Data Structure and Algorithms Design

Vincent Chau （周）

2021.11.3

How The Course is Organized

- Introduction (4 class hours: Instructor)
    - Projects (can be written in Python, C, C++)
    - Necessary knowledges
    - Divided into two main parts: Algorithms Design & Data Structure
- Project Programming (32 class hours)
- Presentation (4 class hours)
    - About one of the assigned projects (randomly selected by instructor)

Evaluation

- Presentations: 30%
- Project Reports: 70%

- One report per project
- MS Word: .doc
- About the projects

Project Report

1. Problem description / demand analysis
   Carefully analyze the experimental requirements and analyze the needs of the experimental contents
2. System structure / algorithm idea
   Basic idea, system framework, and describe the functions and relationships of each module
3. Function module design
   Module design idea, flow chart and algorithm complexity analysis
4. Test results and analysis
   Test data selection or generation method, operation result screenshot, performance diagram
5. Experimental summary
   The problems encountered, the problem-solving process, and summarize the experimental experience
6. Source code
   All source program lists of the project shall be fully annotated

Summary

- Divide and Conquer
- Dynamic programming

Divide and Conquer

Divide and Conquer

Concept

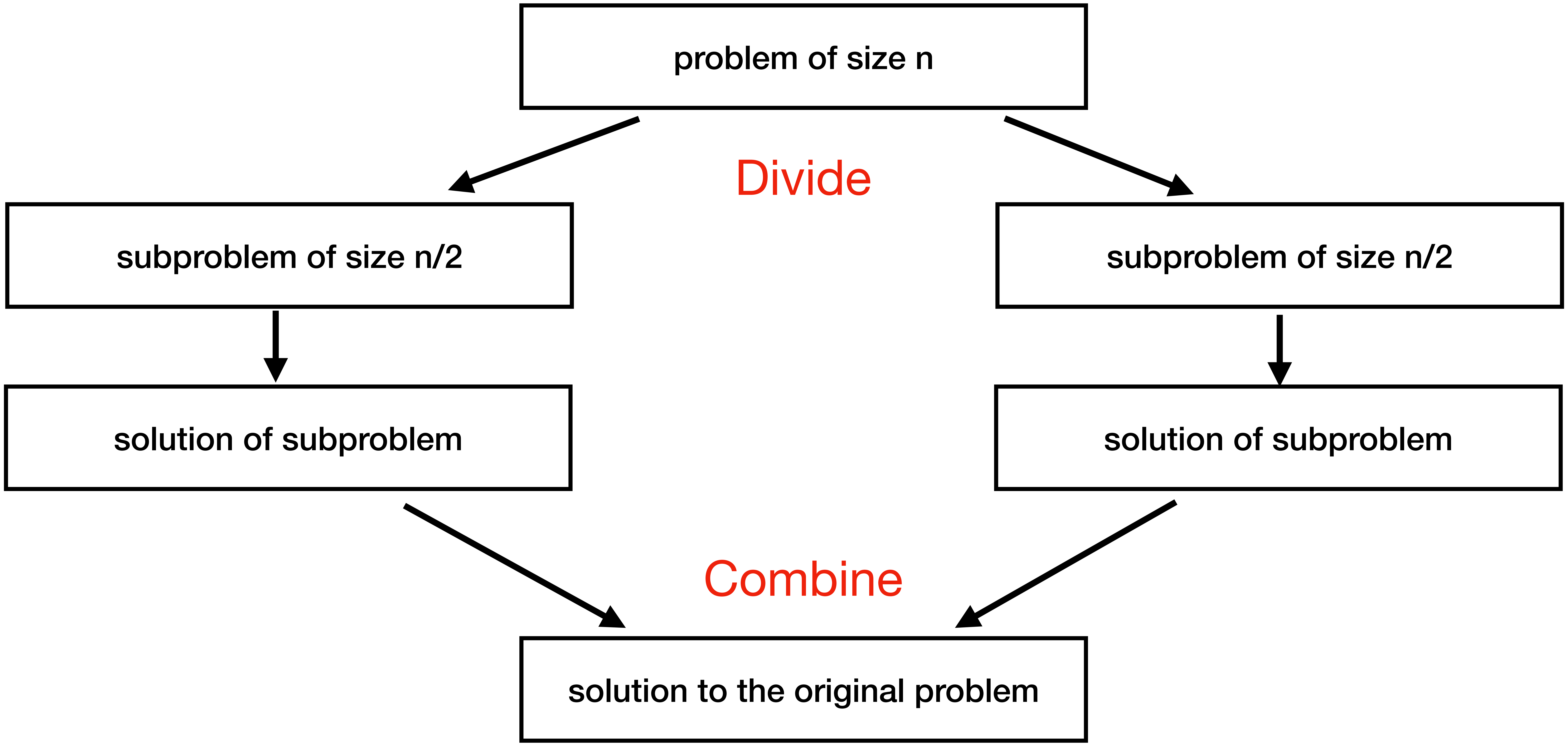
1. Divide a problem into two or smaller instances (ideally of about the same size)

2. Solve the small instances

3. Obtain a solution to the original instance by combining these solution to the smaller instances

Divide and Conquer

Concept



Recurrence equation

Time complexity

Suppose we divide the problem into k sub problems

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ T(n_1) + T(n_2) + \ldots + T(n_k) + f(n) & \text{otherwise} \end{cases}$$

where $g(n)$ is the time to compute answer directly

$T(n)$ is the time for divide-and-conquer method on any input of size n

$f(n)$ is the time for dividing the problem and combining the solutions to sub problems

Recurrence equation

Time complexity

Suppose we divide the problem into b sub problems

$$T(n) = \begin{cases} T(1) & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases}$$

a is the time needed to compute a sub problem

$f(n)$ is the time for dividing the problem and combining the solutions to sub problems

Recurrence equation

Master Theorem

$$T(n) = aT(n/b) + f(n)$$

If $f(n) \in \Theta(n^d)$ where $d \geq 0$

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Example

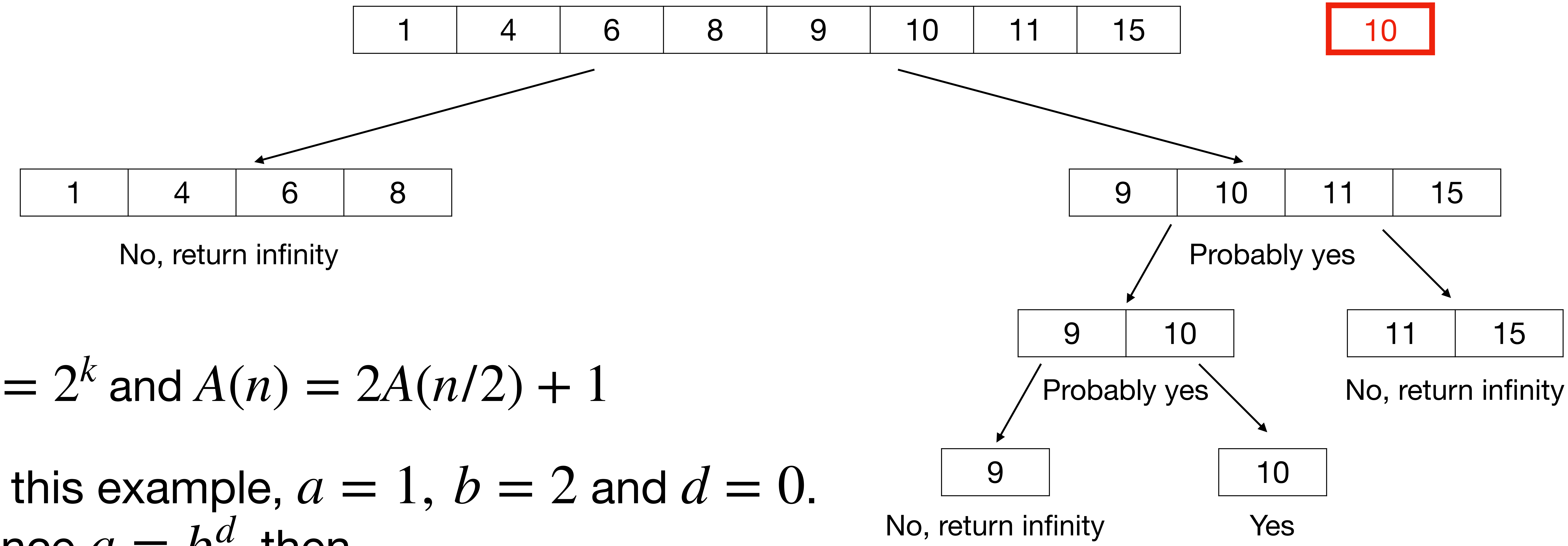
Suppose $n = 2^k$ and $A(n) = 2A(n/2) + 1$

In this example, $a = 2,\ b = 2$ and $d = 0$.
Since $a > b^d$, then

$$A(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \Theta(n)$$

Binary Search

Given a sorted list, and an integer, find the location of the integer in the list.



$n = 2^k$ and $A(n) = 2A(n/2) + 1$

In this example, $a = 1,\ b = 2$ and $d = 0$.
Since $a = b^d$, then

$$A(n) \in \Theta(n^d \log n) = \Theta(\log n)$$

Find the maximum

- Given a list of n elements, the problem is to find the maximum
- A straightforward method is to go through all numbers

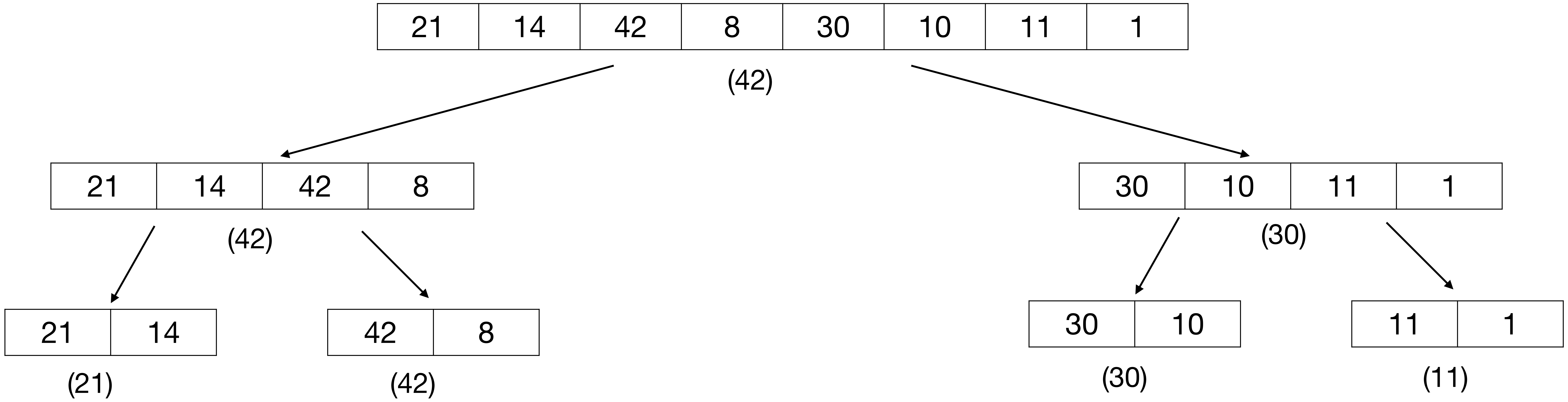
```
max=L(1)
for i = 2,…,n:
 | if L(i)>max then
 |  | max=L(i)
```

Running time: $n - 1$ comparisons

Find the maximum

Divide and Conquer

Given a list of n elements, the problem is to find the maximum items.



number of comparisons =

Find the minimum and the maximum

- Given a list of n elements, the problem is to find the maximum and minimum items.
- A straightforward method is to go through all numbers

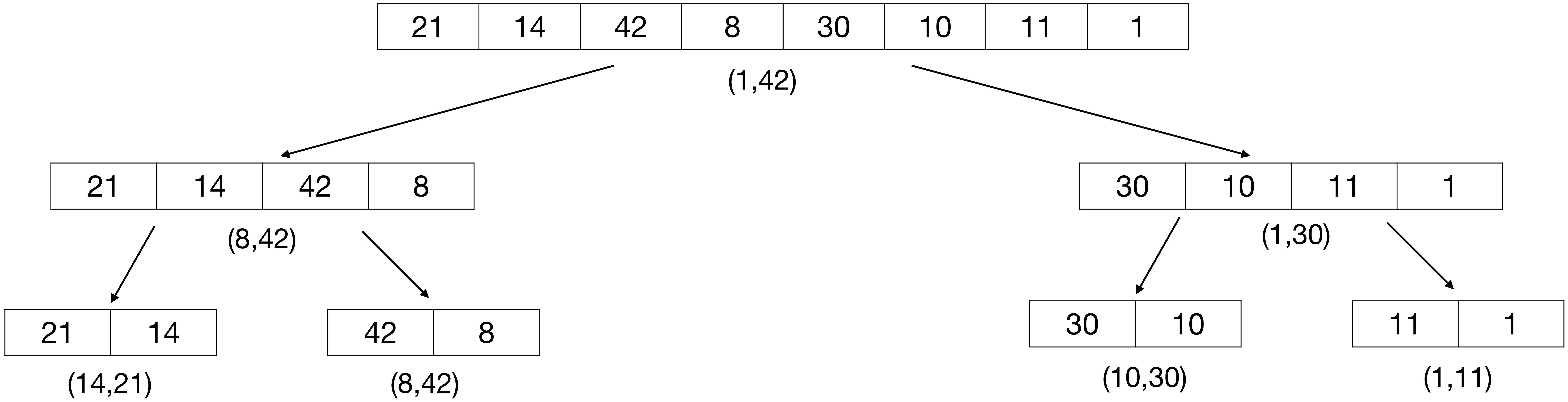
```
min=max=L(1)
for i = 2,…,n:
  | if L(i)>max then
  |   | max=L(i)
  | if L(i)<max then
  |   | min=L(i)
```

Running time: $2(n-1)$ comparisons

Find the minimum and the maximum

Divide and Conquer

Given a list of n elements, the problem is to find the maximum and minimum items.



number of comparisons =

Recurrence equation

Time complexity

$$T(n) = \begin{cases} T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil) + 2 & n > 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$$

If $n = 2^k$

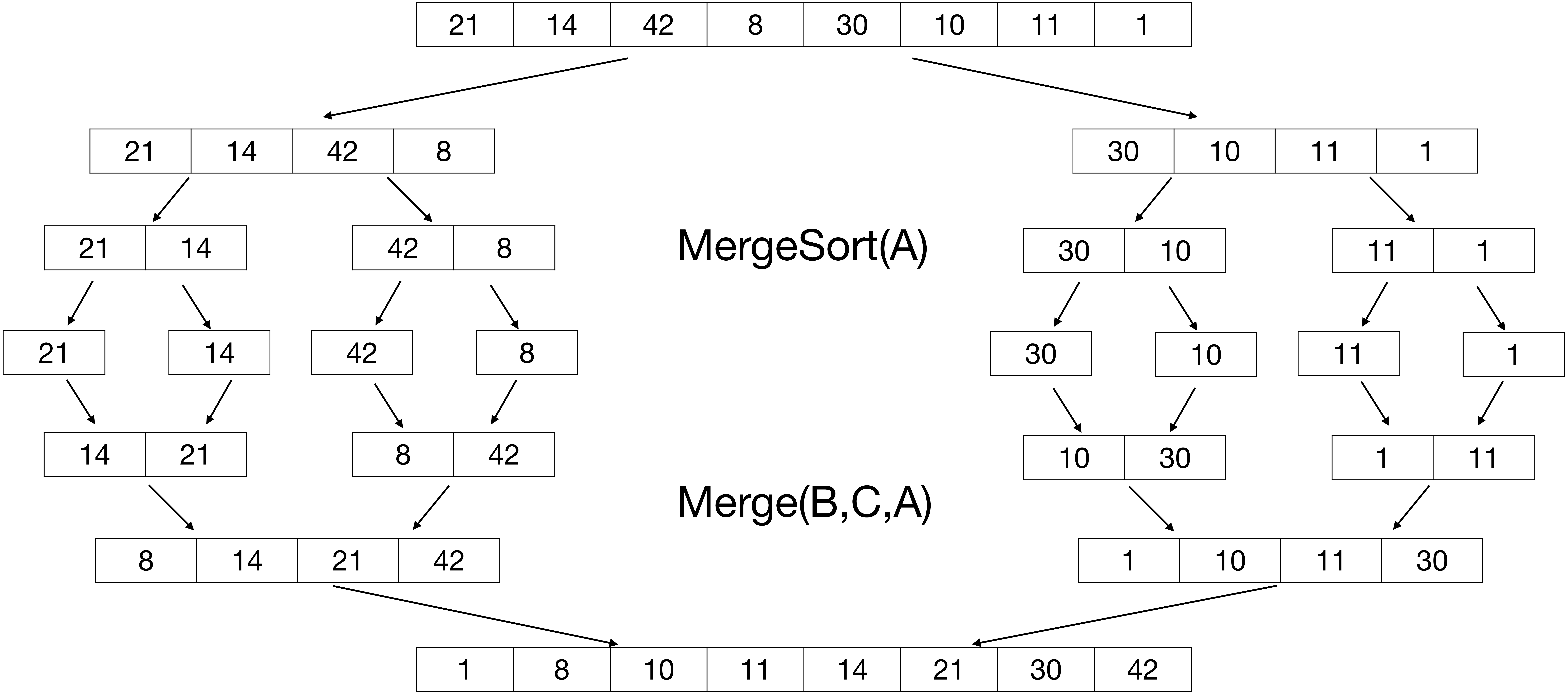
$$\begin{aligned} T(n) &= 2T(n/2) + 2 \\ &= 2(2T(n/2) + 2) + 2 \\ &= 4T(n/4) + 4 + 2 \\ &\vdots \\ &= 2^{k-1}T(2) + \sum_{1 \le i \le k-1} 2^i \\ &= 2^{k-1} + 2^k - 2 \\ &= \frac{3}{2}n - 2 \end{aligned}$$

Merge Sort

- Sorting: given a list of n elements, sort the elements in increasing order
- Based on Divide and Conquer

```
MergeSort(A[0,…,n-1])
if n>1
   copy A[0,…,⌊n/2⌋-1] to B[0,…,⌈n/2⌉-1]
   copy A[⌊n/2⌋,…,n-1] to C[0,…,⌈n/2⌉-1]
   MergeSort(B[0,…,⌈n/2⌉-1])
   MergeSort(C[0,…,⌈n/2⌉-1])
   Merge(B,C,A)
end if
```

Merge Sort



21 14 42 8 30 10 11 1
21 14 42 8
30 10 11 1
21 14
42 8
MergeSort(A)
30 10
11 1
21
14
42
8
30
10
11
1
14 21
8 42
10 30
1 11
Merge(B,C,A)
8 14 21 42
1 10 11 30
1 8 10 11 14 21 30 42

Merge Sort

Merge(B,C,A)

```
Input B[0,…,p-1], C[0,…,q-1]
Output A[0,…,p+q-1]
i←0; j←0; k←0;
while i<p and j<q do
    if B[i]≤C[j]
        A[k]←B[i]
        i←i+1
      else
        A[k]←C[j]
        j←j+1
    end if
    k←k+1
end while
if i=p
    copy C[j … q-1] to A[k … p+q-1]
  else
    copy B[i … p-1] to A[k … p+q-1]
end if
```

B

8	14	21	42

C

1	10	11	30

A

Running time

Merge sort

Running time

$$T(n) = \begin{cases} 2T(\lceil n/2 \rceil) + T_{\text{merge}}(n) & n > 1 \\ 0 & n = 1 \end{cases}$$

Master Theorem

$n = 2^k, f(n) = T_{\text{merge}}(n) = n$

$a = 2,\ b = 2$ and $d = 1$

Since $a = b^d$, then

$T(n) \in \Theta(n^d \log n) = \Theta(n \log n)$

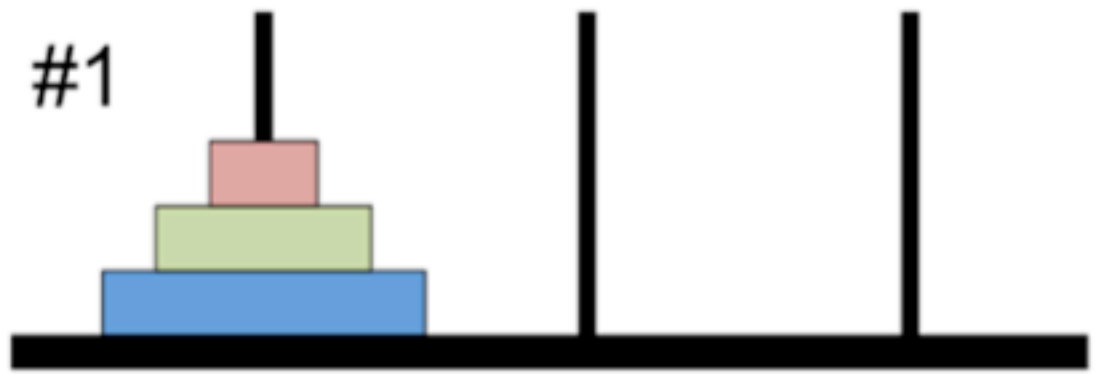
Tower of Hanoi

Given three vertical rods, or towers, and N disks of different sizes, each with a hole in the center so that the rod can slide through it.

The disks are originally stacked on one of the towers in order of descending size (i.e., the largest disc is on the bottom).

The goal of the problem is to move all the disks to a different rod while complying with the following three rules:

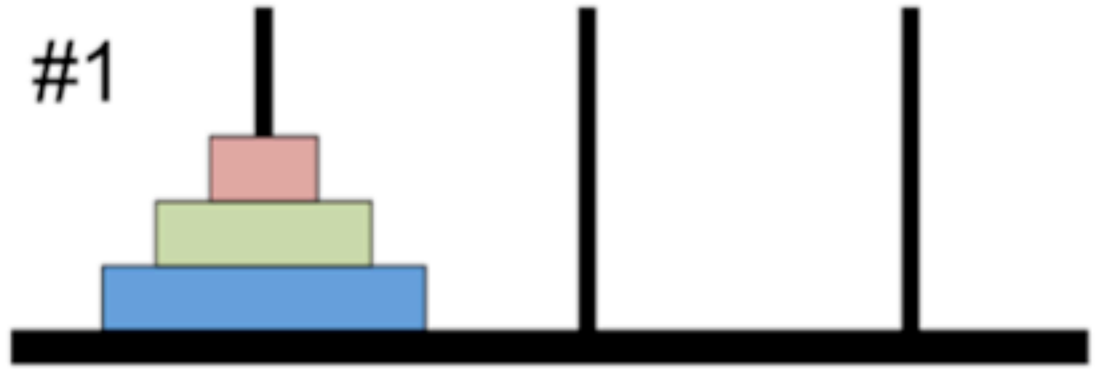
1. Only one disk can be moved at a time
2. Only the disk at the top of a stack may be moved
3. A disk may not be placed on top of a smaller disk



Solution to the Tower of Hanoi

with 3 disks

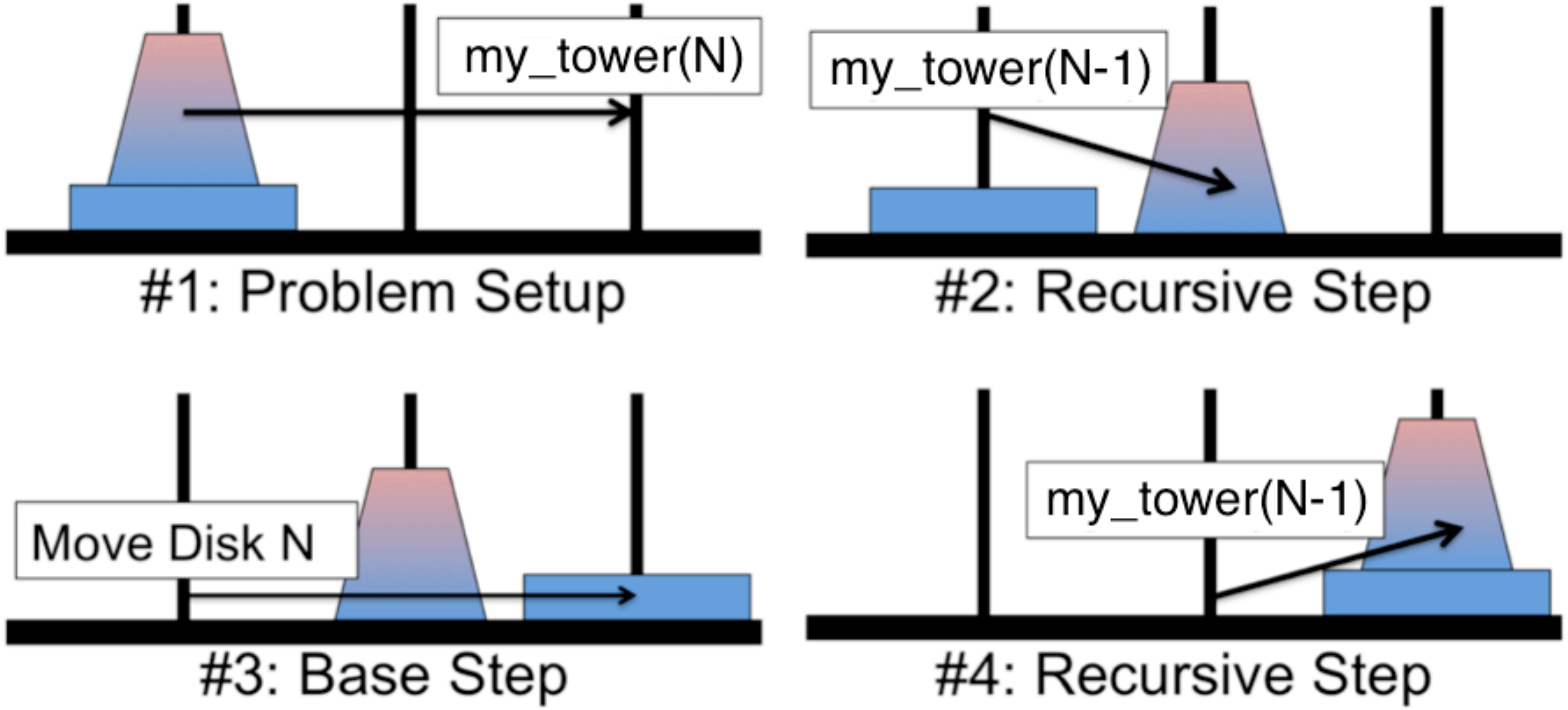
#1



#8

Solution to the Tower of Hanoi

with N disks

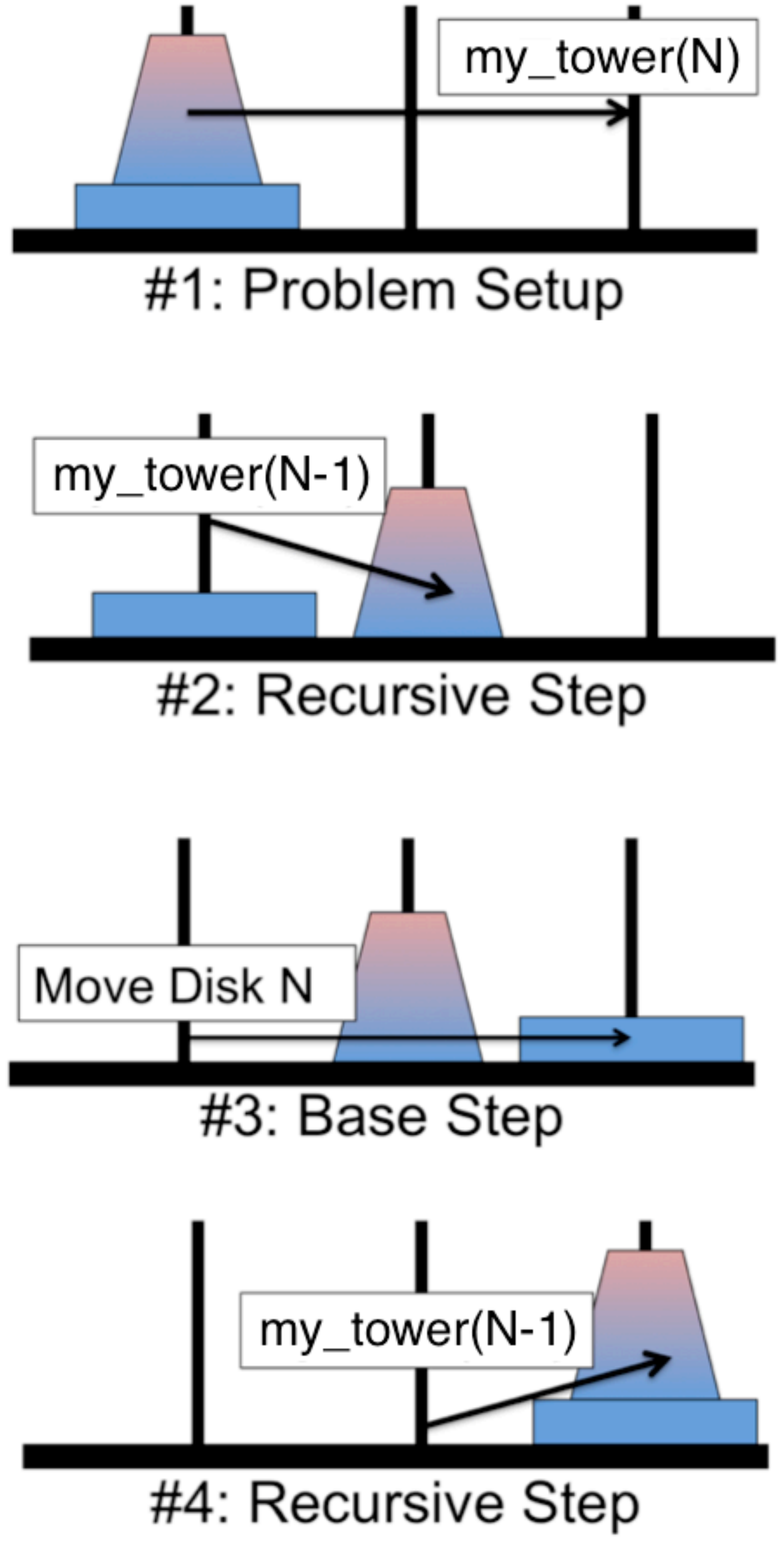


Solution to the Tower of Hanoi

with N disks

```python
def my_towers(N, from_tower, to_tower, alt_tower):
    """
    Displays the moves required to move a tower of size N from the
    'from_tower' to the 'to_tower'.

    'from_tower', 'to_tower' and 'alt_tower' are uniquely either
    1, 2, or 3 referring to tower 1, tower 2, and tower 3.
    """

    if N != 0:
        # recursive call that moves N-1 stack from starting tower
        # to alternate tower
        my_towers(N-1, from_tower, alt_tower, to_tower)

        # display to screen movement of bottom disk from starting
        # tower to final tower
        print("Move disk %d from tower %d to tower %d."\
                  %(N, from_tower, to_tower))

        # recursive call that moves N-1 stack from alternate tower
        # to final tower
        my_towers(N-1, alt_tower, to_tower, from_tower)
```

```python
my_towers(3, 1, 3, 2)
```



Solution to the Tower of Hanoi

with N disks

$$T(n) = \begin{cases} 2T(n-1)+1 & n>1 \\ 1 & n=1 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n-1)+1 \\ &= 2^2T(n-2)+2+1 \\ &\vdots \\ &= 2^iT(n-2)+2^{i-1}+2^{i-2}+\ldots+1 \\ &\vdots \\ &= 2^{n-1}T(1)+2^{n-2}+\ldots+1 \\ &= 2^{n-1}+2^{n-2}+\ldots+1 \\ &= 2^n-1 \end{aligned}$$

Divide And Conquer

Advantages

- The divide and conquer algorithm is easy and effective to solve difficult problems by breaking them into sub-problems
- The divide and conquer algorithm helps to solve the sub-problem independently
- The divide and conquer algorithm makes effective and efficient use of memory caches
- Divide and conquer algorithms does not require any modifications

Disadvantages

- In the divide and conquer algorithm, the recursion of the sub-problem is slow
- For some specific problems, the divide and conquer method becomes complicated in comparison to the iterative method

Dynamic programming

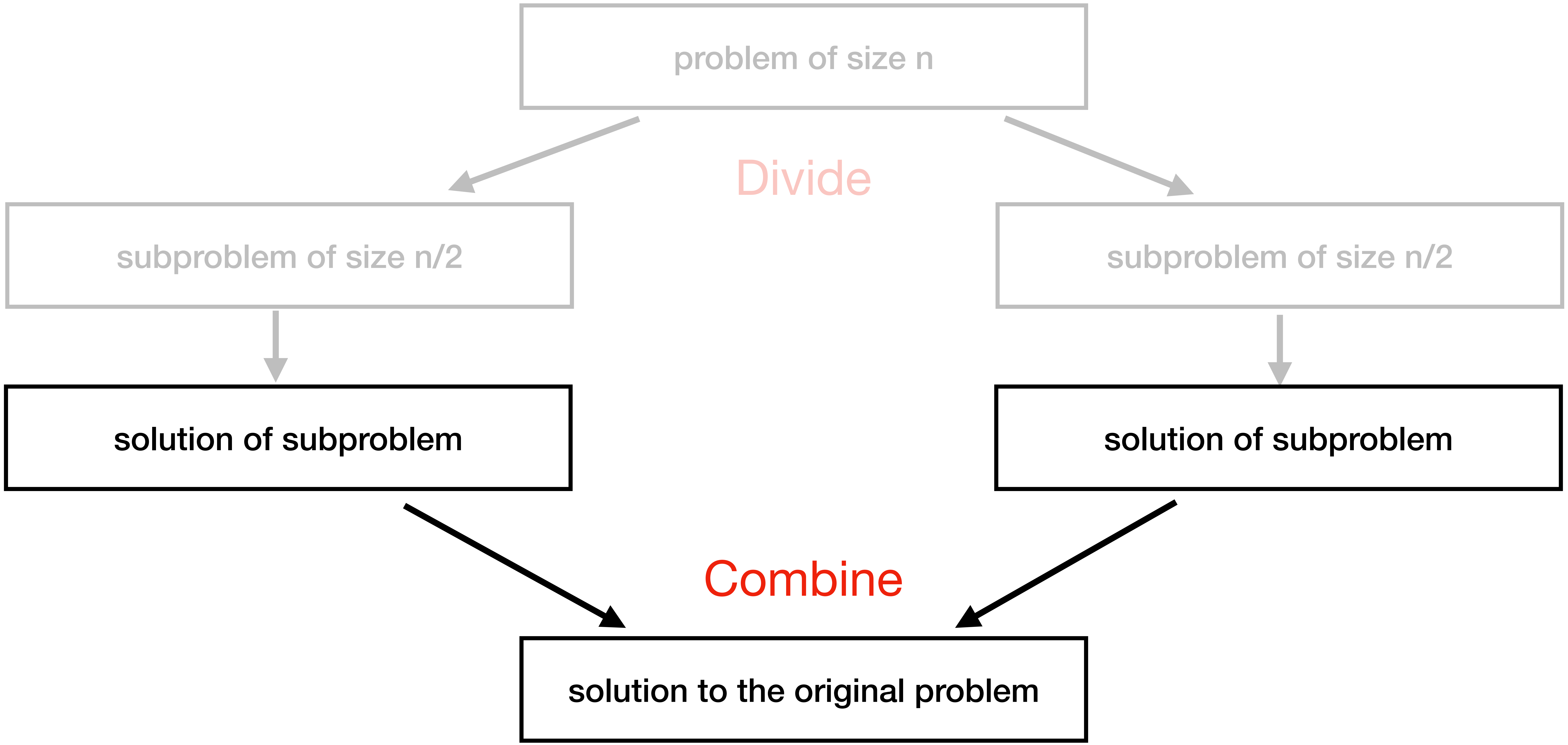
Dynamic Programming

Concept

- Dynamic Programming (commonly referred to as DP) is an algorithmic technique for solving a problem by recursively breaking it down into simpler subproblems and using the fact that the optimal solution to the overall problem depends upon the optimal solution to it’s individual subproblems.
- The technique was developed by **Richard Bellman** in the 1950s.
- DP algorithm solves each subproblem just once and then remembers its answer, thereby avoiding re-computation of the answer for similar subproblem every time.

Dynamic Programming

Concept



Fibonacci numbers

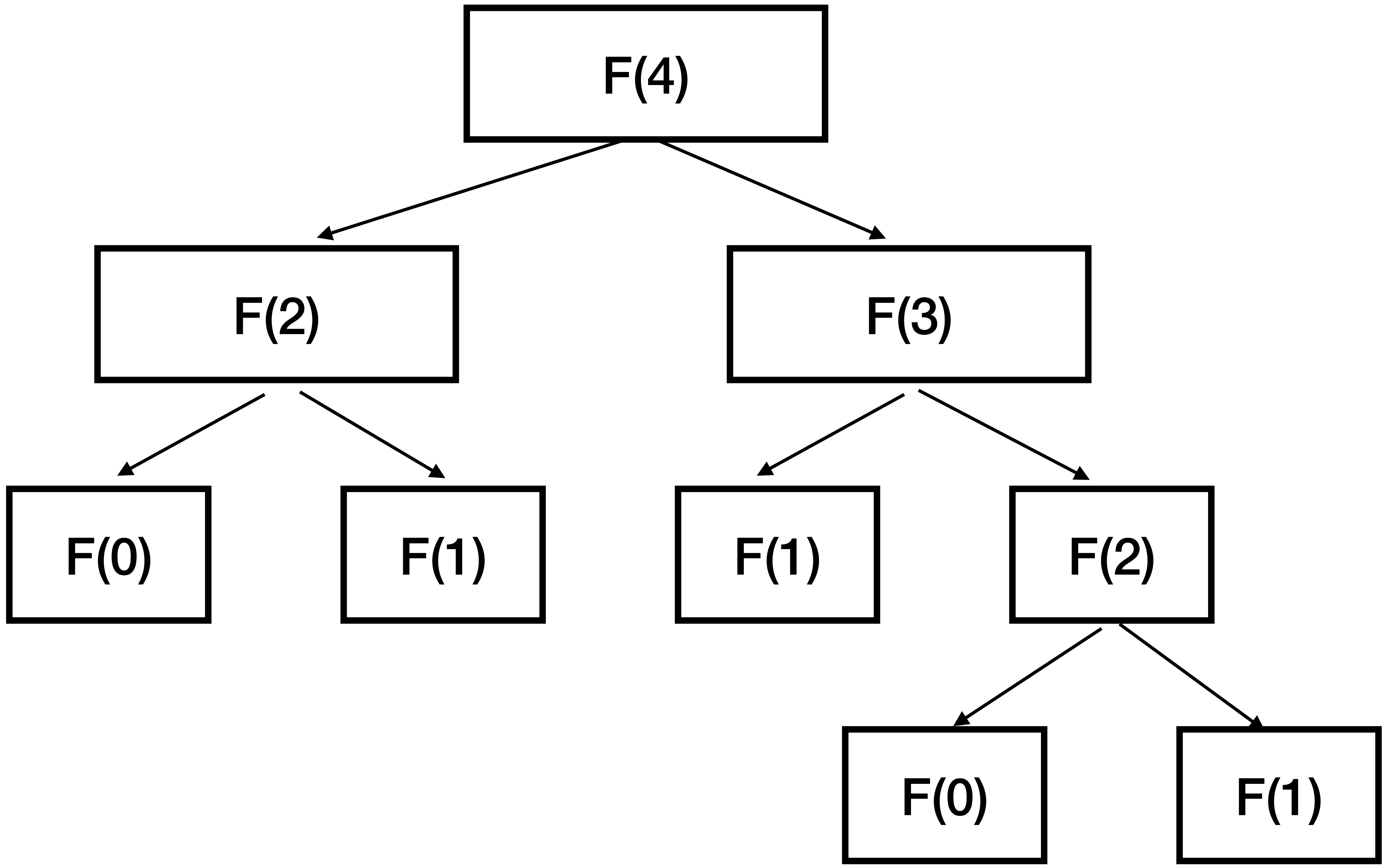
The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

$$F_n = F_{n-1} + F_{n-2}$$

with

$F_0 = 0$ and $F_1 = 1$



exponential running time

Fibonacci numbers

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

$$F_n = F_{n-1} + F_{n-2}$$

with

$F_0 = 0$ and $F_1 = 1$

Iterative way

$F_0 = 0$

$F_1 = 1$

$F_2 =$

$F_3 =$

$F_4 =$

$F_5 =$

…

linear running time

Dynamic Programming

Concept

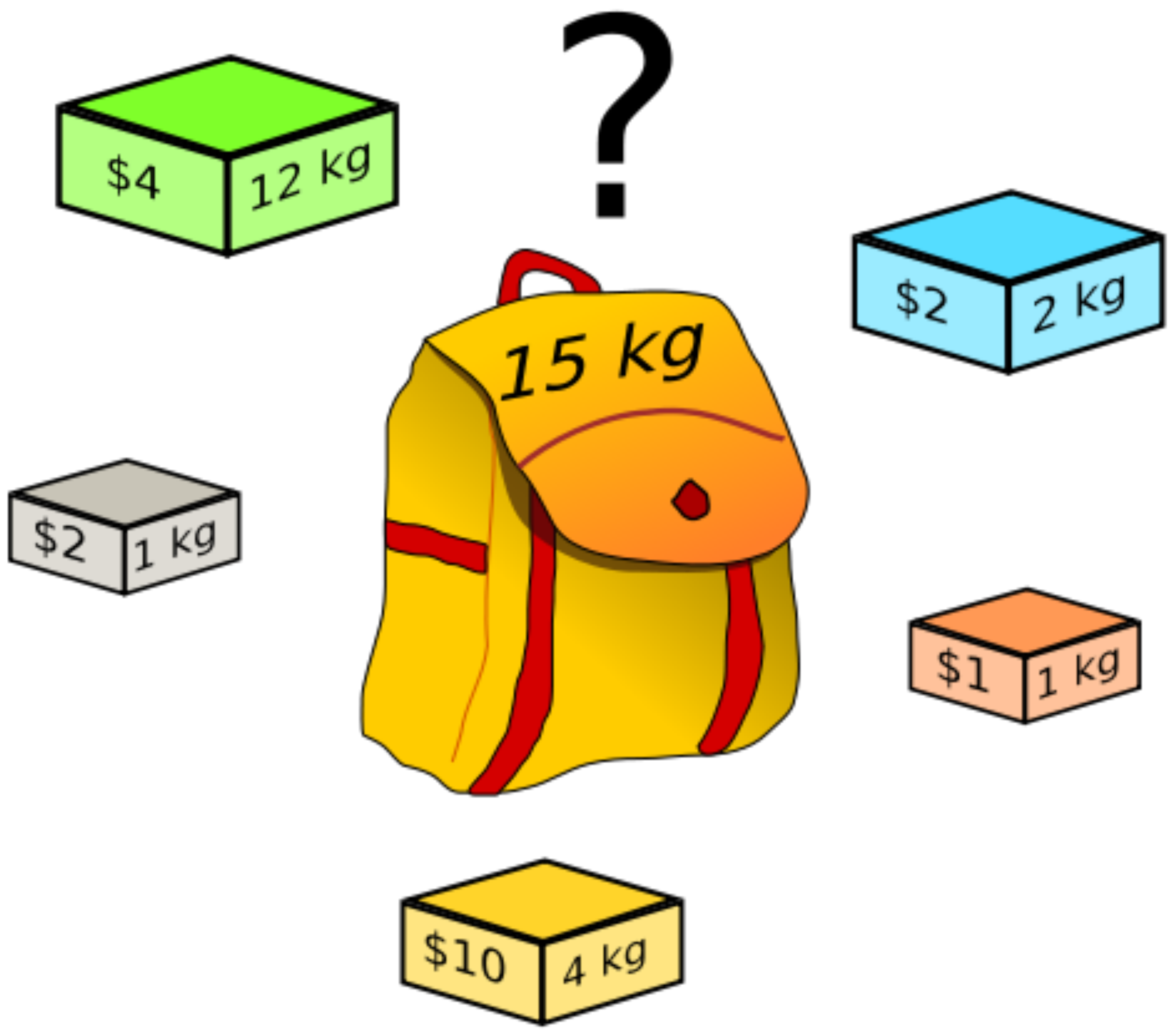
- Dynamic Programming (commonly referred to as DP) is an algorithmic technique for solving a problem by recursively breaking it down into simpler subproblems and using the fact that the optimal solution to the overall problem depends upon the optimal solution to it's individual subproblems.
- The technique was developed by **Richard Bellman** in the 1950s.
- DP algorithm solves each subproblem just once and then remembers its answer, thereby avoiding re-computation of the answer for similar subproblem every time.

Knapsack problem

- Weight capacity K
- Set of n items. Each item i has
  - weight w_i
  - value v_i

Goal: find subset of items such that

- total weight is less than K
- total value is maximized



Knapsack problem

Dynamic Programming formulation

Definition

- $m(i,j)$ is the maximum value (optimal)
  - among items $1,\ldots,i$
  - the total weight of chosen items is at most j

$$m(1,j) = \begin{cases} v_i & \text{if } j \geq w_j \\ 0 & \text{otherwise} \end{cases}$$

$$m(i,j) = \max \begin{cases} m(i-1,j-w_i) + v_i & \text{item } i \text{ is chosen and } j \geq w_i \\ m(i-1,j) & \text{item } i \text{ is not chosen} \end{cases}$$

Knapsack problem

Dynamic Programming formulation

Goal: find subset of items

$$m(i,j) = \max \begin{cases} m(i-1,j-w_i) + v_i & \text{item } i \text{ is chosen and } j \geq w_i \\ m(i-1,j) & \text{item } i \text{ is not chosen} \end{cases}$$

$$m(1,j) = \begin{cases} v_i & \text{if } j \geq w_j \\ 0 & \text{otherwise} \end{cases}$$

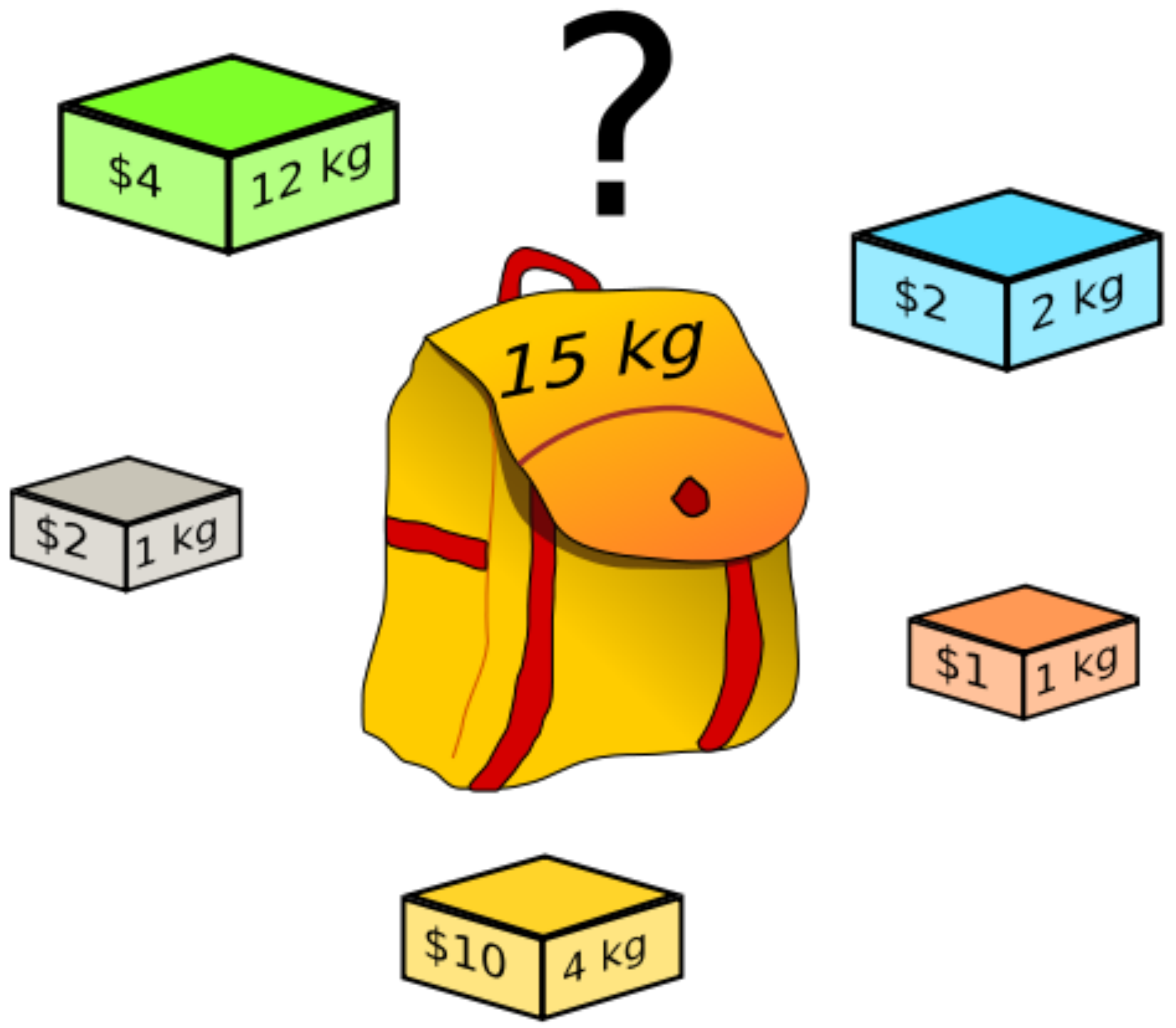
Weight capacity K=15

item i	weight w_i	value v_i
1	12	4
2	1	2
3	4	10
4	1	1
5	2	2

$i \setminus j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6
3	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12
4	2	3	3	10	12	13	13	13	13	13	13	13	13	13	13
5	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Knapsack problem

Dynamic Programming formulation



Weight capacity K=15

item i	weight w_i	value v_i
1	12	4
2	1	2
3	4	10
4	1	1
5	2	2

i \ j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6
3	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12
4	2	3	3	10	12	13	13	13	13	13	13	13	13	13	13
5	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Knapsack problem

Dynamic Programming formulation

Definition

not the unique formulation

- $m(i, j)$ is the maximum value (optimal)
  - among items $1, \ldots, i$
  - the total weight of chosen items is at most j

$$m(1,j) = \begin{cases} v_i & \text{if } j \geq w_j \\ 0 & \text{otherwise} \end{cases}$$

$$m(i,j) = \max \begin{cases} m(i-1, j-w_i) + v_i & \text{item } i \text{ is chosen and } j \geq w_i \\ m(i-1, j) & \text{item } i \text{ is not chosen} \end{cases}$$

Knapsack problem

Dynamic Programming formulation

Definition

- $p(i, k)$ is the minimum weight (optimal)
  - among items $1, \ldots, i$
  - the total value of chosen items is exactly k

$$p(1,k) = \begin{cases} w_1 & \text{if } k = v_j \\ +\infty & \text{otherwise} \end{cases}$$

$$p(i,k) = \min \begin{cases} p(i-1, k-v_i) + w_i & \text{item } i \text{ is chosen and } k \geq v_i \\ p(i-1, k) & \text{item } i \text{ is not chosen} \end{cases}$$

Knapsack problem

which formulation is better?

$$m(i,j) = \max \begin{cases} m(i-1,j-w_i) + v_i \\ m(i-1,j) \end{cases}$$

vs

$$p(i,k) = \min \begin{cases} p(i-1,k-v_i) + w_i \\ p(i-1,k) \end{cases}$$

Knapsack problem

which formulation is better?

$$m(i,j) = \max \begin{cases} m(i-1,j-w_i) + v_i \\ m(i-1,j) \end{cases}$$

$O(nK)$

$1,\ldots,n$ $\quad$ $1,\ldots,K$

vs

$1,\ldots,n$ $\quad$ $1,\ldots,V$

$$p(i,k) = \min \begin{cases} p(i-1,k-v_i) + w_i \\ p(i-1,k) \end{cases}$$

$O(nV)$

Partition Problem

- Given a set of integers $S = \{a_1, a_2, \ldots, a_n\}$
- Goal: Find a subset $U \subseteq S$ such that

$$\sum_{i \in U} a_i = \sum_{i \in S \setminus U} a_i$$

Example: $S = \{1,1,2,3,7,10,12\}$

A valid partition is $\{1,2,3,12\} \cup \{1,7,10\}$

Partition problem

Dynamic programming formulation

Example: $S = \{1,1,2,3,7,10,12\}$ valid partition $\{1,2,3,12\} \cup \{1,7,10\}$

Observation: sum of integers in a subset is $A = \frac{1}{2}\sum_{i \in S} a_i$

$$partition(i,k) = \begin{cases} 1 & \text{if there exists a subset of } \{a_1, \ldots, a_i\} \text{ with sum equal to } k \\ 0 & \text{otherwise} \end{cases}$$

Partition problem

Dynamic programming formulation

$$partition(i,k) = \begin{cases} 1 & \text{if there exists a subset of } \{a_1, \ldots, a_i\} \text{ with sum equal to } k \\ 0 & \text{otherwise} \end{cases}$$

$$partition(i,k) = \max \begin{cases} partition(i, k - a_i) & \text{if } a_i \leq k \\ partition(i,k) & \text{otherwise} \end{cases}$$

$$partition(0,k) = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{if } k > 0 \end{cases}$$

Partition problem

$S = \{1,1,2,3,7,10,12\}$

$A = \frac{1}{2}\sum_{i \in S} a_i = 18$

$$partition(i,k) = \max \begin{cases} partition(i, k - a_i) & \text{if } a_i \leq k \\ partition(i,k) & \text{otherwise} \end{cases}$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Difference with Divide and Conquer

- Characterizing the structure of an optimal solution
- Define the value of an optimal solution recursively
- Using the bottom-up algorithm, calculate the value of an optimal solution
- Using computed information, construct an optimal solution

Exercises

- You may discuss with other classmates about algorithm design
- But coding part and reports must be individual

Investment problem

You are granted a project in which you need to do some archaeological excavation. There are four different places that you can dig, however you only have 10 days for the excavation. The expected rewards for each day are shown in the following table

Place / Days	1	2	3	4
0	0	0	0	0
1	10	3	4	1
2	15	9	6	10
3	16	17	9	13
4	17	26	13	16
5	30	28	15	20

1. Design an efficient algorithm to plan the number of days to spend on each place to maximize the reward. The output should be the following.

```
Place 1, spend X days
Place 2, spend X days
Place 3, spend X days
Place 4, spend X days
Total profit = XX
```

2. Let P number of places, and D the number of total days. What is the complexity time of the algorithm?

Find the minimum

Given a list of numbers, we know that the numbers are decreasing, then increasing. The goal is to find the smallest number in the list. Of course, you go through all the numbers of the list. Design a faster algorithm to find the smallest number in the list.

1.Suppose that any two consecutive numbers are different in the list. How many comparisons do you need to find the minimum if n=20? If n=100?

$$L = [20,18,14,13,12,9,10,12,14,15,16,20,25,30]$$

$$\min(L) = 9$$

2. Suppose that two consecutive numbers may be the same in the list. How do you handle this case? How many comparisons do you need to find the minimum if n=20? If n=100?

$$L = [20,18,14,13,12,9,10,12,14,15,15,15,18,20,25,30]$$