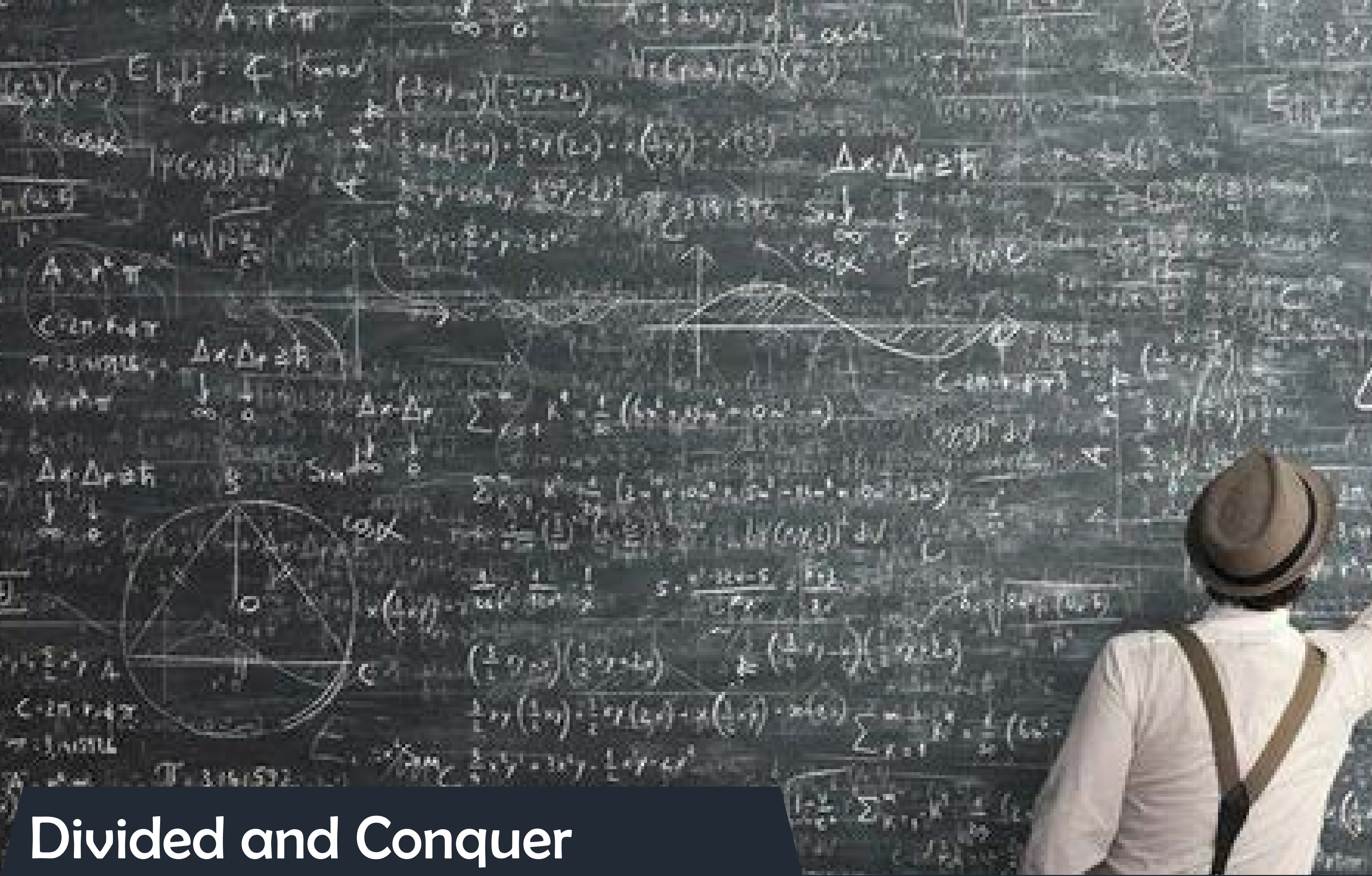


Divided and Conquer

Ja-Hee Kim

Agenda

Ref: https://www.geeksforgeeks.org/divide-and-conquer/

Introduction

Techniques

Examples

Introduction

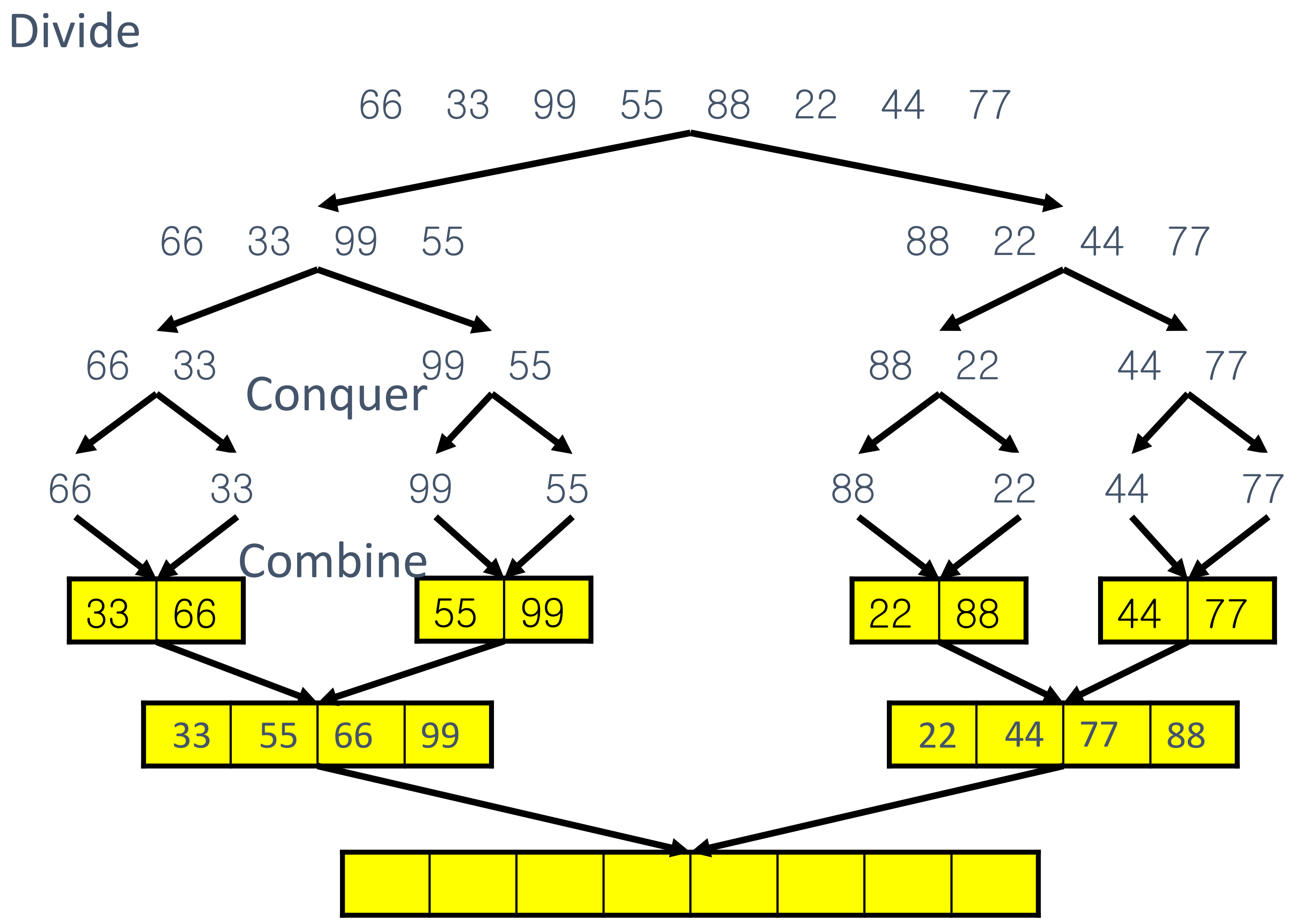
Divide and conquer

- Divide and Conquer is an algorithmic paradigm. A typical Divide and Conquer algorithm solves a problem using following three steps.
  - **Divide**: Break the given problem into sub-problems of same type.
  - **Conquer**: Recursively solve these sub-problems
  - **Combine**: Appropriately combine the answers

Recursive method

- Some recursive methods use the divide and conquer paradigm
- because it solves a problem by reducing it to smaller sub-problems, hoping that their solutions can be used to solve the larger problem.
- Non-recursive method is usually faster.(for example dynamic programming)

Example: Merge sort



Techniques

Mathematical induction

- to prove a given statement about any well-ordered set.
- The proof consists of two steps:
  - The base case: prove that the statement holds for the first natural number n. Usually, n = 0 or n = 1, rarely, n = –1
  - The inductive step: prove that, if the statement holds for some natural number n, then the statement holds for n + 1.
- $n! = 1 \times 2 \times 3 \times \cdots \times n$
- $n! = \begin{cases} 1, & \text{if } n = 0, 1 \\ n(n-1)! & \text{if } n > 1 \end{cases}$

Base case

Induction case

Avoiding recursion

```java
if(n<2) return (long)1;
return recursive(n-1)+recursive(n-2);
```

- Tail recursion

```java
public long tailRecursion(int n, long preFibo, long prePreFibo) {
    long currentFibo;
    if (n < 2) return n*preFibo;
    return tailRecursion(n-1, preFibo+prePreFibo, preFibo);
}
```

- Iteration

```java
public long iteration(int n) {
    long currentFibo=1;
    long preFibo=1,prePreFibo=1;
    for(int i=n; i > 1 ; i--) {
        currentFibo = preFibo+prePreFibo;
        prePreFibo = preFibo;
        preFibo = currentFibo;
    }
    return currentFibo;
}
```

- Using a stack
- Memorize the result

Avoiding recursion

```java
if(n<2) return (long)1;
return recursive(n-1)+recursive(n-2);
```

- Tail recursion
- Iteration
- Using a stack
- Memorize the result

```java
public long usingStack(int n) {
    ArrayDeque<Record> programStack = new ArrayDeque<>(100);
    programStack.push(new Record(n, 1, 1));
    long currentFibo = n;
    while(!programStack.isEmpty()) {
        Record topRecord = programStack.pop();
        currentFibo = topRecord.n;
        long preFibo = topRecord.pre;
        long prePreFibo = topRecord.prePre;
        if(currentFibo < 3)
            currentFibo =preFibo+prePreFibo;
        else
            programStack.push(new Record(currentFibo-1, preFibo+prePreFibo, preFibo));
    }
    return currentFibo;
}
private class Record{
    private long n;
    private long pre, prePre;
    public Record(long n, long pre, long prePre) {
        this.n = n;
        this.pre = pre;
        this.prePre = prePre;
    }
}
```

```java
private long[] fibonacci;
private int num=2;
private static final int MAX=1010;
public Fibonacci() {
    fibonacci = new long[MAX];
    fibonacci[0]=fibonacci[1]=1;
}
public long memorize(int n) {
    if(n<num) return fibonacci[n];
    else if(n==num) {
        fibonacci[n]=fibonacci[n-1]+fibonacci[n-2];
        num++;
        return fibonacci[n];
    }
    else return memorize(n-1)+memorize(n-2);
}
```

Examples

Search and sort

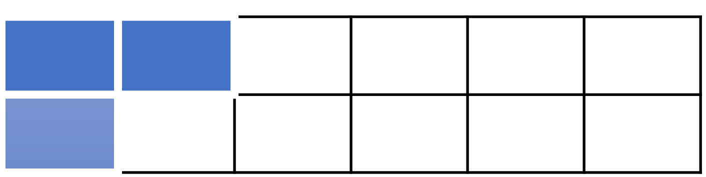
- Binary search uses a recursive method to search an array to find a specified value. The array must be a sorted array:

  a[0]≤a[1]≤a[2]≤. . . ≤ a[finalIndex]

  - If the value is found, its index is returned.
  - If the value is not found, -1 is returned.
- Sort
  - Merge sort
  - Quick sort

Tiling Problem

- Given a “2xn” board and tiles of size “2x1”, count the number of ways to tile the given board using the 2x1 tiles.
- A tile can either be placed horizontally i.e., as a 1x2 tile or vertically i.e., as 2x1 tile.
- Solution
  - Let count(n) be the count of ways to place tiles.
  - Count(n) = $\begin{cases} n & \text{if } n = 1 \text{ or } 2 \\ \text{count}(n-1)+\text{count}(n-2) & \text{otherwise} \end{cases}$

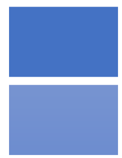


board

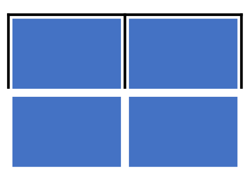
tile

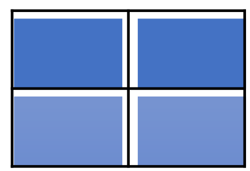
Tiling Problem

- n=1

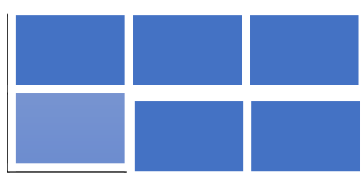


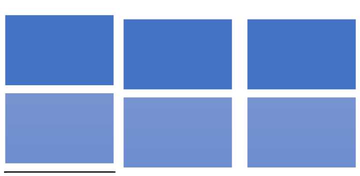
- n=2

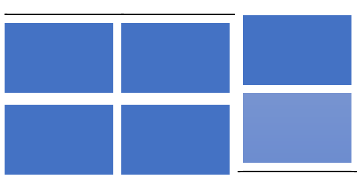




- n=3







Calculate pow(x, n)

- Given two integers x and n, write a function to compute x^n.
- Base case
  - n=0: 1
- Induction
  - n is even: pow(x, n/2) * pow(x, n/2)
  - n is odd: x * pow(x, n/2) * pow(x, n/2)

Karatsuba's algorithm

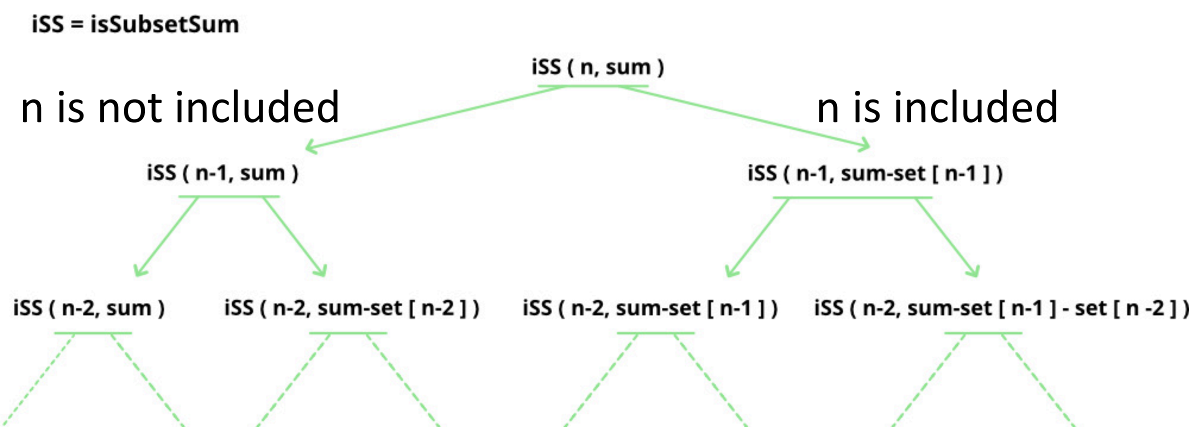
- developed by Russian mathematician Anatoly Karatsuba (early 1960's)
- Given two binary strings that represent value of two integers, find the pr oduct of two strings.
- Idea
  - $x_{0\cdots n} = x_n \times 10^n + x_{0\cdots n-1}, y_{0\cdots n} = y_n \times 10^n + y_{0\cdots n-1}$
  - Base case: $x_{0\cdots n}y_{0\cdots n} = x_0 y_0$ If $n = 0$
  - Induction case
    - $x_{0\cdots n}y_{0\cdots n} = x_n y_n \times 10^{2n} + (x_{0\cdots n-1}y_n + x_n y_{0\cdots n-1}) \times 10^n + x_{0\cdots n-1}y_{0\cdots n-1}$
- Time complexity
  - Native way: $O(n^2)$
  - Karatsuba's algorithm: $O(n^{1.59})$

	2	3	n: the number of digit
X	3	4	
	9	2	$O(n^2)$ for multiplication
6	9		
7	8	2	$O(n)$ for summation

23X34 = (2x3)X100+(3x3+2X4)X10 + 3x4
= 600+170+12
= 782

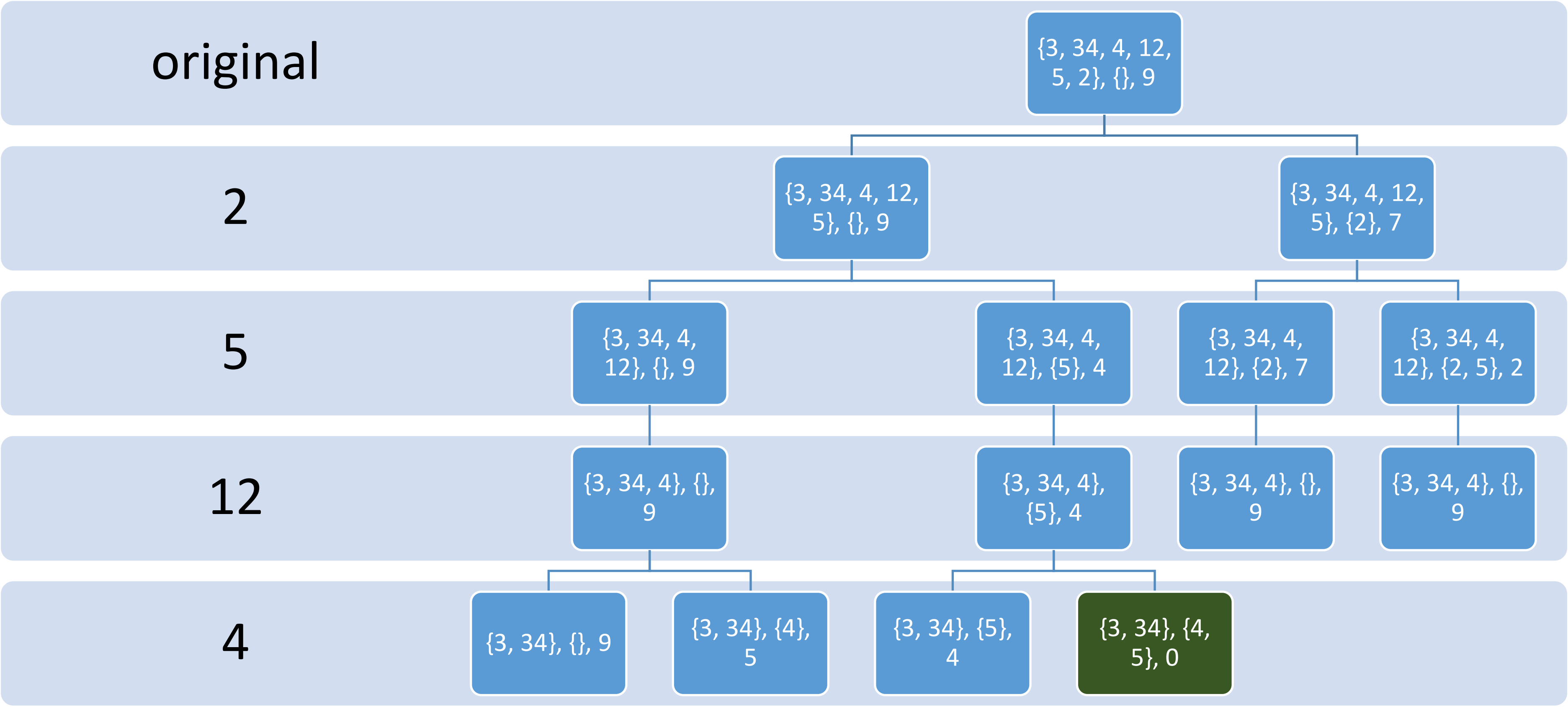
Subset sum

- Given a set of non-negative integers, and a value sum, determine if there is a subset of the given set with sum equal to given sum.
- Algorithm
  - isSubsetSum(set, n, sum) = isSubsetSum(set, n-1, sum) || isSubsetSum(set, n-1, sum-set[n])
  - Base Cases:
    - isSubsetSum(set, n, sum) = false, if sum > 0 and n == 0
    - isSubsetSum(set, n, sum) = true, if sum == 0



Example

- Input: set[] = {3, 34, 4, 12, 5, 2}, sum = 9
- Output: True //There is a subset (4, 5) with sum 9.



Thanks