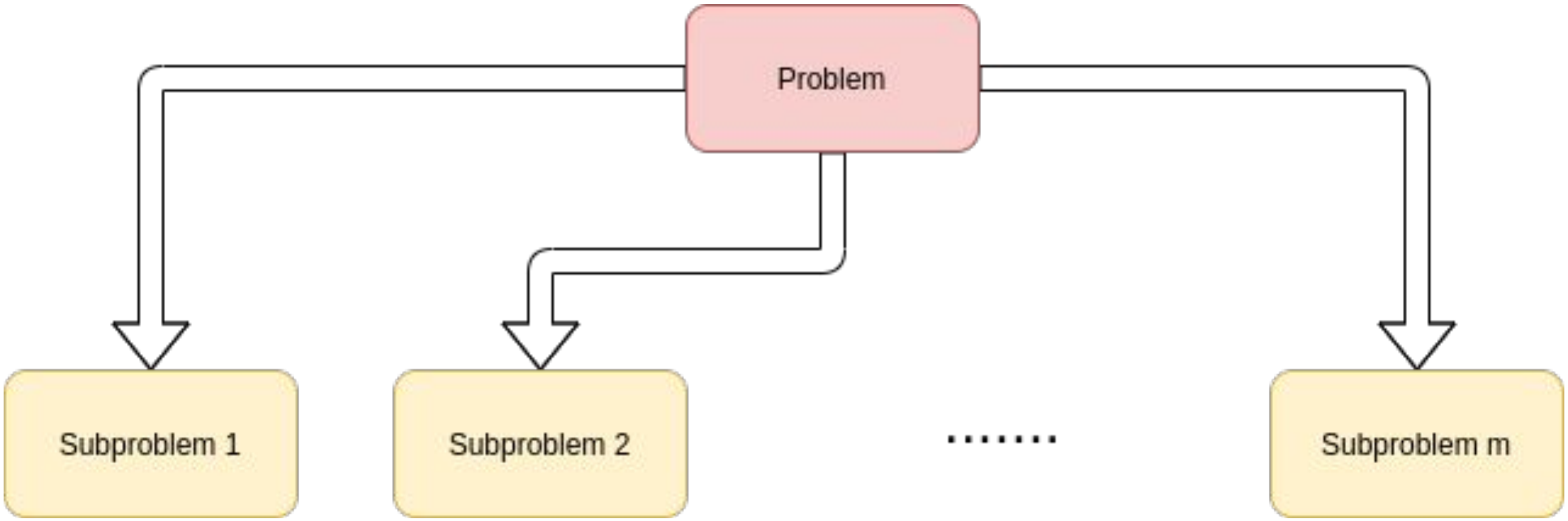
Introduction to Dynamic Programming

UNIVERSITY OF CALGARY

CPSC 413 - Algorithm Design

Benyamin Bashari

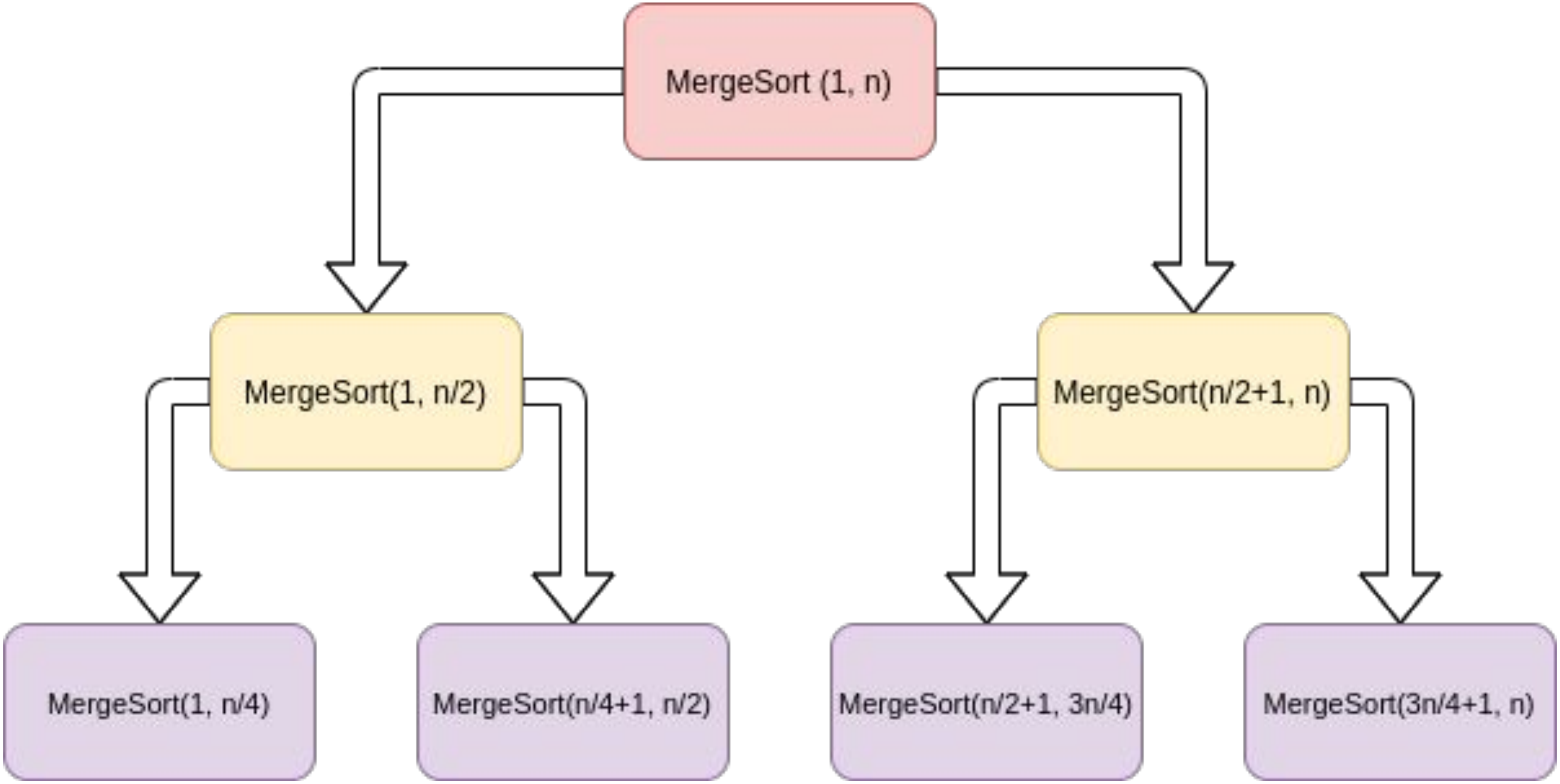
Dynamic Programming



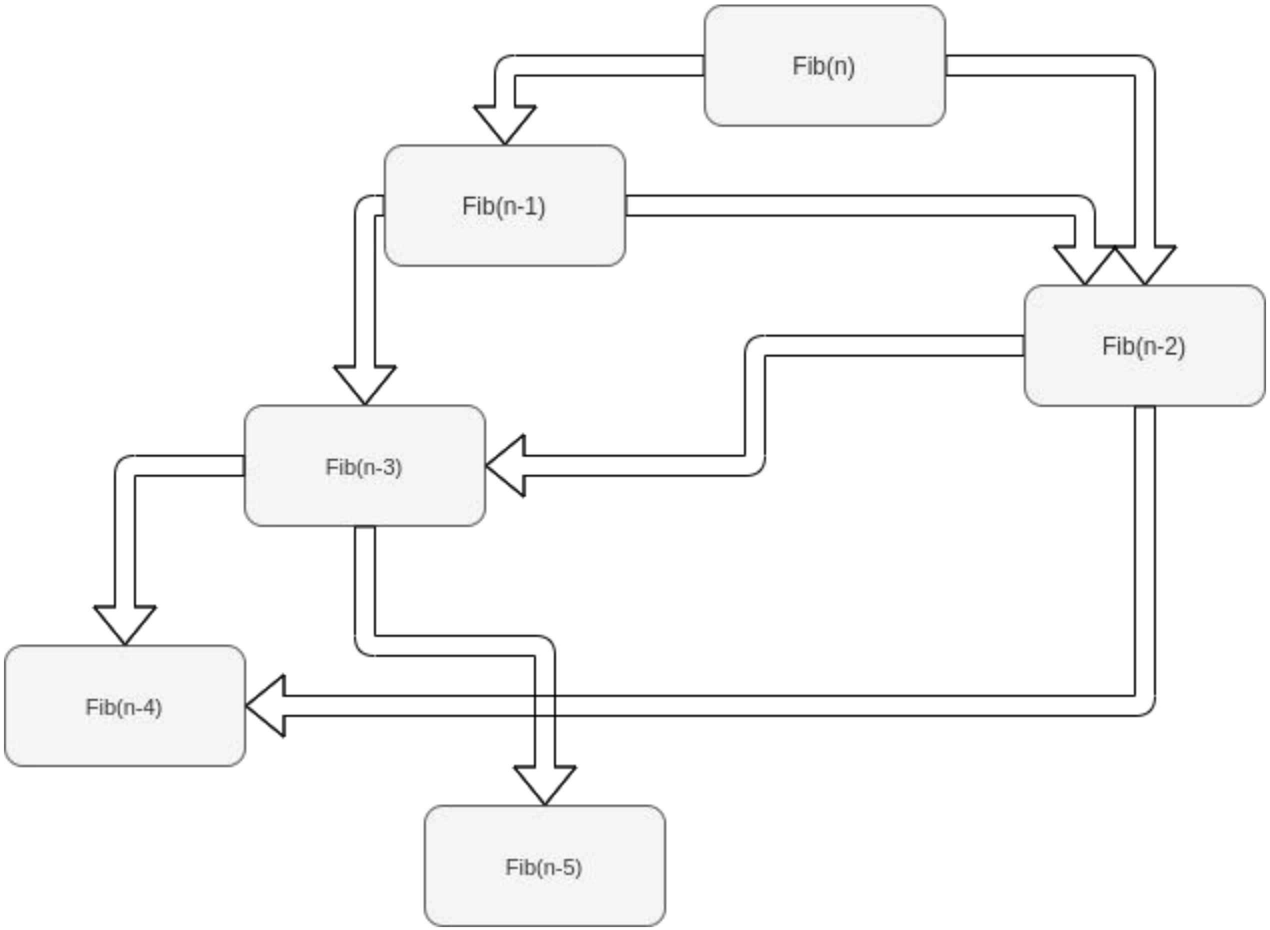
Dynamic Programming

- Solving all subproblems can result to solving the problem
- So what is the difference between Dynamic Programming and Divide and Conquer?
- Questions with efficient Divide and Conquer solution have disjoint subproblems, while questions with efficient Dynamic Programming solution might have similar subproblems.
- If we compute the subproblems in a way that each of them only computed once, then we have an efficient algorithm.

Divide and Conquer



Dynamic Programming



Fibonacci

- ## Bad Implementation

```
int fib(int n) {
   if(n == 0 or n == 1)
       return 1;
   return fib(n-1) + fib(n-2);
}
```

- ## Good Implementation

```
int fib(int n) {
   int arr[n];
   arr[0] = 1;
   arr[1] = 1;
   for(int i = 2 ; i <=  n; i++)
       arr[i] = arr[i-1] + arr[i-2];
   return arr[n];
}
```

Dynamic Programming

- Top Down DP
  - Easier to come up with (Divide and Conquer)
  - We only need to break the problem in different subproblems
  - In other words we only need a recurrence relation of the problem into its subproblems
- Bottom Up DP
  - Harder to come up with
  - We need to find a proper ordering of the subproblems such that, when solving a problem all of its subproblems were already solved.

Example

- We have 4 types of coins
  - 1 Cent
  - 10 Cents
  - 15 Cents
  - 25 Cents
- Remember that greedy algorithm does not return an optimal answer for T = 30
  - Optimal: Choose 2 coins of 15 cents -> Optimal = 2
  - Greedy: Choose 1 coin of 25 cents, and 5 coins of 1 cent -> Greedy = 6

Example

- What is the minimum number of coins required to give change for T cents.
- Consider the last coin that we choose to build T cents.

$$MinCoin(n) = \begin{cases} 1 & n = 1, 10, 15, 25 \\ min(MinCoin(n-1), MinCoin(n-10), & otherwise \\ MinCoin(n-15), MinCoin(n-25)) + 1 & \end{cases}$$

Example (Divide and Conquer Algorithm)

```
int MinCoin(int n) {
    if(n == 1 or n == 10 or n == 15 or n == 25)
        return 1;
    int a = INF, b = INF, c = INF, d = INF;
    if(n - 25 > 0)
        a = MinCoin(n-25);
    if(n - 15 > 0)
        b = MinCoin(n-15);
    if(n - 10 > 0)
        c = MinCoin(n-10);
    if(n - 1 > 0)
        d = MinCoin(n-1);
    return min(a, b, c, d) + 1;
}
```

Example (Runtime of Divide and Conquer)

$$T(n) = T(n-1) + T(n-10) + T(n-15) + T(n-25) + c$$

- Exponential in n

Example (Dynamic Programming Algorithm)

```
int MinCoin(int n) {
   int arr[n];
   arr[1] = arr[10] = arr[15] = arr[25] = 1;
   for(int i = 2 ; i <= n ; i++) {
       if(i == 10 or i == 15 or i == 25)
           continue;
       int a = INF, b = INF, c = INF, d = INF;
       if(n - 25 > 0)
           a = arr[n-25];
       if(n - 15 > 0)
           b = arr[n-15];
       if(n - 10 > 0)
           c = arr[n-10];
       if(n - 1 > 0)
           d = arr[n-1];
       arr[n] = min(a, b, c, d) + 1;
   }
}
```

Example (Runtime of Divide and Conquer)

- $T(n) = c + d.n$, which is $O(n)$
- Also, it is easy to change the algorithm to get which coins we use to give change for n cents.
  - For each element of variable "arr" we only need to know which of arr[n-1], arr[n-10], arr[n-15], arr[n-25] was minimum. For example if arr[n-15] was minimum among them then we used 15 cents coin at that time.