# 25-2 이니로
# 알고리즘 멘토링

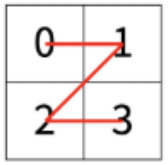
멘토 - 김수성

# Z / 1074

## 백준 1074 / https://www.acmicpc.net/problem/1074

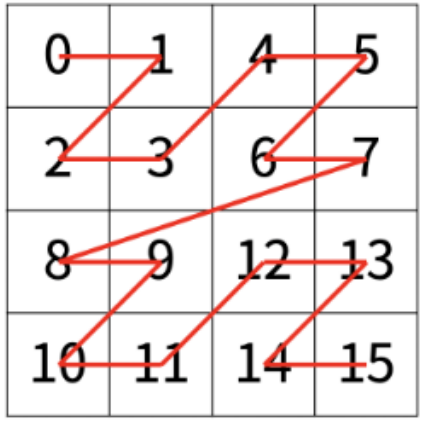
### 문제

한수는 크기가 $2^N \times 2^N$인 2차원 배열을 Z모양으로 탐색하려고 한다. 예를 들어, 2×2배열을 왼쪽 위칸, 오른쪽 위칸, 왼쪽 아래칸, 오른쪽 아래칸 순서대로 방문하면 Z모양이다.



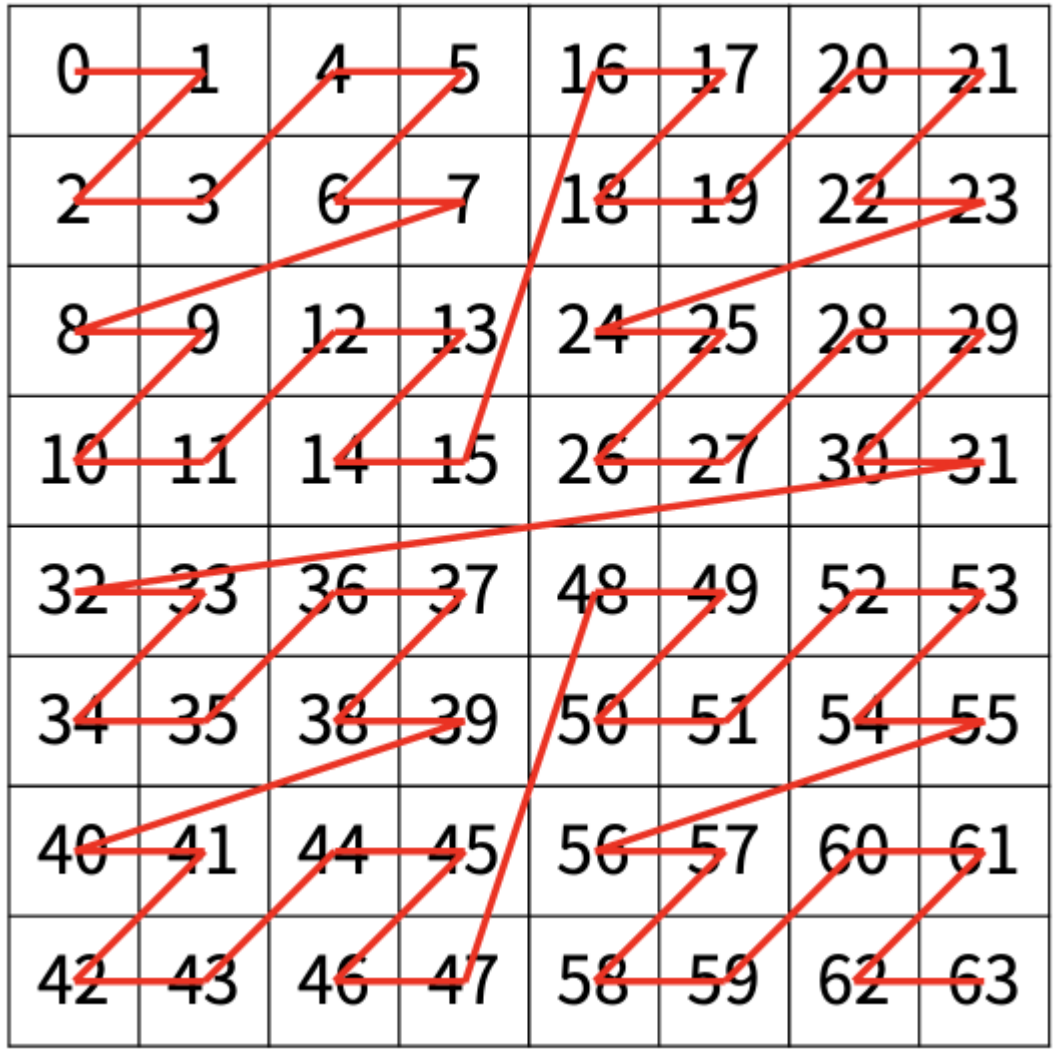

N > 1인 경우, 배열을 크기가 $2^{N-1} \times 2^{N-1}$로 4등분 한 후에 재귀적으로 순서대로 방문한다.

다음 예는 $2^2 \times 2^2$ 크기의 배열을 방문한 순서이다.




N이 주어졌을 때, r행 c열을 몇 번째로 방문하는지 출력하는 프로그램을 작성하시오.

# Z / 1074




예제 입력 1 복사

```
2 3 1
```

예제 출력 1 복사

```
11
```

예제 입력 2 복사

```
3 7 7
```

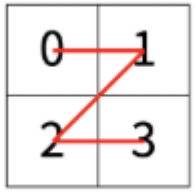
예제 출력 2 복사

```
63
```

# Z / 1074

dnc(size, y, x) =
현재 정사각형 길이가 2^size 일 때 (x, y)의 순서

Base Case
size == 1

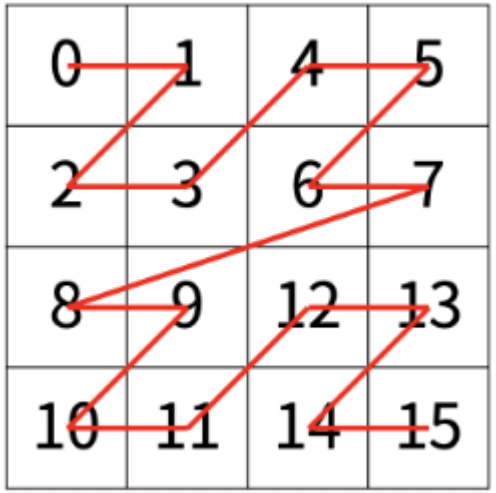



```
ll dnc(ll cur, ll cy, ll cx){
    // Base Case
    if(cur == 1) return 2 * cy + cx;


}
```

(0, 0) = 0
(0, 1) = 1 -> 2 * x + y
(1, 0) = 2
(1, 1) = 3

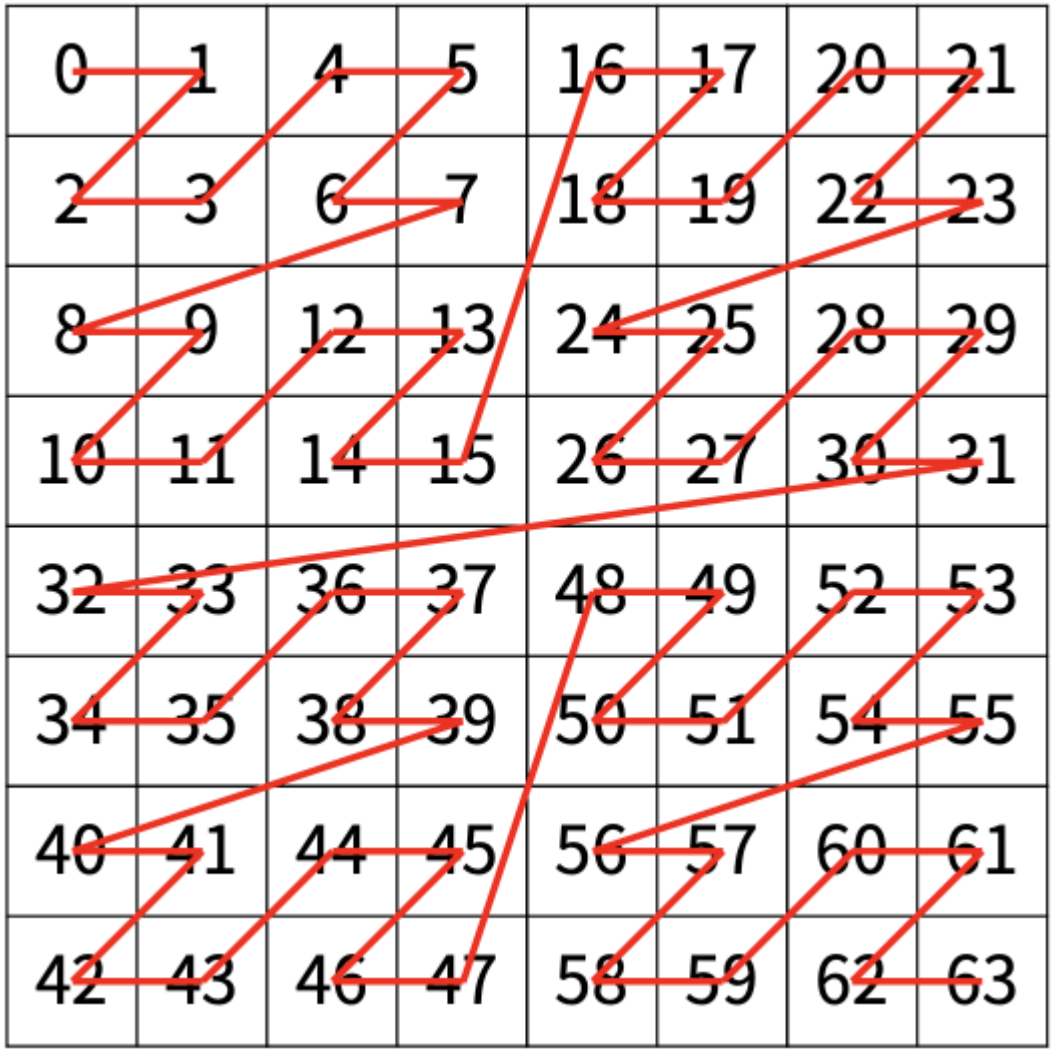
# Z / 1074




->

| | | | |
|---|---|---|---|
| 0 | 1 | 0 + 4 | 1 + 4 |
| 2 | 3 | 2 + 4 | 3 + 4 |
| 0 + 8 | 1 + 8 | 0 + 12 | 1 + 12 |
| 2 + 8 | 3 + 8 | 2 + 12 | 3 + 12 |

# Z / 1074




->

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 4 | 5 | 0 + 16 | 1 | 4 | 5 |
| 2 | 3 | 6 | 7 | 2 | 3 | 6 | 7 |
| 8 | 9 | 12 | 13 | 8 | 9 | 12 | 13 |
| 10 | 11 | 14 | 15 | 10 | 11 | 14 | 15 |
| 0 + 32 | 1 | 4 | 5 | 0 + 48 | 1 | 4 | 5 |
| 2 | 3 | 6 | 7 | 2 | 3 | 6 | 7 |
| 8 | 9 | 12 | 13 | 8 | 9 | 12 | 13 |
| 10 | 11 | 14 | 15 | 10 | 11 | 14 | 15 |

# Z / 1074

| | | | |
|---|---|---|---|
| 0 | 1 | 0 + 4 | 1 + 4 |
| 2 | 3 | 2 + 4 | 3 + 4 |
| 0 + 8 | 1 + 8 | 0 + 12 | 1 + 12 |
| 2 + 8 | 3 + 8 | 2 + 12 | 3 + 12 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 4 | 5 | 0 + 16 | 1 | 4 | 5 |
| 2 | 3 | 6 | 7 | 2 | 3 | 6 | 7 |
| 8 | 9 | 12 | 13 | 8 | 9 | 12 | 13 |
| 10 | 11 | 14 | 15 | 10 | 11 | 14 | 15 |
| 0 + 32 | 1 | 4 | 5 | 0 + 48 | 1 | 4 | 5 |
| 2 | 3 | 6 | 7 | 2 | 3 | 6 | 7 |
| 8 | 9 | 12 | 13 | 8 | 9 | 12 | 13 |
| 10 | 11 | 14 | 15 | 10 | 11 | 14 | 15 |

# Z / 1074

len = 2 ^ (size - 1)
sq = len * len

(0,0) (len, 0)

(0, len)

| | | | |
|---|---|---|---|
| 0 + 0 * sq | 1 + 1 * sq | 0 + 1 * sq | 1 + 1 * sq |
| 2 + 0 * sq | 3 + 0 * sq | 2 + 1 * sq | 3 + 1 * sq |
| 0 + 2 * sq | 1 + 2 * sq | 0 + 3 * sq | 1 + 3 * sq |
| 2 + 2 * sq | 3 + 2 * sq | 2 + 3 * sq | 3 + 3 * sq |

# Z / 1074

(0,0) (len, 0)

(0, len)

| | | | |
|---|---|---|---|
| 0 + 0 * sq | 1 + 1 * sq | 0 + 1 * sq | 1 + 1 * sq |
| 2 + 0 * sq | 3 + 0 * sq | 2 + 1 * sq | 3 + 1 * sq |
| 0 + 2 * sq | 1 + 2 * sq | 0 + 3 * sq | 1 + 3 * sq |
| 2 + 2 * sq | 3 + 2 * sq | 2 + 3 * sq | 3 + 3 * sq |

```
ll dnc(ll cur, ll cy, ll cx){
    // Base Case
    if(cur == 1) return 2 * cy + cx;

    // 현재 정사각형의 길이의 절반
    ll len = (1ll << (cur - 1));

    // 왼쪽 dx = 0, 오른쪽 dx = 1
    // 위쪽 dy = 0, 아래쪽 dy = 1
    ll dy = cy / len, dx = cx / len;

    // 현재 정사각형의 1/4의 크기에 위치를 곱해서 더함
    return dnc(cur - 1, cy % len, cx % len) + (2 * dy + dx) * len * len;
}
```

# Z / 1074

## 시간복잡도

$T(n) = T(n - 1) + O(1)$

-> $O(n)$

```
ll dnc(ll cur, ll cy, ll cx){
    // Base Case
    if(cur == 1) return 2 * cy + cx;

    // 현재 정사각형의 길이의 절반
    ll len = (1ll << (cur - 1));

    // 왼쪽 dx = 0, 오른쪽 dx = 1
    // 위쪽 dy = 0, 아래쪽 dy = 1
    ll dy = cy / len, dx = cx / len;

    // 현재 정사각형의 1/4의 크기에 위치를 곱해서 더함
    return dnc(cur - 1, cy % len, cx % len) + (2 * dy + dx) * len * len;
}
```

# Z / 1074

## C++

```cpp
#include <iostream>
using namespace std;
using ll = long long;
ll n, m, k;
```

```cpp
int main(){
    cin >> n >> m >> k;
    cout << dnc(n, m, k);
    return 0;
}
```

```cpp
ll dnc(ll cur, ll cy, ll cx){
    // Base Case
    if(cur == 1) return 2 * cy + cx;

    // 현재 정사각형의 길이의 절반
    ll len = (1ll << (cur - 1));

    // 왼쪽 dx = 0, 오른쪽 dx = 1
    // 위쪽 dy = 0, 아래쪽 dy = 1
    ll dy = cy / len, dx = cx / len;

    // 현재 정사각형의 1/4의 크기에 위치를 곱해서 더함
    return dnc(cur - 1, cy % len, cx % len) + (2 * dy + dx) * len * len;
}
```

# Z / 1074

## Python

```python
n, m, k = map(int, input().split())
print(dnc(n, m, k))
```

```python
def dnc(cur: int, cy: int, cx: int) -> int:
    # Base Case
    if cur == 1:
        return 2 * cy + cx

    # 현재 정사각형의 길이의 절반
    len = 1 << (cur - 1)

    # 왼쪽 dx = 0, 오른쪽 dx = 1
    # 위쪽 dy = 0, 아래쪽 dy = 1
    dy = cy // len
    dx = cx // len

    # 현재 정사각형의 1/4의 크기에 위치를 곱해서 더함
    return dnc(cur - 1, cy % len, cx % len) + (2 * dy + dx) * len * len
```

# 질문?

# 별 찍기 - 10 / 2447

## 백준 2447 / https://www.acmicpc.net/problem/2447

### 문제

재귀적인 패턴으로 별을 찍어 보자. N이 3의 거듭제곱(3, 9, 27, ...)이라고 할 때, 크기 N의 패턴은 N×N 정사각형 모양이다.

크기 3의 패턴은 가운데에 공백이 있고, 가운데를 제외한 모든 칸에 별이 하나씩 있는 패턴이다.

```
***
* *
***
```

N이 3보다 클 경우, 크기 N의 패턴은 공백으로 채워진 가운데의 (N/3)×(N/3) 정사각형을 크기 N/3의 패턴으로 둘러싼 형태이다. 예를 들어 크기 27의 패턴은 예제 출력 1과 같다.

### 입력

첫째 줄에 N이 주어진다. N은 3의 거듭제곱이다. 즉 어떤 정수 k에 대해 $N=3^k$이며, 이때 $1 \le k < 8$이다.

# 별 찍기 - 10 / 2447

dnc(size, y, x) =

위치가 (x, y)일 때 별을 찍어야 하는 곳으로 분할

Base Case

size == 1

별을 찍으면 됨

```
bool a[MAX][MAX];

void dnc(ll cur, ll cy, ll cx){
    // Base Case
    if(cur == 1){
        a[cy][cx] = 1;
        return;
    }
}
```

a[i][j] = 1이면 별을 찍고 아니면 공백 출력

# 별 찍기 - 10 / 2447

dnc(size, y, x) =

위치가 (x, y)일 때 별을 찍어야 하는 곳으로 분할

size > 1

가운데를 제외한 곳에 별을 찍어야 함

-> 가운데를 제외하고 분할

```
void dnc(ll cur, ll cy, ll cx){
    // Base Case
    if(cur == 1){
        a[cy][cx] = 1;
        return;
    }

    for(int i = 0;i < 3;i++){
        for(int j = 0;j < 3;j++){
            // 가운데면 건너 뜀
            if(i == 1 && j == 1) continue;
            // 분할 후 정사각형 크기
            ll nxt = cur / 3;

            // 분할
            dnc(nxt, cy + i * nxt, cx + j * nxt);
        }
    }
}
```

# 별 찍기 - 10 / 2447

dnc(size, y, x) =

위치가 (x, y)일 때 별을 찍어야 하는 곳으로 분할

마지막으로 a[i][j] == 1 일 때 * 아니면 공백 출력

```
int main(){
    cin >> n; dnc(n, 0, 0);
    for(int i = 0;i < n;i++){
        for(int j = 0;j < n;j++) cout << (a[i][j] ? '*' : ' ');
        cout << "\n";
    }

    return 0;
}
```

# 별 찍기 - 10 / 2447

## 시간복잡도

$T(n) = 8 * T(n / 3) + O(1)$

-> n ^ log3 8 -> 약 n^1.9

```
void dnc(ll cur, ll cy, ll cx){
    // Base Case
    if(cur == 1){
        a[cy][cx] = 1;
        return;
    }

    for(int i = 0;i < 3;i++){
        for(int j = 0;j < 3;j++){
            // 가운데면 건너 뜀
            if(i == 1 && j == 1) continue;
            // 분할 후 정사각형 크기
            ll nxt = cur / 3;

            // 분할
            dnc(nxt, cy + i * nxt, cx + j * nxt);
        }
    }
}
```

# 별 찍기 - 10 / 2447

## C++

```cpp
#include <iostream>
using namespace std;
using ll = long long;

const ll MAX = 7010;
ll n, m, k;
bool a[MAX][MAX];
```

```cpp
void dnc(ll cur, ll cy, ll cx){
    // Base Case
    if(cur == 1){
        a[cy][cx] = 1;
        return;
    }

    for(int i = 0;i < 3;i++){
        for(int j = 0;j < 3;j++){
            // 가운데면 건너 뜀
            if(i == 1 && j == 1) continue;
            // 분할 후 정사각형 크기
            ll nxt = cur / 3;

            // 분할
            dnc(nxt, cy + i * nxt, cx + j * nxt);
        }
    }
}
```

```cpp
int main(){
    cin >> n; dnc(n, 0, 0);
    for(int i = 0;i < n;i++){
        for(int j = 0;j < n;j++) cout << (a[i][j] ? '*' : ' ');
        cout << "\n";
    }

    return 0;
}
```

# 별 찍기 - 10 / 2447

## Python

```python
import sys
sys.setrecursionlimit(10**7)
input = sys.stdin.readline

MAX = 7010
a = [[False] * MAX for _ in range(MAX)]
```

```python
n = int(input())
dnc(n, 0, 0)

for i in range(n):
    for j in range(n):
        print('*' if a[i][j] else ' ', end = '')
    print()
```

```python
def dnc(cur, cy, cx):
    # Base Case
    if cur == 1:
        a[cy][cx] = True
        return

    nxt = cur // 3
    for i in range(3):
        for j in range(3):
            # 가운데면 건너 뜀
            if i == 1 and j == 1:
                continue

            # 분할
            dnc(nxt, cy + i * nxt, cx + j * nxt)
```

# 5주차 - 그리디

# 그리디 알고리즘

현재 상태에서 가장 좋은 선택을 연속적으로 하는 알고리즘

모든 문제가 현재 상태에서 가장 좋은 선택이
전체에서도 가장 좋은 선택임은 보장할 수 없음

-> 현재 상태에서 가장 좋은 선택이
전체에서도 가장 좋은 선택임을 증명

# 동전 0 / 11047

## 백준 11047 / https://www.acmicpc.net/problem/11047

### 문제

준규가 가지고 있는 동전은 총 N종류이고, 각각의 동전을 매우 많이 가지고 있다.

동전을 적절히 사용해서 그 가치의 합을 K로 만들려고 한다. 이때 필요한 동전 개수의 최솟값을 구하는 프로그램을 작성하시오.

### 입력

첫째 줄에 N과 K가 주어진다. ($1 \le N \le 10$, $1 \le K \le 100,000,000$)

둘째 줄부터 N개의 줄에 동전의 가치 $A_i$가 오름차순으로 주어진다. ($1 \le A_i \le 1,000,000$, $A_1 = 1$, $i \ge 2$인 경우에 $A_i$는 $A_{i-1}$의 배수)

### 출력

첫째 줄에 K원을 만드는데 필요한 동전 개수의 최솟값을 출력한다.

# 동전 0 / 11047

예제 입력 1 복사

```
10 4200
1
5
10
50
100
500
1000
5000
10000
50000
```

예제 출력 1 복사

```
6
```

4200 = 1000 * 4 + 200 * 2 -> 6

# 동전 0 / 11047

예제 입력 2 복사

```
10 4790
1
5
10
50
100
500
1000
5000
10000
50000
```

예제 출력 2 복사

```
12
```

4790 = 1000 * 4 + 500 * 1 + 100 * 2 + 50 * 1 + 10 * 4 -> 12

# 동전 0 / 11047

4200 = 1000 * 4 + 200 * 2 -> 6
4790 = 1000 * 4 + 500 * 1 + 100 * 2 + 50 * 1 + 10 * 4 -> 12

최소의 동전을 쓰기 위해서는 가장 큰 동전부터 사용하면 됨

5000원의 동전과 1000의 동전이 있다고 할 때
10000원을 만들기 위해서는 5000원은 2개로 만들 수 있지만
1000원은 10개가 필요함

-> 항상 큰 동전부터 사용하는 것이 이득

# 동전 0 / 11047

이 문제는 동전들의 관계가 서로 배수, 약수이기 때문에
이와 같은 그리디 알고리즘이 성립

EX) 1, 3, 4의 동전으로 6을 만들 때
그리디 알고리즘 -> 4, 1, 1 선택 -> 3
정답 -> 3, 3 선택 -> 6

이는 나중에 배울 DP로 해결 가능

# 동전 0 / 11047

C++

```cpp
#include <iostream>
using namespace std;
using ll = long long;

const ll MAX = 11;
ll n, m, a[MAX];
```

```cpp
int main(){
    cin >> n >> m;
    for(int i = 1;i <= n;i++) cin >> a[i];

    ll result = 0;
    for(int i = n;i >= 1;i--){
        ll div = m / a[i];
        result += div;
        m -= div * a[i];
    }

    cout << result;
    return 0;
}
```

# 동전 0 / 11047

## Python

```python
import sys
input = sys.stdin.readline

n, m = map(int, input().split())
a = [int(input()) for _ in range(n)]

result = 0
for i in range(n - 1, -1, -1):
    div = m // a[i]
    result += div
    m -= div * a[i]

print(result)
```

# 질문?

# ATM / 11399

## 백준 11399 / https://www.acmicpc.net/problem/11399

### 문제

인하은행에는 ATM이 1대밖에 없다. 지금 이 ATM앞에 N명의 사람들이 줄을 서있다. 사람은 1번부터 N번까지 번호가 매겨져 있으며, i번 사람이 돈을 인출하는데 걸리는 시간은 $P_i$분이다.

사람들이 줄을 서는 순서에 따라서, 돈을 인출하는데 필요한 시간의 합이 달라지게 된다. 예를 들어, 총 5명이 있고, $P_1 = 3$, $P_2 = 1$, $P_3 = 4$, $P_4 = 3$, $P_5 = 2$ 인 경우를 생각해보자. [1, 2, 3, 4, 5] 순서로 줄을 선다면, 1번 사람은 3분만에 돈을 뽑을 수 있다. 2번 사람은 1번 사람이 돈을 뽑을 때 까지 기다려야 하기 때문에, 3+1 = 4분이 걸리게 된다. 3번 사람은 1번, 2번 사람이 돈을 뽑을 때까지 기다려야 하기 때문에, 총 3+1+4 = 8분이 필요하게 된다. 4번 사람은 3+1+4+3 = 11분, 5번 사람은 3+1+4+3+2 = 13분이 걸리게 된다. 이 경우에 각 사람이 돈을 인출하는데 필요한 시간의 합은 3+4+8+11+13 = 39분이 된다.

줄을 [2, 5, 1, 4, 3] 순서로 줄을 서면, 2번 사람은 1분만에, 5번 사람은 1+2 = 3분, 1번 사람은 1+2+3 = 6분, 4번 사람은 1+2+3+3 = 9분, 3번 사람은 1+2+3+3+4 = 13분이 걸리게 된다. 각 사람이 돈을 인출하는데 필요한 시간의 합은 1+3+6+9+13 = 32분이다. 이 방법보다 더 필요한 시간의 합을 최소로 만들 수는 없다.

줄을 서 있는 사람의 수 N과 각 사람이 돈을 인출하는데 걸리는 시간 $P_i$가 주어졌을 때, 각 사람이 돈을 인출하는데 필요한 시간의 합의 최솟값을 구하는 프로그램을 작성하시오.

### 입력

첫째 줄에 사람의 수 N($1 \le N \le 1,000$)이 주어진다. 둘째 줄에는 각 사람이 돈을 인출하는데 걸리는 시간 $P_i$가 주어진다. ($1 \le P_i \le 1,000$)

### 출력

첫째 줄에 각 사람이 돈을 인출하는데 필요한 시간의 합의 최솟값을 출력한다.

# ATM / 11399

예제 입력 1 복사

```
5
3 1 4 3 2
```

예제 출력 1 복사

```
32
```

1 2 3 3 4 순서대로 돈을 인출하면
1 +
1 + 2 +
1 + 2 + 3 +
1 + 2 + 3 + 3 +
1 + 2 + 3 + 3 + 4
= 1 + 3 + 6 + 9 + 13 = 32

# ATM / 11399

1 2 3 3 4 순서대로 돈을 인출하면
= 1 + 3 + 6 + 9 + 13 = 32

인출 하는데 걸리는 시간이 작은 사람부터 배치하면 될 것 같음

# ATM / 11399

증명

인출 하는데 걸리는 시간이 P1, P2인 사람 2명이 있다고 생각하자
P1 -> P2 순으로 인출하면 P1 + P1 + P2 = 2 * P1 + P2
P2 -> P1 순으로 인출하면 P2 + P2 + P1 = 2 * P2 + P1

두 식에 P1 + P2를 빼면
P1 -> P2는 P1, P2 -> P1은 P2만 남음
P1, P2 중 작은 것 부터 보면 됨

# ATM / 11399

C++

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
using ll = long long;

const ll MAX = 1010;
ll n, a[MAX];

int main(){
    cin >> n;
    for(int i = 1;i <= n;i++) cin >> a[i];
    sort(a + 1, a + n + 1);

    ll sum = 0, result = 0;
    for(int i = 1;i <= n;i++){
        sum += a[i];
        result += sum;
    }

    cout << result;

    return 0;
}
```

# ATM / 11399

## Python

```python
import sys
input = sys.stdin.readline

n = int(input())
a = list(map(int, input().split()))
a.sort()

sum = 0
result = 0
for x in a:
    sum += x
    result += sum

print(result)
```

# 질문?

# 회의실 배정 / 11399

## 백준 1931 / https://www.acmicpc.net/problem/1931

### 문제

한 개의 회의실이 있는데 이를 사용하고자 하는 N개의 회의에 대하여 회의실 사용표를 만들려고 한다. 각 회의 I에 대해 시작시간과 끝나는 시간이 주어져 있고, 각 회의가 겹치지 않게 하면서 회의실을 사용할 수 있는 회의의 최대 개수를 찾아보자. 단, 회의는 한번 시작하면 중간에 중단될 수 없으며 한 회의가 끝나는 것과 동시에 다음 회의가 시작될 수 있다. 회의의 시작시간과 끝나는 시간이 같을 수도 있다. 이 경우에는 시작하자마자 끝나는 것으로 생각하면 된다.

### 입력

첫째 줄에 회의의 수 N($1 \le N \le 100,000$)이 주어진다. 둘째 줄부터 N+1 줄까지 각 회의의 정보가 주어지는데 이것은 공백을 사이에 두고 회의의 시작시간과 끝나는 시간이 주어진다. 시작 시간과 끝나는 시간은 $2^{31}$-1보다 작거나 같은 자연수 또는 0이다.

### 출력

첫째 줄에 최대 사용할 수 있는 회의의 최대 개수를 출력한다.

# 회의실 배정 / 11399

## 예제 입력 1 복사

```
11
1 4
3 5
0 6
5 7
3 8
5 9
6 10
8 11
8 12
2 13
12 14
```

## 예제 출력 1 복사

```
4
```

## 힌트

(1,4), (5,7), (8,11), (12,14) 를 이용할 수 있다.

# 회의실 배정 / 11399

생각 해 볼만한 접근

1. 잡아먹는 시간이 작은 순으로 정렬
2. 시작 시간이 빠른 순으로 정렬
3. 종료 시간이 빠른 순으로 정렬

# 회의실 배정 / 11399

1. 잡아먹는 시간이 작은 순으로 정렬

반례)

(1, 4), (3, 5), (4, 7)

정렬을 하면 다음과 같음

(3, 5), (1, 4), (4, 7)

# 회의실 배정 / 11399

2. 시작 시간이 빠른 순으로 정렬

반례)

(1, 10), (2, 3), (4, 5)

정렬을 하면 다음과 같음

(1, 10), (2, 3), (4, 5)

# 회의실 배정 / 11399

### 3. 종료 시간이 빠른 순으로 정렬

(1, 4), (3, 5), (0, 6), (5, 7),
(3, 8), (5, 9), (6, 10), (8, 11),
(8, 12), (2, 13), (12, 14)

예제 입력 1 복사

```
11
1 4
3 5
0 6
5 7
3 8
5 9
6 10
8 11
8 12
2 13
12 14
```

예제 출력 1 복사

```
4
```

# 회의실 배정 / 11399

증명

위의 알고리즘은 앞에서부터
선택 할 수 있는 구간만 선택 함

2개의 구간 P1, P2가 있고
P1의 종료 시간이 P2의 종료 시간보다 작음

P1

P2

# 회의실 배정 / 11399

증명


항상 종료 시간이 빠른 구간을 선택 하는 것이
뒤에 선택 할 수 있는 구간의 크기가 더 큼

# 회의실 배정 / 11399

C++

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
using ll = long long;

const ll MAX = 101010;
ll n;
pair <ll, ll> a[MAX];
```

```cpp
int main(){
    cin >> n;
    for(int i = 1;i <= n;i++) cin >> a[i].second >> a[i].first;
    sort(a + 1, a + n + 1);

    ll la = -1, result = 0;
    for(int i = 1;i <= n;i++){
        if(a[i].second < la) continue;
        la = a[i].first; result++;
    }

    cout << result;
    return 0;
}
```

# 회의실 배정 / 11399

Python

```python
import sys
input = sys.stdin.readline

n = int(input())
a = []
for _ in range(n):
    s, e = map(int, input().split())
    a.append((e, s))

a.sort()

la = -1
result = 0
for e, s in a:
    if s < la:
        continue
    la = e
    result += 1

print(result)
```

# 질문?

# 과제

동전 0 – https://www.acmicpc.net/problem/11047

ATM - https://www.acmicpc.net/problem/11399

회의실 배정 – https://www.acmicpc.net/problem/1931

파일 합치기 3 – https://www.acmicpc.net/problem/13975

구두 수선공 - https://www.acmicpc.net/problem/14908

# 고생하셨습니다