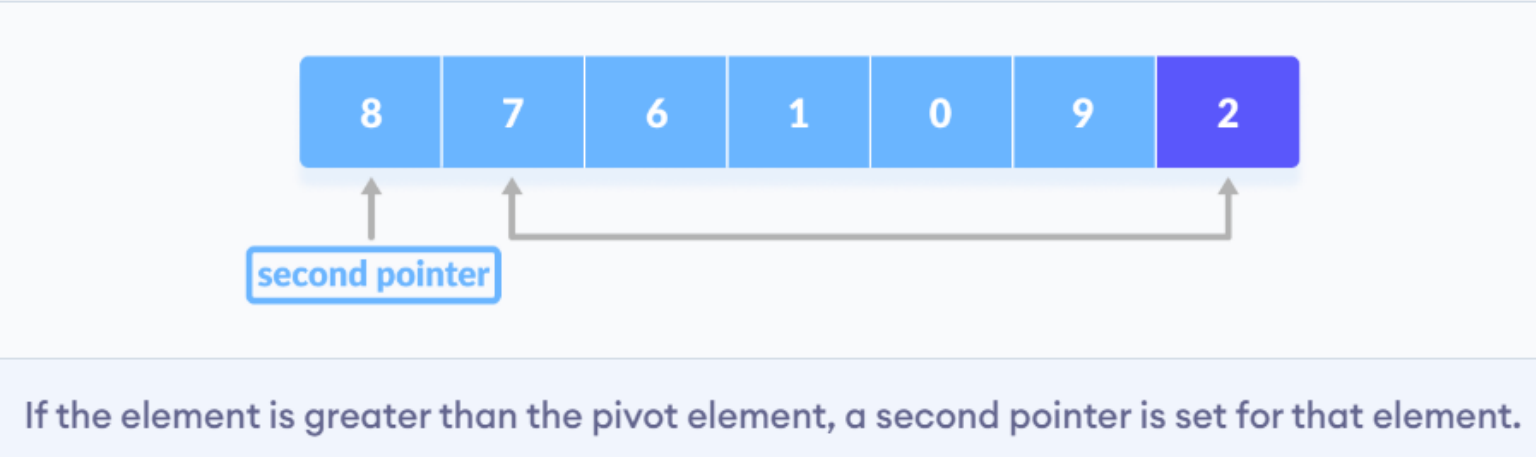


Comparison of pivot element with element beginning from the first index

[3]. Programiz,, QuickSort https://www.programiz.com/dsa/quick-sort

3. Divide and Conquer Applications …(cont'd)

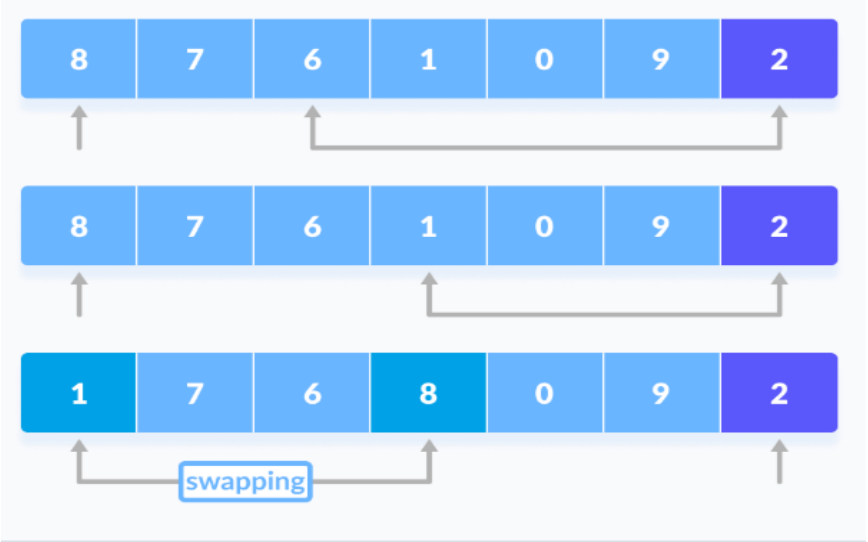
- If the element is greater than the pivot element, a second pointer is set for that element.



[3]. Programiz,, QuickSort https://www.programiz.com/dsa/quick-sort

3. Divide and Conquer Applications …(cont'd)

- Now, pivot is compared with other elements.
- If an element smaller than the pivot element is reached, the smaller element is swapped with the greater element found earlier.



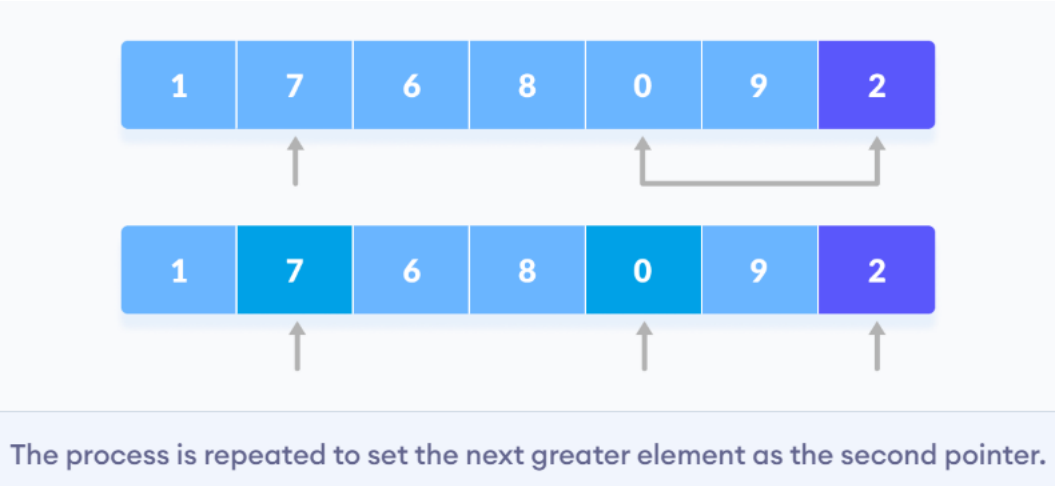
Pivot is compared with other elements.

[3]. Programiz,, QuickSort https://www.programiz.com/dsa/quick-sort

3. Divide and Conquer Applications …(cont'd)

Step-4:

- Again, the process is repeated to set the next greater element as the second pointer.
- And, swap it with another smaller element.

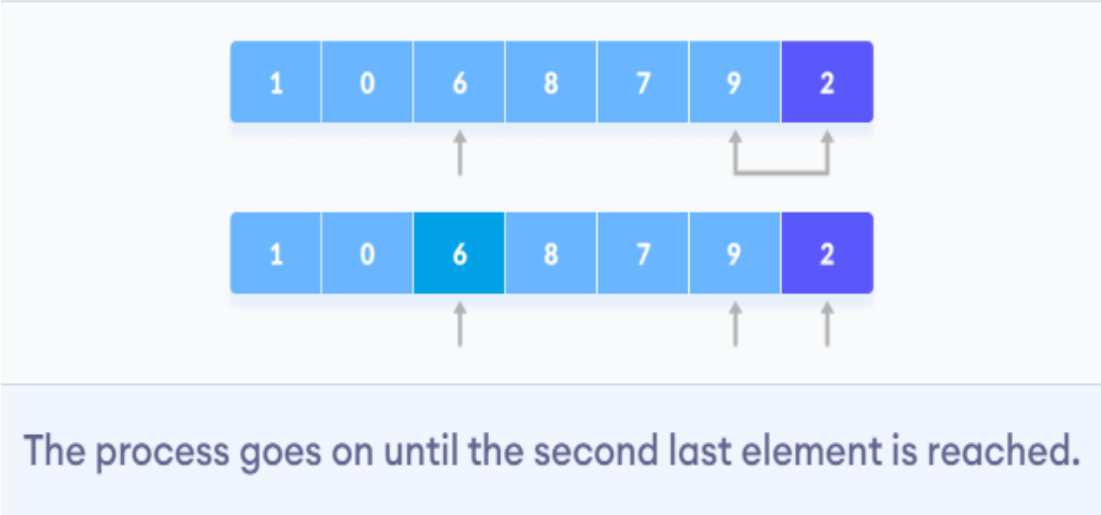


The process is repeated to set the next greater element as the second pointer.

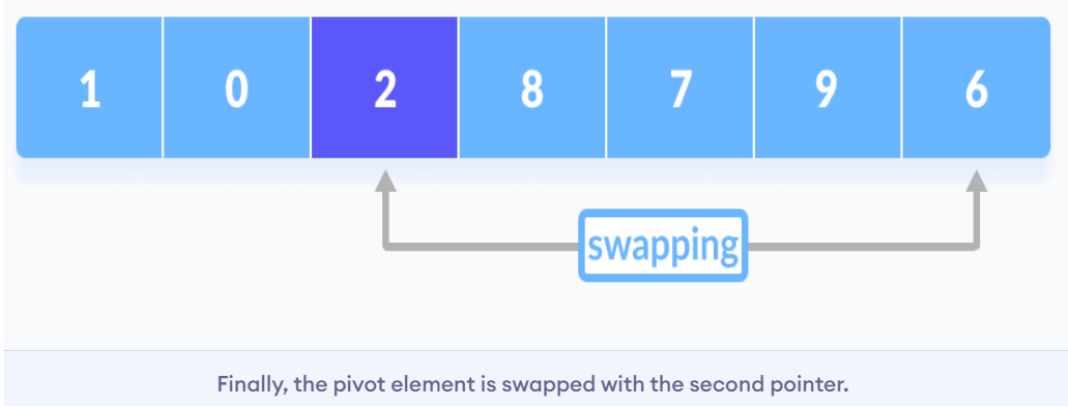
[3]. Programiz,, QuickSort https://www.programiz.com/dsa/quick-sort

3. Divide and Conquer Applications …(cont'd)

Step-5: The process goes on until the second last element is reached



Step-6: Finally, the pivot element is swapped with the second pointer.



3. Divide and Conquer Applications …(cont'd)

3. Divide Subarrays

- Pivot elements are again chosen for the left and the right sub-parts separately. And, **step 2** is repeated.

3. Divide and Conquer Applications …(cont'd)

Time Complexities of Quicksort

Worst Case Complexity [Big-O]: O(n2)

- It occurs when the pivot element picked is either the greatest or the smallest element.
- This condition leads to the case in which the pivot element lies in an extreme end of the sorted array.
- However, the quicksort algorithm has better performance for scattered pivots.

[3]. Programiz,, QuickSort https://www.programiz.com/dsa/quick-sort

3. Divide and Conquer Applications …(cont'd)

…..Time Complexities of Quicksort

Best Case Complexity [Big-omega]: O(n*log n)

- It occurs when the pivot element is always the middle element or near to the middle element.

Average Case Complexity [Big-theta]: O(n*log n)

- It occurs when the above conditions do not occur

Quicksort algorithm is used when

- the programming language is good for recursion
- time complexity matters
- space complexity matters

[3]. Programiz, QuickSort https://www.programiz.com/dsa/quick-sort

Summary

- Divide and conquer algorithm is a strategy of solving a large problem by breaking the problem into smaller sub-problems, solving the sub-problems, and combining them to get the desired output.

- Divide and Conquer Algorithms involves the following steps :- Divide: Divide the given problem into sub-problems using recursion, Conquer: Solve the smaller sub-problems recursively. If the subproblem is small enough, then solve it directly, and Combine Combine the solutions of the sub-problems that are part of the recursive process to solve the actual problem.
- Recursion is a powerful technique that can be used to solve a wide variety of problems. However, it is important to use recursion carefully, as it can lead to stack overflows if not used properly.
- Binary Search is searching algorithm for finding an element's position in a sorted array. In this approach, the element is always searched in the middle of a portion of an array.
- Merge Sort is a divide and conquers algorithm, it divides the given array into equal parts and then merges the 2 sorted parts, merge() and mergeSort().
- Quicksort Algorithm is based on the divide and conquer approach where an array is divided into subarrays by selecting a pivot element (element selected from the array)

References

1. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, MIT Press, 2009.

2. Geeks for Geeks, Recursive Algorithms, Kartik, 2024. https://www.geeksforgeeks.org/recursion-algorithms/

3. Programiz,Binary Search (With Code). https://www.programiz.com/dsa/binary-search

Thank You!

For your attention