

# MAXIMUM SUM

20022627 정주원

# 순서

- ⊙ 문제 분석

- ⊙ 문제 해결 방법

- Brute Force
- Dynamic Programming1
- Dynamic Programming2

# 문제 분석

- $N \times N$  배열에서 합이 최대가 되는 부분 찾기

0	-2	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2

SUM: 15

# 문제 분석

- ◎  $N \times N$  배열에서 합이 최대가 되는 부분 찾기

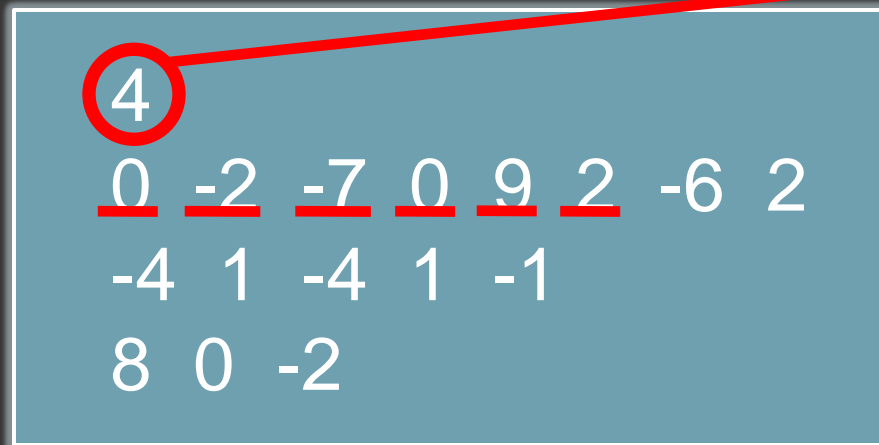
- ◎ 입력

- $N$  : 입력 받을 배열의 크기( $N \times N$  배열)
  - 범위 :  $(0, 100]$
- $N^2$ 개의 숫자
  - 범위 :  $[-127, 127]$

# 문제 분석

- $N \times N$  배열에서 합이 최대가 되는 부분 찾기

- 입력



4	0	-2	-7	0	9	2	-6	2
-4	1	-4	1	-1				
8	0	-2						



0	<div>N=4 -2</div>	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2

# 문제 분석

- ◎  $N \times N$  배열에서 합이 최대가 되는 부분 찾기
- ◎ 입력
- ◎ 출력
  - 최대가 되는 부분의 합

# 문제 분석

- $N \times N$  배열에서 합이 최대가 되는 부분 찾기

- 입력

- 출력

15

0	-2	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2

# 문제 분석

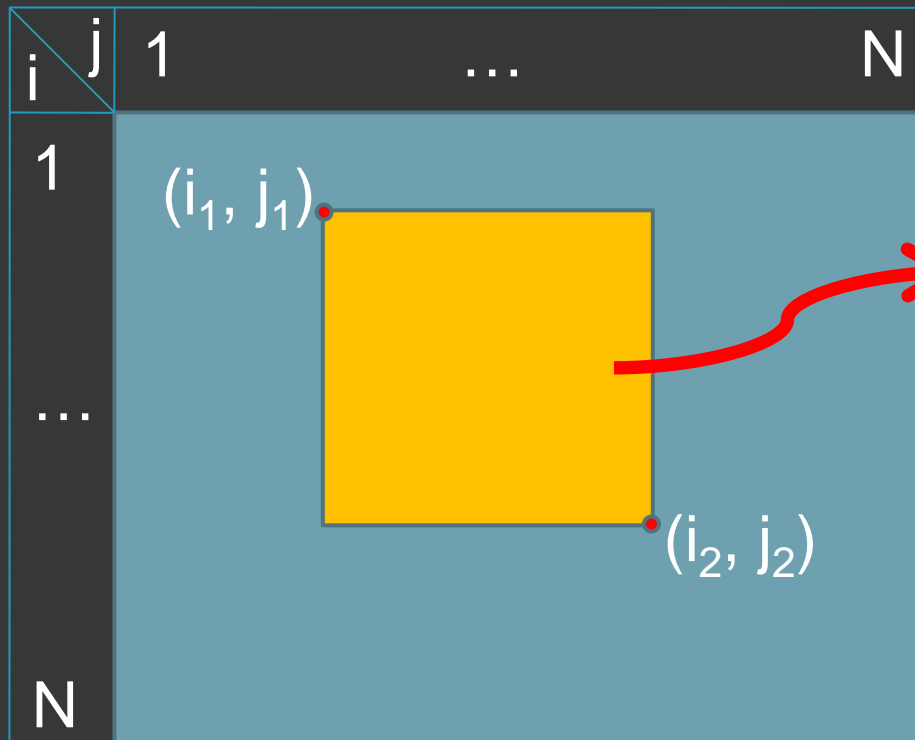
- ◎  $N \times N$  배열에서 합이 최대가 되는 부분 찾기
- ◎ 입력
- ◎ 출력
- ◎ 입력되는 테스트 케이스의 수는 1



# Brute Force

- ◎ 가능한 모든 사각형 탐색
  - 두 점  $(i_1, j_1), (i_2, j_2)$  이용
  - $i_1 \leq i_2, j_1 \leq j_2$
  - 시간 복잡도 :  $O(N^4)$
- ◎ 각 사각형 안의 원소들의 합 계산
  - $\text{data}[i_1, j_1]$  부터  $\text{data}[i_2, j_2]$ 까지 합
  - 시간 복잡도 :  $O(N^2)$
- ◎ 전체 시간 복잡도 :  $O(N^6)$

# Brute Force



```
subSum = 0
for m = i1..i2
  for n = j1..j2
    subSum += data[m][n]
If (subSum > max)
  max = subSum
```

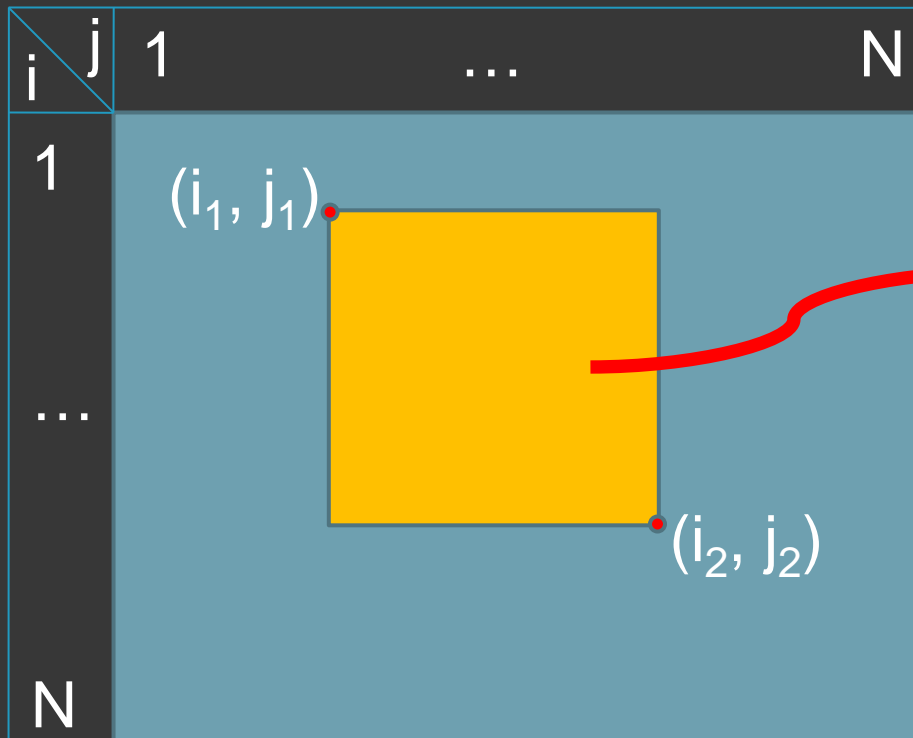
# Brute Force

#	Problem	Verdict	Language	Run Time
6369169	Maximum Sum	Time limit exceeded	ANSI C	3.000

# Dynamic Programming 1

## ◎ 개선할 부분

- 각 사각형 안의 원소들의 합 계산



```
subSum = 0
for m = i1..i2
  for n = j1..j2
    subSum += data[m][n]
If (subSum > max)
  max = subSum
```

# Dynamic Programming 1

## ◎ sum[N][N]

- sum[i][j] : data[0,0]부터 data[i, j]까지 합

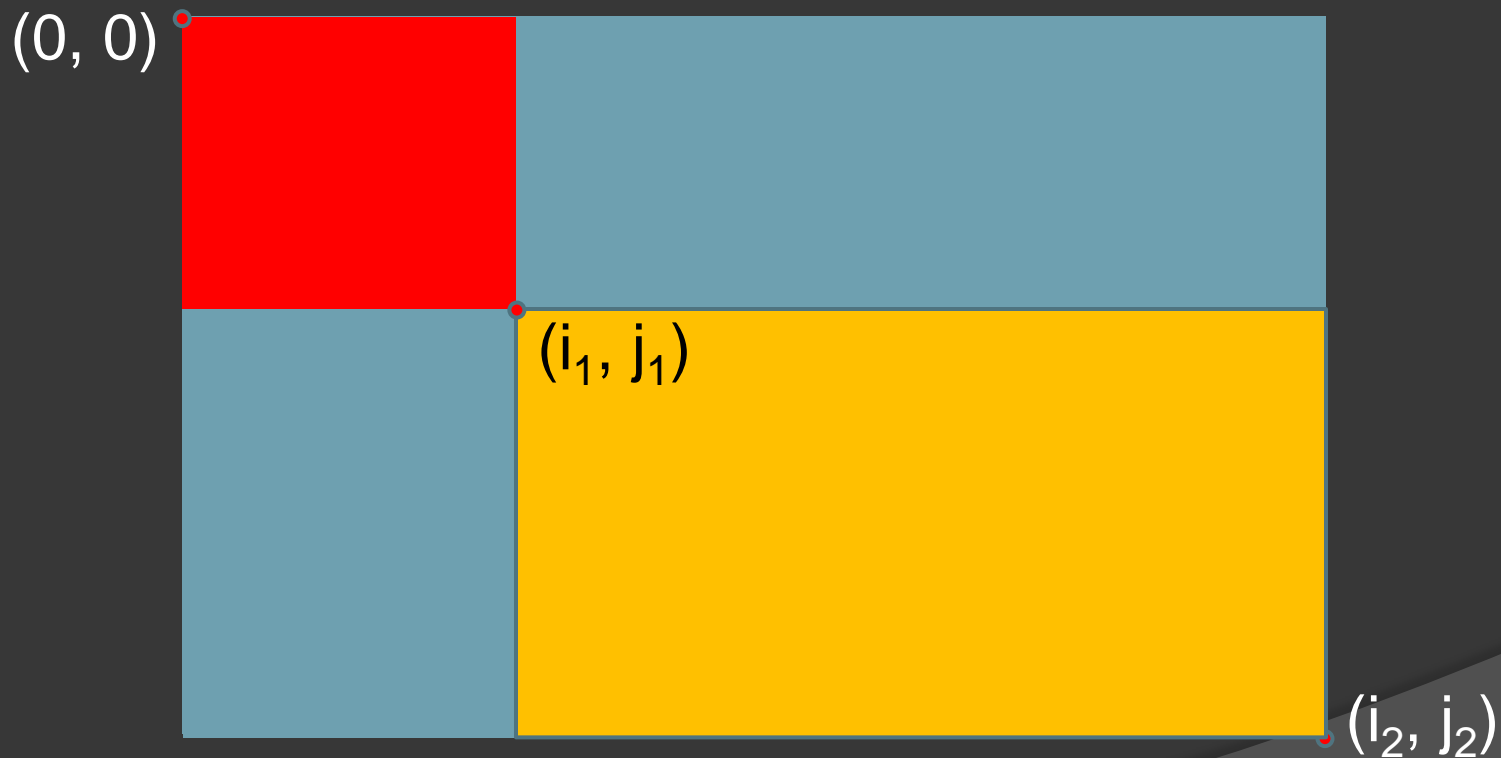
data			
0	-2	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2



sum			
0	-2	-9	-9
9	9	-4	-2
5	6	-11	-8
4	13	-4	-3

# Dynamic Programming 1

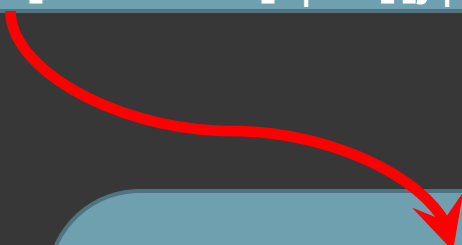
⦿  $(i_1, j_1)$ 부터  $(i_2, j_2)$ 의 합 구하기



# Dynamic Programming 1

## ◎ $(i_1, j_1)$ 부터 $(i_2, j_2)$ 의 합 구하기

- $$\text{subSum} = \text{sum}[i_2, j_2] - \text{sum}[i_1 - 1, j_2] - \text{sum}[i_2, j_1 - 1] + \text{sum}[i_1 - 1][j_1 - 1]$$
- 시간복잡도 :  $O(1)$



```
subSum = 0
for m = i1..i2
  for n = j1..j2
    subSum += data[m][n]
If (subSum > max)
  max = subSum
```

# Dynamic Programming 1

## ◎ $(i_1, j_1)$ 부터 $(i_2, j_2)$ 의 합 구하기

- $\text{subSum} = \text{sum}[i_2, j_2] - \text{sum}[i_1 - 1, j_2] - \text{sum}[i_2, j_1 - 1] + \text{sum}[i_1 - 1][j_1 - 1]$
- 시간복잡도 :  $O(1)$

## ◎ 전체 시간복잡도 : $O(N^4)$



# Dynamic Programming 2

- ◎ 1차원에서 최대 부분합 구하기

- ◎ Kadane's algorithm

- 시간 복잡도 :  $O(n)$

# Kadane's algorithm

array X



- ◎ maxEndingHere
  - $i-1$ 을 포함한 최대 subarray
- ◎ maxSoFar
  - $i-1$ 까지의 최대 subarray

# Kadane's algorithm

array X



- ⦿ maxEndingHere
  - $\max(\text{maxEndingHere} + X_i, X_i)$
- ⦿ maxSoFar
  - $\max(\text{maxSoFar}, \text{maxEndingHere})$

# Kadane's algorithm

```
maxSoFar = -inf
maxEndingHere = 0
for i = 1...n
    maxEndingHere = max(maxEndingHere + x[i], x[i])
    maxSoFar = max(maxSoFar, maxEndingHere)
```

# Dynamic Programming 2

## ◎ 1차원 문제로 바꾸기

k	9	2	-6	2
	-4	1	-4	1
i	-1	8	0	-2

4	11	-10	1
---	----	-----	---

## ◎ maximum subarray 찾기 : $O(n)$

# Dynamic Programming 2

```
for i = 1 ... n
  for k = 1 ... i
    /* maximum subarray 찾기: O(n) */
```

◎ 전체 시간복잡도 :  $O(n^3)$

# Dynamic Programming 2

⦿  $i = 1, k = 1$

9	2	-6	2
-4	1	-4	1
-1	8	0	-2

9	2	-6	2
---	---	----	---

maxEndingHere:  
9, 11, 5, 7

maxSoFar = 11

Max = 11

# Dynamic Programming 2

⦿  $i = 2, k = 1$

9	2	-6	2
-4	1	-4	1
-1	8	0	-2

5	3	-10	3
---	---	-----	---

maxSoFar = 8

Max = 11



# Dynamic Programming 2

⦿  $i = 2, k = 2$

9	2	-6	2
-4	1	-4	1
-1	8	0	-2

-4	1	-4	1
----	---	----	---

maxSoFar = 1

Max = 11

# Dynamic Programming 2

⦿  $i = 3, k = 1$

9	2	-6	2
-4	1	-4	1
-1	8	0	-2

4	11	-10	1
---	----	-----	---

maxSoFar = 15

Max = 15

# Dynamic Programming 2

⦿  $i = 3, k = 2$

9	2	-6	2
-4	1	-4	1
-1	8	0	-2

-5	9	-4	-1
----	---	----	----

maxSoFar = 9

Max = 15

# Dynamic Programming 2

⦿  $i = 3, k = 3$

9	2	-6	2
-4	1	-4	1
-1	8	0	-2

-1	8	0	-2
----	---	---	----

maxSoFar = 8

Max = 15

# Dynamic Programming 2

## ◎ 최종 알고리즘

```
maxSoFar = -inf ;
for i = 1 ... n {
  for k = 1 ... i {
    maxEndHere = 0
    for j = 1 ... n {
      cur = sum[i][j] - sum[i][j-1] - sum[k-1][j] +
            sum[k-1][j-1]
      maxEndHere = max(maxEndHere+cur, cur)
      maxSoFar = max(maxSoFar, maxEndHere)
    }
  }
}
```