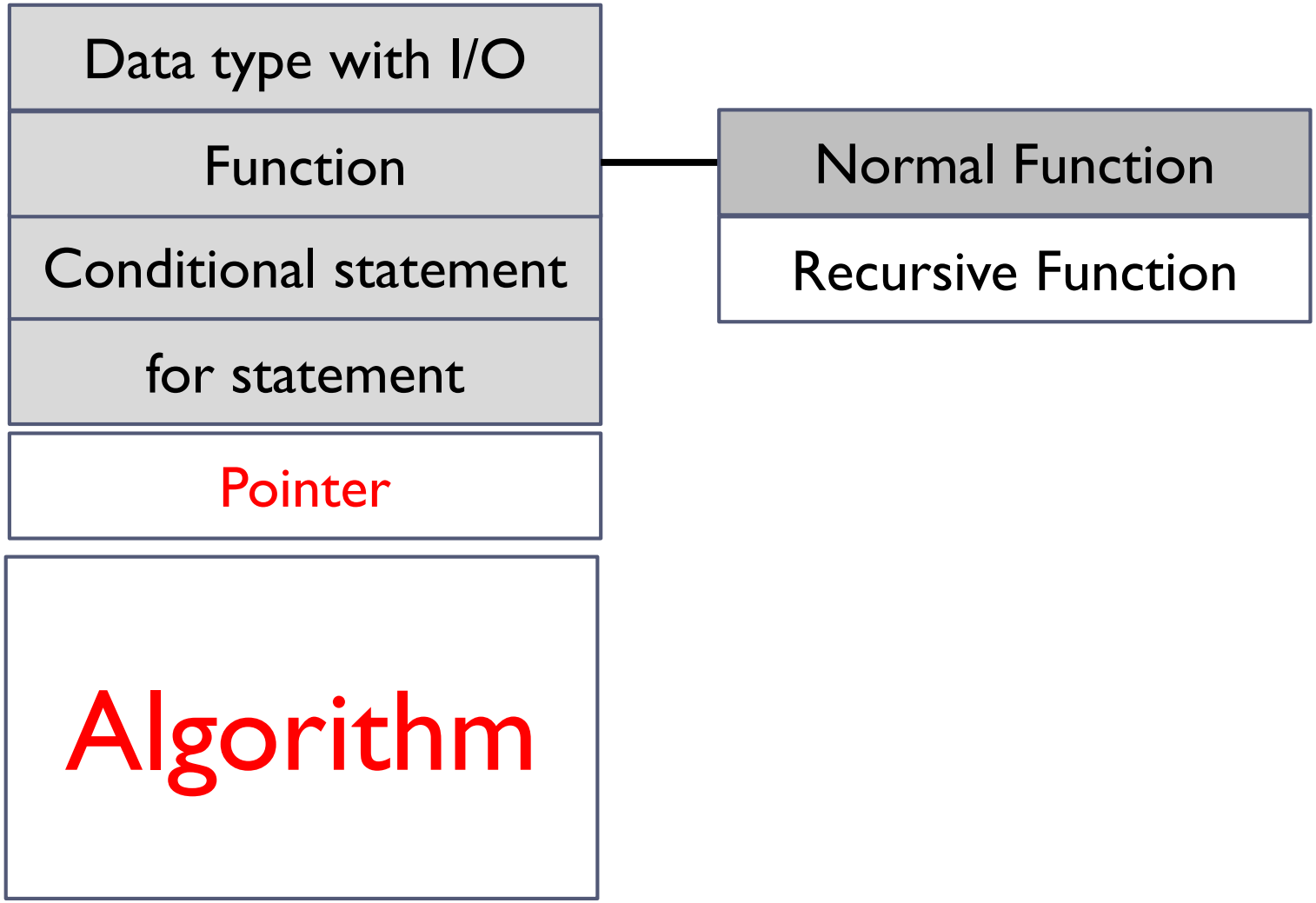
2013 Fall Semester SMP

Day 4 - Recursion

Contents

- Data type with I/O
- Function
- Conditional statement
- for statement
- Pointer
- Algorithm

Contents


Data type with I/O
Function
Conditional statement
for statement
Pointer
Algorithm
Normal Function
Recursive Function

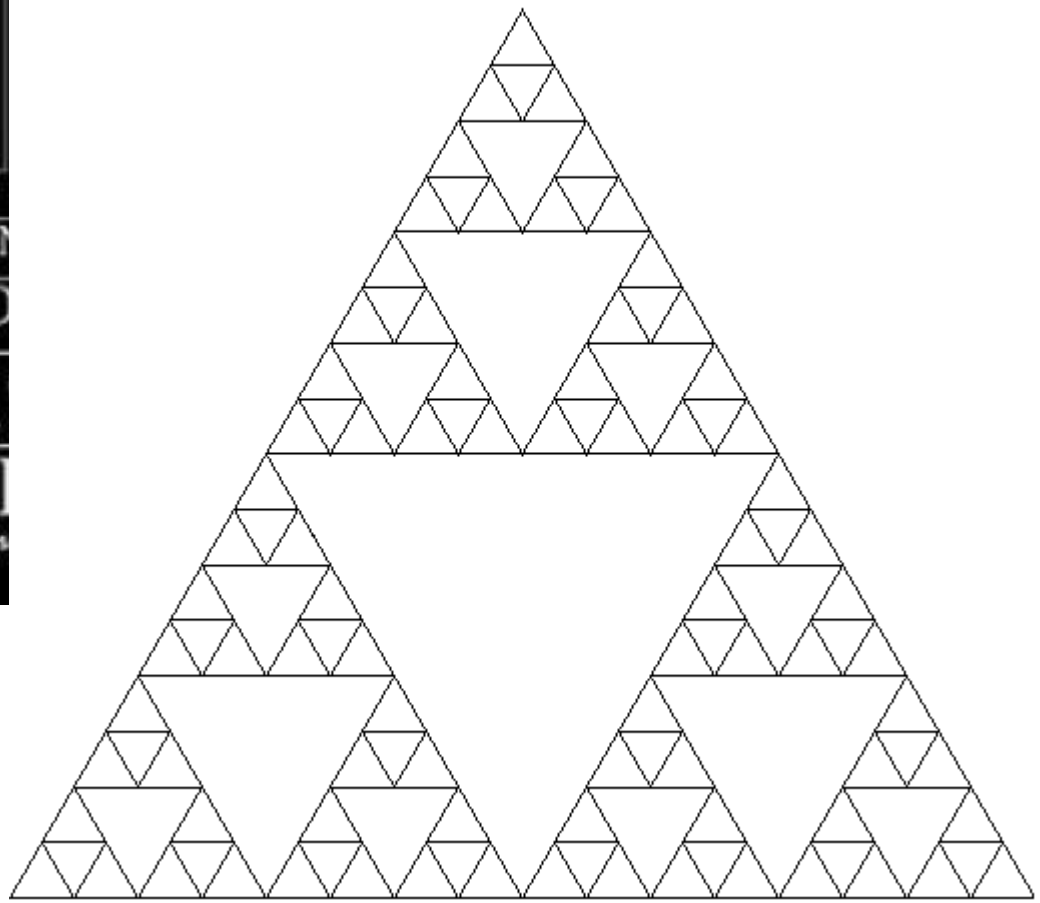
Recursion

- **Recursion** is the process of repeating items in a **self-similar way** (from wiki)
- **Recursive function** is implemented in a programming language, whose implementation references itself
  - 자기 자신을 호출하는 함수
  - Like..

```
1:  #include <cstdio>
2:  #include <algorithm>
3:  using namespace std;
4:  int main(){
5:      main();
6:      return 0;
7:  }
8:
```

Recursion


RECURSION
RECURSION
RECURSION
RECURSION
RECURSION
RECURSION
RECURSION
May I see your passport, please?
May I see your passport, please?



Recursion Example

- Fibonacci number
  - 재귀적으로 정의됨 (Recurrence relation)
  - $F_n = F_{n-1} + F_{n-2}$

```
 1: #include <cstdio>
 2: #include <algorithm>
 3: using namespace std;
 4: int get_fibo(int x){
 5:     if(x == 0) return 1;
 6:     else if(x == 1) return 1;
 7:     else return get_fibo(x-1) + get_fibo(x-2);
 8: }
 9: int main(){
10:     printf("%d",get_fibo(5));
11:     return 0;
12: }
```

Recursion Example

- Factorial
  - 마찬가지로 점화식이 있음
  - $n! = n \times (n-1)!$

```
 1: #include <cstdio>
 2: #include <algorithm>
 3: using namespace std;
 4: int get_fact(int x){
 5:     if(x == 0) return 1;
 6:     else return x * get_fact(x-1);
 7: }
 8: int main(){
 9:     printf("%d",get_fact(5));
10:     return 0;
11: }
```

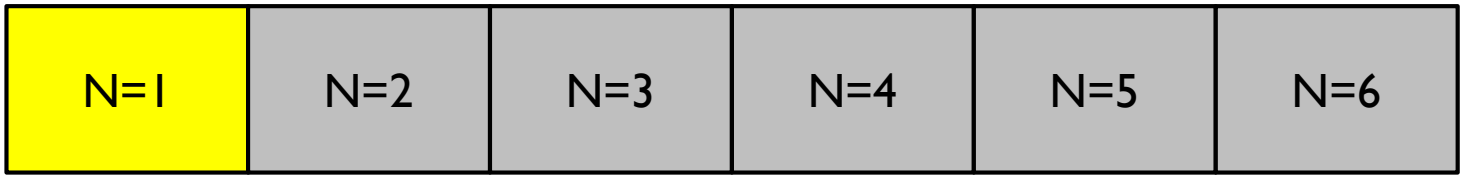
Recursion & Induction

- 재귀는 수학적 귀납법과 매우 비슷하다
  - 수학적 귀납법의 경우, 명제를 증명할 때 우선
  - Base condition을 설정하고,
  - 재귀적인 관계를 이용해서 증명한다
- $Prove\ that\ \sum_{i=1}^{n} i = \frac{n(n+1)}{2}\ for\ n \in N$
  - Base condition (n=1) : $\sum_{i=1}^{n} i = 1 = \frac{1(1+1)}{2}$
  - Suppose that $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$ then

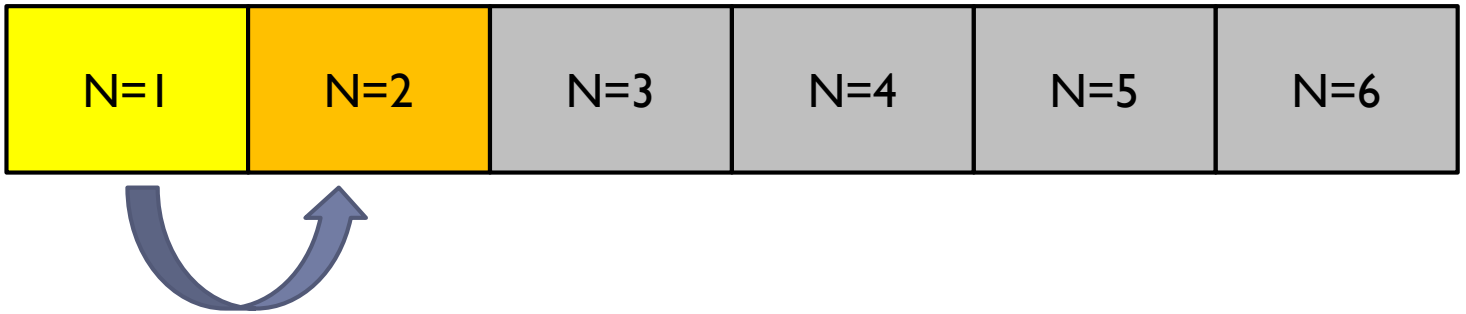
$$\sum_{i=1}^{k+1} i = (k+1) + \boxed{\sum_{i=1}^{k} i} = (k+1) + \boxed{\frac{k(k+1)}{2}} = \frac{(k+1)(k+2)}{2}$$

By assumption

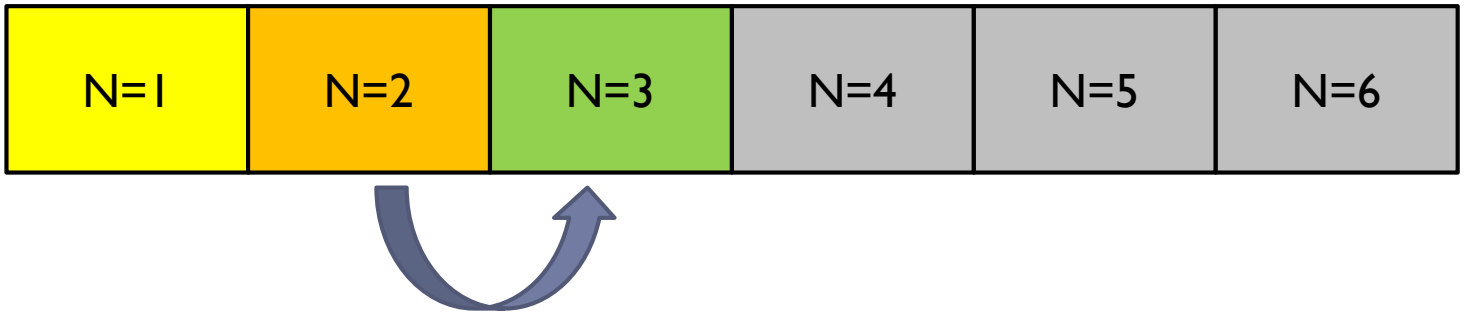
Recursion & Induction



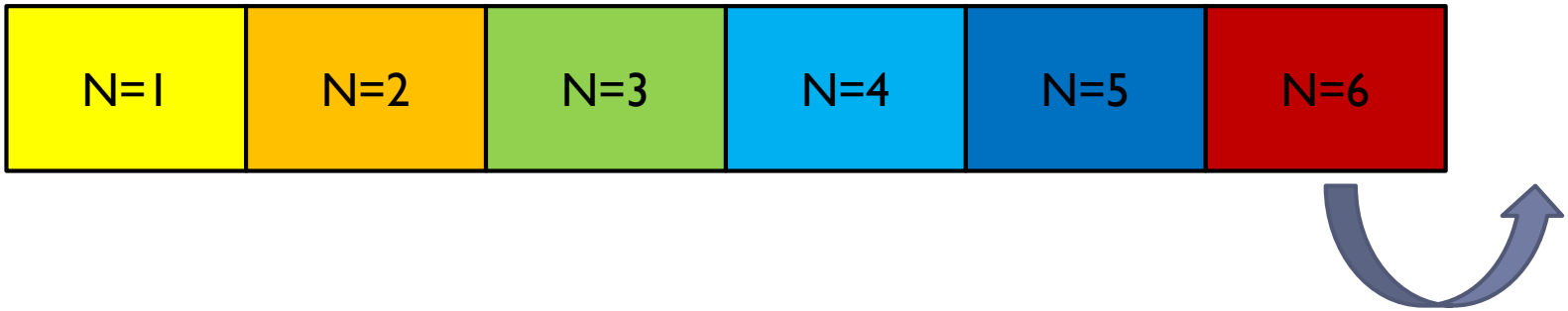
Recursion & Induction



Recursion & Induction



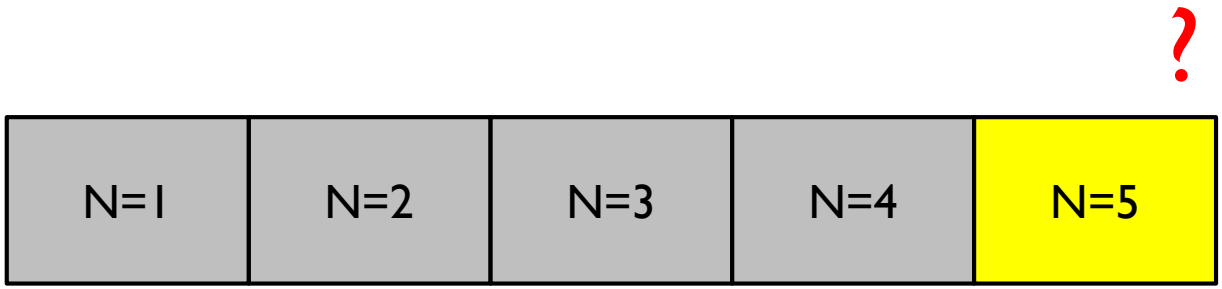
Recursion & Induction



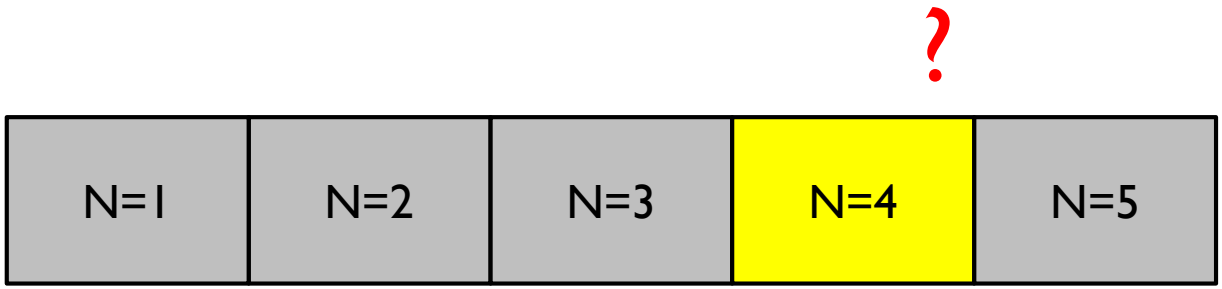
Recursion & Induction

- 재귀는 수학적 귀납법과 매우 비슷하다
  - **But**, 증명 순서가 반대
  - **n = k-1 일 때 성립한다고 가정하면, n = k 일 때 성립한다.**
  - **당연히 초기조건이 있어야 함**
- Factorial을 다시 생각해보면…

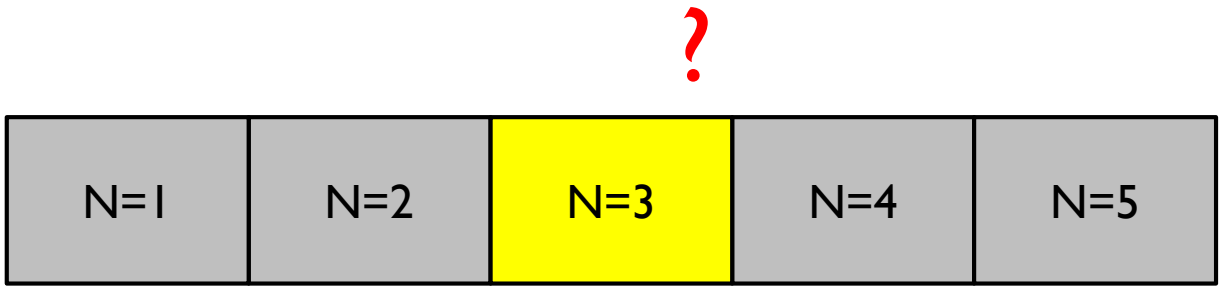
Recursion & Induction



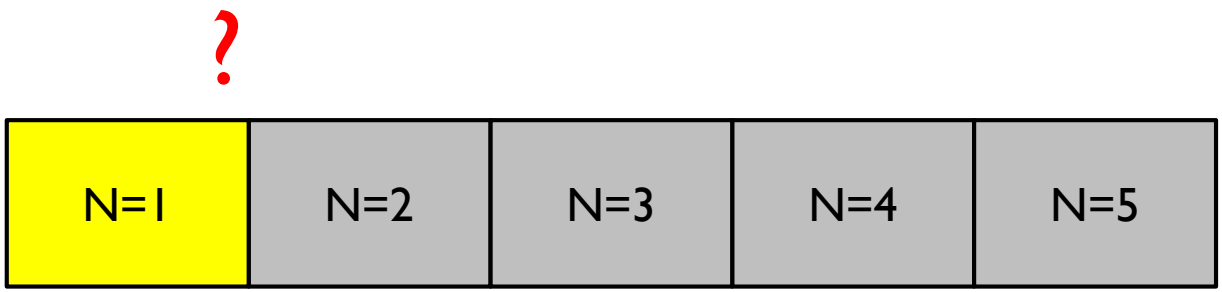
Recursion & Induction



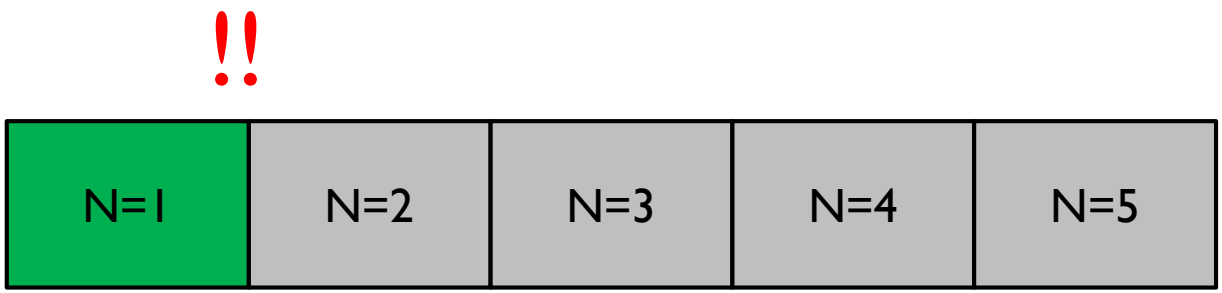
Recursion & Induction



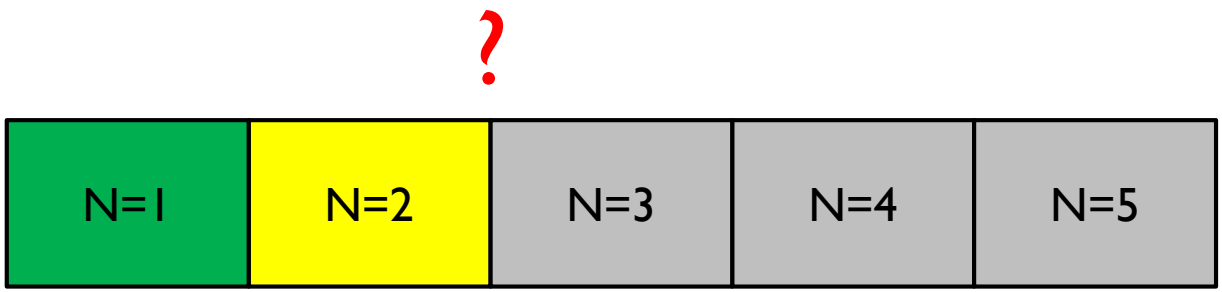
Recursion & Induction



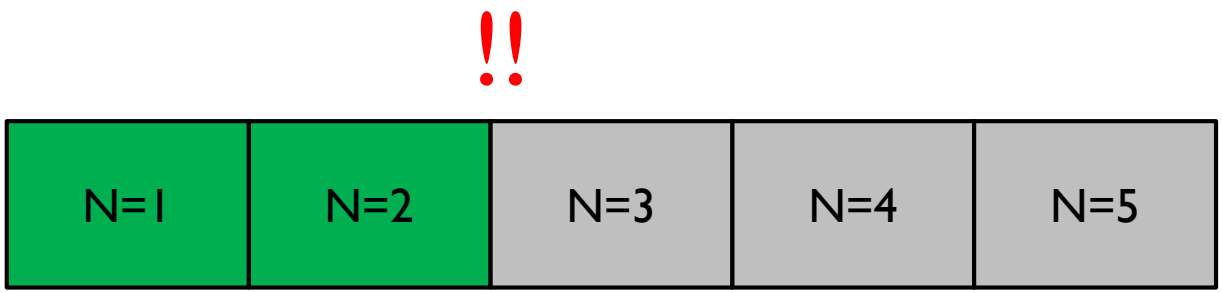
Recursion & Induction



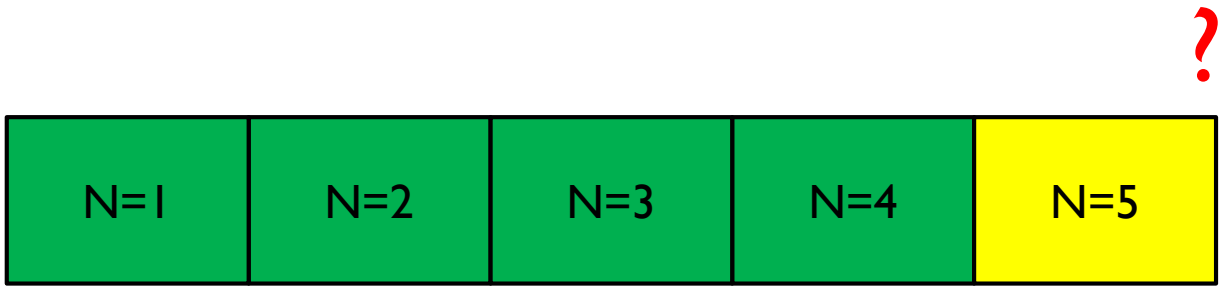
Recursion & Induction



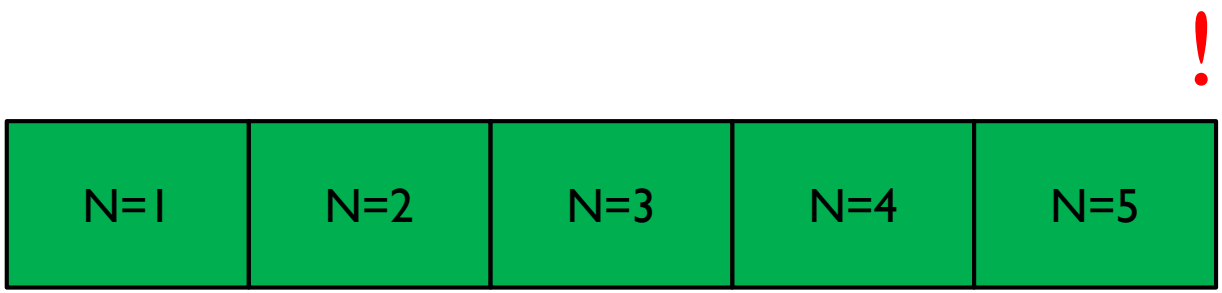
Recursion & Induction



Recursion & Induction



Recursion & Induction



Recursion Implementation

- Function Decl {
  Base Condition
  Process
  }

```
 1: #include <cstdio>
 2: #include <algorithm>
 3: using namespace std;
 4: int get_fibo(int x){
 5:     if(x == 0) return 1;
 6:     else if(x == 1) return 1;
 7:     else{
 8:         int one = get_fibo(x-1);
 9:         int two = get_fibo(x-2);
10:         return one + two;
11:     }
12: }
13: int main(){
14:     printf("%d",get_fibo(4));
15:     return 0;
16: }
```

Recursion Implementation

- 우선, 함수의 정의를 세운다
  - **귀납법**에서 증명하고자 하는 명제를 세우는 것과 비슷
  - int get_fact(int x) : x! 를 계산하여 return하는 함수
  - int get_fibo(int x) : Fx 를 계산하여 return하는 함수
- 그리고 귀납법처럼 생각하여 식을 세운다
  - return x * get_fact(x-1)
  - return get_fibo(x-1) + get_fibo(x-2)
- ~~참 쉽죠?~~

Recursion

- Recursion은 코딩을 함에 있어 첫 번째 난관
  - 극뽁! 해야 인생이 편해집니다
  - ~~착하게 삽시다~~
- 여러가지 예제를 다루면서 익숙해 지는 것이 관건

- More Issue?

Recursion

- Divide & Conquer
- Dynamic Programming

- Back Tracking

Recursion

- **Divide & Conquer**
- Dynamic Programming

- Back Tracking

Divide & Conquer

- **Divide & Conquer (D&C)** is an important algorithm design paradigm based on multi-branched recursion. (wiki)
  - 말 그대로 분할해서 정ㅋ벅ㅋ
  - 구체적으로 알아봅시다

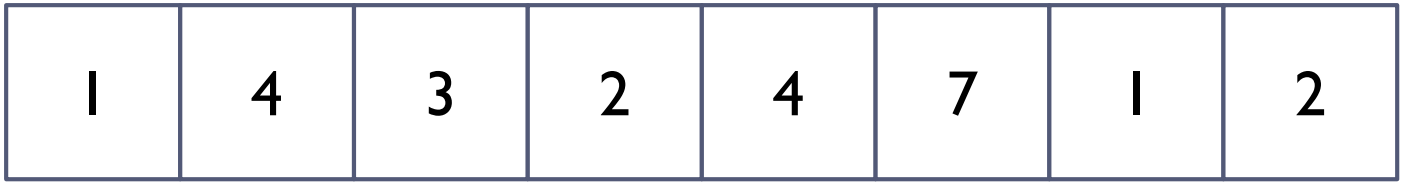
Merge Sort

- Divide & Conquer의 가장 기본적인 형태
  - 일단 나누고, 각각에 대해서 작업을 수행한 다음
  - 하나로 합친다!
- 매우 중요함
  - Divide & Conquer의 대표적 모델로써 중요하고
  - O(n log n) 의 효율적인 정렬 알고리즘이라 중요하다
  - 응용이 될 가능성이 매우 큼

Merge Sort

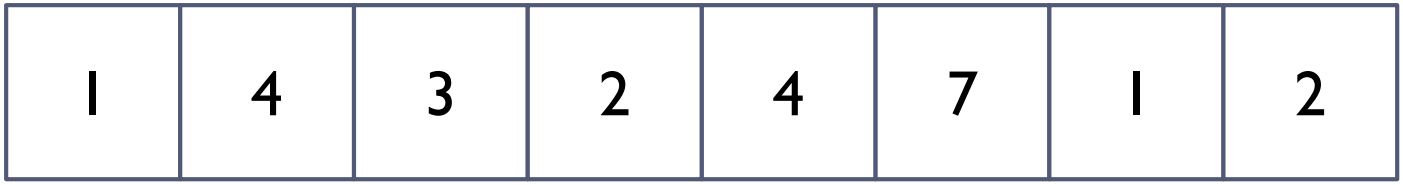
- Merge Sort의 절차
  - 1. 일단 두 부분으로 나눈다
  - 2. 각각의 부분을 어떻게든 정렬한다.
  - 3. 두 부분을 합친다

Merge Sort



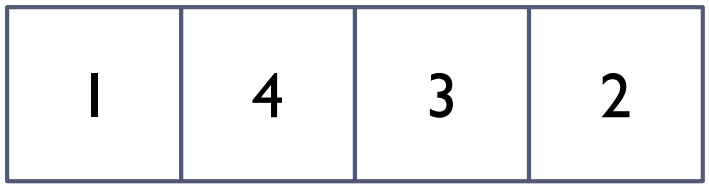
Merge Sort

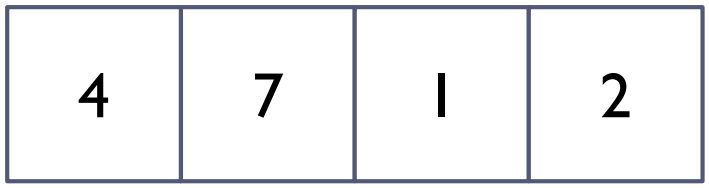
- 1. 일단 나눈다



Merge Sort

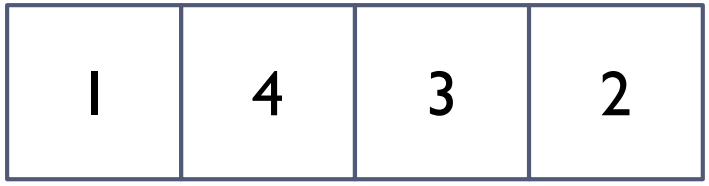
- 1. 일단 나눈다

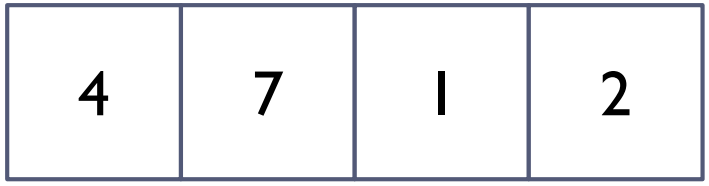




Merge Sort

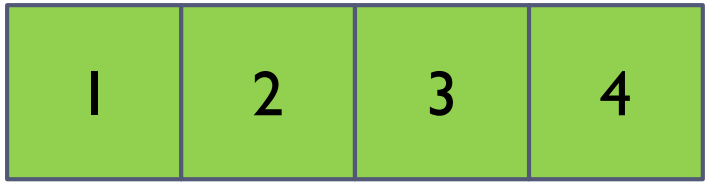
- 2. 어떻게든 둘을 정렬한다

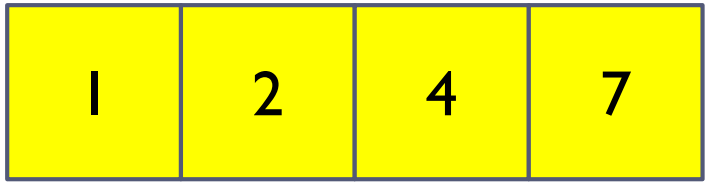




Merge Sort

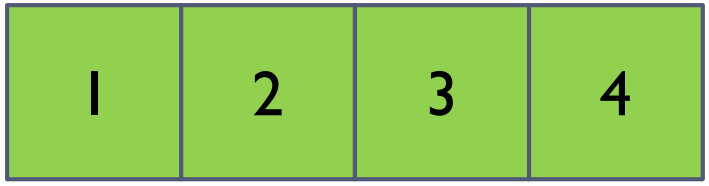
- 2. 어떻게든 둘을 정렬한다

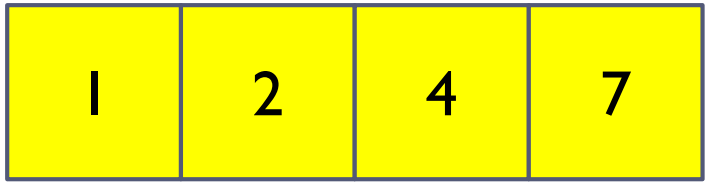




Merge Sort

- 3. 다시 하나로 합친다





Merge Sort

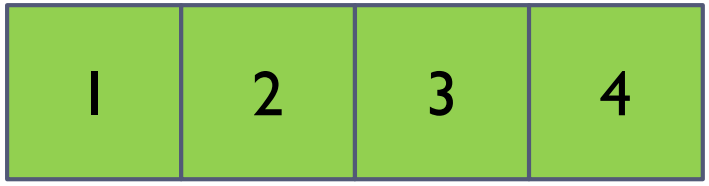
- 3. 다시 하나로 합친다

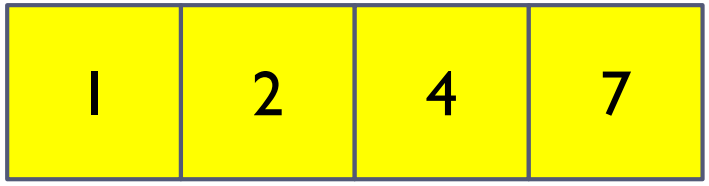
Merge Sort

- 3. 다시 하나로 합친다 ?????

Merge Sort

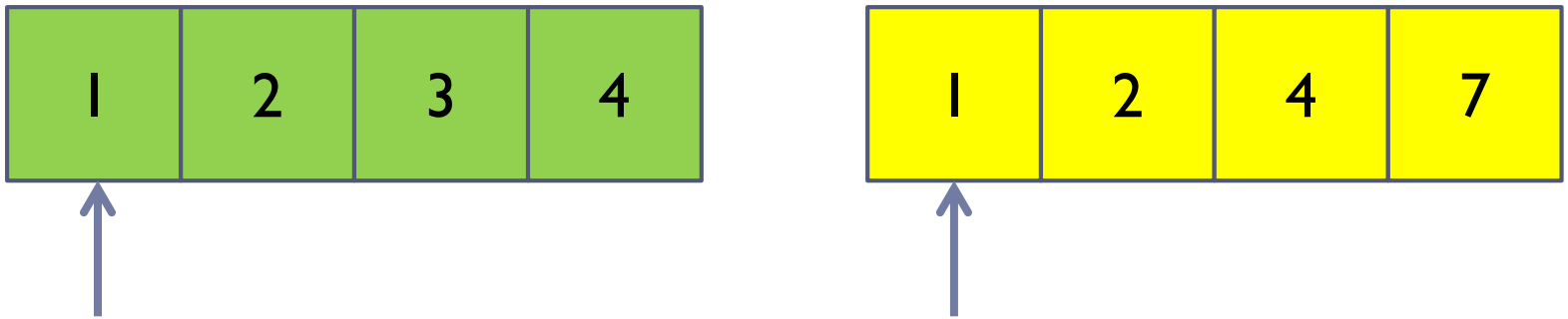
- 3. 다시 하나로 합친다





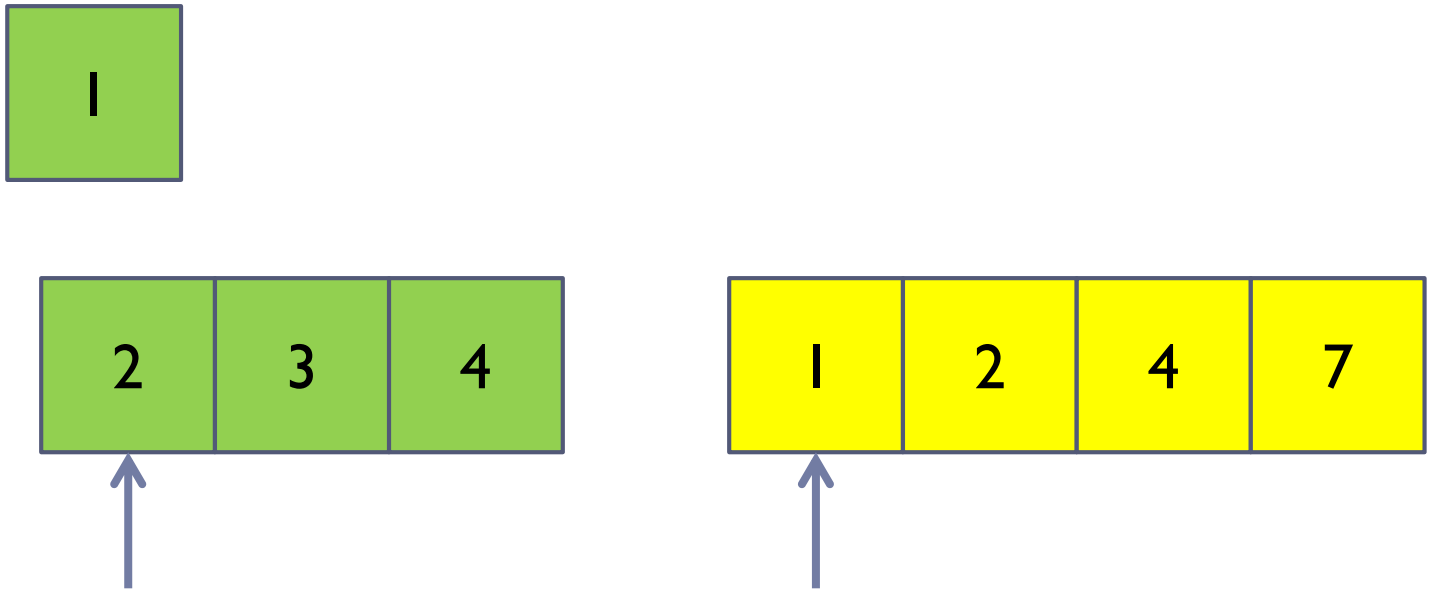
Merge Sort

- 3. 다시 하나로 합친다



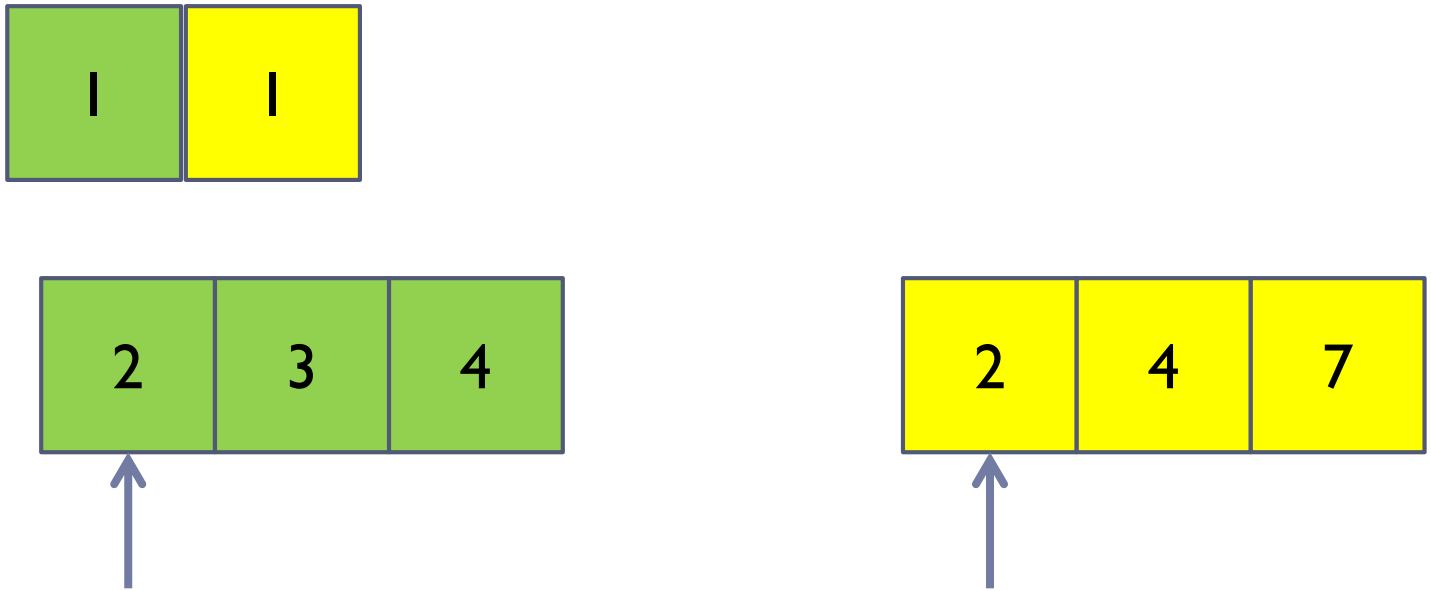
Merge Sort

- 3. 다시 하나로 합친다



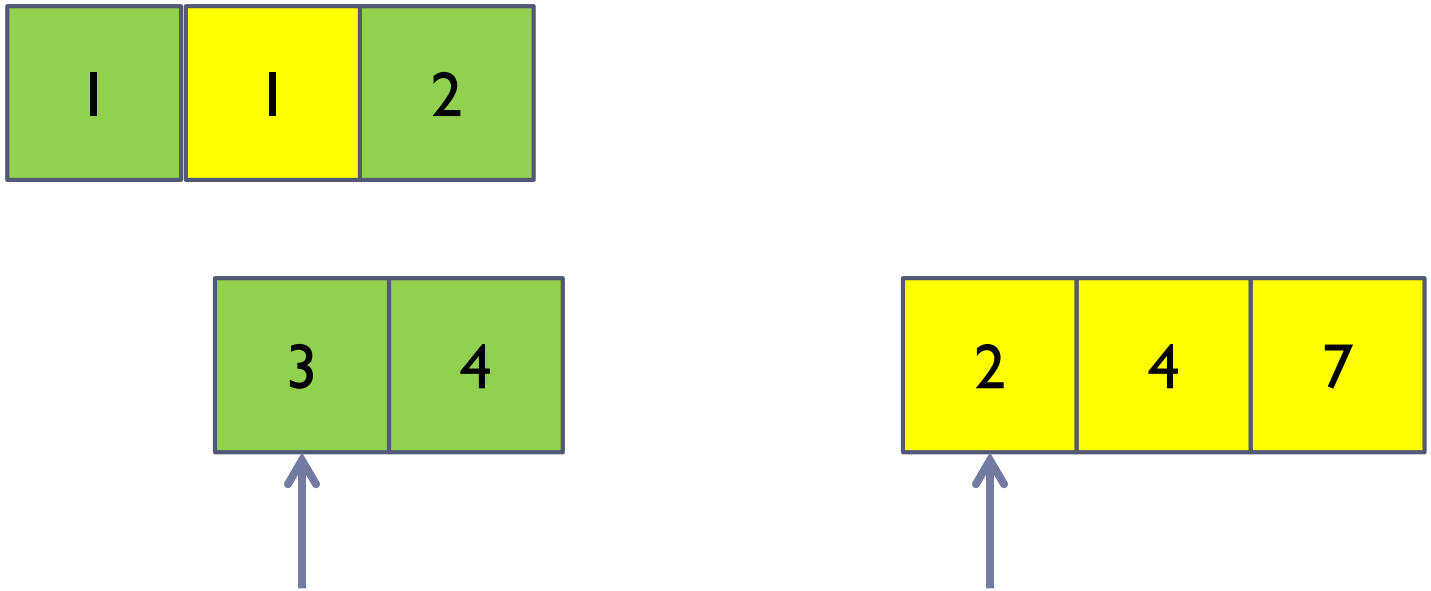
Merge Sort

- 3. 다시 하나로 합친다



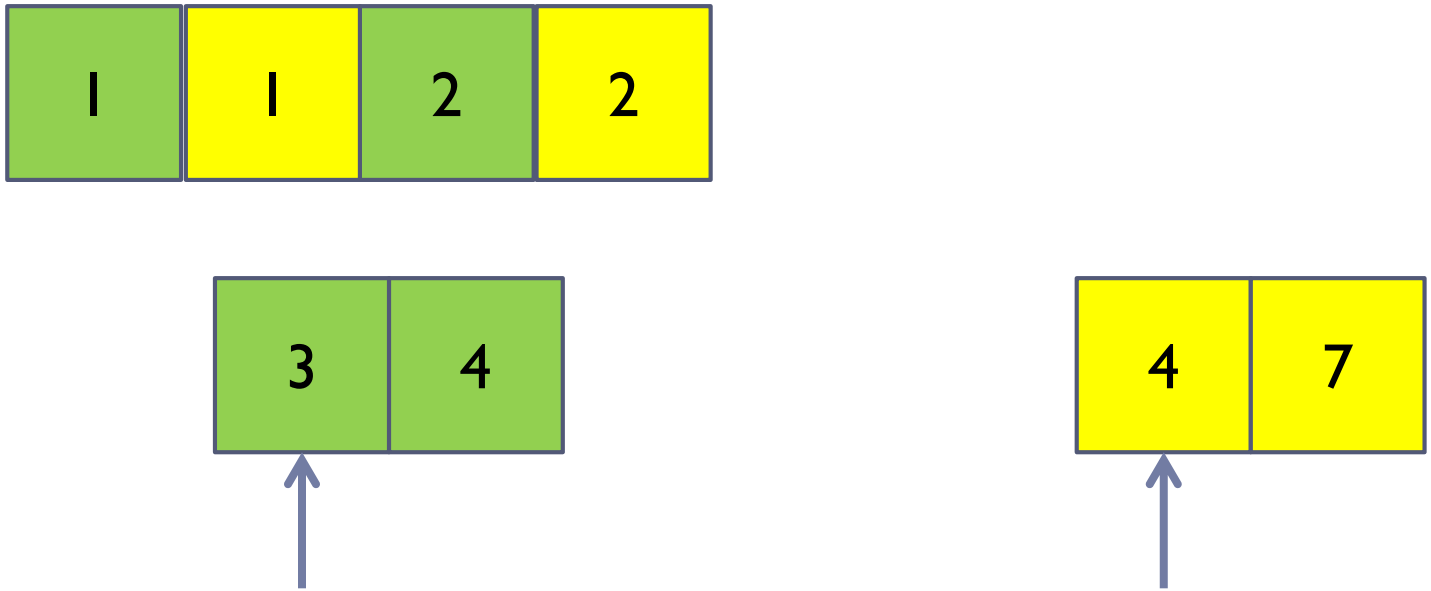
Merge Sort

- 3. 다시 하나로 합친다



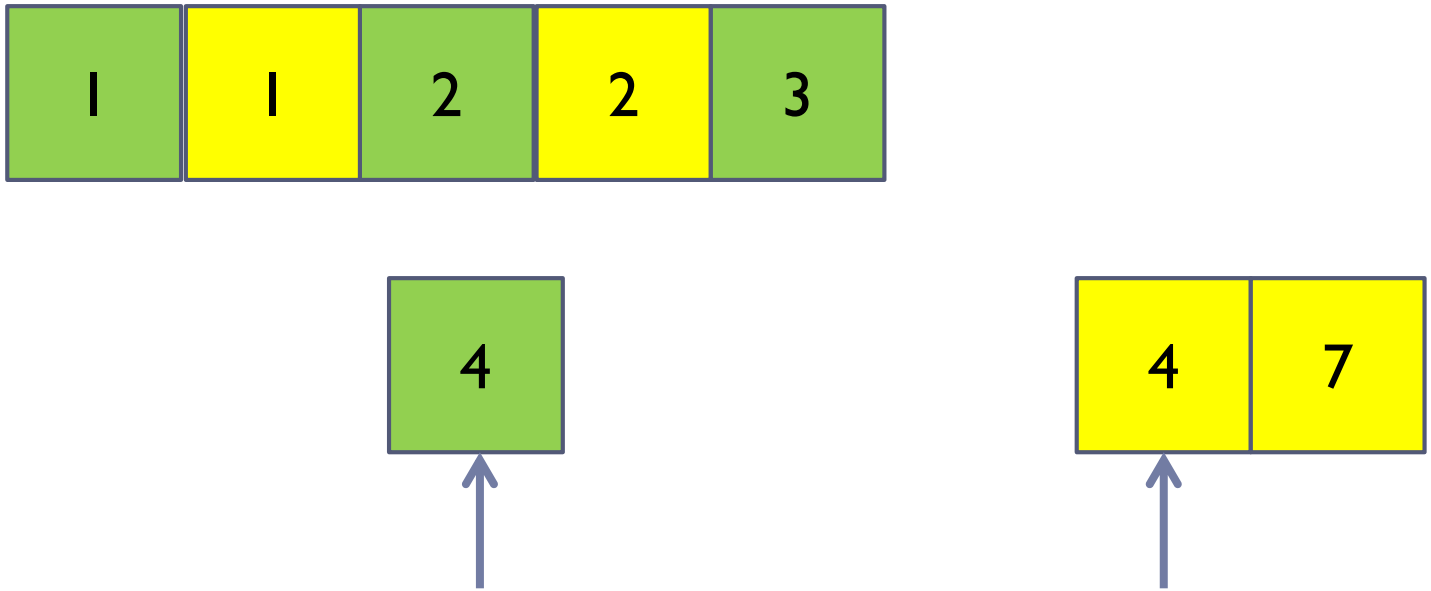
Merge Sort

- 3. 다시 하나로 합친다



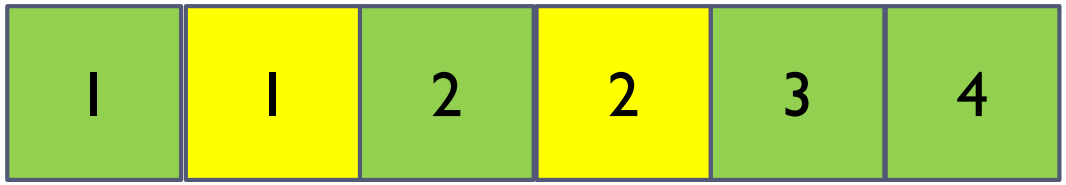
Merge Sort

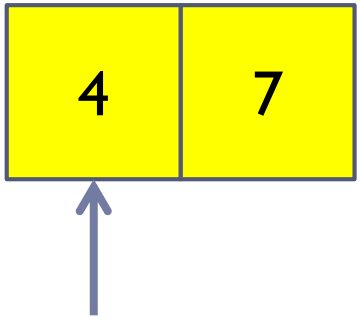
- 3. 다시 하나로 합친다



Merge Sort

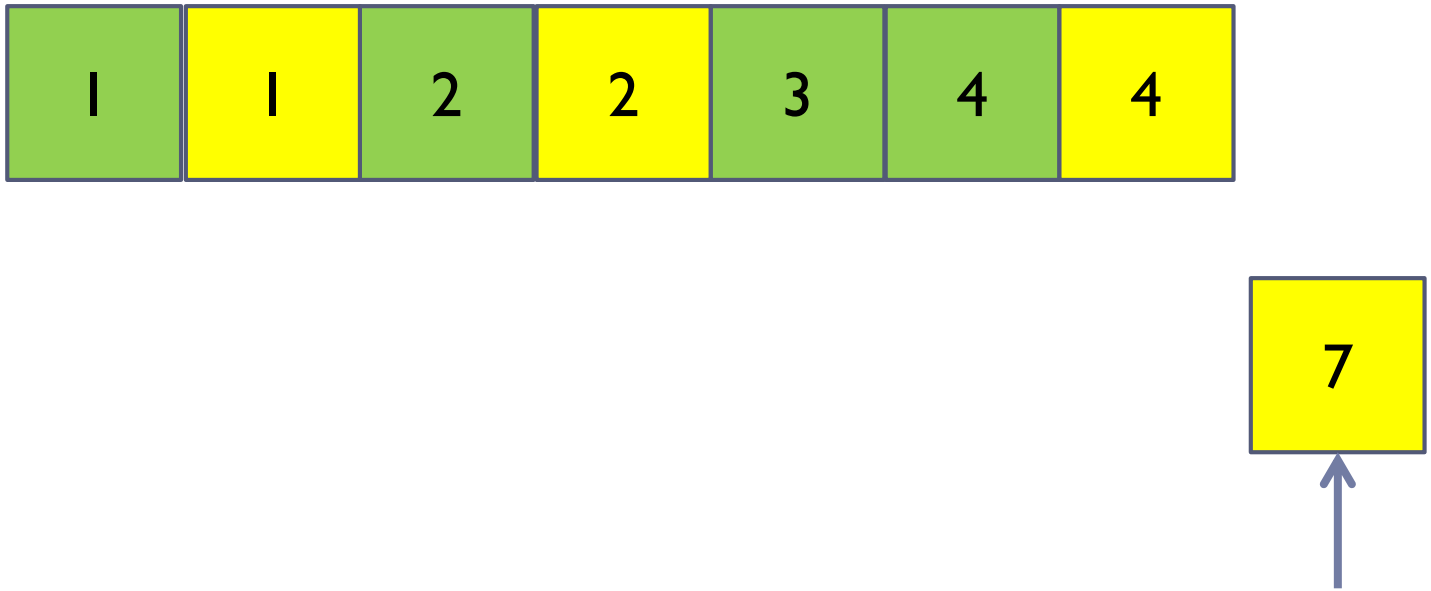
- 3. 다시 하나로 합친다





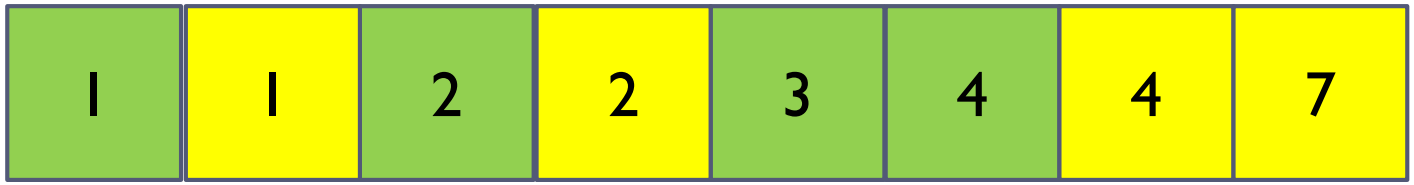
Merge Sort

- 3. 다시 하나로 합친다



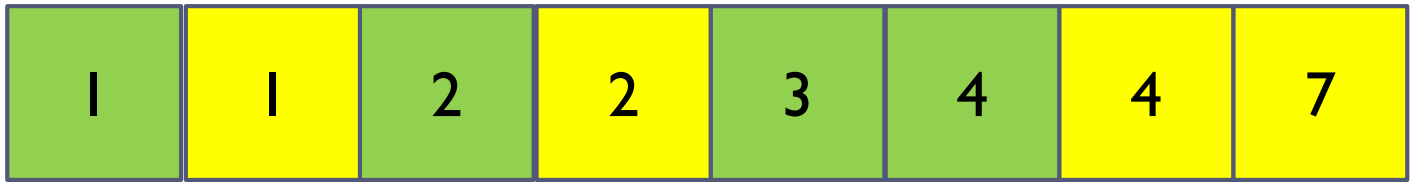
Merge Sort

- 3. 다시 하나로 합친다



Merge Sort

- 3. 다시 하나로 합친다 (!!!)



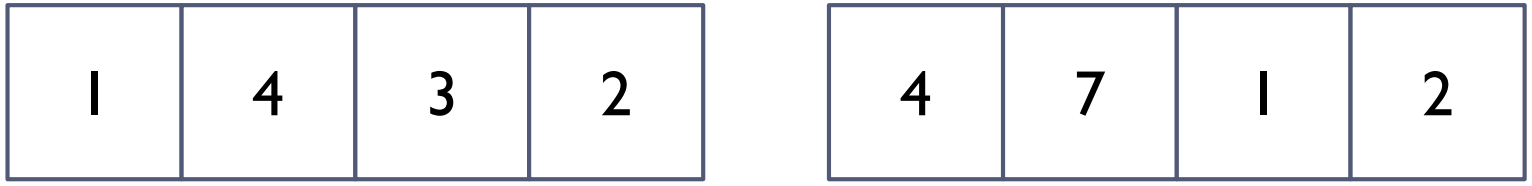
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!

1	4	3	2	4	7	1	2

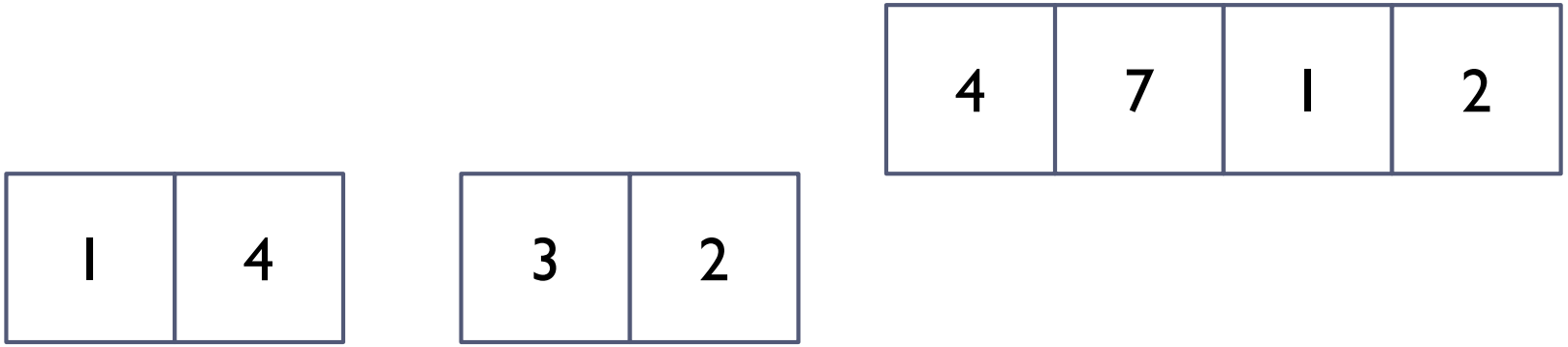
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



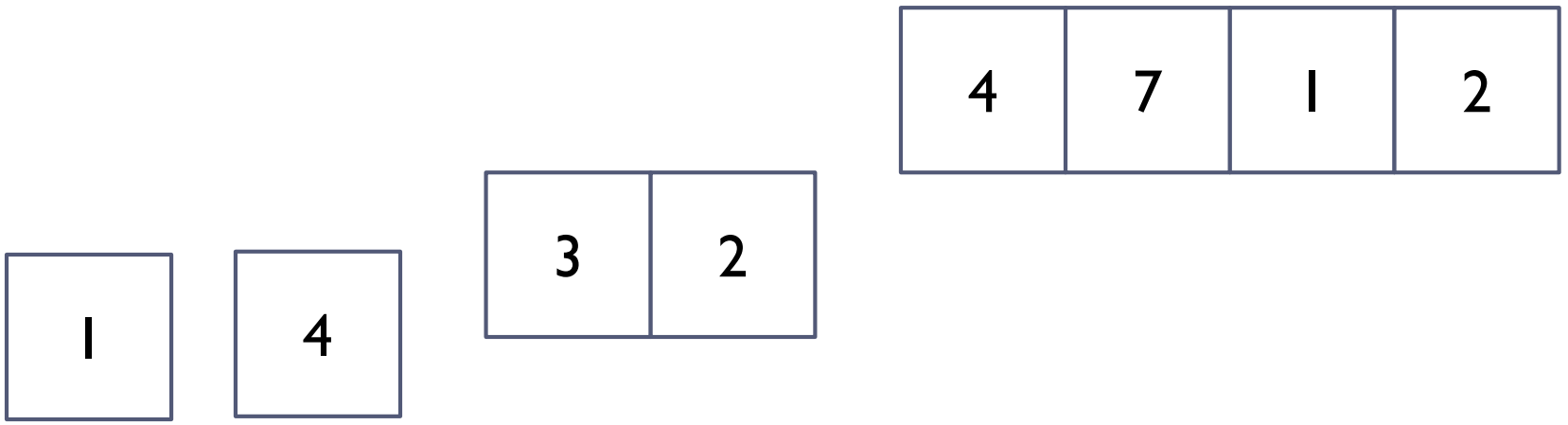
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



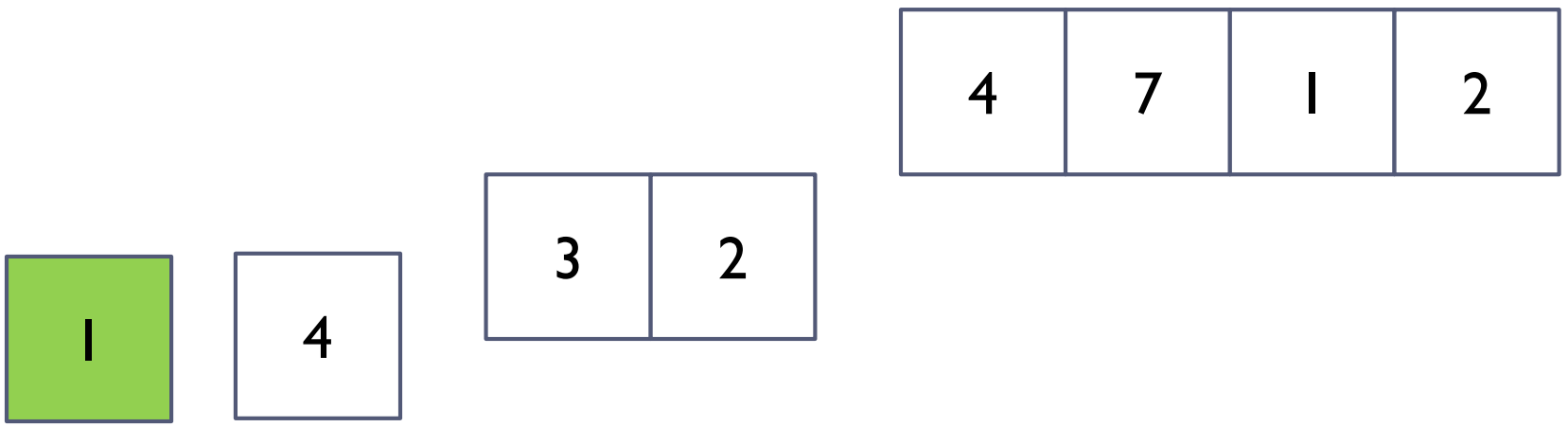
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



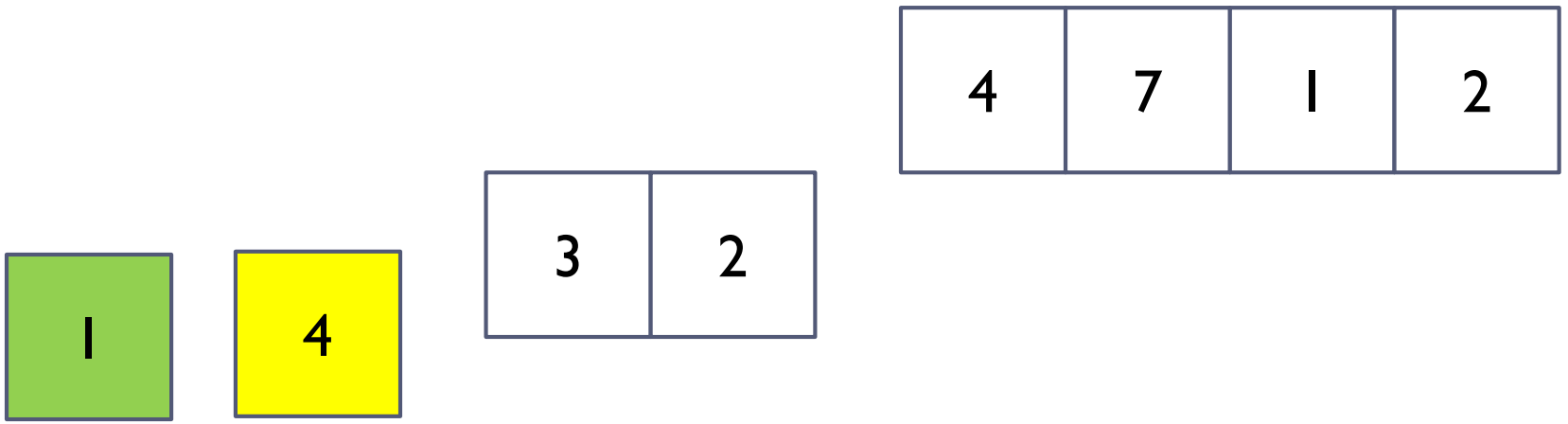
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



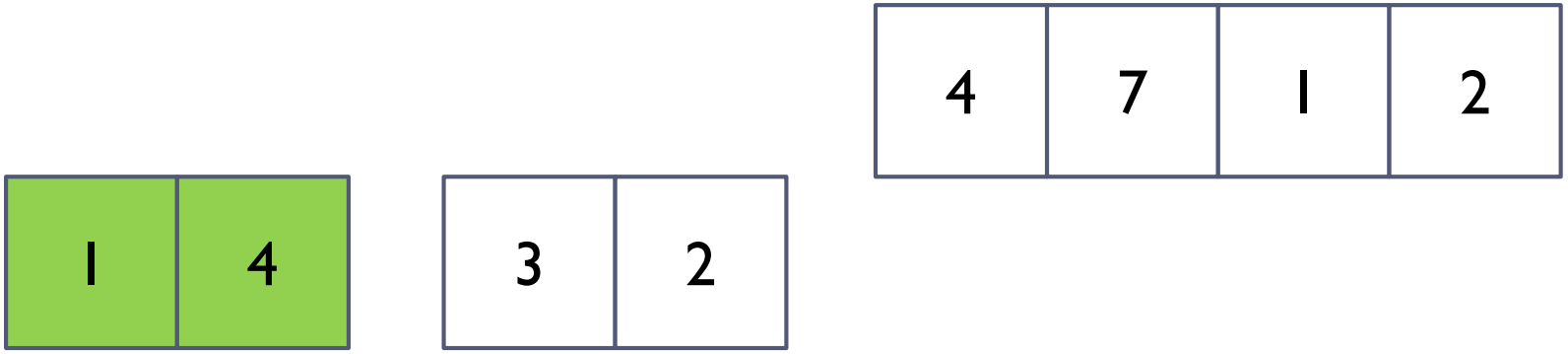
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



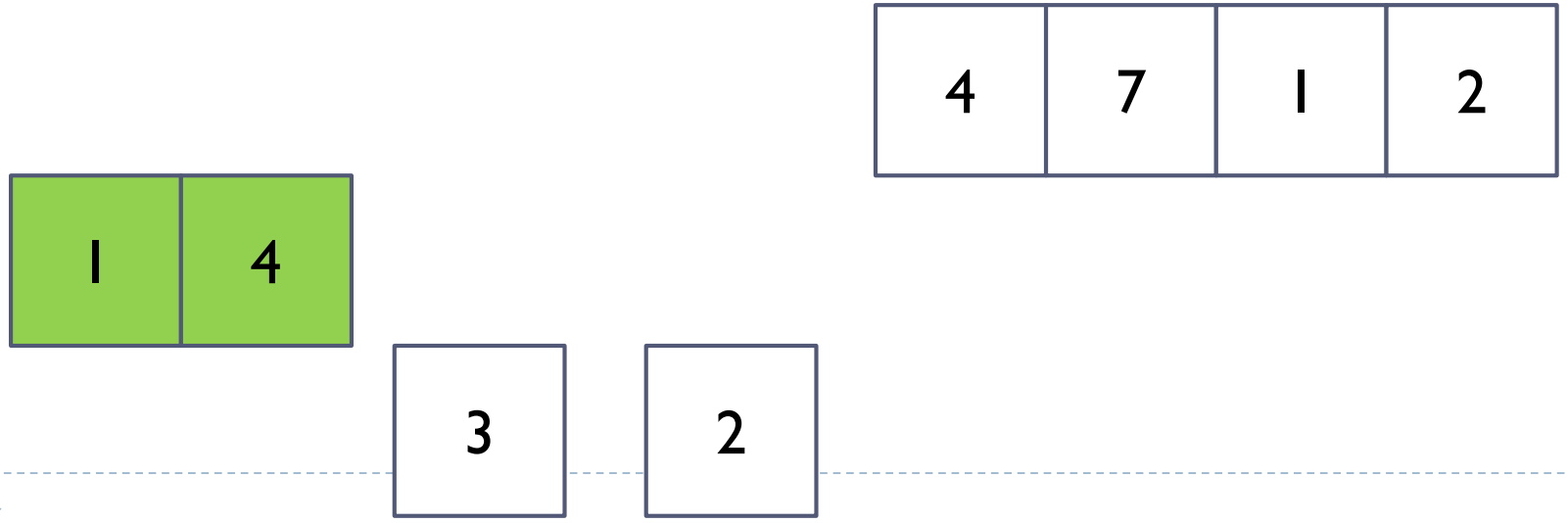
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



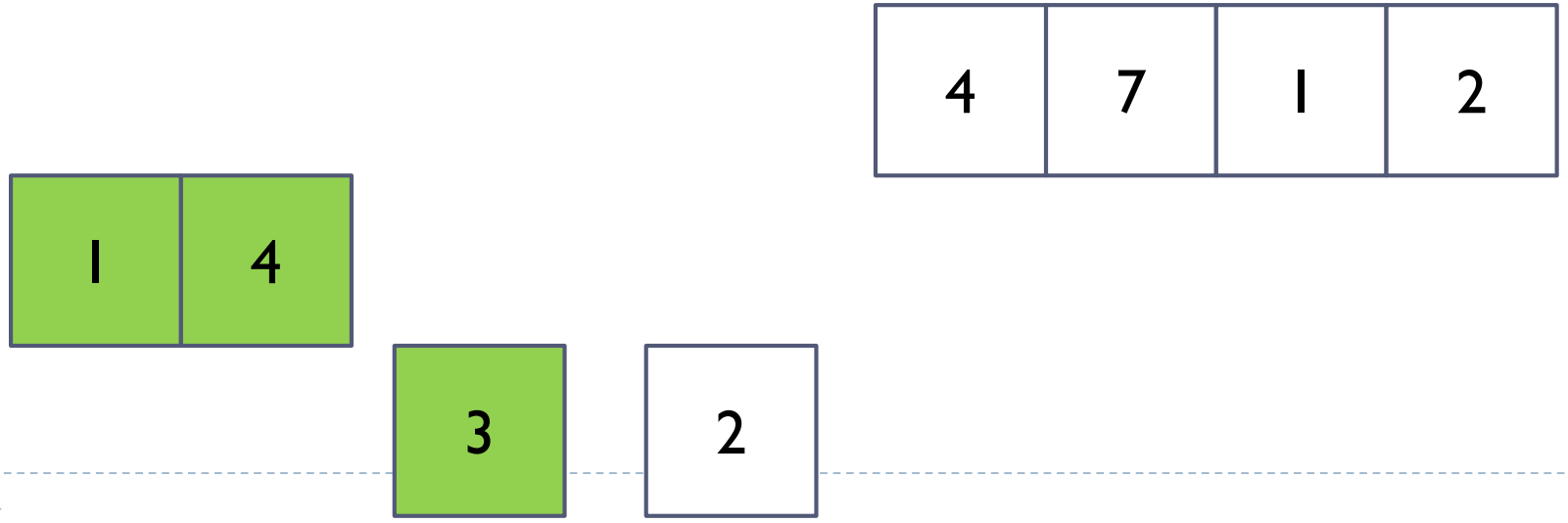
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



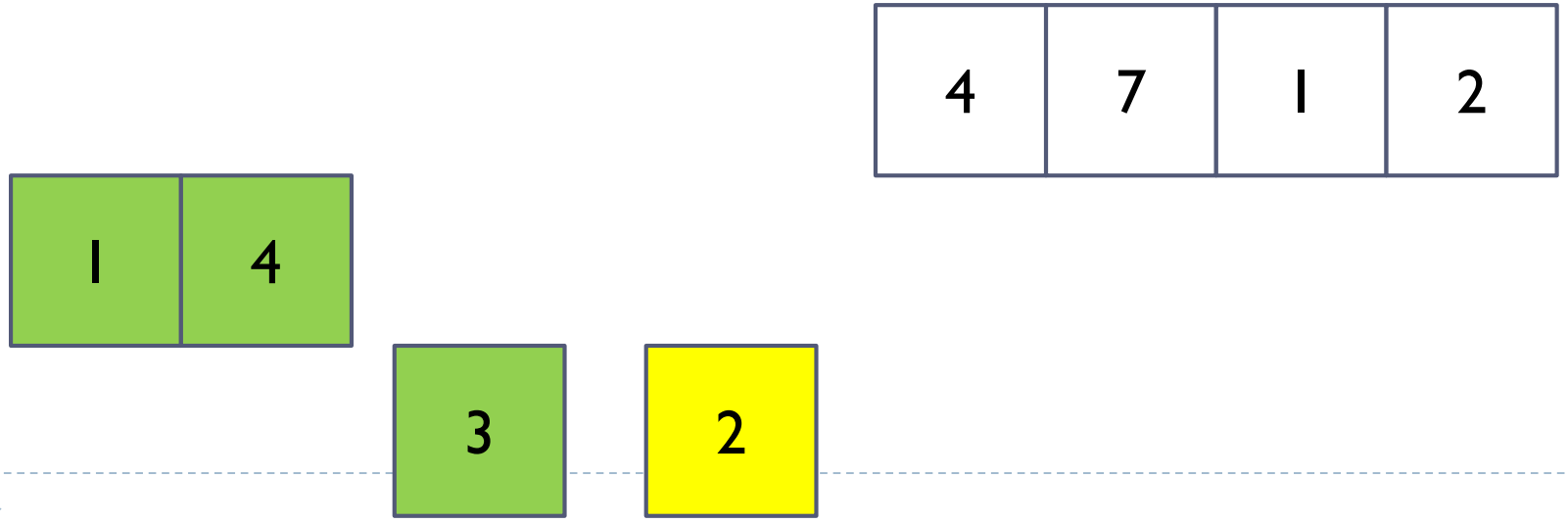
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



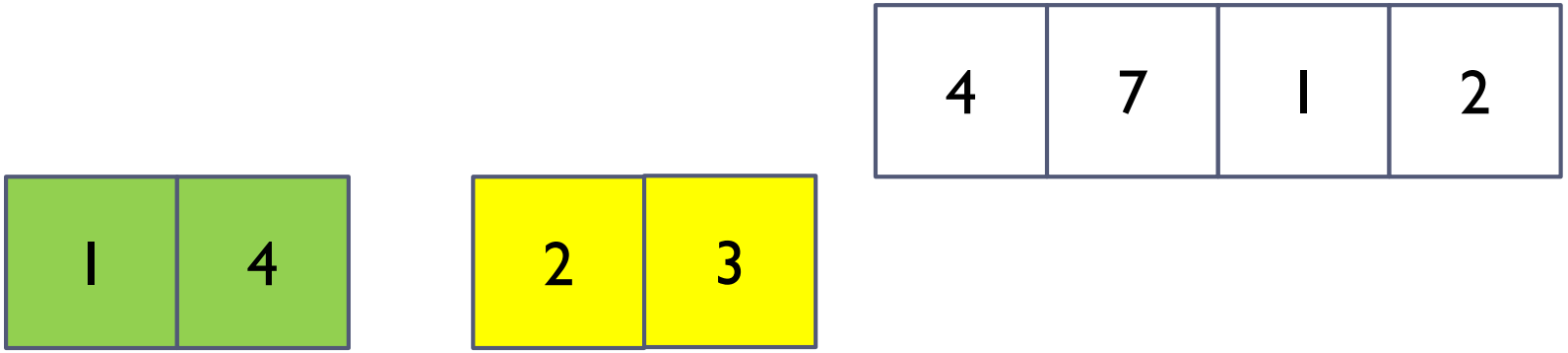
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



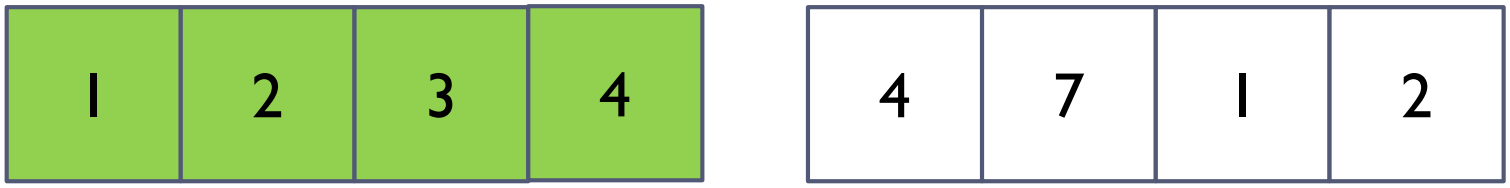
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



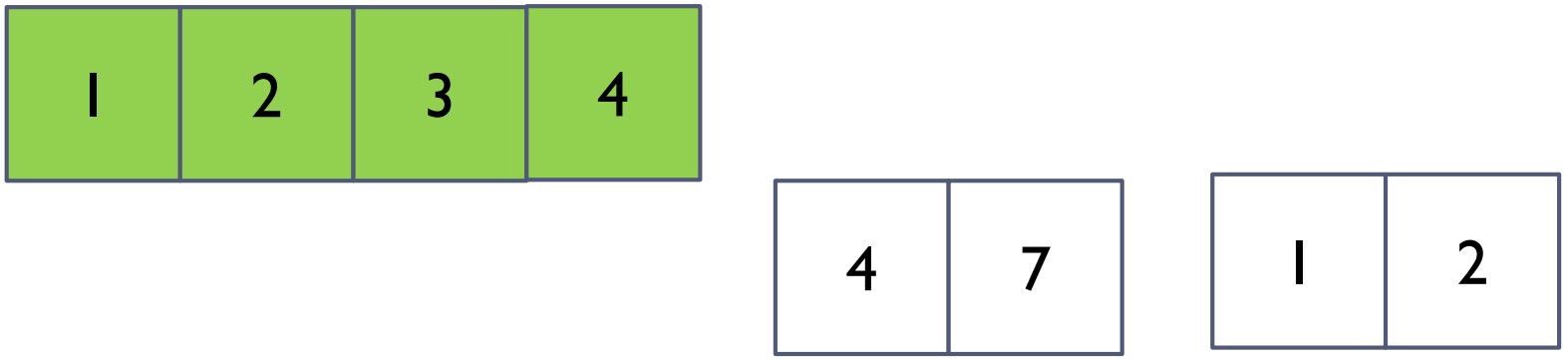
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



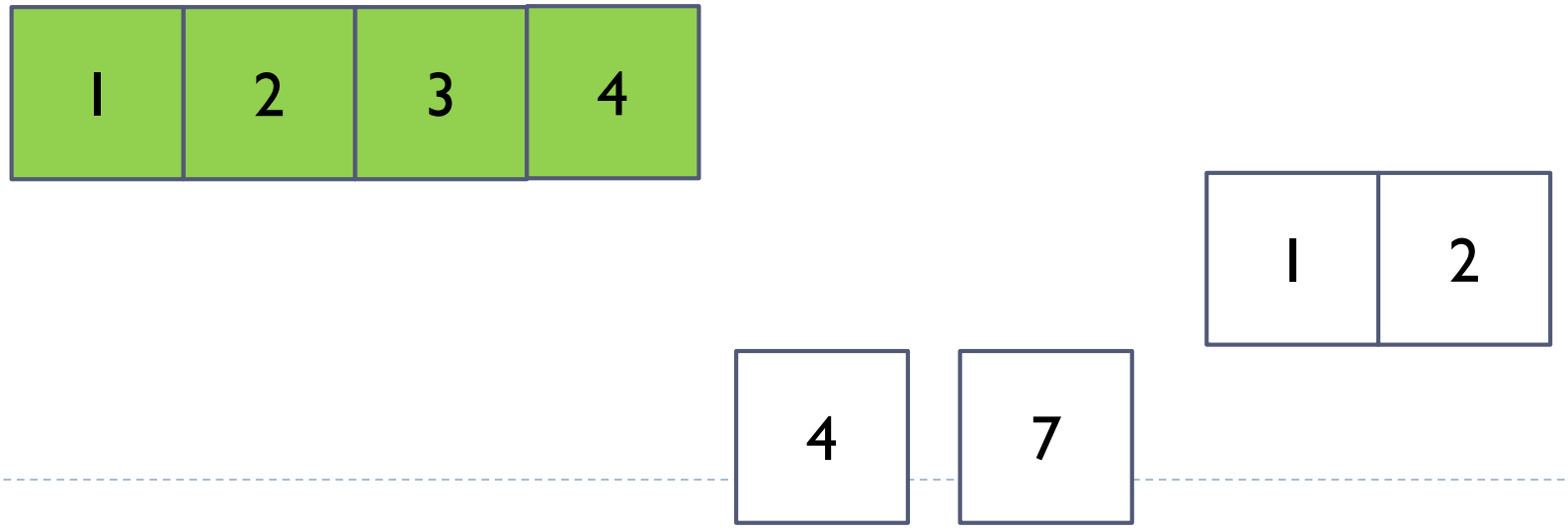
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



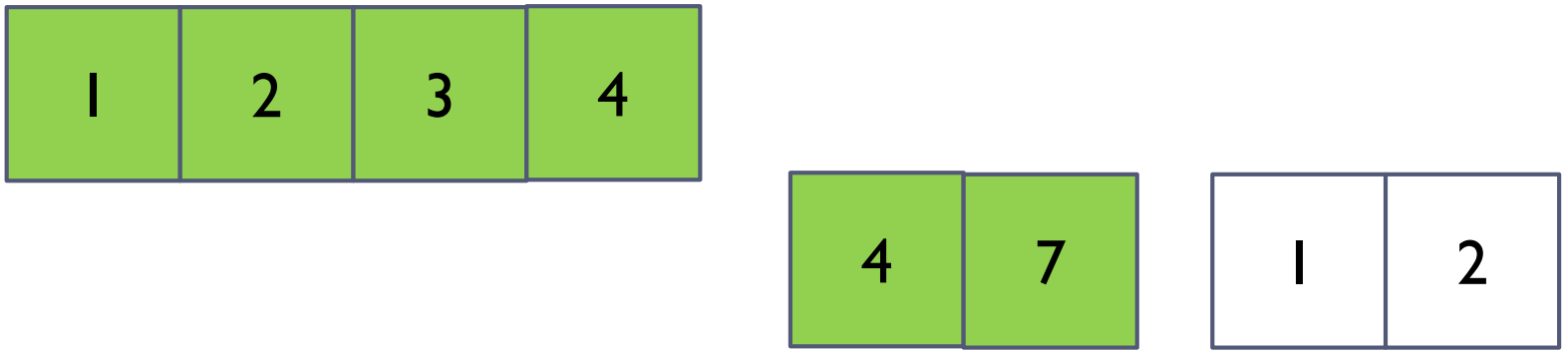
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



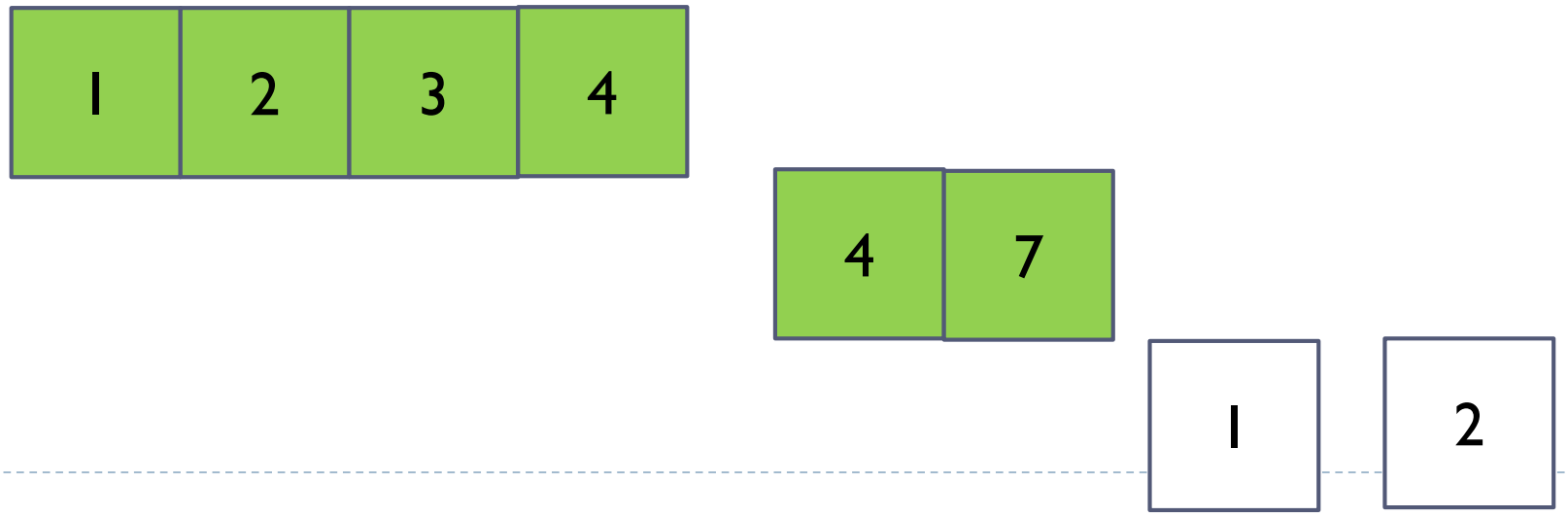
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



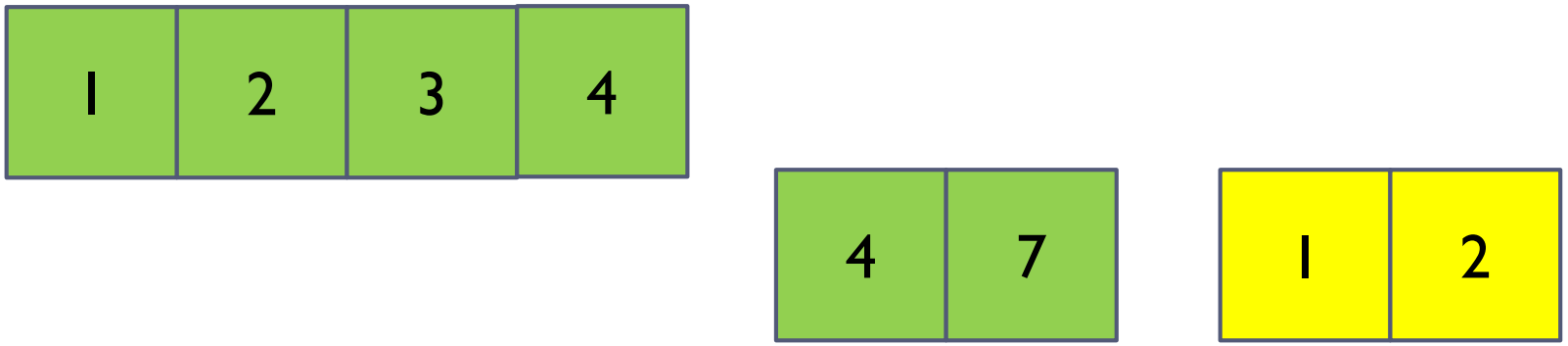
Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



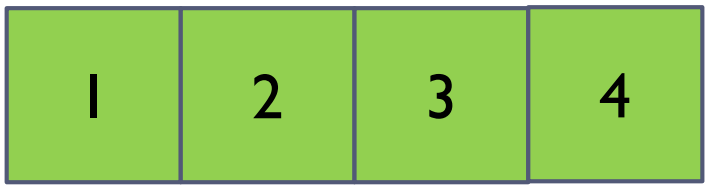
Merge Sort

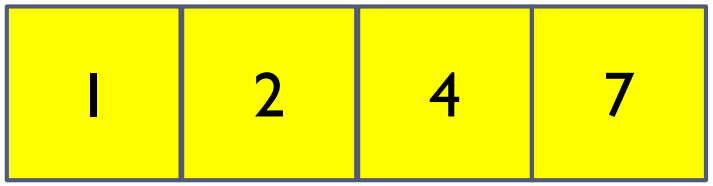
- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!



Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!





Merge Sort

- 각각의 부분을 어떻게든 정렬한다??
  - ➔ 또 다시 Merge Sort로 정렬!

Divide & Conquer

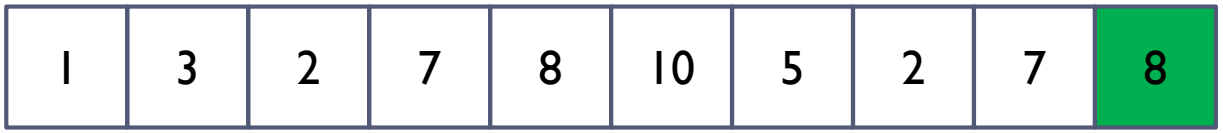
```
 4:  int n, m;
 5:  int Data[30];
 6:  void MergeSort(int start, int end){
 7:      if(start >= end) return;
 8:      int mid = (start + end) / 2;
 9:      MergeSort(start, mid);
10:      MergeSort(mid+1, end);
11:      // ---------------- merging  --------------
12:      int temp[30] = {0,}, temp_len = 0; // temporary for merging
13:      int p1 = start, p2 = mid+1;
14:      while(p1 <= mid && p2 <= end){
15:          if(Data[p1] > Data[p2]){
16:              temp[temp_len++] = Data[p2];
17:              p2++;
18:          }
19:          else{
20:              temp[temp_len++] = Data[p1];
21:              p1++;
22:          }
23:      }
24:      if(p1 >= mid+1){ // right array has element
25:          while(p2 <= end){
26:              temp[temp_len++] = Data[p2];
27:              p2++;
28:          }
29:      }
30:      else if(p2 >= end+1){ // left array has element
31:          while(p1 <= mid){
32:              temp[temp_len++] = Data[p1];
33:              p1++;
34:          }
35:      }
36:      for(int i=start,j=0;i<=end;i++,j++) Data[i] = temp[j];
37:      return;
38:  }
```

Divide & Conquer

```
39:  int main(){
40:      int n;
41:      scanf("%d",&n);
42:      for(int i=1;i<=n;i++) scanf("%d",&Data[i]);
43:      MergeSort(1, n);
44:      for(int i=1;i<=n;i++) printf("%d ",Data[i]);
45:      printf("₩n");
46:      return 0;
47:  }
```

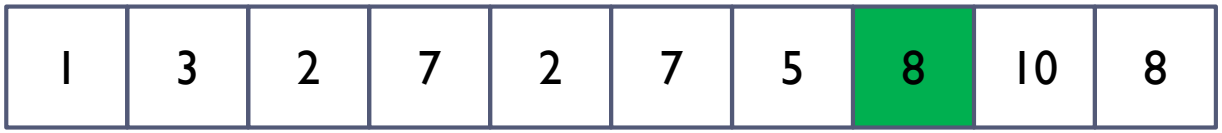
Divide & Conquer

- Merge Sort, Quick Sort가 대표적인 D&C.
  - Quick Sort 역시 마찬가지!



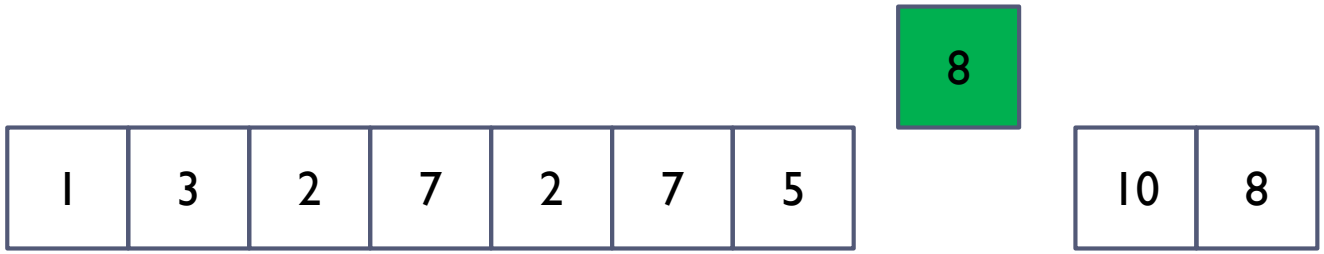
Divide & Conquer

- Merge Sort, Quick Sort가 대표적인 D&C.
  - Quick Sort 역시 마찬가지!



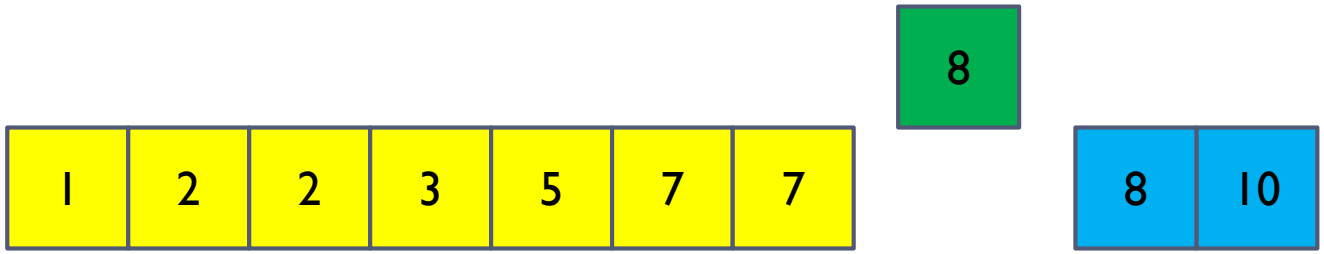
Divide & Conquer

- Merge Sort, Quick Sort가 대표적인 D&C.
  - Quick Sort 역시 마찬가지!



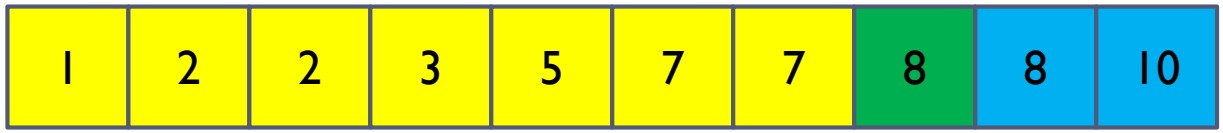
Divide & Conquer

- Merge Sort, Quick Sort가 대표적인 D&C.
  - Quick Sort 역시 마찬가지!



Divide & Conquer

- Merge Sort, Quick Sort가 대표적인 D&C.
  - Quick Sort 역시 마찬가지!



Divide & Conquer

- Merge Sort, Quick Sort가 대표
  - Quick Sort 역시 마찬가지!

1	2	2	3	5	7	7	8	8	10

Divide & Conquer

- 문제를 나누어서 부분문제를 해결 후 전체문제 해결!
  - 부분 부분을 정렬한 후 전체 정렬
- D&C is **very** hard
  - D&C라고 판단하기가 어렵고
  - 판단을 하고 난 후에는 모델링이 난감합니다
  - For example,

Divide & Conquer

- 2차원에서 비교 불가능한 쌍의 개수를 찾아라
  - 비교가능? (x1, y1) < (x2, y2) iff x1 < x2 & y1 < y2
  - (1, 9) (7, 10) (4, 1) (3, 4) (5, 3) (6, 5) (2, 3) 라고 하자
  - 쉽게 하기 위해서 중복된 x좌표는 없다고 하자

- Indexed Tree를 사용하면 간단.
- Divide & Conquer로 해결!

Divide & Conquer

1 9	7 10	4 1	3 4	5 3	6 5	2 3

Divide & Conquer

1 9	2 3	3 4	4 1	5 3	6 5	7 10

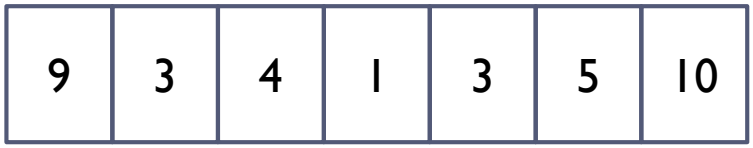
x순으로 정렬 (stable sort)

Divide & Conquer

1	2	3	4	5	6	7
9	3	4	1	3	5	10

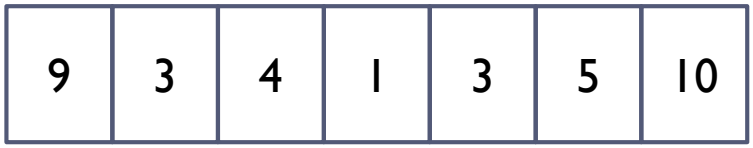
이제 x를 제외하고 y만 가지면,

Divide & Conquer



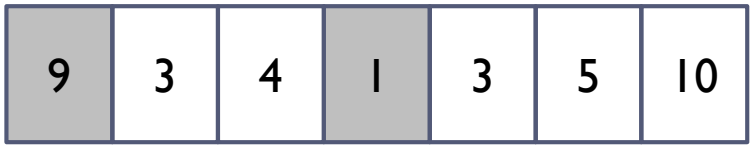
이제 x를 제외하고 y만 가지면,

Divide & Conquer



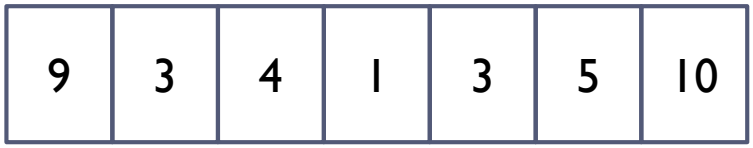
자신보다 앞에 있는 숫자들보다 작으면 안됨!

Divide & Conquer



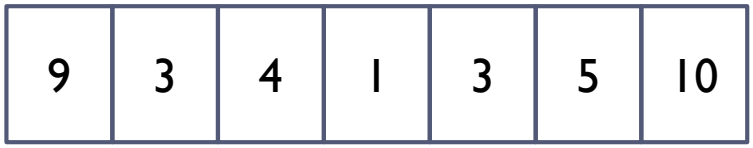
자신보다 앞에 있는 숫자들보다 작으면 안됨!

Divide & Conquer



이러한 쌍의 개수를 모두 찾자

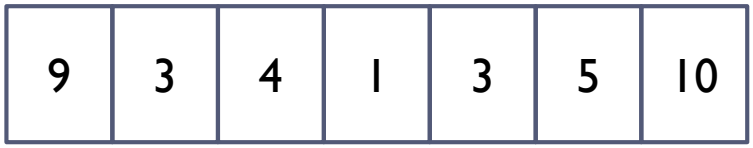
Divide & Conquer



I. 나누어서 각각의 쌍의 개수를 모두 찾는다

II. 합치면서 새로 발생하는 쌍의 개수를 찾는다

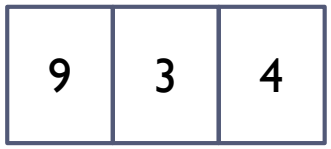
Divide & Conquer

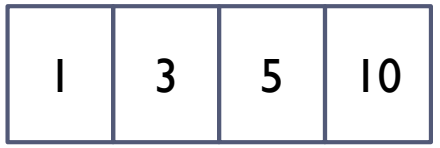


I. 나누어서 각각의 쌍의 개수를 모두 찾는다

II. 합치면서 새로 발생하는 쌍의 개수를 찾는다

Divide & Conquer



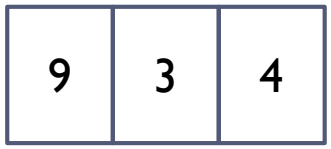


I. 나누어서 각각의 쌍의 개수를 모두 찾는다

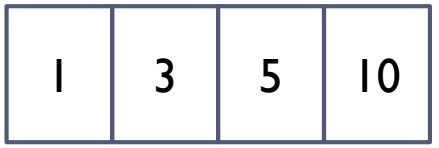
II. 합치면서 새로 발생하는 쌍의 개수를 찾는다

Divide & Conquer

2개



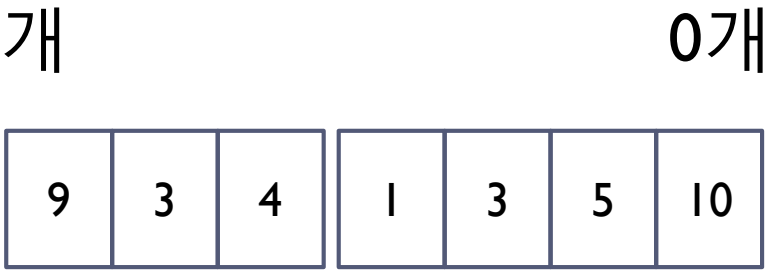
0개



I. 나누어서 각각의 쌍의 개수를 모두 찾는다

II. 합치면서 새로 발생하는 쌍의 개수를 찾는다

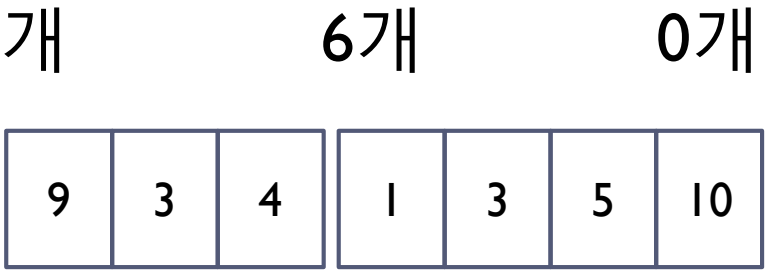
Divide & Conquer



I. 나누어서 각각의 쌍의 개수를 모두 찾는다

II. 합치면서 새로 발생하는 쌍의 개수를 찾는다

Divide & Conquer

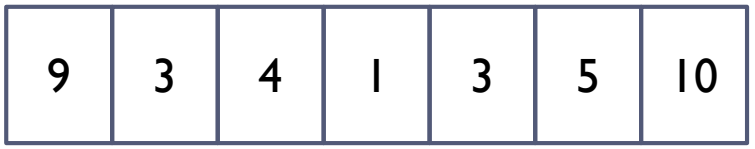


I. 나누어서 각각의 쌍의 개수를 모두 찾는다

II. 합치면서 새로 발생하는 쌍의 개수를 찾는다

Divide & Conquer

= 8개!

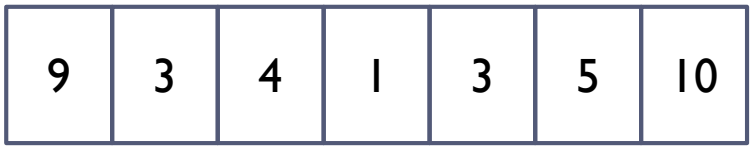


I. 나누어서 각각의 쌍의 개수를 모두 찾는다

II. 합치면서 새로 발생하는 쌍의 개수를 찾는다

Divide & Conquer

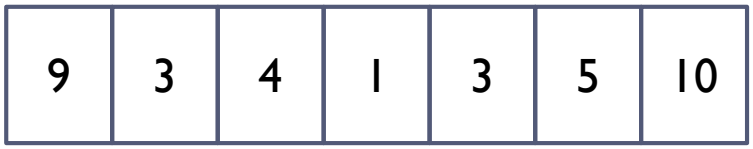
I. 은 단순히 나누어 재귀호출을 하면 됨



I. 나누어서 각각의 쌍의 개수를 모두 찾는다
II. 합치면서 새로 발생하는 쌍의 개수를 찾는다

Divide & Conquer

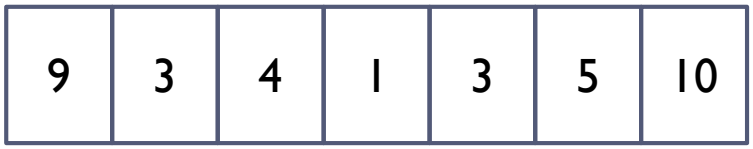
합치면서 새로 발생하는 것은 어떻게 찾나?



I. 나누어서 각각의 쌍의 개수를 모두 찾는다
II. 합치면서 새로 발생하는 쌍의 개수를 찾는다

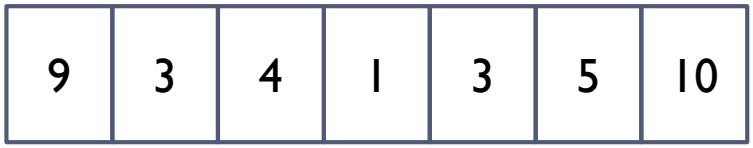
Divide & Conquer

정렬이 되어있다면?

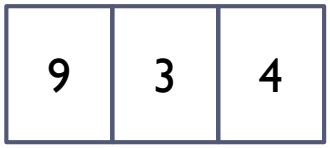


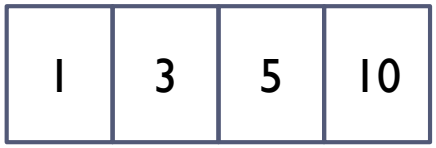
I. 나누어서 각각의 쌍의 개수를 모두 찾는다
II. 합치면서 새로 발생하는 쌍의 개수를 찾는다

Divide & Conquer



Divide & Conquer





Divide & Conquer

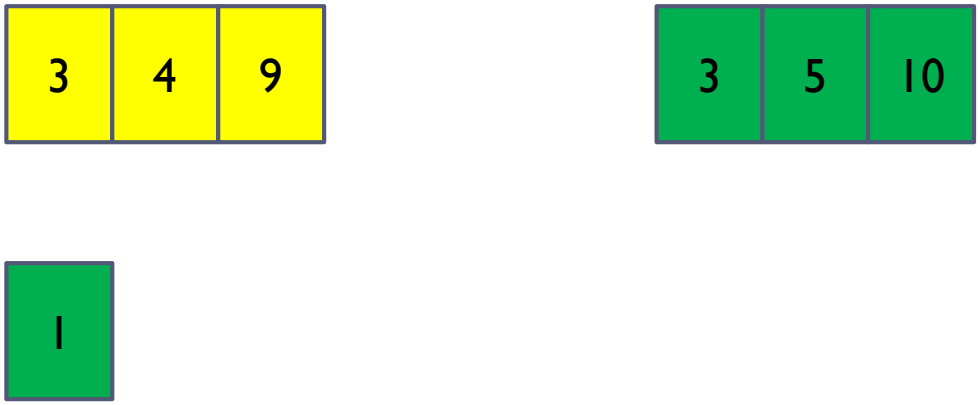
Divide & Conquer

왼쪽과 오른쪽이 정렬이 되어있다.

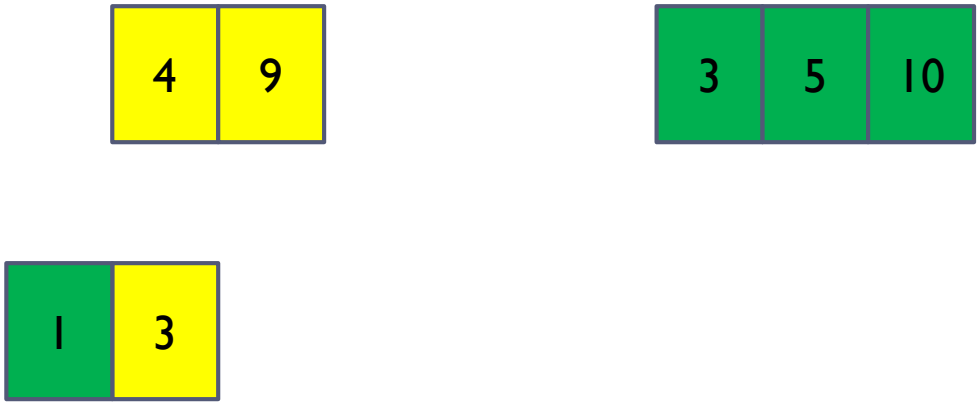
이제 Merging을 하면?

Divide & Conquer

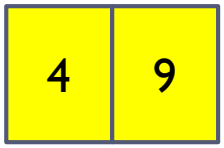
Divide & Conquer

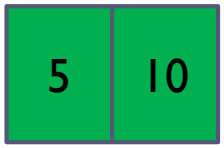


Divide & Conquer

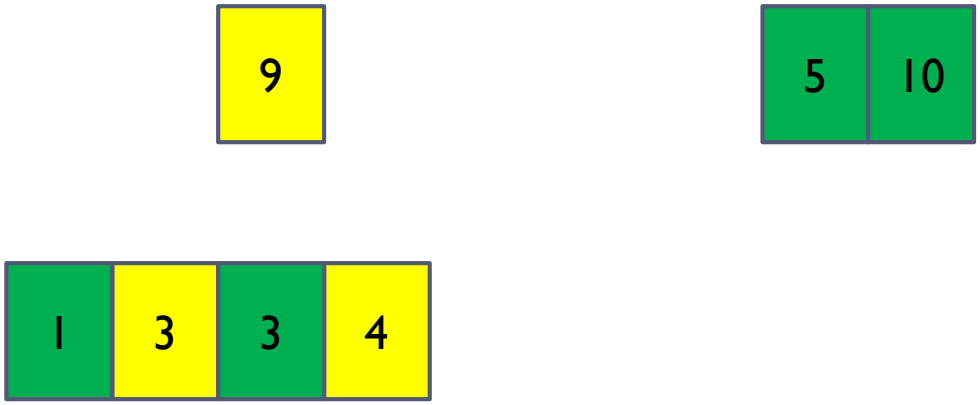


Divide & Conquer

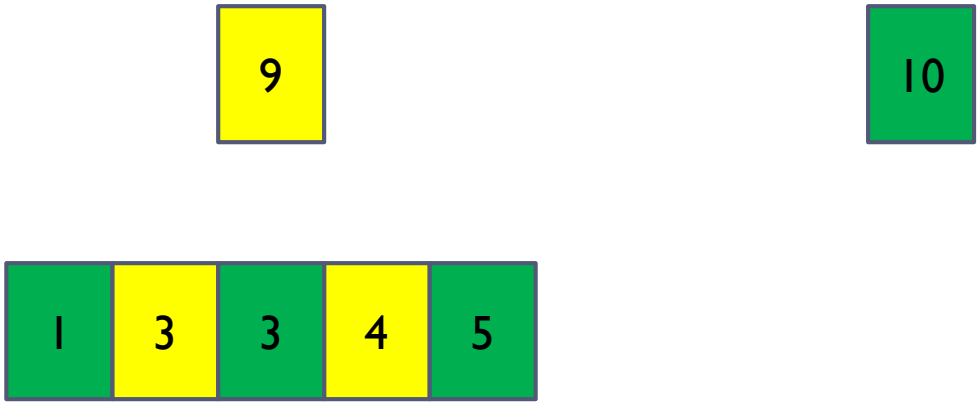




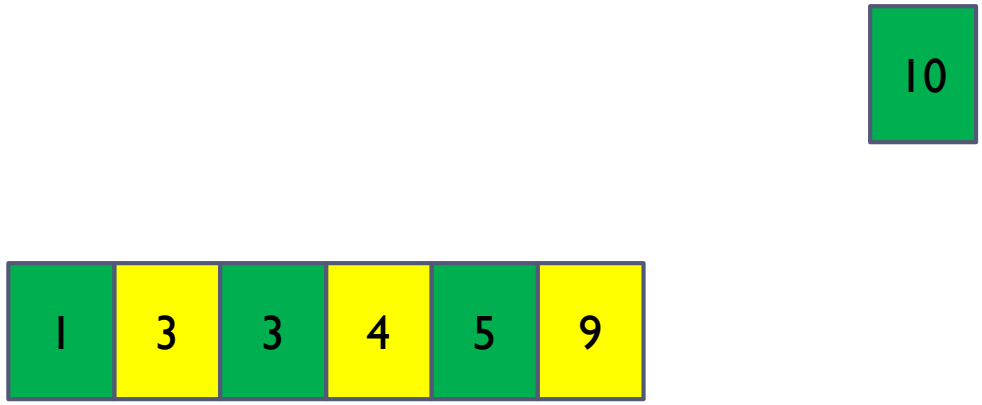
Divide & Conquer



Divide & Conquer



Divide & Conquer



Divide & Conquer

Divide & Conquer

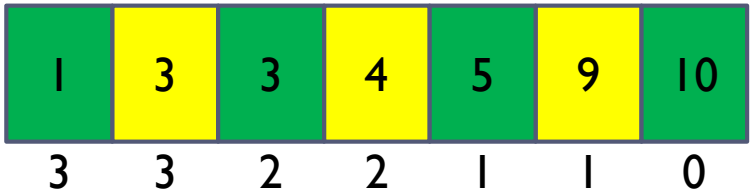
초록색은 오른쪽, 노란색은 왼쪽 묶음

초록색이 앞에 있으면 안됨!

Divide & Conquer

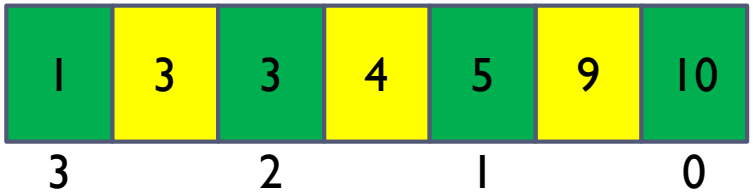
초록색 입장에서 내 뒤에 노란색이 몇 개나 있나?

Divide & Conquer



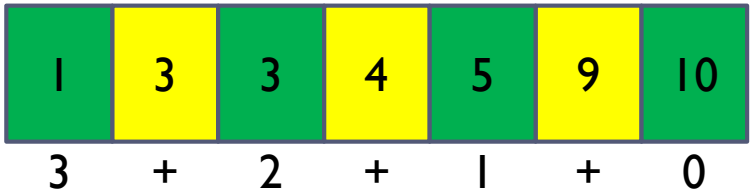
초록색 입장에서 내 뒤에 노란색이 몇 개나 있나?

Divide & Conquer



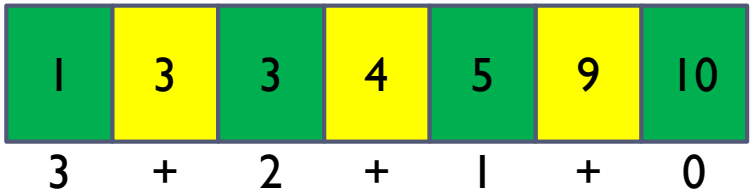
초록색 입장에서 내 뒤에 노란색이 몇 개나 있나?

Divide & Conquer



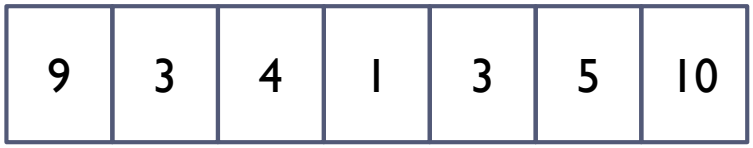
초록색 입장에서 내 뒤에 노란색이 몇 개나 있나?

Divide & Conquer



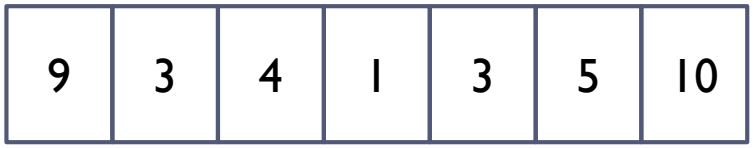
합치면서 발생하는 쌍의 수가 6개!

Divide & Conquer



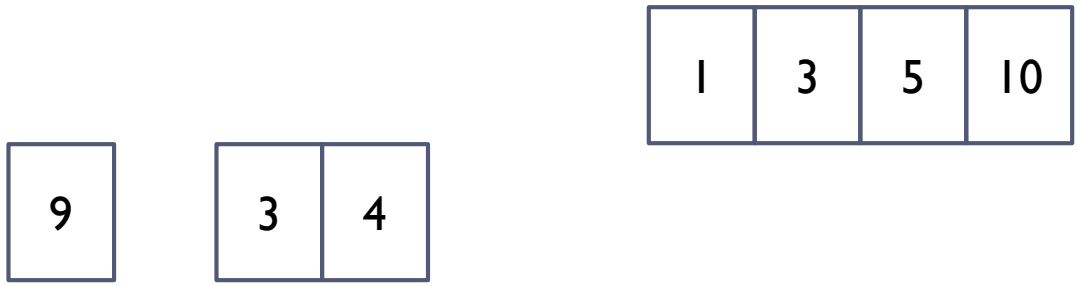
Overall procedure

Divide & Conquer

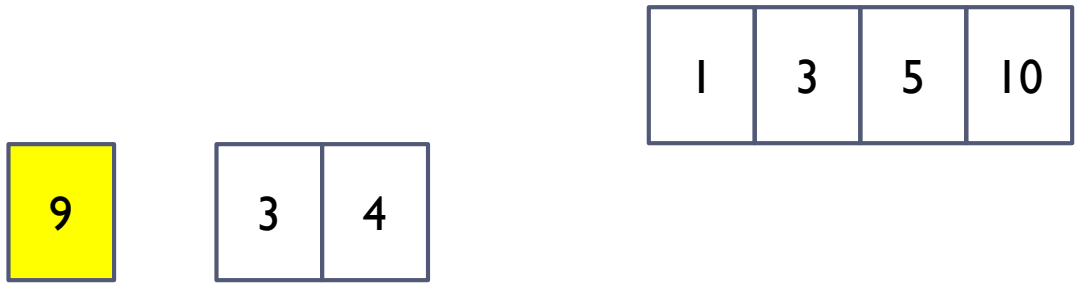


Divide & Conquer

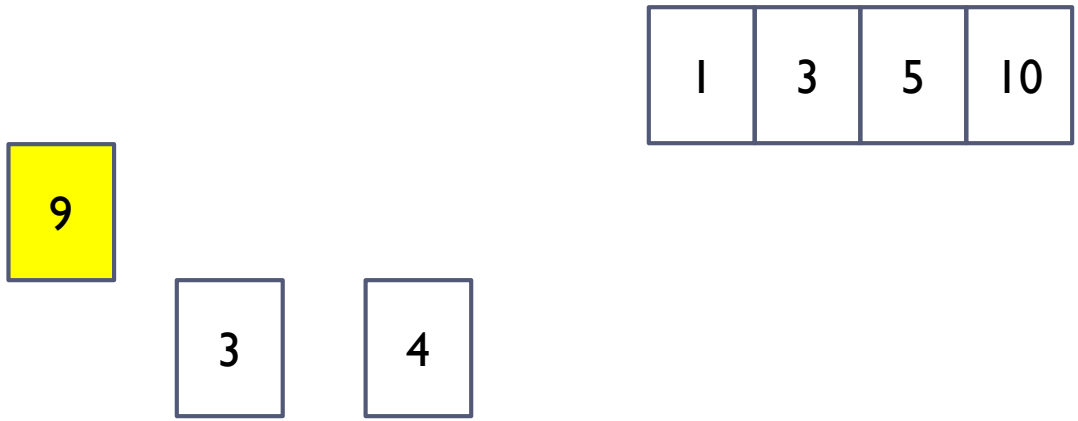
Divide & Conquer



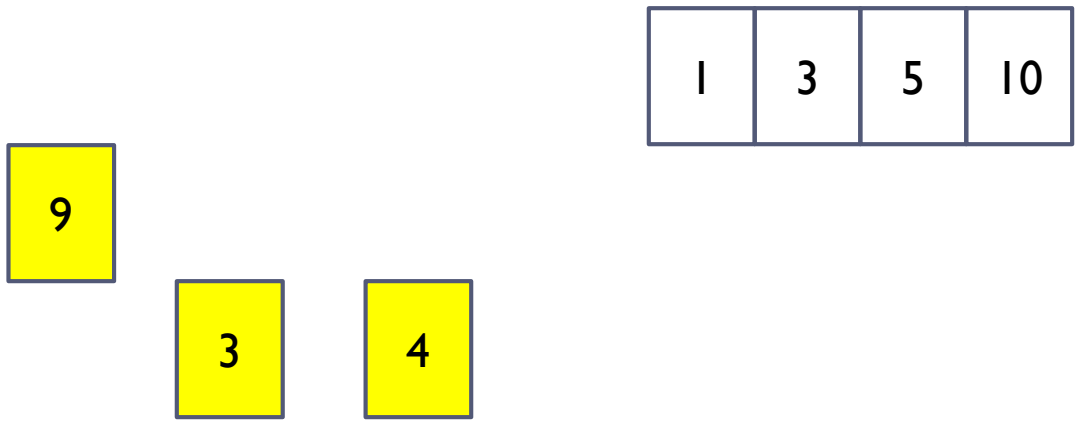
Divide & Conquer



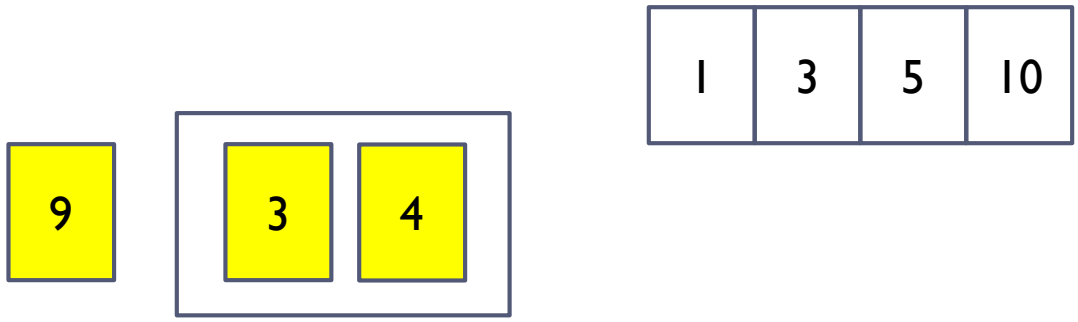
Divide & Conquer



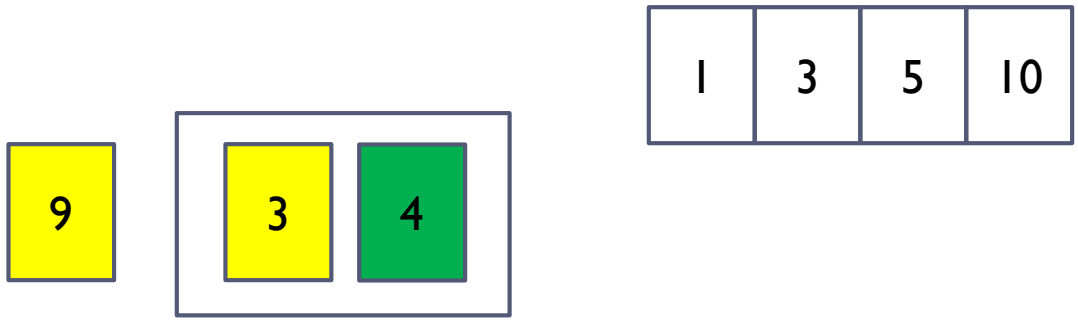
Divide & Conquer



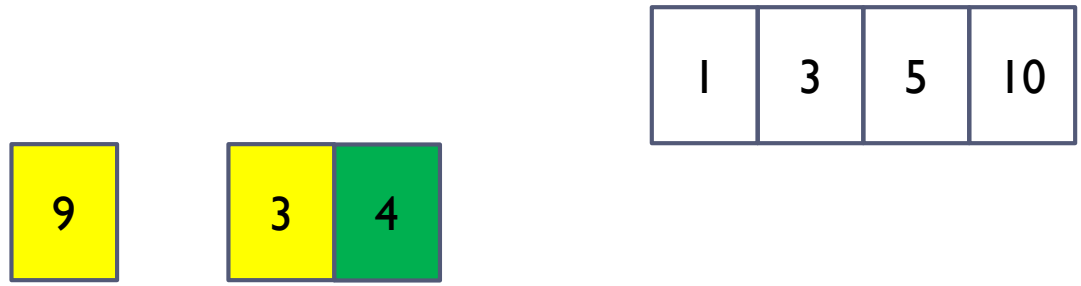
Divide & Conquer



Divide & Conquer

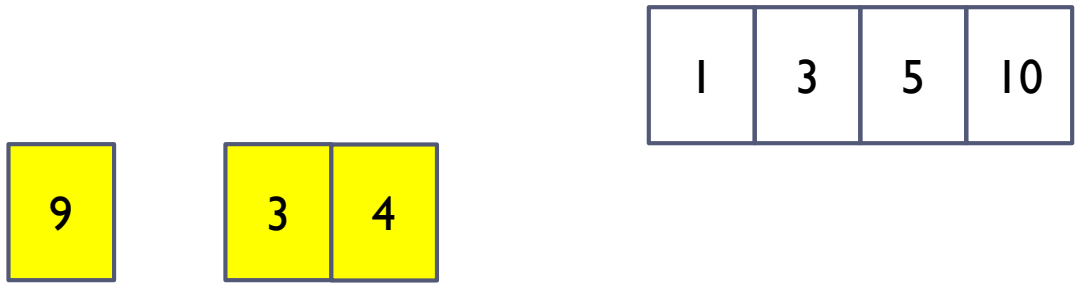


Divide & Conquer



0

Divide & Conquer



0

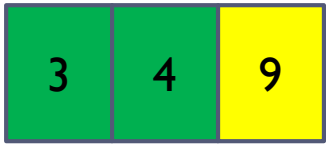
Divide & Conquer

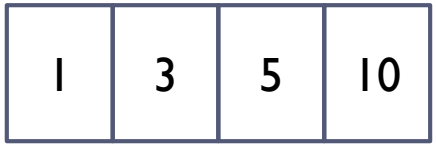
0

Divide & Conquer

0

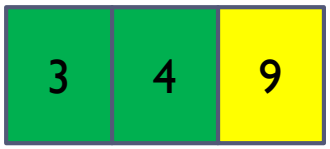
Divide & Conquer

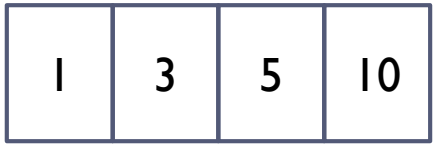




0

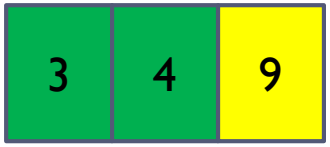
Divide & Conquer

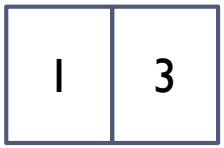


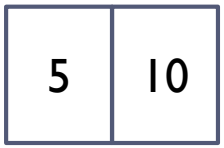


0 + 2

Divide & Conquer

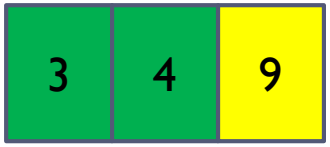


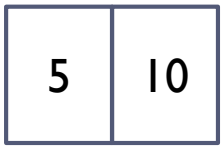




0 + 2

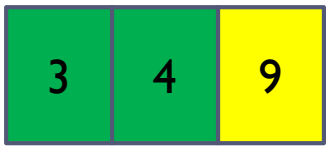
Divide & Conquer

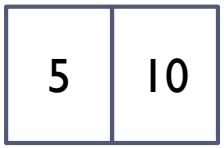


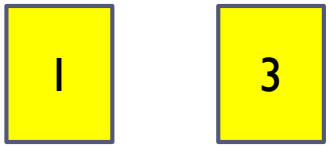


0 + 2

Divide & Conquer

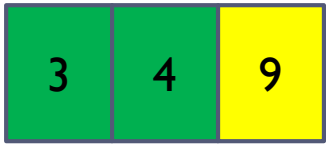


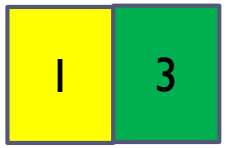


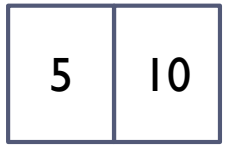


0 + 2

Divide & Conquer

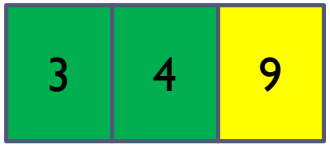


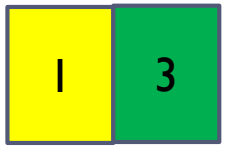


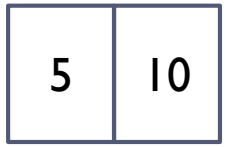


0 + 2

Divide & Conquer

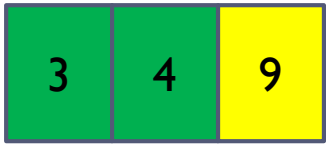


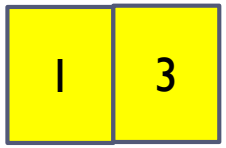


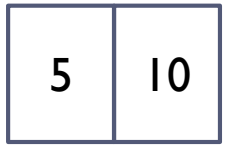


0 + 2 + 0

Divide & Conquer

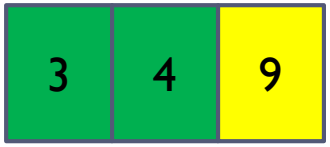


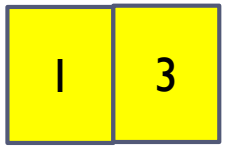


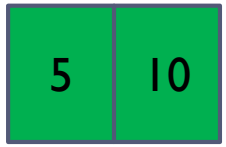


0 + 2 + 0

Divide & Conquer





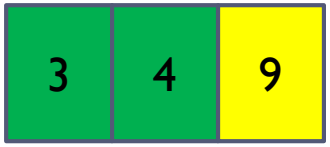


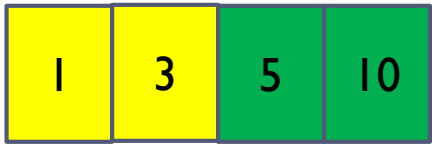
0 + 2 + 0 + 0

Divide & Conquer

0 + 2 + 0 + 0

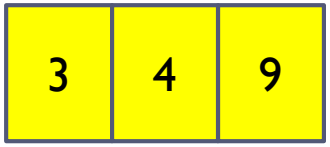
Divide & Conquer

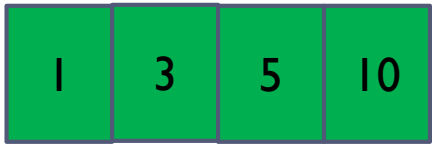




0 + 2 + 0 + 0 + 0

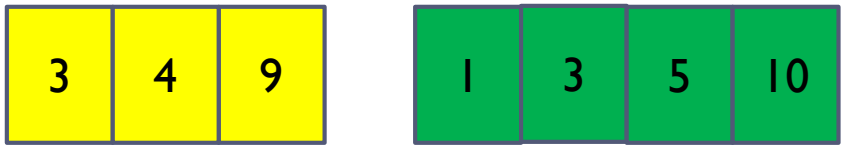
Divide & Conquer





0 + 2 + 0 + 0 + 0

Divide & Conquer



0 + 2 + 0 + 0 + 0

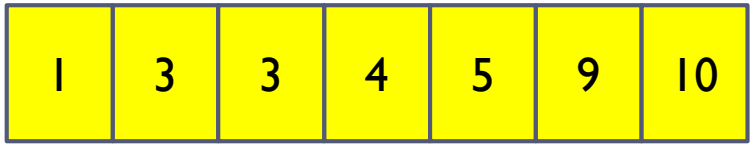
Divide & Conquer

0 + 2 + 0 + 0 + 0

Divide & Conquer

0 + 2 + 0 + 0 + 0 + 6

Divide & Conquer



0 + 2 + 0 + 0 + 0 + 6 = 8 !

Divide & Conquer

- 3차원도 D&C로 풀릴 것입니다. (예전에 푼 기억이…)
- D&C IS HARD!!
  - 근데 그만큼 D&C로 어려운 문제 내기도 힘듭니다
  - 대표적인 D&C 문제들은 꼭 알아둡시다

Divide & Conquer

- Example
  - N을 입력받아 1 2 3 … (N-1) N (N-1) … 3 2 1 출력
    - input = 4 → output = 1 2 3 4 3 2 1
    - int Print (int s int n) : The output starts from s and peak point is n
  - 1D array에서 최솟값 찾기
    - int Min(int array, int index) := The minimum value among array[0] ~ array[index]
  - N개의 수를 입력 받아 연속부분 최대합을 출력
    - 함수 정의는 알아서 생각해보시길
  - Hanoi's Tower
    - 판의 개수를 입력 받아 옮기는 과정을 출력한다
    - void Hanoi(int n, char a, char b, char c)
      - 첫 번째 기둥의 이름이 a, 두 번째 기둥의 이름이 b, 세 번째 기둥의 이름이 c이며, n개의 판이 a에 있을 때 옮기는 과정을 출력하는 함수