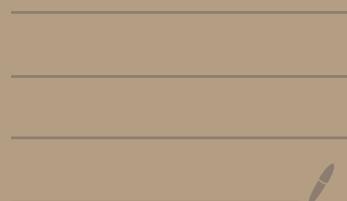


# Divide & Conquer

---



```

[1 1 0 0 0 0 1 1]
[1 1 0 0 0 0 1 1]
[0 0 0 0 1 1 0 0]
[0 0 0 0 1 1 0 0]
[1 0 0 0 1 1 1 1]
[0 1 0 0 1 1 1 1]
[0 0 1 1 1 1 1 1]
[0 0 1 1 1 1 1 1]

```

8x8      Count=0      n = int(input())  
 for i in range(n):  
 Origin\_Matrix.append(list(input().split(" ")))

fun check\_색종이(x,y,n):

check = Origin\_Matrix[x][y]

for i in range(x, x+n)      ) index 범위의  
 for j in range(y, y+n)      ) 이유: x, y가  
 if check와 다른게 아니라면      항상 시작점이기  
 나오면:      때문에

$\Rightarrow$  다음장

블록 단지를 조사하세요  
 $+n, +\frac{n}{2}, +\frac{n}{4}, \dots$

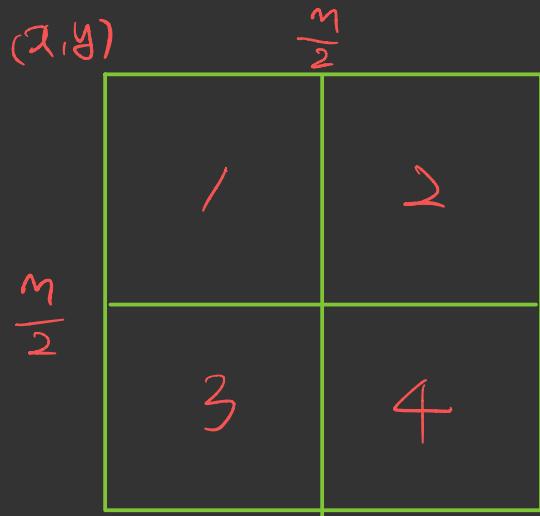
을 해주어야 한다.

색종이 만들기

check - 색종이  $(x, y, n/2)$ : 1사분면

- ‘’  $(x, y+n/2, n/2)$  2사분면
- ‘’  $(x+n/2, y, n/2)$  3사분면
- ‘’  $(x+n/2, y+n/2, n/2)$  4사분면

$\Rightarrow 743 |$  틀렸음  
최소 간격 13  
divide



## 종이의 개수

```
def check_color_paper(x, y, n):
    check = origin_matrix[x][y]
    for i in range(x, x+n):
        for j in range(y, y+n):
            if check != origin_matrix[i][j]:
                check_color_paper(x, y, n//3) # 1사분면
                check_color_paper(x, y+n//3, n//3) # 2사분면
                "      (x, y+2*n//3, n//3) # 3사분면
                "      (x+n//3, y, n//3) # 4사분면
                :
                :
```

def check\_color\_paper (cont.)

if check == -1 :

minus\_count += 1

elif check == 0 :

zero\_Count += 1

elif check == 1 :

plus\_Count += 1

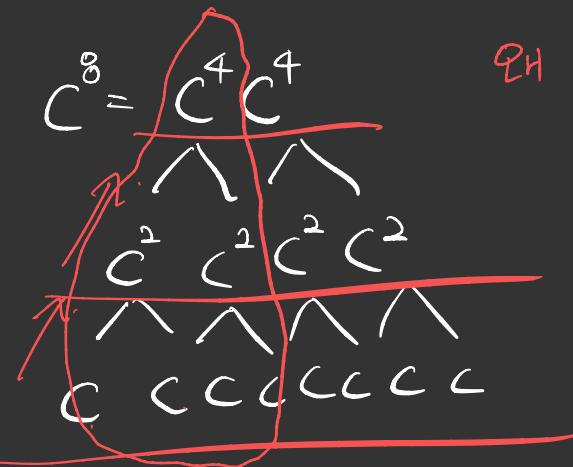
곱셈

## \* 분할정복을 이용한 거듭제곱 \*

$C^n$  연산은  $C$ 를  $n$ 번 곱하므로  $O(n)$ 이다.

그러나 분할정복을 사용하면  $O(\log n)$ 으로 시간복잡도를 줄일 수 있다.

$$C^n = \begin{cases} C^{\frac{n}{2}} C^{\frac{n}{2}} & (n \text{은 짝수}) \\ C^{\frac{n-1}{2}} C^{\frac{n-1}{2}} C & (n \text{은 홀수}) \end{cases}$$



왜 return하는 과정은  
시간복잡도에  
고려를 안 할까?

곱셈 cont.

왜 각 연산마다 "%C" 를 해야하지?

$$(A+B)\%M = ((A\%M) + (B\%M))\%M$$

$$(A*B)\%M = ((A\%M) * (B\%M))\%M$$

$$(A-B)\%M = ((A\%M) - (B\%M))\%M$$

이항 계수3

### \* 피타고라스의 소정리 \*

$$a^p \% P = a \% P \quad (P는 소수, a는 정수)$$

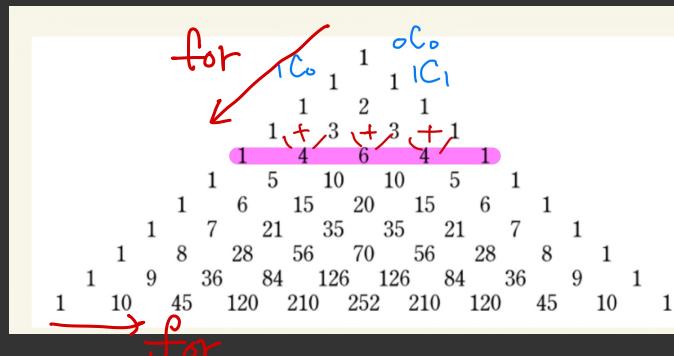
$a$ 와  $P$ 가 서로소이면

$$a^{P-1} \% P = 1 \% P$$

왜 파스칼의 삼각형을 이용하여 풀면 안될까?

i) 문제에서 Input N의 범위가 크기 때문.

시간복잡도:  $O(n^2)$  ( $\because$  이중 for문)



$$\underbrace{a \cdot a^{P-2}}_{\text{a의 } P\text{-제곱으로 연산에 대한 역원}} \equiv 1 \pmod{P}$$

$$mC_k = \frac{n!}{k!(n-k)!}$$

$$\frac{1}{k!(n-k)!} \stackrel{?}{=} \text{어떻게 구하지?}$$

$$[K!(n-k)!]^{P-2} \rightarrow K!(n-k)! \pmod{\text{연산의 }} \text{역원}$$

$$\frac{n!}{K!(n-k)!} \% P$$

$O(n)$

$$\frac{m!}{k!(n-k)!} [K!(n-k)!]^{P-2} \% P$$

$O(\log n)$

### 이항 계수3 cont.

$$nC_k \% P = \left( \frac{n!}{k!(n-k)!} \right) \% P = \frac{n! \% P}{\underbrace{[k!(n-k)!]} \% P} \quad \leftarrow \text{이렇게 했을 경우 문제점}$$

나머지가 0이되면

Error가 발생한다.

행렬 곱셈

A      B

$$\begin{pmatrix} m_1m_1 & m_1m_2 \\ m_2m_1 & m_2m_2 \\ m_3m_1 & m_3m_2 \end{pmatrix} \begin{pmatrix} m_1k_1 & m_1k_2 & m_1k_3 \\ m_2k_1 & m_2k_2 & m_2k_3 \end{pmatrix}$$

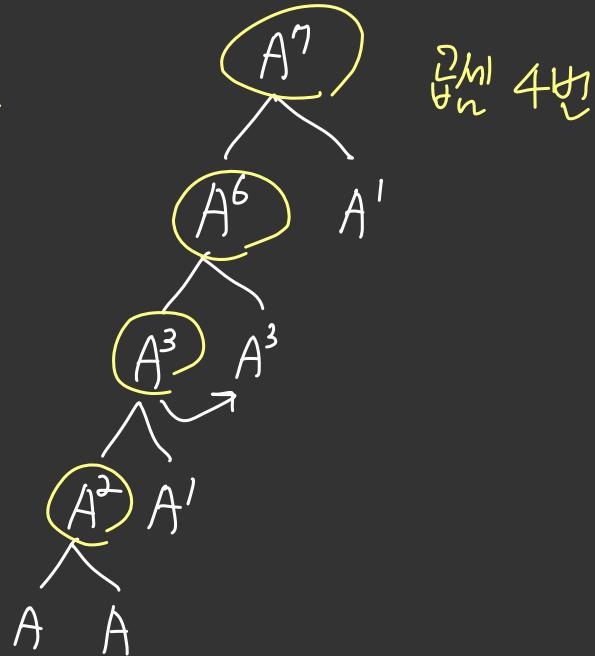
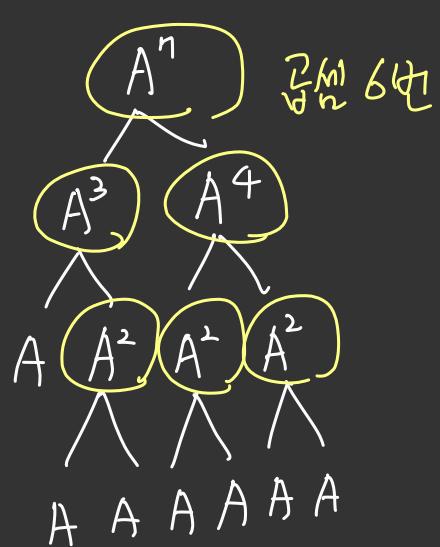
$$= \begin{cases} A[1][1] \times B[1][1] + A[1][2] \times B[2][1] & A[1][1] \times B[1][2] + A[1][2] \times B[2][2] \dots \\ A[2][1] \times B[1][1] + A[2][2] \times B[2][1] & A \dots \\ A[3][1] \times B[1][1] + A[3][2] \times B[2][1] & \dots \end{cases}$$

$C[n][k] = [A[n][m] \times B[m][k]]$  의 누적합 (  $m$ : 1부터 끝까지 )

행렬 제곱

ref. 1629번 → 가등재<sub>2x2</sub>을 분할정복을 이용하여 풀었다.

$$A^8 = A \times A \times A \cdots \times A \Rightarrow \times 7\text{번}$$



피보나치 수 6

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

$$\rightarrow \begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} \dots \textcircled{a}$$

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} \dots \textcircled{b}$$

$$\rightarrow \begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}}_A \underbrace{\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}}_{\mathbf{u}_n} \quad \begin{array}{l} \xrightarrow{\textcircled{a}} \text{식에서 } Ax=b \text{ 꼴로 만들기 위해} \\ \left( \begin{matrix} F_{n+1} \\ F_n \end{matrix} \right) \text{을 공통인자로 갖는 식 } \textcircled{b} \text{를} \\ \text{추가} \end{array}$$

피보나치 수 6 cont.

$$u_0 = \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$u_1 = A \cdot u_0$$

$$u_2 = A \cdot u_1 = A \cdot (A \cdot u_0) = A^2 u_0$$

$$u_3 = A \cdot u_2 = A \cdot (A \cdot u_1) = A^3 u_0$$

$$u_4 = A^4 u_0$$

$$\vdots$$

$$u_n = A^{\frac{n}{2}} u_0 \rightarrow A^{\frac{n}{2}} \text{ 이용하여 } u_n \text{ 를 구할 수 있다.}$$

$$\rightarrow \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

피보나치 수 6 cont.

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix}$$

$$A^2 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} F_3 & F_2 \\ F_2 & F_1 \end{pmatrix}$$

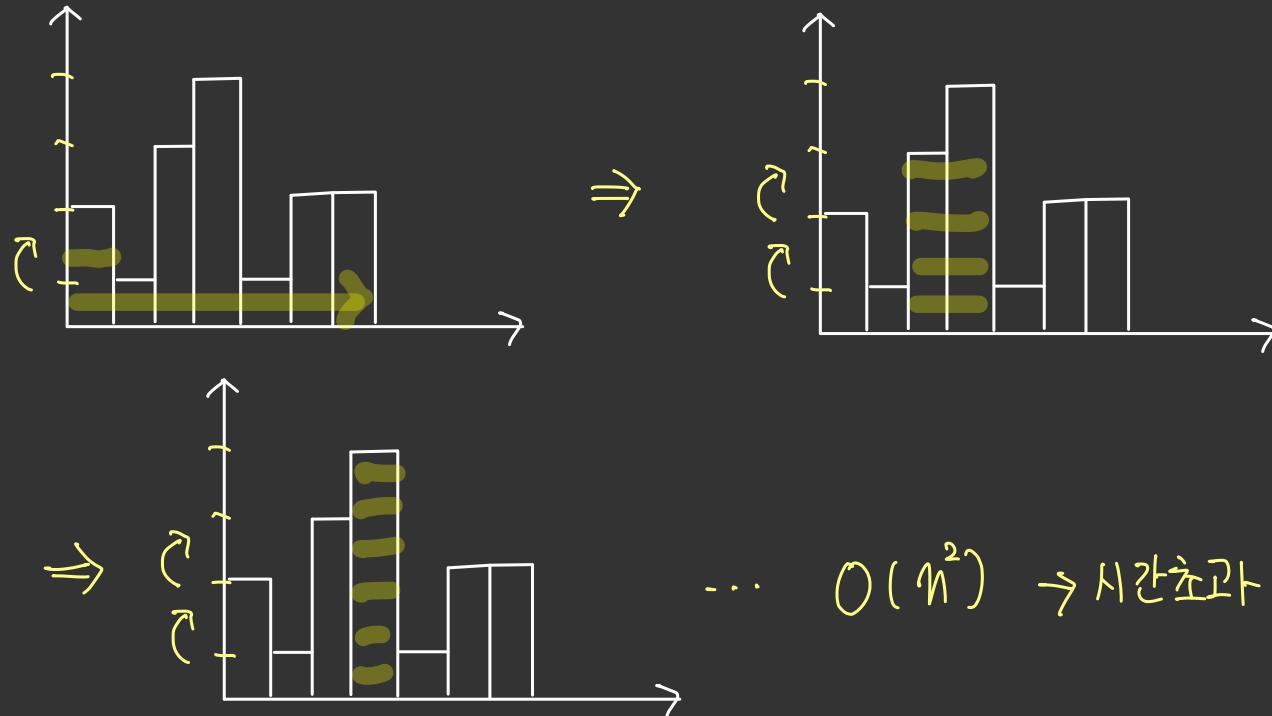
$$A^3 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_3 & F_2 \\ F_2 & F_1 \end{pmatrix} = \begin{pmatrix} F_4 & F_3 \\ F_3 & F_2 \end{pmatrix}$$

⋮

$$A^n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

## 히스토그램에서 가장 큰 직사각형

방법 1. 2 번째 막대의 높이를 하나씩 올려가면서 가능한 히스토그램의 면적을  
넓여보는 방식

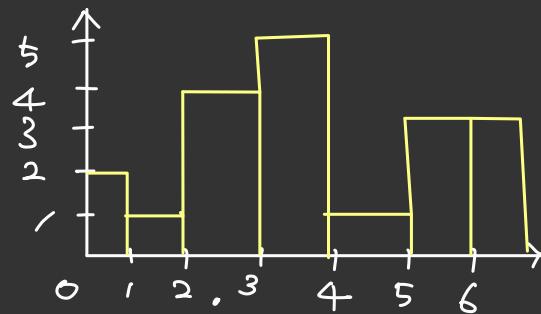


히스토그램에서 가장 큰 직사각형 Cont.

2. 2

Index

0	1	2	3	4	5	6
2	1	4	5	1	3	3



$i = 0$



$i = 1$



POP

$$(i \times 2) = 2$$

$i = 1$



$i = 2$



$i = 3$



$i = 4$



$$i - stack[-1] - 1 = width$$

$$\text{Pop } (4 - 2 - 1) \times 5 = 5$$

$$\text{Pop } (4 - 1 - 1) \times 4 = 8$$

$i = 4$



$i = 5$



$i = 6$



히스토그램에서 가장 큰 직사각형 cont.

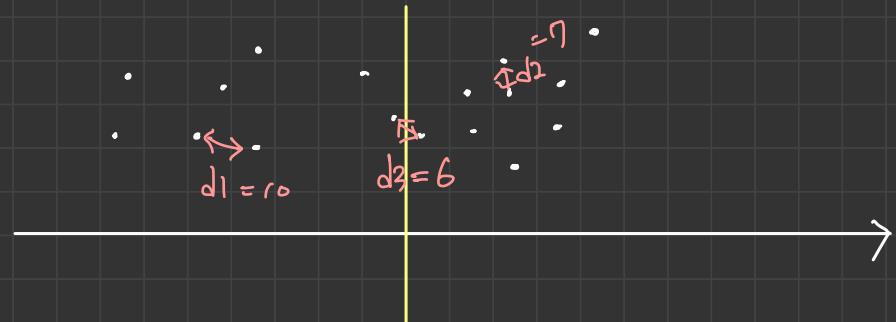
$i = 7$	<del><math>6(3)</math></del>	$\rightarrow \text{Pop } (7 - 5 - 1) \times 3 = 3$
	<del><math>5(3)</math></del>	$\rightarrow \text{Pop } (7 - 4 - 1) \times 3 = 6$
	<del><math>4(1)</math></del>	$\rightarrow \text{Pop } (7 - 1 - 1) \times 1 = 5$
	<del><math>1(1)</math></del>	$\rightarrow \text{Pop } 7 \times 1 = 7$

## 가장 가까운 두 점

① 모든 점을 기좌표 기준 정렬

② 중간 지점을 기준으로 두 개로 분할해서 구간거리가

일정 수준까지 작아지도록 재귀적으로 분할



③ merge 될 때, 좌측 우측에서 각각 가장 가까운 두 점 사이를 구함

그 중 작은 값 선택

해당값과 중간 기준위치를 지나는 왼쪽 점과 오른쪽 점 사이의 거리를 비교

ex)  $\min(d_1, d_2) = 7 = d \rightarrow$  중간 부분에서 " $d = 7 > d_3$ "를 만족하는  $d_3$ 을 구함.

가장 가까운 두 점 . cont.

①-1. 출발을 하리가 최소 단위인 점 한 개가 되었을 때는 거리를 계산할 수 없으므로 무한대를 반환한다.

③-1. 왼쪽과 오른쪽 영역에서의 최솟값  $d$ 가 구해졌으면 가능한 점의 후보를 차트표의 거리가  $d$  미만인 점들로 추릴 수 있다.

③-2. 차트표는 처리가 끝났으니 싱크포를 기준으로 정렬하고 후보군을 추린다.

③-3. 후보군이 모였으니 두 점간 거리를 계산한다.

이 때, 계산된 거리가  $d$ 보다 작다면  $d$ 를 갱신해준다.

만약, 계산된 거리가  $d$ 보다 크거나 같다면 푸아蠢을 탐출한다.

( $\infty$  기준으로 정렬하였기 때문)

모든 템색이 끝나면 최솟값으로 갱신된  $d$ 를 반환한다.

가장 가까운 두 점 . Cont.

③)-4. 만약 같은 점이 존재한다면 거리의 최솟값은 0이므로 이를 반복한다.