# 분할 정복

190521(화) - 전현승

경북대학교 알고리즘 문제해결 동아리 GORI

# 목차

- 분할 정복?
- 분할 정복의 응용
- 분할 정복의 장단점
- 예시 : 머지 소트
- 11728 – 배열 합치기
- 예시 : 수의 N제곱
- 1629 – 곱셈
- 번외 : 행렬 제곱
- 2447 – 별 찍기 - 10

# 분할 정복?

- Divide & Conquer
- **분할** + **정복**!

- 그대로 해결할 수 없는 문제를 **작고 간단하며 유형이 비슷한 문제로 나눠서** 문제 해결 (**분할**)
- 부분문제의 답을 합쳐서 전체 문제의 정답을 구해야 할 때도 있음 (**정복**)

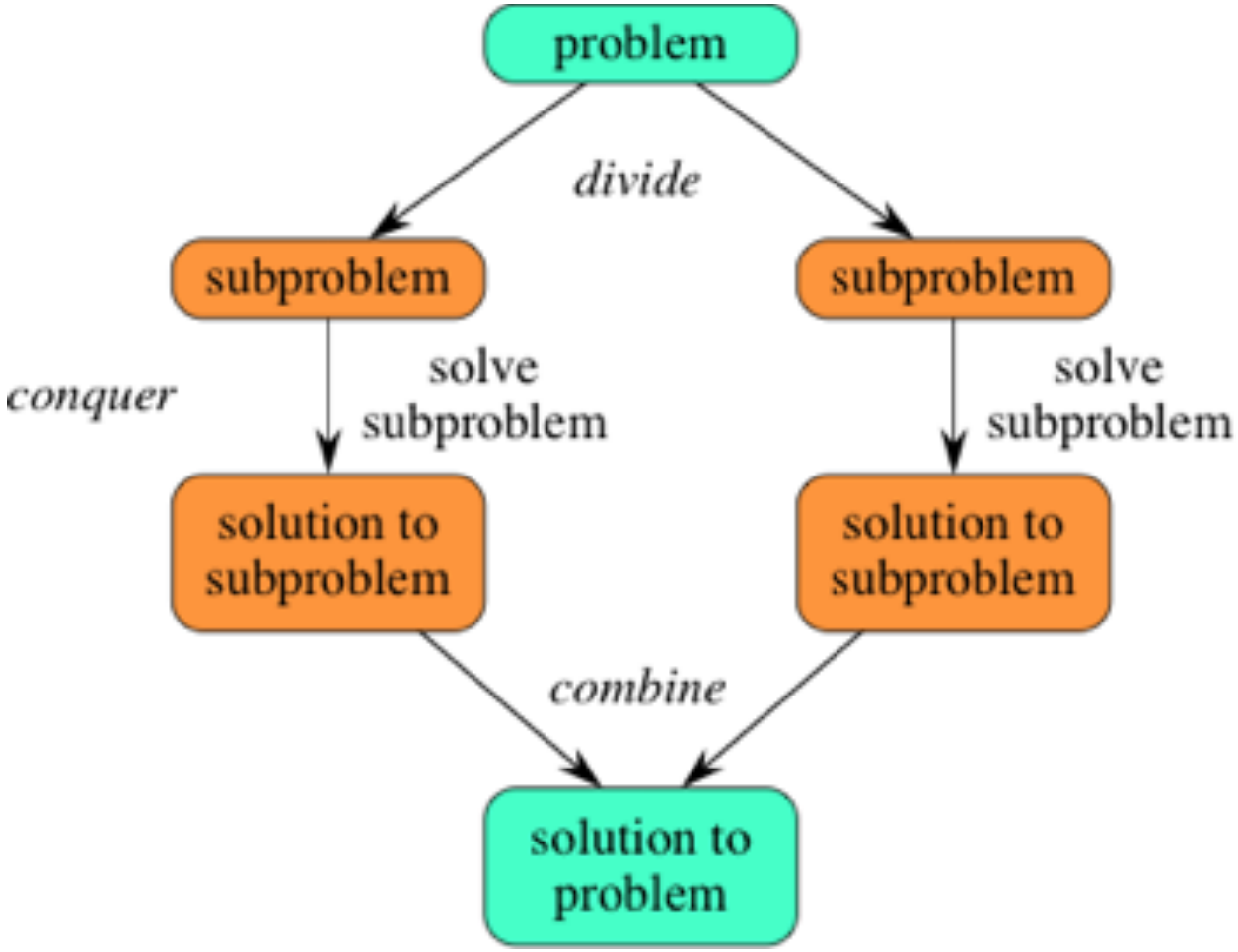
- 보통 재귀함수 이용

# 분할 정복?

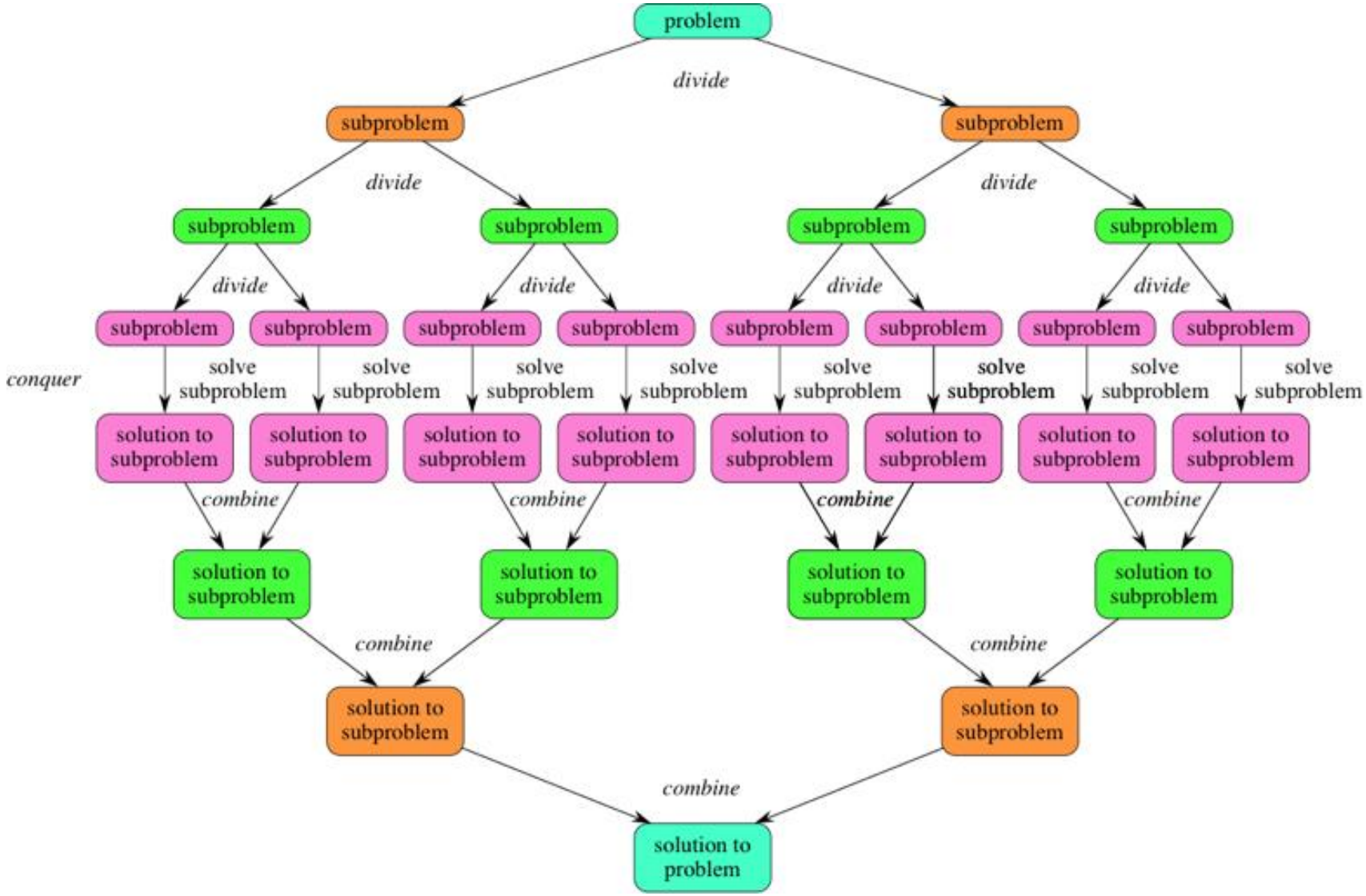
- … 작고 간단하며 유형이 비슷한 문제로 나눠서 …
- **→ 부분문제로 나눠서 풀기**
- 어디서 많이 봤는데?

- 다이나믹 프로그래밍 : 부분문제가 겹침 → 메모이제이션으로 해결
- 분할 정복 : 부분문제가 겹치지 않음

# 분할 정복?


problem
divide
subproblem
subproblem
conquer
solve
subproblem
solve
subproblem
solution to
subproblem
solution to
subproblem
combine
solution to
problem

# 분할 정복?


problem
divide
subproblem
subproblem
divide
divide
subproblem
subproblem
subproblem
subproblem
divide
divide
divide
divide
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
conquer
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
combine
combine
combine
combine
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
combine
combine
solution to subproblem
solution to subproblem
combine
solution to problem

# 분할 정복?

| Divide & Conquer | Dynamic Programming |
|---|---|
| 1. Partitions a problem into independent smaller sub-problems | 1. Partitions a problem into overlapping sub-problems |
| 2. Doesn't store solutions of sub-problems. (Identical sub-problems may arise - results in the same computations are performed repeatedly.) | 2. Stores solutions of sub-problems: thus avoids calculations of same quantity twice |
| 3. Top down algorithms: which logically progresses from the initial instance down to the smallest sub-instances via intermediate sub-instances. | 3. Bottom up algorithms: in which the smallest sub-problems are explicitly solved first and the results of these used to construct solutions to progressively larger sub-instances |

# 분할 정복의 응용

- (DP처럼) 특정 알고리즘 지칭 X
- 다양한 알고리즘들의 base가 되는 아이디어

- 머지 소트, 퀵 소트
- 수 제곱, 행렬 제곱
- 큰 수 곱셈 (카라추바 알고리즘)
- 라인 스위핑
- 고속 푸리에 변환
- 이분 탐색
- 등등…

# 분할 정복의 장단점

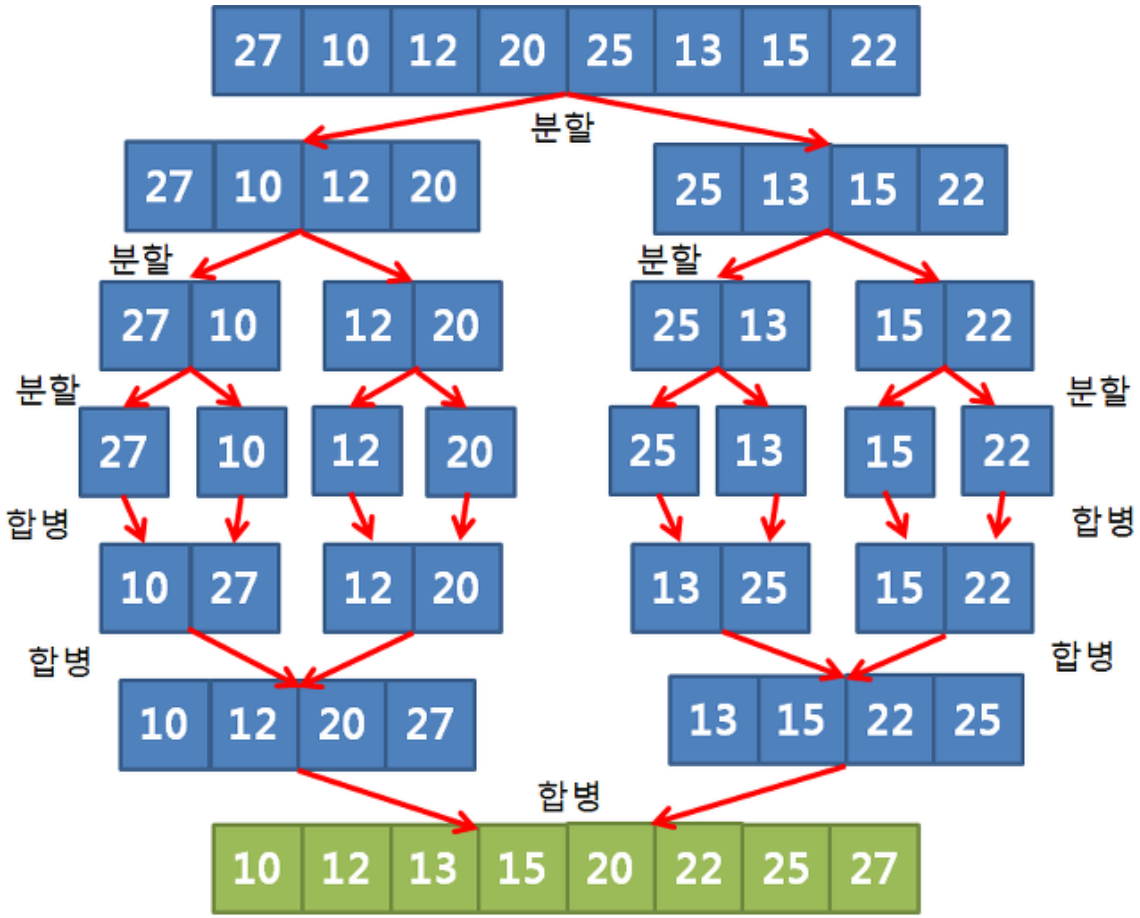
- 장점
  - (원래 못 풀 문제를) 문제를 분할함으로써 풀 수 있다
  - 효율적인 문제풀이 (문제를 쪼개가면서 푸니 로그 시간으로 줄어듦)

- 단점
  - 재귀함수 호출로 인한 오버헤드
  - 문제가 어디서부터 간단한지 판단 어려움 (직관적이지 않음)

- 분할 방법, 최소 문제 기준, 작은 문제를 이용해 큰 문제를 푸는 방법 등에서 퍼포먼스 차이가 큼

# 예시 : 머지 소트

- Merge sort (병합 정렬)



# 예시 : 머지 소트

**ALGORITHM 9** A Recursive Merge Sort.

**procedure** *mergesort*($L = a_1, \ldots, a_n$)
**if** $n > 1$ **then**
    $m := \lfloor n/2 \rfloor$
    $L_1 := a_1, a_2, \ldots, a_m$
    $L_2 := a_{m+1}, a_{m+2}, \ldots, a_n$
    $L := merge(mergesort(L_1), mergesort(L_2))$
{$L$ is now sorted into elements in nondecreasing order}

- 배열 크기가 1이 될 때까지 계속 반으로 쪼갠다 (분할)
- msort(0~N) → merge(msort(0~N/2), msort(N/2+1, N))

# 예시 : 머지 소트

**ALGORITHM 10** Merging Two Lists.

**procedure** *merge*($L_1$, $L_2$: sorted lists)
$L$ := empty list
**while** $L_1$ and $L_2$ are both nonempty
  remove smaller of first elements of $L_1$ and $L_2$ from its list; put it at the right end of $L$
  **if** this removal makes one list empty **then** remove all elements from the other list and
    append them to $L$
**return** $L$ {$L$ is the merged list with elements in increasing order}

- 다 쪼갠 뒤 합치는 건? (정복)
- 핵심 : 정렬된 두 배열을 최대한 빨리 합치는 법

# 예시 : 머지 소트

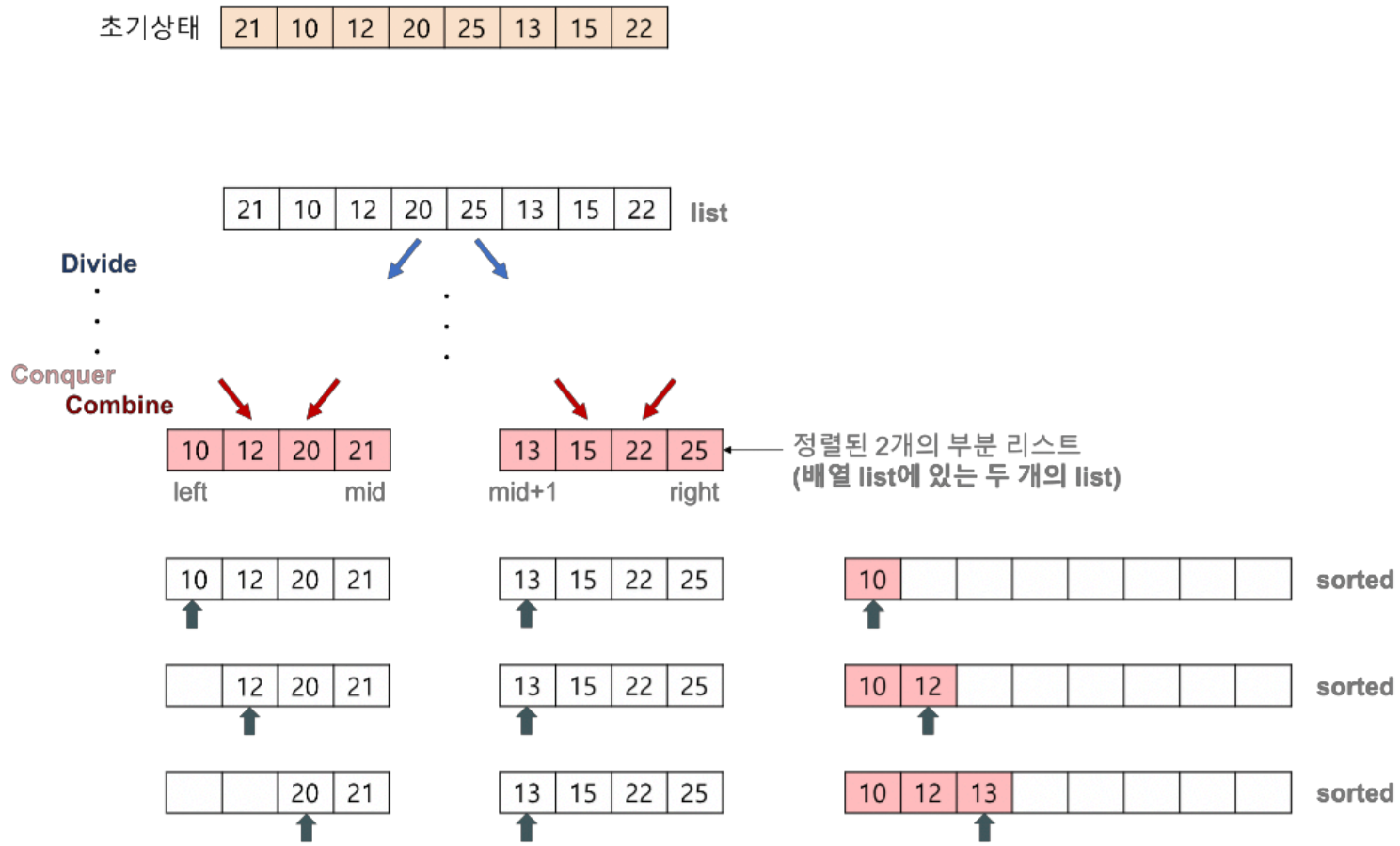
- We need : 두 배열을 가리키는 포인터 두 개, 새 배열을 담을 vector

- 초기화: 비어 있는 vector 생성,
- 두 포인터는 각각 배열의 첫번째 요소 참조
- 두 포인터가 가리키는 요소를 비교해 작은 요소를 vector에 넣음
- 그리고 넣은 요소를 가리키는 포인터를 증가시킴(다음 원소를 가리킴)

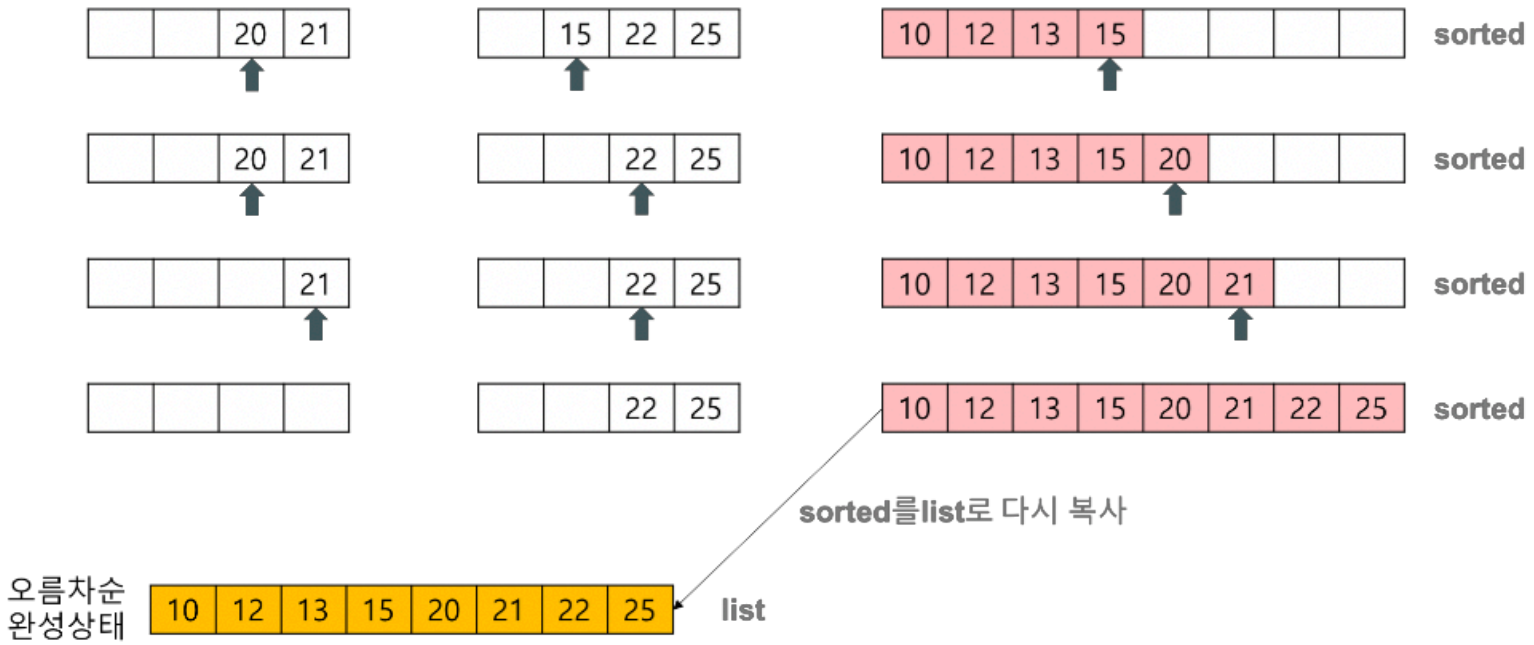
- 이 작업을 두 배열을 모두 vector에 넣을 때 까지 반복 → O(N)

# 예시 : 머지 소트

- 정렬된 4개짜리 두 배열을 합치는 걸 보자



# 예시 : 머지 소트




- 이렇게 두 배열을 합쳐서 새로운 정렬된 배열을 만들 수 있다

# 예시 : 머지 소트

- 배열을 반씩 쪼개서 (O(lgN))
- 두 배열을 합치기 (O(N))

- 머지 소트의 시간 복잡도 : O(NlgN)

# 11728 – 배열 합치기

- 정렬되어있는 두 배열 A와 B가 주어진다. 두 배열을 합친 다음 정렬해서 출력하는 프로그램을 작성하시오.

- 머지소트에서 두 배열 합치기 연습
- 사실 정렬 문제는 그냥 std::sort 쓰면 되니…
- 배열 합치기 연습이라도

# 11728 – 배열 합치기

```
int a_idx = 0, b_idx = 0, c_idx = 0;

while (a_idx < n && b_idx < m) {
    if (a[a_idx] < b[b_idx]) {
        c[c_idx++] = a[a_idx++];
    } else {
        c[c_idx++] = b[b_idx++];
    }
}

while (a_idx < n) c[c_idx++] = a[a_idx++];

while (b_idx < m) c[c_idx++] = b[b_idx++];
```

# 예시 : 수의 N제곱

- 어떠한 수의 B의 N제곱을 구하고 싶다
- 단순히 N번 곱하면? B * B * B * … → O(N)
- 시간복잡도를 더 줄일 수 없을까?

- 거듭제곱의 성질을 이용하면
- 분할 정복을 통해 개선된 알고리즘을 짤 수 있다

# 예시 : 수의 N제곱

- N이 짝수일 경우, $B^N = B^{\frac{N}{2}} \times B^{\frac{N}{2}}$
- N이 홀수일 경우, $B^N = B^{N-1} \times B$
- 반씩 쪼갤 수 있다!

- 예) B^10 → B^5 * B^5 → (B * B^2 * B^2) * (B * B^2 * B^2) → …
- B^N을 구하는 데 필요한 연산의 수를 줄일 수 있다

# 1629 – 곱셈

- 자연수 A를 B번 곱한 수를 알고 싶다. 단 구하려는 수가 매우 커질 수 있으므로 이를 C로 나눈 나머지를 구하는 프로그램을 작성하시오.
- A, B, C는 모두 2,147,483,647 이하의 자연수이다.

- 그냥 B번 곱했다간 O(N)으로 시간초과
- 아까 배운 방법을 써 보자

# 1629 – 곱셈

```
typedef long long ll;
ll solution(ll b, ll n, ll m) {  // (b^n)%m
    if (n == 0LL) {
        return 1;
    }

    ll ret = solution(b, n / 2, m) % m;  // 함수 호출 줄이기 위해 변수에 담아두기
    if (n % 2 == 0) {
        return (ret * ret) % m;
    } else {
        return (((ret * ret) % m) * b) % m;
    }
}

cout << solution(a, b, c) << '\n';
```

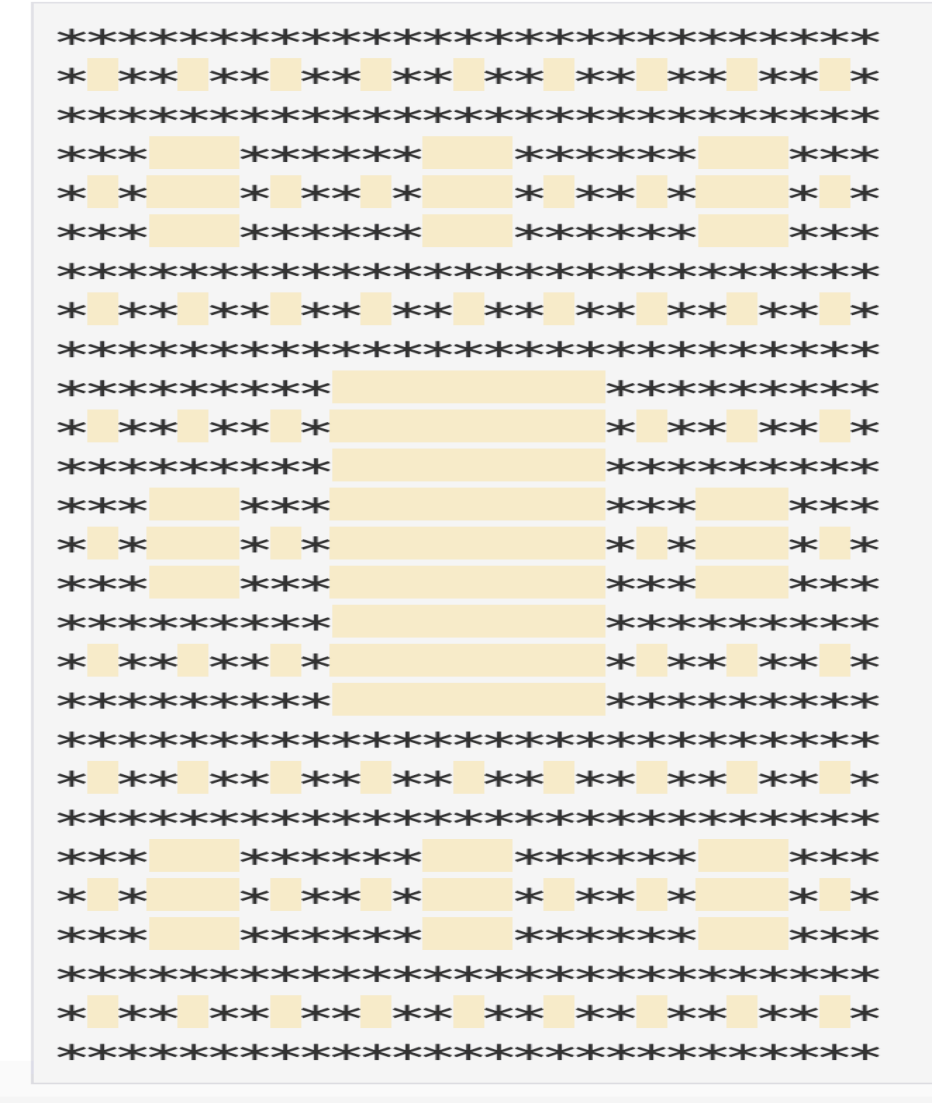
# 번외 : 행렬 제곱

- N*K 행렬과 K*M 행렬의 곱 : O(NMK)
- N*N 행렬의 제곱 : O(N^3)
- N*N 행렬의 B제곱 : O(B * N^3)
- 너무 오래 걸린다

- 앞에서 썼던 아이디어에서 숫자만 행렬로 바꾸면 된다
- 행렬 class 또는 struct 만들고, 곱셈 연산자 overloading (또는 곱셈 함수 직접 작성)
- 10830 – 행렬 제곱

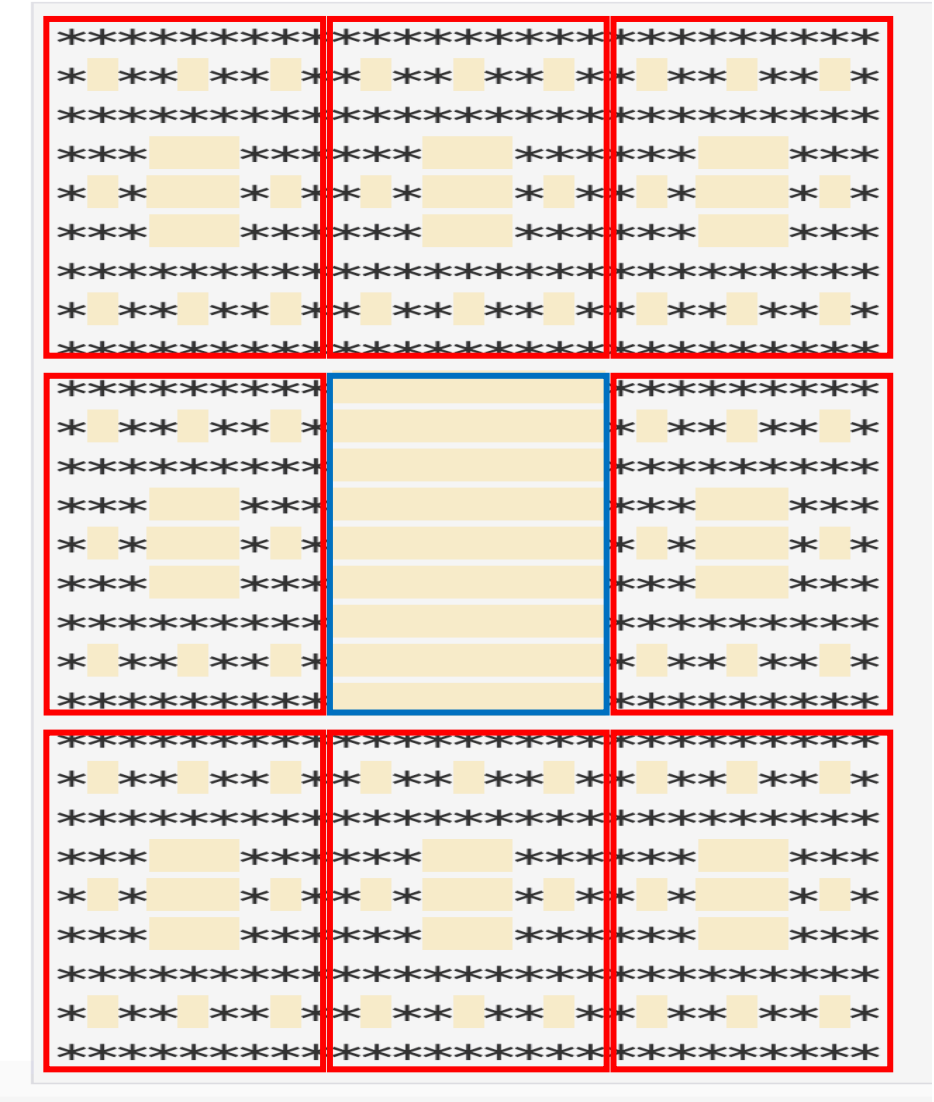
# 2447 – 별 찍기 - 10

- ???
- 그냥 반복문으로 찍기엔 좀 힘들어 보인다
- 부분문제로 나눌 수 있을까?



# 2447 – 별 찍기 - 10

- 모양이 반복됨을 알 수 있다
- 3*3 분할해서 가운데만 빈칸, 나머지 8칸은 다시 재귀적 구조
- N = 1일 때가 최소 케이스 (‘*’ 하나 출력)



# 2447 – 별 찍기 - 10

- http://boj.kr/4a0ae9c9cce64d36913ad6375aca25ca

- solution(y, x, n) → 왼쪽 위 좌표가 (y, x)이고, n*n 크기의 사각형
- 한 재귀함수 내에서 총 8개의 재귀함수 호출 (8부분)
- 가운데 부분은 그 자리에서 공백으로 채우기

- 최소 케이스까지 분할했을 경우 (N = 1) '*' 하나 출력

# 참고

- https://www.slideshare.net/fvsandoval/data-structure-and-algorithm-divide-and-conquer (분할정복 그림)
- https://deokmans.tistory.com/10 (머지소트 그림)
- https://gmlwjd9405.github.io/2018/05/08/algorithm-merge-sort.html (머지소트 그림)

끝