Divide and Conquer Algorithm

Mayank Pratap Singh

24 July 2024

1 Introduction to Divide and Conquer

Divide and Conquer is a fundamental algorithm design paradigm that works by recursively breaking down a problem into smaller sub-problems, solving each sub-problem individually, and then combining their solutions to solve the original problem. This approach is used in various algorithms and has applications in numerous fields, including computer science, mathematics, and engineering.

2 Steps of Divide and Conquer

The Divide and Conquer approach consists of three main steps:

1. **Divide:** Split the original problem into smaller sub-problems.
2. **Conquer:** Solve each sub-problem recursively.
3. **Combine:** Combine the solutions of the sub-problems to form the solution to the original problem.

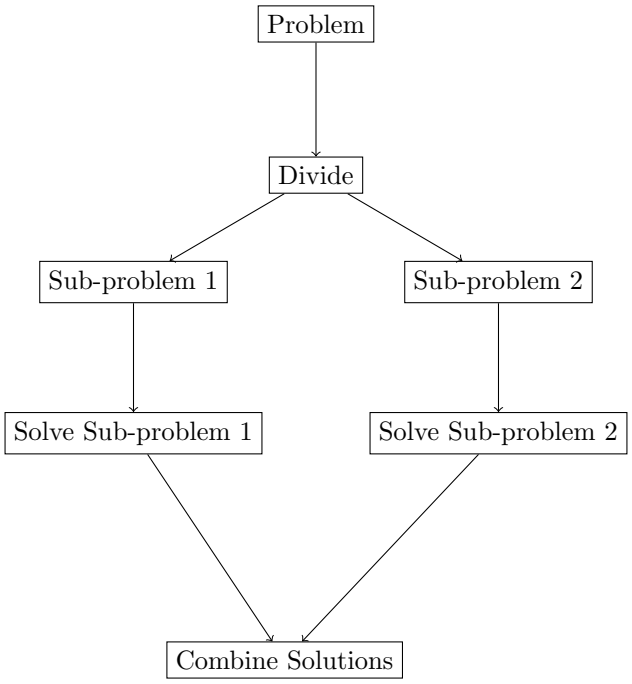


Figure 1: Divide and Conquer Process

3 Pseudocode for Divide and Conquer

Below is the pseudocode for a generic Divide and Conquer algorithm:

```
function DivideAndConquer(a, i, j):
    if (isSmall(a, i, j)):
        return solution(a, i, j)
    else:
        m = divide(a, i, j)
        b = DivideAndConquer(a, i, m)
        c = DivideAndConquer(a, m+1, j)
        return combine(b, c)
```

4 Example: Binary Search

Binary Search is a classic example of a Divide and Conquer algorithm. It works by repeatedly dividing a sorted array in half until the target value is found.

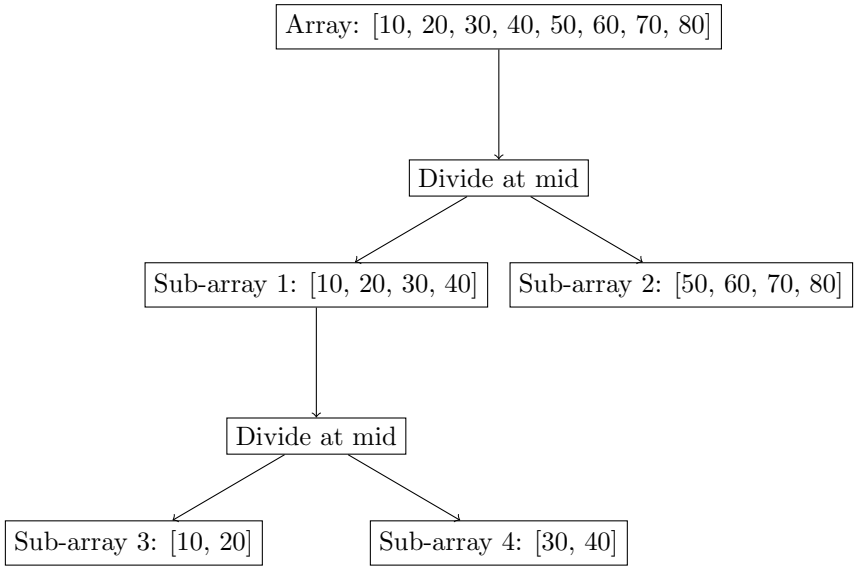


Figure 2: Binary Search Process

```
function BinarySearch(a, i, j, target):
    while (i <= j):
        mid = i + (j - i) / 2
        if (a[mid] == target):
            return mid
        elif (a[mid] < target):
```

```
        return BinarySearch(a, mid+1, j, target)
    else:
        return BinarySearch(a, i, mid-1, target)
```

5 Real-time Example: Big Data and MapReduce

Big Data refers to large and complex data sets that require advanced methods for storage, processing, and analysis. MapReduce is a programming model used for processing large data sets with a distributed algorithm on a cluster.

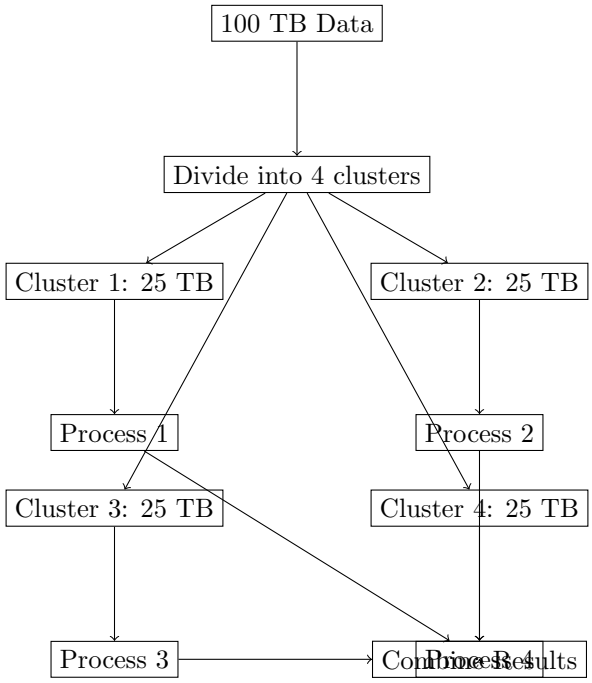


Figure 3: Big Data Processing using MapReduce

6 Conclusion

Divide and Conquer is a powerful algorithm design paradigm that can solve complex problems efficiently by breaking them down into smaller sub-problems. It is used in various algorithms like Binary Search, Merge Sort, and in real-world applications like Big Data processing with MapReduce.

7 Introduction to Merge Sort

Merge Sort is a clever way to sort a bunch of numbers (or any items) in order. It uses a special strategy called "Divide and Conquer" to make the job easier. Let's break it down step by step.

7.1 Steps of Merge Sort

Imagine you have a big pile of Lego blocks of different sizes, and you want to arrange them from smallest to largest. Here's how Merge Sort helps you do that:

1. **Divide:** First, split your big pile into two smaller piles. Keep splitting each pile into two more piles until each pile has only one Lego block.
2. **Conquer:** Next, start looking at pairs of Lego blocks. Compare them and arrange them in the right order. Keep doing this with all the piles.
3. **Combine:** Finally, combine the sorted piles back together. Each time you combine two piles, make sure the blocks stay in the right order.

8 Pseudocode for Merge Sort

Here's the pseudocode for Merge Sort. Think of it as a recipe that tells you how to sort the Lego blocks:

```
function MergeSort(arr, i, j):
    if (i < j):
        mid = i + (j - i) / 2
        MergeSort(arr, i, mid)
        MergeSort(arr, mid + 1, j)
        Merge(arr, i, mid, j)
```

9 Merge Procedure

The merge procedure is like combining two sorted piles of Lego blocks into one big sorted pile. Here's how it works:

```
function Merge(arr, i, mid, j):
    n1 = mid - i + 1
    n2 = j - mid

    left = new array of size n1
    right = new array of size n2
```

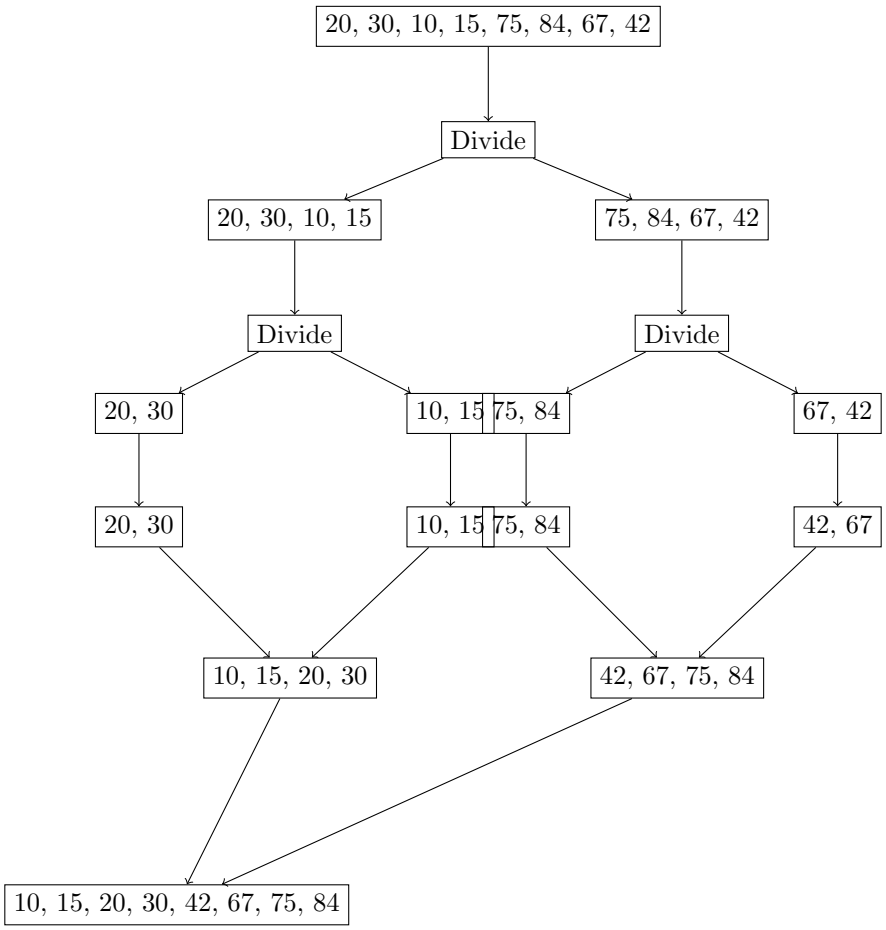


Figure 4: Merge Sort Process

```
for k from 0 to n1-1:
    left[k] = arr[i + k]
for k from 0 to n2-1:
    right[k] = arr[mid + 1 + k]

k = i
l = 0
r = 0

while l < n1 and r < n2:
    if left[l] <= right[r]:
        arr[k] = left[l]
        l += 1
```

```
        else:
            arr[k] = right[r]
            r += 1
        k += 1

    while l < n1:
        arr[k] = left[l]
        l += 1
        k += 1

    while r < n2:
        arr[k] = right[r]
        r += 1
        k += 1
```

10 Example: Sorting a List of Numbers

Let's sort the list: [20, 30, 10, 15, 75, 84, 67, 42]

1. Start with the list: [20, 30, 10, 15, 75, 84, 67, 42]
2. Divide it into two parts: [20, 30, 10, 15] and [75, 84, 67, 42]
3. Keep dividing: [20, 30], [10, 15], [75, 84], [67, 42]
4. Divide until single elements: [20], [30], [10], [15], [75], [84], [67], [42]
5. Start merging sorted pairs: [20, 30], [10, 15], [75, 84], [42, 67]
6. Merge sorted pairs into bigger sorted lists: [10, 15, 20, 30], [42, 67, 75, 84]
7. Finally, merge the sorted lists: [10, 15, 20, 30, 42, 67, 75, 84]

11 Conclusion

Merge Sort is like a magic trick that helps you sort things quickly and efficiently. By dividing the big problem into smaller problems, solving each one, and then putting everything back together, you can sort even the biggest pile of Lego blocks (or numbers) with ease.

12 Merge Sort Algorithm

Merge Sort is an efficient, stable, comparison-based, divide and conquer sorting algorithm. Most implementations produce a stable sort, meaning that the implementation preserves the input order of equal elements in the sorted output.

12.1 Steps of Merge Sort

1. **Divide:** Divide the array into two halves.
2. **Conquer:** Recursively sort both halves.
3. **Combine:** Merge the two sorted halves to produce the sorted array.

12.2 Pseudocode for Merge Sort

```
function MergeSort(arr, i, j):
    if (i < j):
        mid = i + (j - i) / 2
        MergeSort(arr, i, mid)
        MergeSort(arr, mid + 1, j)
        Merge(arr, i, mid, j)
```

12.3 Merge Procedure

The merge procedure combines two sorted sub-arrays into a single sorted array.

```
function Merge(arr, i, mid, j):
    n1 = mid - i + 1
    n2 = j - mid

    left = new array of size n1
    right = new array of size n2

    for k from 0 to n1-1:
        left[k] = arr[i + k]
    for k from 0 to n2-1:
        right[k] = arr[mid + 1 + k]

    k = i
    l = 0
    r = 0

    while l < n1 and r < n2:
        if left[l] <= right[r]:
            arr[k] = left[l]
            l += 1
        else:
            arr[k] = right[r]
            r += 1
        k += 1

    while l < n1:
```

```
        arr[k] = left[l]
        l += 1
        k += 1

    while r < n2:
        arr[k] = right[r]
        r += 1
        k += 1
```

12.4 Time Complexity Analysis

$$\begin{aligned} T(n) &= 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

12.5 Space Complexity Analysis

Merge Sort requires extra space for the temporary arrays used during merging.

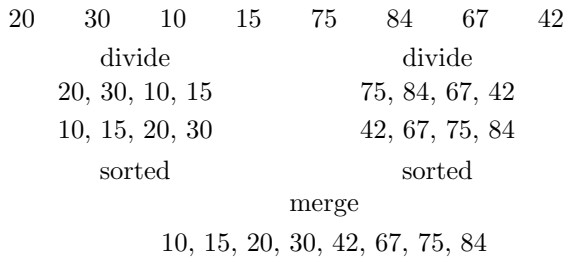
$$\text{Space Complexity} = O(n)$$

13 Detailed Example with Merge Sort

Consider the array: [20, 30, 10, 15, 75, 84, 67, 42]

- Divide the array into two halves: [20, 30, 10, 15] and [75, 84, 67, 42]
- Recursively divide the halves: [20, 30], [10, 15], [75, 84], [67, 42]
- Further divide until single elements are obtained: [20], [30], [10], [15], [75], [84], [67], [42]
- Start merging sorted arrays:
  - Merge [20] and [30] into [20, 30]
  - Merge [10] and [15] into [10, 15]
  - Merge [75] and [84] into [75, 84]
  - Merge [67] and [42] into [42, 67]
- Continue merging:
  - Merge [20, 30] and [10, 15] into [10, 15, 20, 30]
  - Merge [75, 84] and [42, 67] into [42, 67, 75, 84]
- Final merge:
  - Merge [10, 15, 20, 30] and [42, 67, 75, 84] into [10, 15, 20, 30, 42, 67, 75, 84]

13.1 Visualization



14 Conclusion

The Divide and Conquer algorithm is a powerful strategy that can be used to solve complex problems by breaking them down into simpler sub-problems. Merge Sort is a classic example of a Divide and Conquer algorithm, demonstrating how dividing a problem into smaller parts and then combining the solutions can lead to an efficient overall solution.