# 

# Heap

## 0. heapq

heap은 최대값이나 최소값을 뽑아내기에 최적화된 자료구조이다.
파이썬에서는 heap 구현할 때, heapq라는 내장 라이브러리를 사용한다.

```
import heapq
heapq.heappush(heap, 4)
heapq.heappush(heap, 1)
heapq.heappush(heap, 7)
heapq.heappush(heap, 3)
print(heap)
#[1, 3, 7, 4]
```

출력했을 때는 순서가 뒤죽박죽 섞인 것처럼 보이지만

```
print(heapq.heappop(heap))
print(heap)
# 1
# [3, 4, 7]
```

heapq.heappop이라는 함수로 최소값을 빼낼 수 있다.

heapq.heapify라는 함수로 기존 리스트를 힙으로 변환하는것도 가능하다.

```
heap = [4, 1, 7, 3, 8, 5]
heapq.heapify(heap)
print(heap)
# [1, 3, 5, 4, 8, 7]
```

heapq라이브러르는 우선순위 큐를 구현하는데에도 사용할 수 있다.

아래와 같이 원소가 두개인 튜플이나 리스트를  heapq.heappush를 하면, 0번 째 원소가 작은 것부터 빠져나온다.

```
import heapq

h = []
heapq.heappush(h, (3, "third"))
heapq.heappush(h, (2, "second"))
heapq.heappush(h, (4, "fourth"))
heapq.heappush(h, (1, "first"))

print(heapq.heappop(h))
print(heapq.heappop(h))
print(heapq.heappop(h))
print(heapq.heappop(h))

#(1, 'first')
#(2, 'second')
#(3, 'third')
#(4, 'fourth')
```

# 1. 백준 1715 카드정렬하기

## 문제

정렬된 두 묶음의 숫자 카드가 있다고 하자. 각 묶음의 카드의 수를 A, B라 하면 보통 두 묶음을 합쳐서 하나로 만드는 데에는 A+B 번의 비교를 해야 한다. 이를테면, 20장의 숫자 카드 묶음과 30장의 숫자 카드 묶음을 합치려면 50번의 비교가 필요하다.

매우 많은 숫자 카드 묶음이 책상 위에 놓여 있다. 이들을 두 묶음씩 골라 서로 합쳐나간다면, 고르는 순서에 따라서 비교 횟수가 매우 달라진다. 예를 들어 10장, 20장, 40장의 묶음이 있다면 10장과 20장을 합친 뒤, 합친 30장 묶음과 40장을 합친다면 (10 + 20) + (30 + 40) = 100번의 비교가 필요하다. 그러나 10장과 40장을 합친 뒤, 합친 50장 묶음과 20장을 합친다면 (10 + 40) + (50 + 20) = 120 번의 비교가 필요하므로 덜 효율적인 방법이다.

N개의 숫자 카드 묶음의 각각의 크기가 주어질 때, 최소한 몇 번의 비교가 필요한지를 구하는 프로그램을 작성하시오.

## 입력

첫째 줄에 N이 주어진다. (1 ≤ N ≤ 100,000) 이어서 N개의 줄에 걸쳐 숫자 카드 묶음의 각각의 크기가 주어진다. 숫자 카드 묶음의 크기는 1,000보다 작거나 같은 양의 정수이다.

## 출력

첫째 줄에 최소 비교 횟수를 출력한다.

## 예시 및 출력

예제 입력 1 복사

```
3
10
20
40
```

예제 출력 1 복사

```
100
```

## 구현 아이디어

1. 가장 작은 값과 그 다음으로 작은 값을 Heap에서 꺼낸다.
2. 더한 뒤 다시 Heap에 넣는다.(더한 값을 total 변수에 저장한다.)
3. Heap에 하나만 남을 때까지 1~2를 반복한다.
4. total을 출력한다.

## 코드

```
import heapq

n = int(input())

heap = []
result = 0
for _ in range(n):
  heapq.heappush(heap, int((input())))

while len(heap) != 1:
  one = heapq.heappop(heap)
  two = heapq.heappop(heap)
  result += one+two
  heapq.heappush(heap, one+two)

print(result)
```

---

# 2. 프로그래머스 디스크 컨트롤러

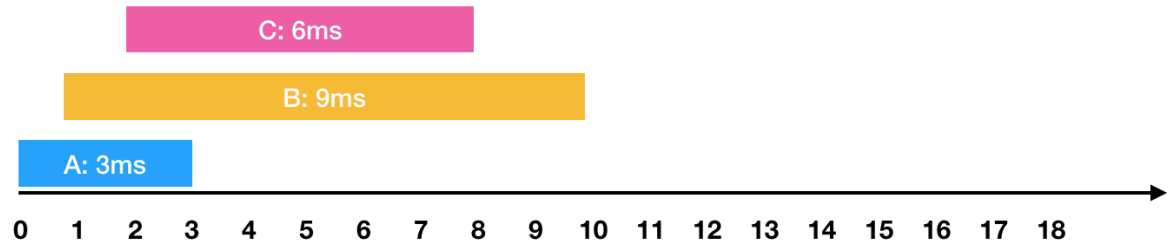
## 문제

하드디스크는 한 번에 하나의 작업만 수행할 수 있습니다. 디스크 컨트롤러를 구현하는 방법은 여러 가지가 있습니다. 가장 일반적인 방법은 요청이 들어온 순서대로 처리하는 것입니다.

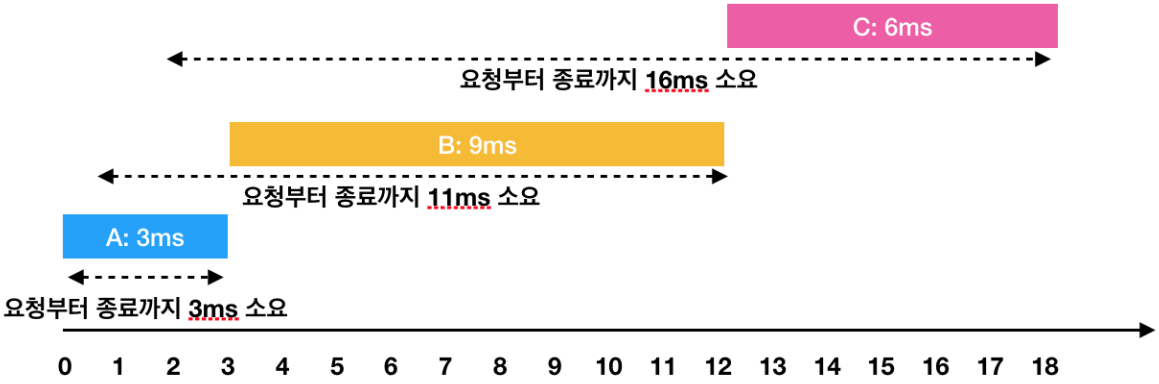
예를들어

```
- 0ms 시점에 3ms가 소요되는 A작업 요청
- 1ms 시점에 9ms가 소요되는 B작업 요청
- 2ms 시점에 6ms가 소요되는 C작업 요청
```

와 같은 요청이 들어왔습니다. 이를 그림으로 표현하면 아래와 같습니다.



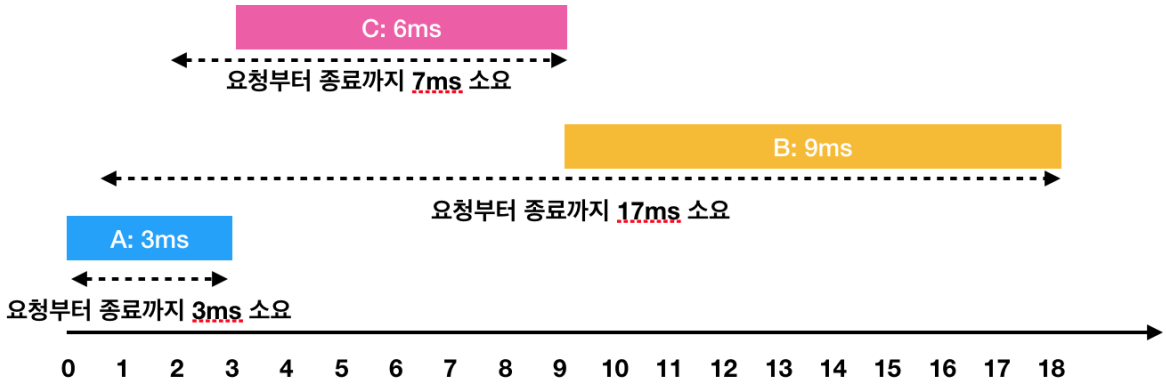

한 번에 하나의 요청만을 수행할 수 있기 때문에 각각의 작업을 요청받은 순서대로 처리하면 다음과 같이 처리 됩니다.




```
- A: 3ms 시점에 작업 완료 (요청에서 종료까지 : 3ms)
- B: 1ms부터 대기하다가, 3ms 시점에 작업을 시작해서 12ms 시점에 작업 완료(요청에서 종료까지 : 11ms)
- C: 2ms부터 대기하다가, 12ms 시점에 작업을 시작해서 18ms 시점에 작업 완료(요청에서 종료까지 : 16ms)
```

이 때 각 작업의 요청부터 종료까지 걸린 시간의 평균은 10ms(= (3 + 11 + 16) / 3)가 됩니다.

하지만 A → C → B 순서대로 처리하면




```
- A: 3ms 시점에 작업 완료 (요청에서 종료까지 : 3ms)
- B: 1ms부터 대기하다가, 3ms 시점에 작업을 시작해서 12ms 시점에 작업 완료(요청에서 종료까지 : 11ms)
- C: 2ms부터 대기하다가, 12ms 시점에 작업을 시작해서 18ms 시점에 작업 완료(요청에서 종료까지 : 16ms)
```

이렇게 A → C → B의 순서로 처리하면 각 작업의 요청부터 종료까지 걸린 시간의 평균은 9ms(= (3 + 7 + 17) / 3)가 됩니다.

각 작업에 대해 [작업이 요청되는 시점, 작업의 소요시간]을 담은 2차원 배열 jobs가 매개변수로 주어질 때, 작업의 요청부터 종료까지 걸린 시간의 평균을 가장 줄이는 방법으로 처리하면 평균이 얼마가 되는지 return 하도록solution 함수를 작성해주세요. (단, 소수점 이하의 수는 버립니다)

# 입력 및 출력

## 예시 및 출력

입출력 예

| jobs | return |
|---|---|
| [[0, 3], [1, 9], [2, 6]] | 9 |

입출력 예 설명

문제에 주어진 예와 같습니다.

- 0ms 시점에 3ms 걸리는 작업 요청이 들어옵니다.
- 1ms 시점에 9ms 걸리는 작업 요청이 들어옵니다.
- 2ms 시점에 6ms 걸리는 작업 요청이 들어옵니다.

## 구현 아이디어

1. 가장 실행시간이 짧은 작업부터 해야 평균시간이 짧아진다.
2. 작업을 하고나서는 start와 end를 지정해서 그 사이에 들어온 작업들을 우선순위 큐에 넣어둔다.
3. 모든 작업을 마치고 작업의 개수만큼 나누면 평균이 나온다.

## 코드

```
import heapq

def solution(jobs):
    answer = 0
    end = 0
    i = 0
    start = -1
    heap = []

    while i < len(jobs):
        for j in jobs:
            if start < j[0] <= end:
                heapq.heappush(heap, [j[1], j[0]])
        if heap:
            cur = heapq.heappop(heap)
            start = end
            end += cur[0]
            answer += end - cur[1]
            i +=1
        else:
            end += 1
    return answer // len(jobs)
```