Divide & Conquer

Seo Ju Won
2019.07.04

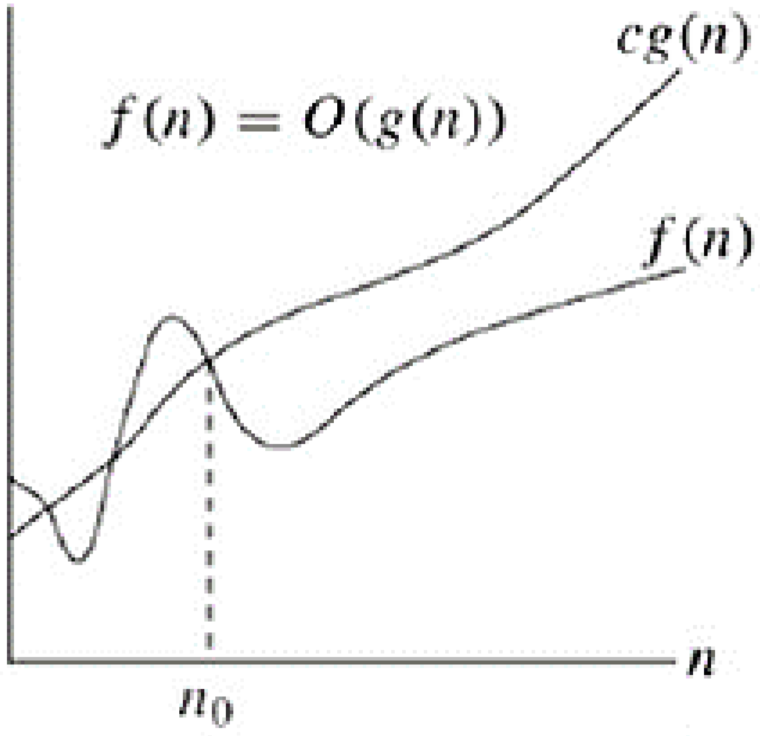
Table of Contents

1. Introduction

Big-O notation

- $f(n)$ is $O(g(n))$,

  if there are positive constants c and n_0 such that $f(n) \leq cg(n)$ for $n \geq n_0$

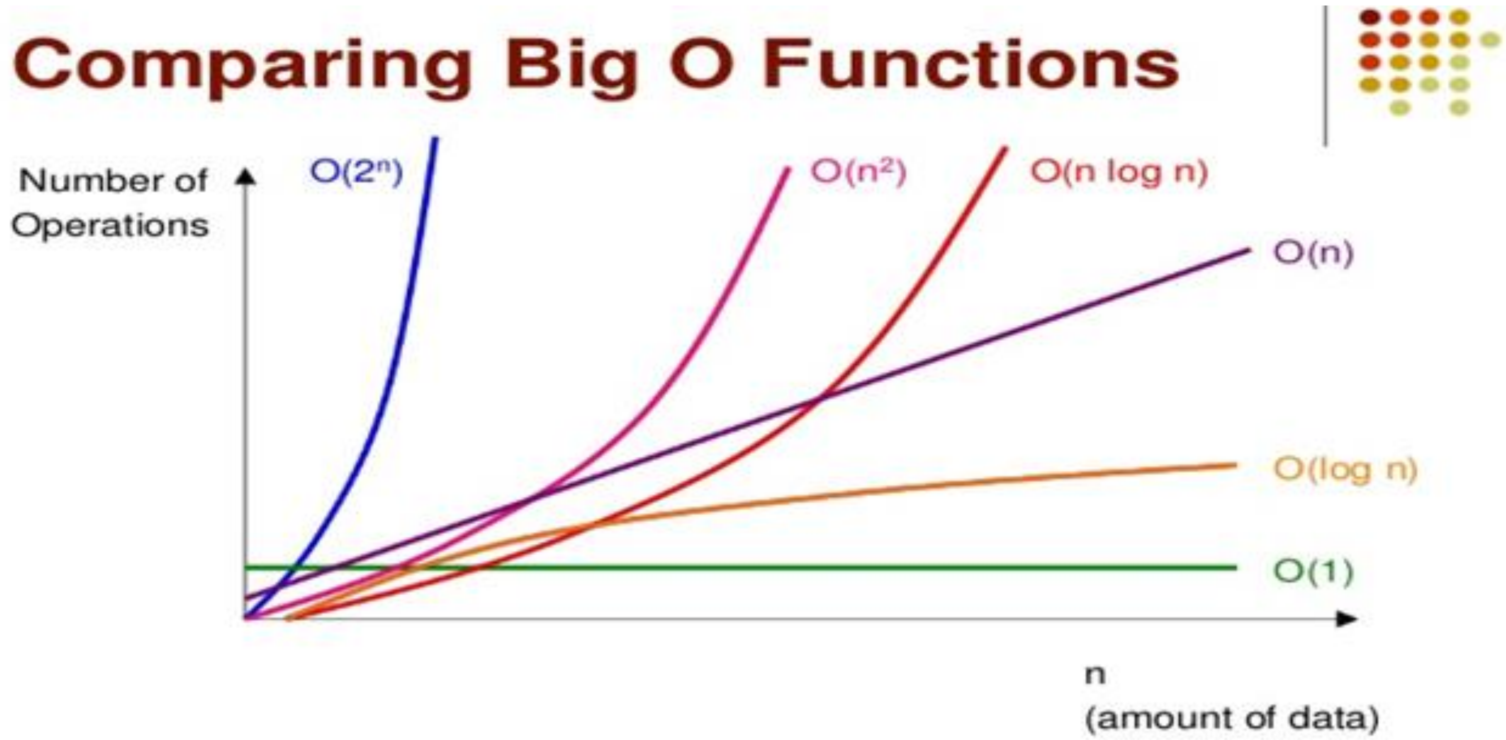


Big-O notation

$f(n)=4n+5$: $O(n)$

$f(n)=7n^2+40$: $O(n^2)$

Big-O notation

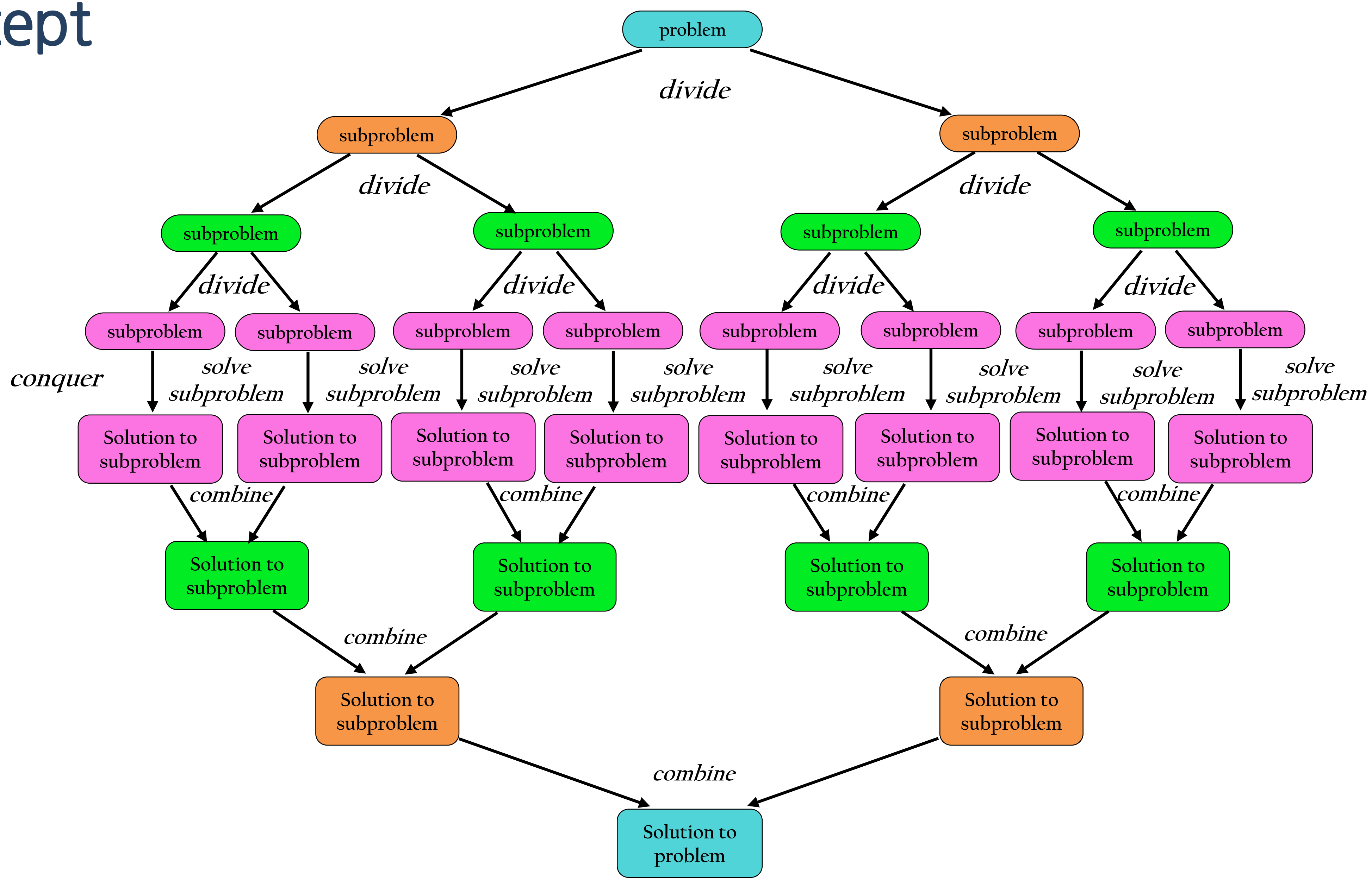


Performance :

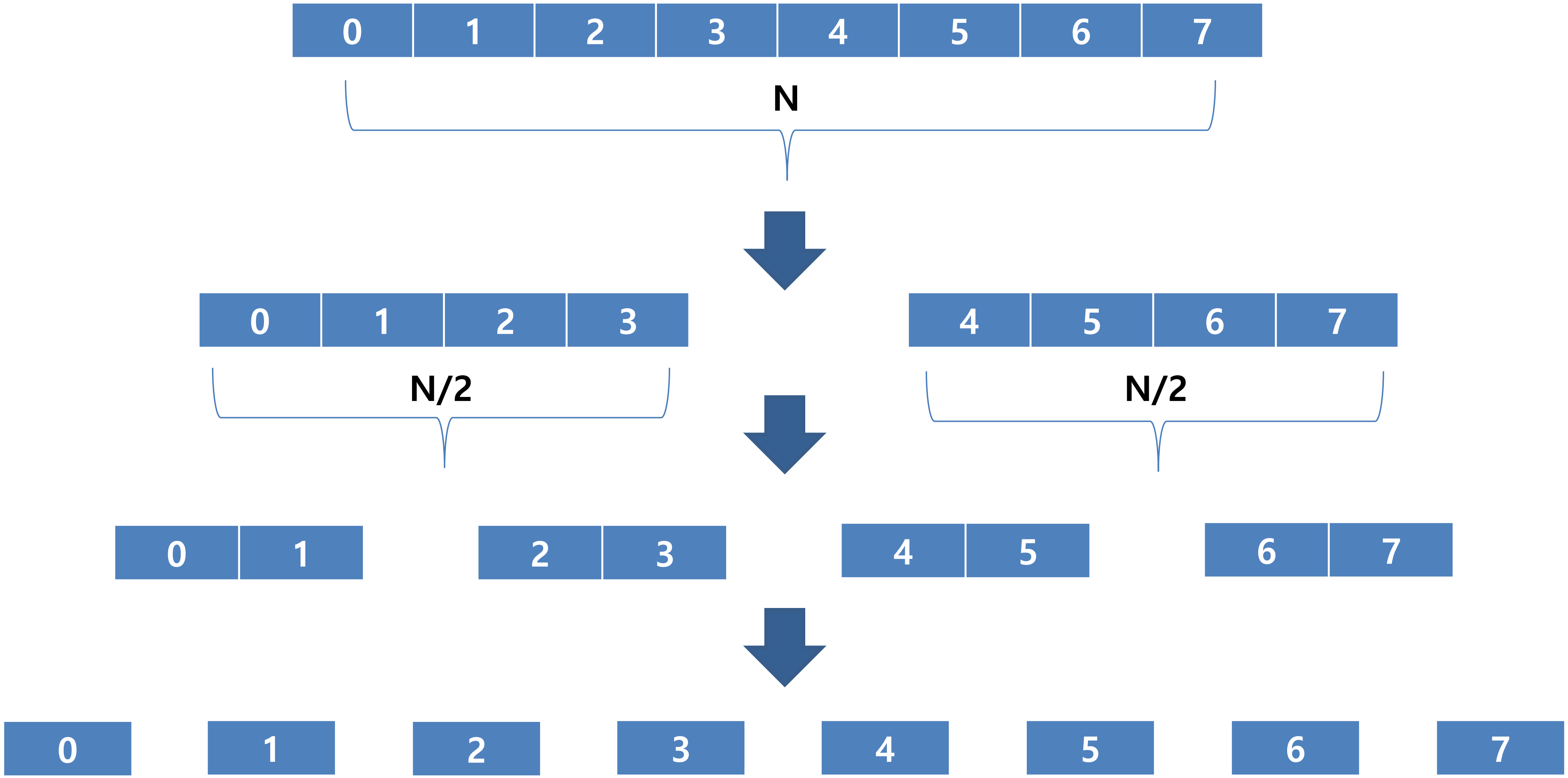
O(1) > O(logn) > O(n) > O(nlogn) > O(n^2) > O(2^n)

2. Divide & Conquer

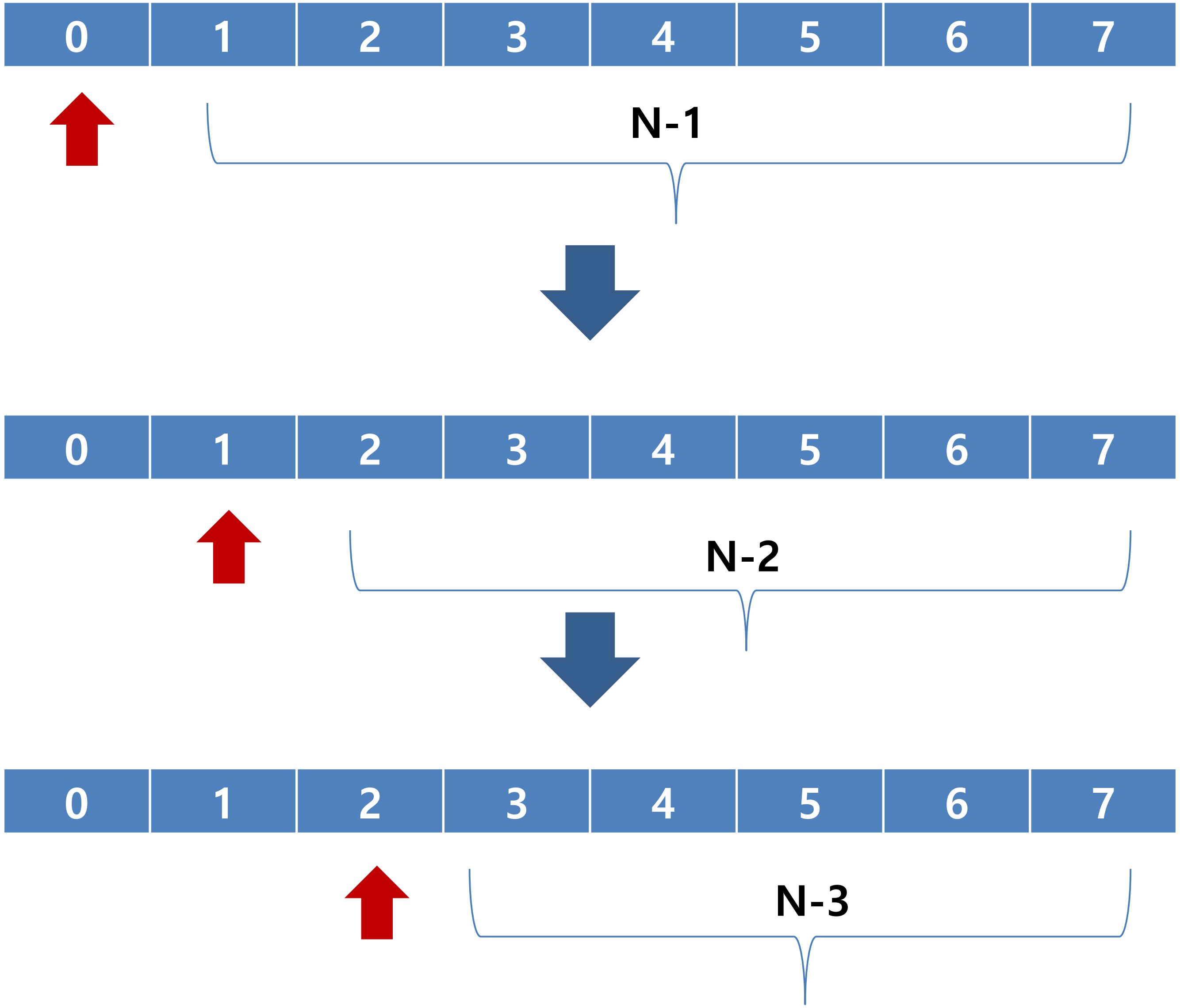
Concept


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
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

Concept



Concept



Master Theorem

Combining the three cases above gives us the following "master theorem".

Theorem 1 *The recurrence*

Number of problem

Size of problem

$$
\begin{aligned}
T(n) &= aT(n/b) + cn^k \\
T(1) &= c,
\end{aligned}
$$

Time of conquer step
for each sub-problem

where a, b, c, and k are all constants, solves to:

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

Master Theorem

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

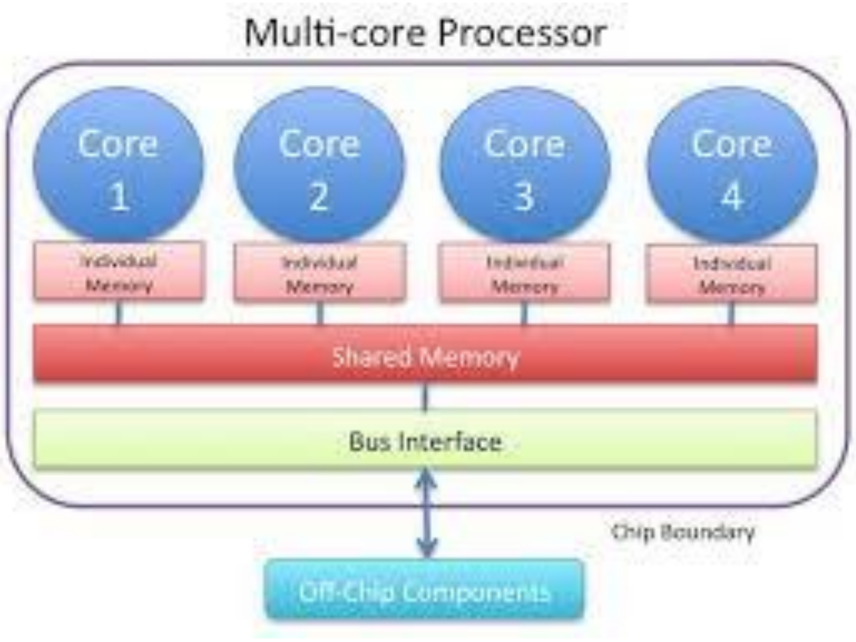
A=2 B=2 K=1

T(n)=2T(n/2)+n

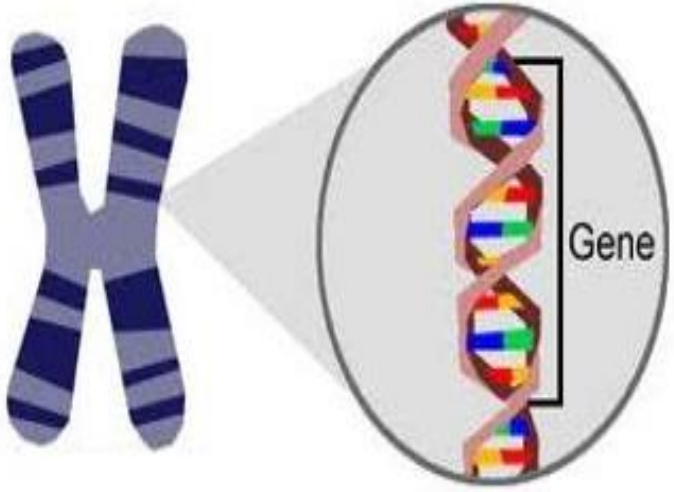
$$T(n) \in \Theta(n^k) \text{ if } a < b^k$$
$$\boxed{T(n) \in \Theta(n^k \log n) \text{ if } a = b^k}$$
$$T(n) \in \Theta(n^{\log_b a}) \text{ if } a > b^k$$

then $T(n) = \Theta(n^{\log_2 2} \log_2 n) = \Theta(n \log_2 n)$

Applications



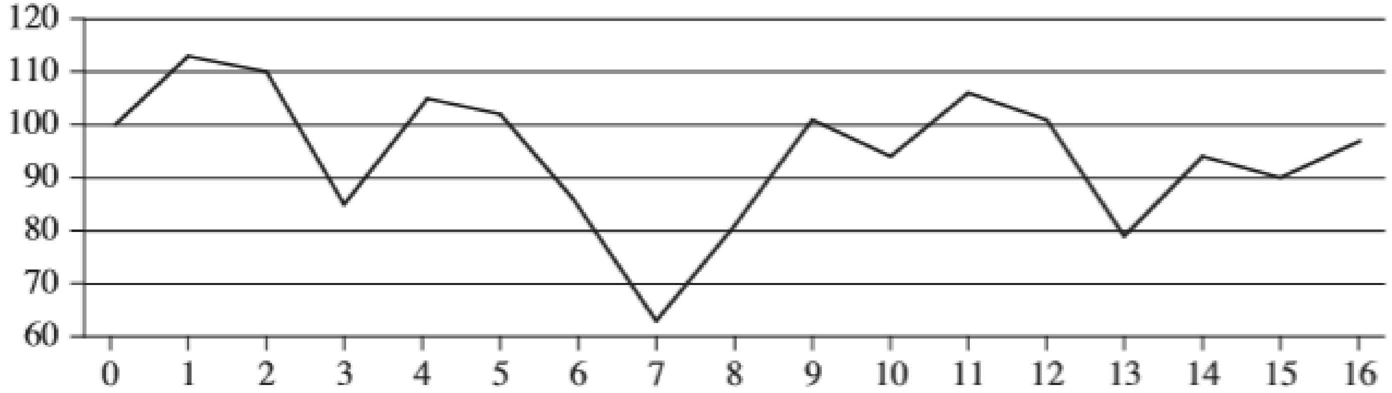
Merge Sort



Quick Sort

3. Maximum-subarray

Problem



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

maximum subarray

1. Brute-Force

1	-2	2	4	5	-6	3

Sum [illegible]

Max value -999

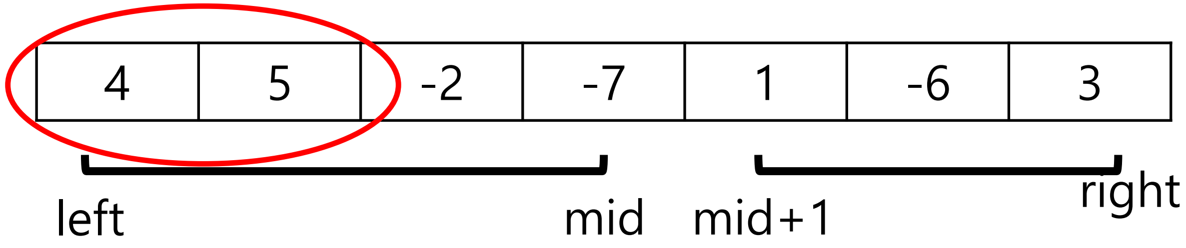
1. Brute-Force

```
for(int i=0; i<n; i++){

    int sum=0;

    for (int j = i; j < n; j++) {
            sum = sum + arr[j];
            ret = max(ret, sum);
        }

}
```

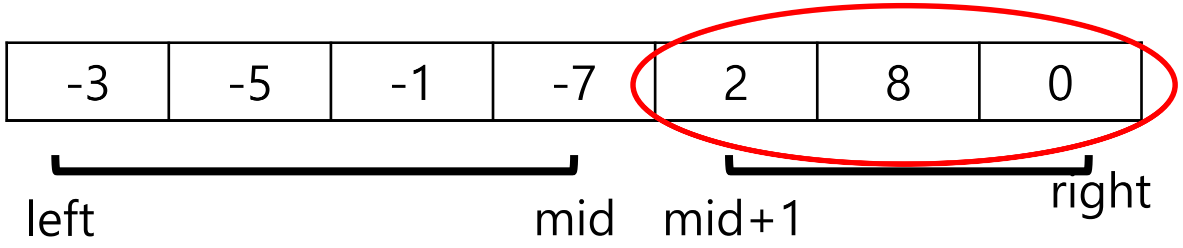
(n^2+n)/2 -> O(n^2)

2. Divide & Conquer

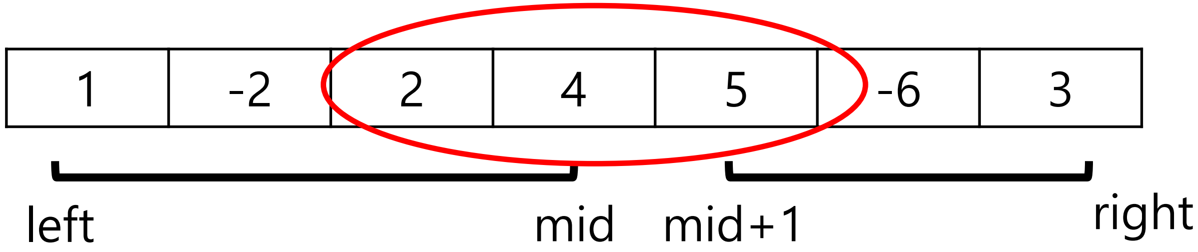
Case 1. Max value in the range of [left , mid]



Case 2. Max value in the range of [mid+1 , right]



Case 3. Max value overlap between the [left, mid] and [mid+1, right]



2. Divide & Conquer

1. If base case, return value
2. Find mid value
3. Divide the problem to sub-problem and compare case1 max value and case2 max value.
4. Find the case3 max value.
5. Return the highest value by comparing the maximum value of case 3 and single value(case1, case2)

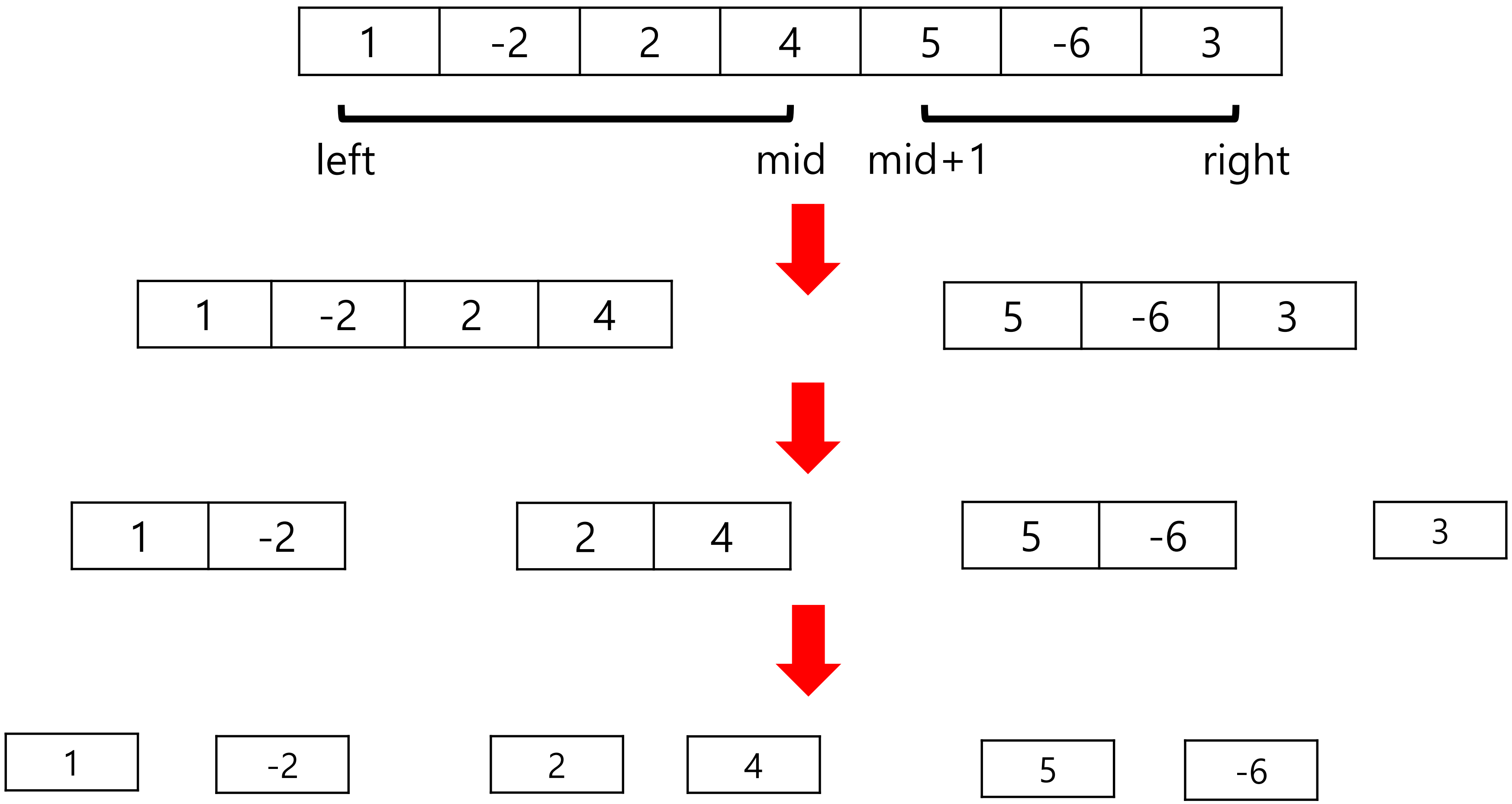
```
int recur(int left, int right) {

    //base case
    if (left == right) {
        return arr[left];
    }

    //recursive case
    int mid = (left + right) / 2;

    int single = max(recur(left, mid), recur(mid + 1, right));

    for (int i = mid; i >= left; i--) {
        sum = sum + arr[i];
        left_sum = max(left_sum, sum);
    }

    sum = 0;
    for (int i = mid + 1; i <= right; i++) {
        sum = sum + arr[i];
        right_sum = max(right_sum, sum);
    }

    return max(left_sum + right_sum, single);
}
```

Case 1

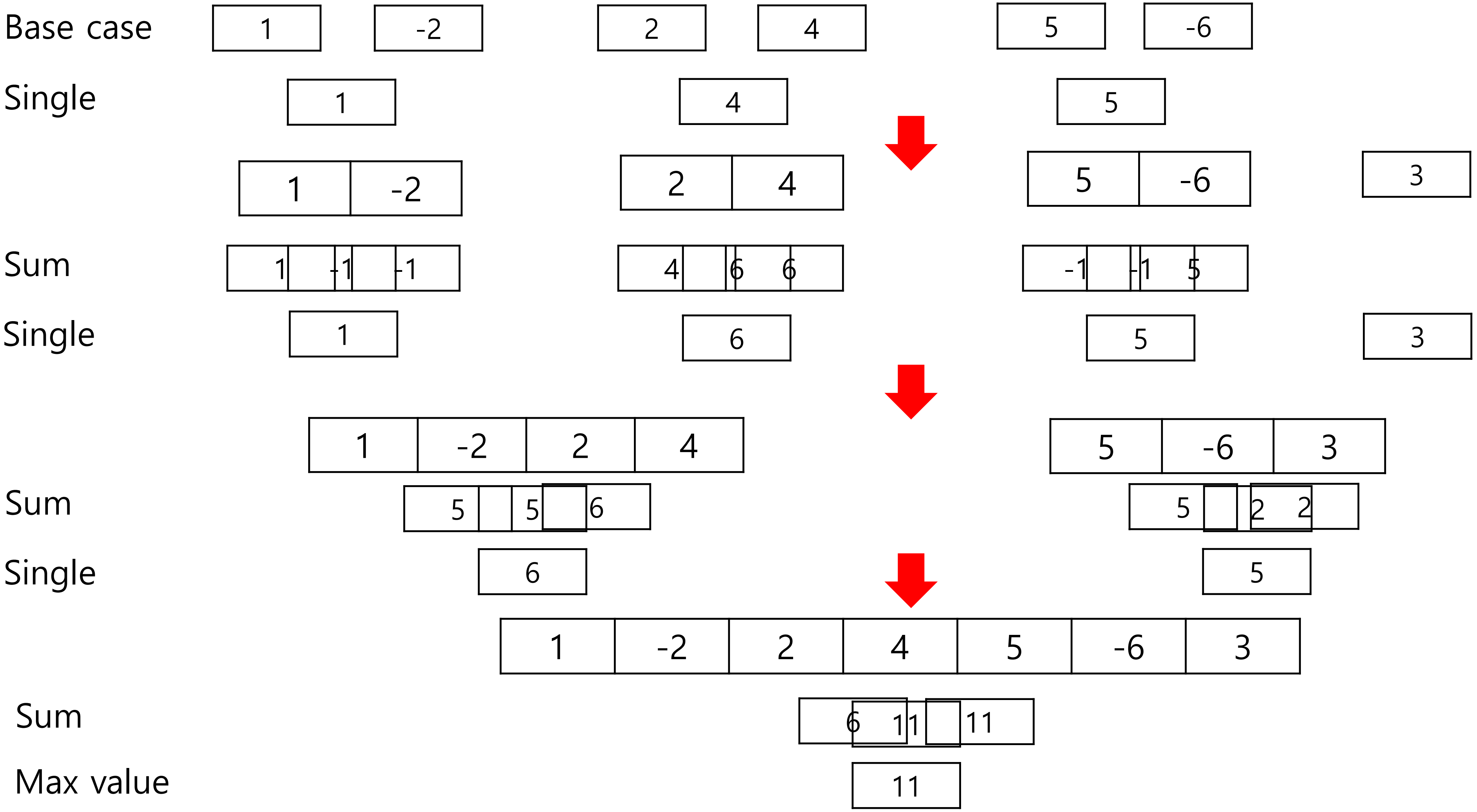
Case 2

Case 3

2. Divide



2. Conquer & Combine



2. Divide & Conquer

- $\Theta(1)$
- $2T(n/2)$
- $\Theta(n)$

(mid - left +1) + (right - (mid + 1) +1)
= right - left +1

```
int recur(int left, int right) {

    //base case
    if (left == right) {
        return arr[left];
    }

    //recursive case
    int mid = (left + right) / 2;

    int single = max(recur(left, mid), recur(mid + 1, right));

    for (int i = mid; i >= left; i--) {
        sum = sum + arr[i];
        left_sum = max(left_sum, sum);
    }

    sum = 0;
    for (int i = mid + 1; i <= right; i++) {
        sum = sum + arr[i];
        right_sum = max(right_sum, sum);
    }

    return max(left_sum + right_sum, single);
}
```

2. Divide & Conquer

$T(n) = 2T(n/2) + \Theta(n)$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. $\boxed{\text{If } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \lg n).}$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Using master theorem, a=2, b=2, $f(n) = \Theta(n)$

$\therefore T(n) = \Theta(n\log n)$

3. Dynamic programing

$$f(i) = \begin{cases} max(0, f(i-1)) + arr[i] & i > 0, \\ arr[i] & i == 0 \end{cases}$$

```
for(int i=0; i<n; i++){

    sum = max(sum,0) + arr[i];
    ret = max(ret, sum);

}
```

O(n) Time

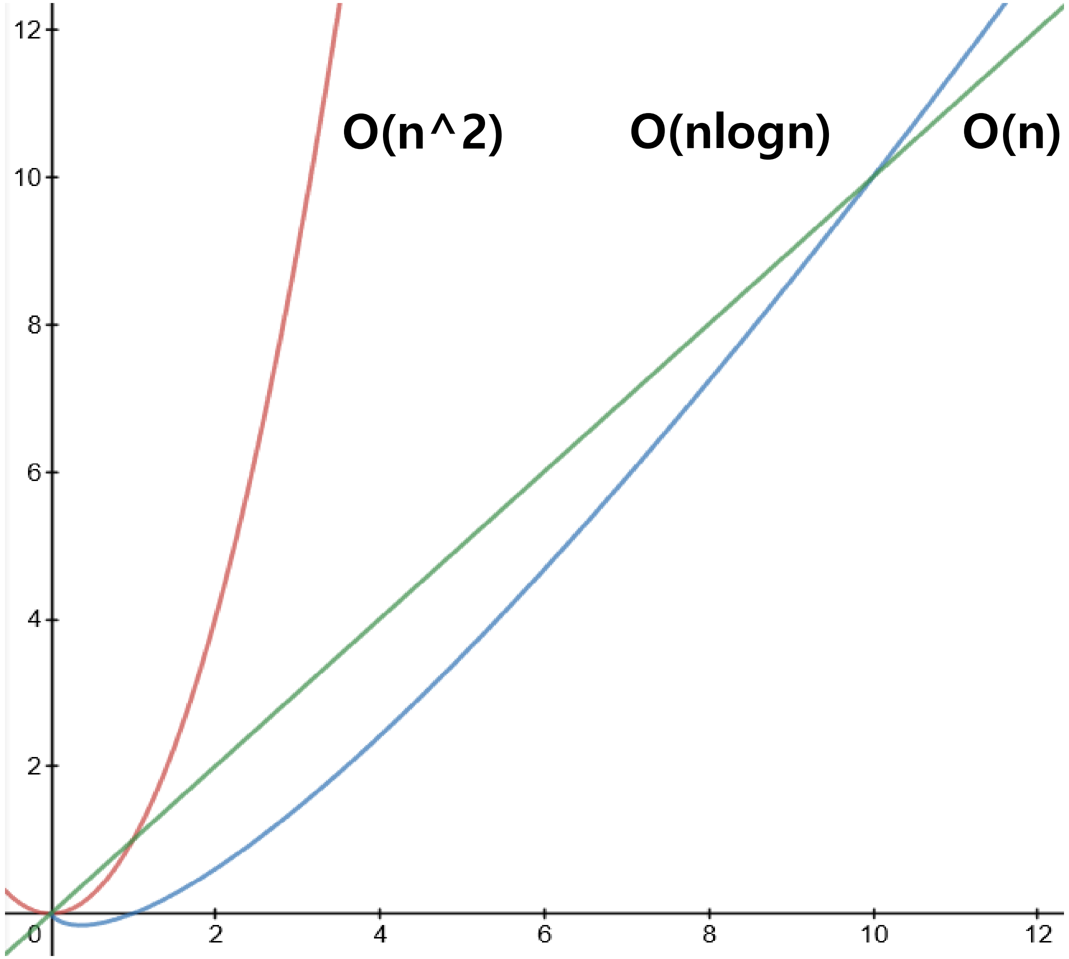
3. Dynamic programing

1	-2	2	4	5	-6	3

Sum -999

Max value -999

Analysis



N>10 이상일 경우

Performance : O(n^2) < O(nlogn) < O(n)

Analysis

1. Brute-Force => O(n^2)

```
n의 개수 : 10000
max value : 6569
걸린 시간 : 0.953000sec
```

2. Divide & Conquer => O(nlogn)

```
n의 개수 : 10000
max value : 6569
걸린 시간 : 0.004000sec
```

3. Dynamic Programing => O(n)

```
n의 개수 : 10000
max value : 6569
걸린 시간 : 0.001000sec
```

4. Conclusion

Conclusion

- **Advantages**

- Use Divide & Conquer, we can solve problems in less time than Brute-force.
- It can be available in many fields.
  (such as, Matrix multiplication, Parallelization, Bioinformatics)

- **Disadvantages**

- There is a possibility of stack overflow due to recursive.
- There is a possibility of better way such as dynamic programing.

Q & A

Thank you!