Divide and Conquer

Divide and Conquer is an algorithmic pattern.

A divide and conquer algorithm is a strategy of solving a large problem by,

1. Breaking the larger problem into smaller sub-problems.
2. Solving the sub-problems
3. Combining them the sub-problem to get the desired output.

Note:

To use the divide and conquer algorithm, recursion is used.

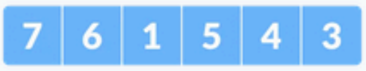
How Divide and Conquer Algorithms Work ?

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 recursion process to solve the actual problem.

Example:

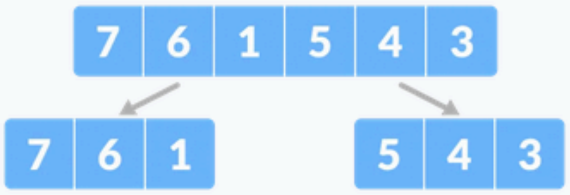
Here, we will sort an array using the divide and conquer approach (ie. merge sort).

1. Let the given array be:



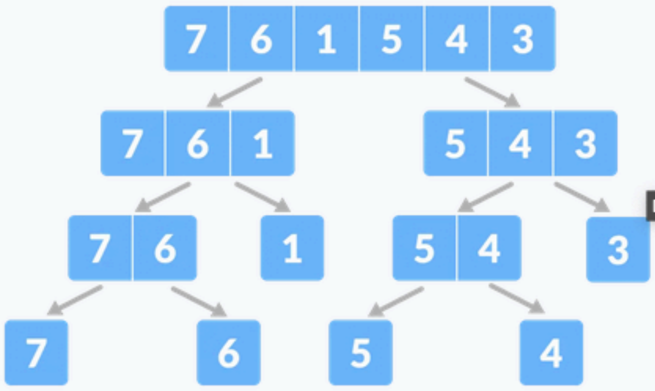
Array for merge sort

2. **Divide** the array into two halves.



Divide the array into two subparts

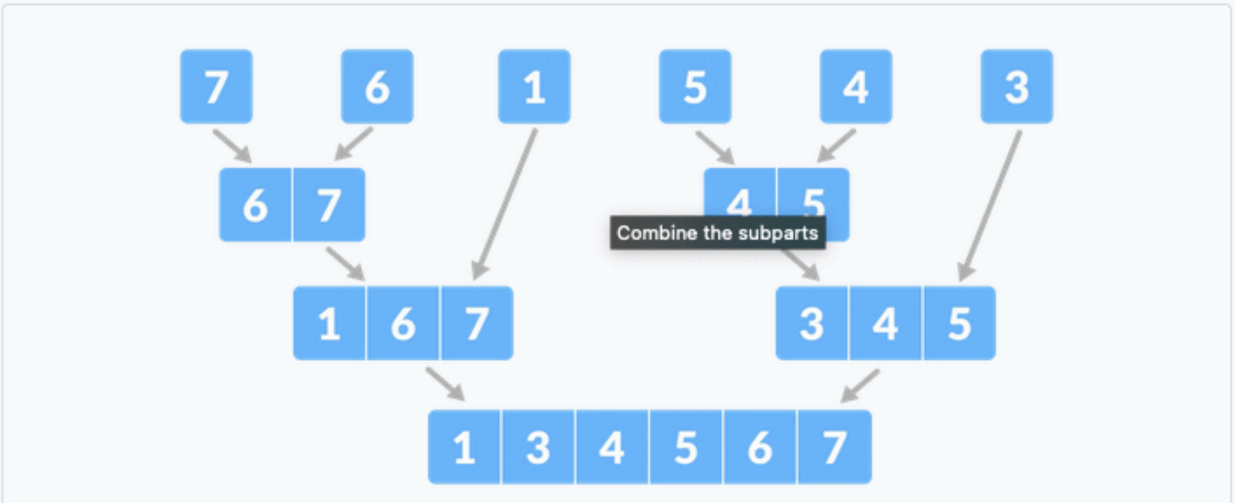
Again, divide each subpart recursively into two halves until you get individual elements.



Divide the array into smaller subparts, merge sort

Divide the array into smaller subparts

3. Now, combine the individual elements in a sorted manner. Here, **conquer** and **combine** steps go side by side.



Combine the subparts

```
T(n) = aT(n/b) + f(n),
Where,
n = Size of input
a = Number of subproblems in the recursion
n/b = Size of each subproblem. All subproblems
      are asumed to have the same size.
f(n) = cost of the work done outside the recursive call,
       which includes the cost of dividing the problem
       and cost of merging the solutions.
```

Time Complexity

The complexity of the divide and conquer algorithm is calculated using the Master Theorem.

Example to find the time complexity of a recursive problem.

For a merge sort, the equation can be written as:

```
T(n) = aT(n/b) + f(n)
     = 2T(n/2) + O(n)
Where,
a = 2 (each time, a problem is divided into 2 subproblems)
n/b = n/2 (size of each sub problem is half of the input)
f(n) = time taken to divide the problem and merging
       the subproblems
T(n/2) = O(n log n) (To understand this,
          please refer to the master theorem.)

Now, T(n) = 2T(n log n) + O(n)
          ≈ O(n log n)
```

Advantage of Divide and Conquer Algorithm :

* This approach is suitable for multiprocessing systems.
* It takes efficient use of memory caches.
* It is more proficient than that of its counterpart Brute Force technique.
* It efficiently uses cache memory without occupying much space because it solves simple subproblems within the cache memory instead of accessing the slower main memory.

Disadvantage of Divide and Conquer Algorithm

* Since most of its algorithms are designed by incorporating recursion, so it necessitates high memory management.
* An explicit stack may overuse the space.
* It may even crash the system if the recursion is performed rigorously greater than the stack present in the CPU.

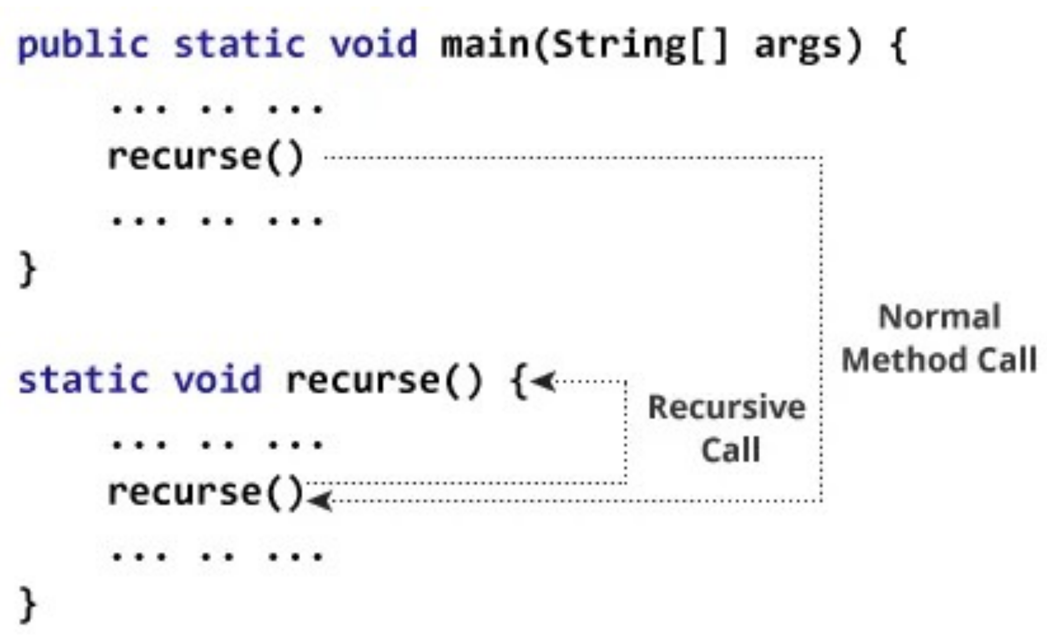
Application of Divide and Conquer Algorithm :

1. Binary Search
2. Quick Sort
3. Merge Sort
4. Strassen's Matrix Multiplication
5. Karatsuba Algorithm

Recursion: A method that calls itself is known as recursive method. And, this process is known as recursion.

☆ This technique provides a way to break complicated problems down into simple problems which are easier to solve.

Example:
Two parallel mirrors facing each other. Any object in between them would be reflect recursively.


```java
public static void main(String[] args) {
    ... .. ...
    recurse()
    ... .. ...
}

static void recurse() {
    ... .. ...
    recurse()
    ... .. ...
}
```

Working of Java Recursion

☆ In the above example, we have called the `recurse()` method from inside the `main` method. (normal method call). And, inside the recurse() method, we are again calling the same recurse method. This is a recursive call.

☆ In order to stop the recursive call, we need to provide some conditions inside the method. Otherwise, the method will be called infinitely.

Hence, we use the if...else statement (or similar approach) to terminate the recursive call inside the method.

Program:

```java
package com.dhruvlenka;
//Factorial of a Number Using Recursion
public class Factorial {
    static int factorial(int num){
        // termination condition
      if(num == 0){
          return 1;
      } else {
          // recursive call
          return num * factorial(num-1);
      }
    }
    public static void main(String[] args) {
     int number, result; // var declaration
     number = 20; //assigning value
     result = factorial(number);
     System.out.println("Factorial of " + number + " = " + result);
    }
}
```