

True False 잘 이용하기(2920)

```

1  arr=list(map(int,input().split()))
2
3  ascending=True
4  descending=True
5
6  for i in range(len(arr)-1):
7      if arr[i+1]>arr[i]:
8          descending=False
9      elif arr[i+1]<arr[i]:
10         ascending=False
11
12     if ascending:
13         print("ascending")
14     elif descending:
15         print("descending")
16     else:
17         print("mixed")

```

이런식으로 처음에는 True 로 사용했다가 리스트를 차근차근 진행해나가면서 반대의 경우가 존재한다면 False로 바꿔준다.

절반으로 나눠서 계속 찾기 (2798)

```

14  hap=sorted(hap)
15  start=0
16  end=len(hap)-1
17  while start<=end:
18      mid = (start + end) // 2
19      if hap[mid]<=M:
20          result=hap[mid]
21          start=mid+1
22      elif hap[mid]>M:
23          end=mid-1
24
25  print(result)

```

항상 start와 end를 바꿔줄때는 1을 더하고 빼는것을 까먹지 말자 !

1. 카드 중 3개씩 뽑는 모든 경우의 수는 $C(n, 3)$ 이며, n 은 최대 100입니다.

2. 따라서 단순히 3중 반복문으로 모든 경우의 수를 확인하여 문제를 해결할 수 있습니다.

$$\frac{n(n-1)(n-2)}{3!} \Rightarrow \text{경우}$$

$$\frac{100 \cdot 99 \cdot 98}{6} \approx 1,617,000$$

M = 21



파이썬은 1초에 2000만 정도 연산을 하는데 n 이 최대 100일 경우는 n^3 이므로 최대 100만정도 밖에 안되므로 충분히 완전탐색으로 탐색 가능하다. (단순히 3중 반복문으로 가능하다)

카드를 차근차근 뽑을때는 3중 반복문 쓸때 (2798)

```

1 N,M=map(int,input().split())
2 arr=list(map(int,input().split()))
3 hap=set()
4
5 for i in range(len(arr)):
6     for j in range(len(arr)):
7         if arr[i]==arr[j]:
8             continue
9         for k in range(len(arr)):
10            if arr[i]==arr[k] or arr[j]==arr[k]:
11                continue
12            hap.add(arr[i]+arr[j]+arr[k])
13
14 hap=sorted(hap)
15 start=0
16 end=len(hap)-1
17 while start<=end:
18     mid = (start + end) // 2
19     if hap[mid]<=M:
20         result=hap[mid]
21         start=mid+1
22     elif hap[mid]>M:
23         end=mid-1
24
25 print(result)

```

(왼쪽은 내가 푼것)
(아래거는 나동빈이 푼거)

```

n, m = list(map(int, input().split(' ')))
data = list(map(int, input().split(' ')))

result = 0
length = len(data)

count = 0
for i in range(0, length):
    for j in range(i + 1, length):
        for k in range(j + 1, length):
            sum_value = data[i] + data[j] + data[k]
            if sum_value <= m:
                result = max(result, sum_value)

print(result)

```

굳이 위처럼 카드를 i, j, k 를 할 때 일일이 0부터 할필요가 없다. 이미 선택됐으므로 앞에 경우는 건너뛰고 넘어가면 된다.

또한 나눠서 찾을 필요 없이 조건에 맞다면 다음식을 이용하여 더한값을 계산하면서 바로 구하는것도 가능하다.

result=max(result,sum_value)

for i in range(10,8) 이런식으로 앞에거가 크면 자동으로 아무런 실행을 하지않는다 따로 빼줄 필요가 없다 !

n,m=[10,9] 이런식으로 되어있으면 미지수와 리스트 데이터 개수가 같으면 자동으로 그 변수에 리스트 위치에 맞는값이 지정된다.

```
1 result=["1","2","3","4"] 1234
2 print("".join(result))
```

```
1 result=["1","2","3","4"] 1 2 3 4
2 print(" ".join(result))
```

```
1 result=["1","2","3","4"] 1
2 print("\n".join(result)) 2
                             3
                             4
```

“사이에입력하고싶은값”join(리스트이름) 하면은 리스트 안에 있는 원소들을 각각 사이에 “사이에 입력하고싶은값을 넣어서 출력시켜준다

(1874)

```
1 n=int(input())
2 count=1
3 stack=[]
4 result=[]
5
6 for i in range(n):
7     number=int(input())
8     while count<=number:
9         stack.append(count)
10        result.append("+")
11        count+=1
12    if stack[-1]==number:
13        stack.pop()
14        result.append("-")
15    else:
16        print("NO")
17        exit(0)
18
19 print("\n".join(result))
```

. 한번 넣은 숫자는 다시 못넣을 경우는 count로 1부터 하나씩 올리면된다. 예를들어 n 이 8이라 면 8까지 넣고 난후에는 더이상 count가 9가 되어 아무리해도 number 가 9보다 커질수가 없어 while 문안에 들어갈수 없고 이미 꺼내진것들은 다빠졌기 때문에 남은것들만 쌓여 있는데 그중 에서 스택에 마지막거랑 꺼내는것이 같다면 꺼 낼수 있는것이고 만약에 같지 않다면 꺼내지 못하는것이기 때문에 no를 출력하게 된다.

만약 8이 되기전에 5까지 넣고 5를 꺼냈는데 그 다음에 3이라는 숫자가 나온다면은 숫자가 더 작기때문에 더이상 넣지도 못하고 마지막 숫자 도 같지 않기때문에 더이상 꺼내지 못하고 no를 출력하게 되는것이다.

숫자가 지금까지 넣은 숫자보다 크다면 더 넣으 면 되지만 그거보다 작다면 그냥 꺼내는수밖에

없는데 꺼내는경우의 숫자가 stack 의 마지막거랑 같지않다면 꺼내지 못하는것이다.

인덱스와 데이터 한번에 저장하는 방법 (enumerate) (1966)

```
n_list=[[int(x),idx] for idx,x in enumerate(input().split())]
```

```
1 t=int(input())
2
3 for i in range(t):
4     n,m=list(map(int,input().split()))
5     data=list(map(int,input().split()))
6     data2=[]
7     for j in range(len(data)):
8         data2.append([data[j],j])
9
10    count=0
11    while True:
12        length = len(data2)
13        max_value=data2[0][0]
14        for l in range(1,length):
15            if max_value<data2[l][0]:
16                max_value=data2[l][0]
17
18        while data2[0][0]!=max_value:
19            data2.append(data2[0])
20            del data2[0]
21        count+=1
22
23        if data2[0][1]==m:
24            print(count)
25            break
26        del data2[0]
```

이렇게 일일이 찾을 필요없이 그냥

max함수 사용하면된다.

max(queue,key=lambda x:x[0])

x[0] 기준중에 제일 큰값요소니까

(,) 튜플형태로 뽑아서 마지막에[0]을 써준것이
다.

```
test_case = int(input())
for _ in range(test_case):
    n, m = list(map(int, input().split(' ')))
    queue = list(map(int, input().split(' ')))
    queue = [(i, idx) for idx, i in enumerate(queue)]

    count = 0
    while True:
        if queue[0][0] == max(queue, key=lambda x: x[0])[0]:
            count += 1
            if queue[0][1] == m:
                print(count)
                break
            else:
                queue.pop(0)
        else:
            queue.append(queue.pop(0))
```

이렇게 리스트를 형성한다음에

for idx, i in enumerate(list) 를 통하여 인덱스번호
호랑 리스트요소를 같이 저장할수 있다.

del queue[0] 말고 queue.pop(0) 이렇게 써도된
다.

queue.append(queue.pop(0)) 을 통하여 0번째
데이터를 추가하면서 0번째데이터를 제거할수
있다.

```

16 t=int(input())
17
18 for _ in range(t):
19     data=list(input())
20     length=len(data)
21     result=[]
22     save=[]
23     for j in range(length):
24         if not result and (data[j]=="<" or data[j]=="-"):
25             continue
26
27         elif not save and data[j]==">":
28             continue
29
30         elif data[j]=="<" and result:
31             save.append(result.pop())
32
33         elif data[j]==">" and save:
34             result.append(save.pop())
35
36         elif data[j]=="-" and result:
37             result.pop()
38
39     else:
40         result.append(data[j])
41     print("".join(result)+"".join(save[::-1]))

```

있으면 진행하고 없으면 진행 안하면 된다. 따로 continue 로 뺄 필요없이 조건문안에 조건문을 하나 더만들어줘서 편하게 생각할수도 있다.

(5397)

```

test_case = int(input())

for _ in range(test_case):
    left_stack = []
    right_stack = []
    data = input()
    for i in data:
        if i == '-':
            if left_stack:
                left_stack.pop()
        elif i == '<':
            if left_stack:
                right_stack.append(left_stack.pop())
        elif i == '>':
            if right_stack:
                left_stack.append(right_stack.pop())
        else:
            left_stack.append(i)
    left_stack.extend(reversed(right_stack))
    print(''.join(left_stack))

```

1	a=[1,2,3,4]	[1, 2, 3, 4, 8, 7, 6, 5]
2	b=[8,7,6,5]	[1, 2, 3, 4, 8, 7, 6, 5, 5, 6, 7, 8]
3	a.extend(b)	[8, 7, 6, 5]
4	print(a)	[5, 6, 7, 8]
5	a.extend(reversed(b))	두개의 리스트를 합쳐줄때는 extend 사용
6	print(a)	
7	print(b)	리스트를 뒤집어 줄때는 sorted sort 처럼 reversed 는 원본 유지
8	b.reverse()	지 b.reverse는 원본자체를 뒤집는다.
9	print(b)	

리스트를 + 로 단순히 합치기 가능하다.

```
1 a=[1,2,3]    [1, 2, 3, 3, 4, 5]
2 b=[3,4,5]
3 print(a+b)
```

(10930)

문제 : 문자열 S 가 주어졌을 때, SHA-256 해시값을 구하는 프로그램을 작성하시오.

입력 : 첫째 줄에 문자열 S 가 주어진다. S 는 알파벳 대문자와 소문자,

그리고 숫자로만 이루어져 있으며, 길이는 최대 50이다.

출력 : 첫째 줄에 S 의 SHA-256 해시값을 출력한다.

```
import hashlib
```

입력값의 encode() 를 만든다.

```
input_data = input()
```

hashlib.sha256(인코드 값).hexdigest()

```
encoded_data = input_data.encode()
```

로 해쉬 생성

```
result = hashlib.sha256(encoded_data).hexdigest()
```

```
print(result)
```

1. hashlib의 sha256 함수를 이용하면 SHA 256 해시를 구할 수 있습니다.
2. hashlib.sha256(문자열의 바이트 객체).hexdigest() : 해시 결과 문자열

(1920) 문제

1. 특정 정수의 등장 여부를 간단히 체크하면 됩니다.
2. Python에서는 dictionary 자료형을 해시처럼 사용할 수 있습니다.
3. 본 문제는 set 자료형을 이용해 더욱 간단히 풀 수 있습니다.

2번말고 3번으로 풀었다.

```
1 n=int(input())
2 arr=[int(x) for x in input().split()]
3 m=int(input())
4 arr2=[int(x) for x in input().split()]
5 arr.sort()
6
7 for i in arr2:
8     start=0
9     end=len(arr)-1
10
11     while start<=end:
12         mid = int((start + end) / 2)
13         if arr[mid]<=i:
14             result=arr[mid]
15             start=mid+1
16         elif arr[mid]>i:
17             end=mid-1
18         elif arr[mid]==i:
19             print(1)
20             break
21
22     if start>end:
23
24         print(0)
```

```
n = int(input())
array = set(map(int, input().split()))
m = int(input())
x = list(map(int, input().split()))
```

```
for i in x:
    if i not in array:
        print('0')
    else:
        print('1')
```

중복여부를 지워주기 위해 set을 사용하면 시간차이가 엄청나게 난다. 반드시 어떤 요소가 어떤 리스트 안에 있는 요소인지 체크하기 위해서는 어떤 리스트를 set 으로 바꿔주어 시간 절약을 하자.

```

14 import sys
15 t=int(sys.stdin.readline())
16
17 for _ in range(t):
18     friend_list=[]
19     f = int(sys.stdin.readline())
20     for i in range(f):
21         a,b=sys.stdin.readline().split()
22         count=0
23         check=0
24         if not friend_list[0]:
25             count+=2
26             friend_list[0].append(a)
27             friend_list[0].append(b)
28             check=2
29         else:
30             for j in range(len(friend_list)):
31                 if check==2:
32                     break
33                 elif a not in friend_list[j] and b not in friend_list[j]:
34                     continue
35                 elif a in friend_list[j] and b in friend_list[j]:
36                     check=2
37                     count+=len(friend_list[j])
38                 elif a in friend_list[j] and check==0:
39                     check+=1
40                     count+=len(friend_list[j])
41                     c=j
42                     acheck=True
43                 elif b in friend_list[j] and check==0:
44                     check+=1
45                     count+=len(friend_list[j])
46                     c=j
47                     bcheck=True
48                 elif a in friend_list[j] and check==1:
49                     check=2
50                     count+=len(friend_list[j])
51                     friend_list[c]=friend_list[c]+friend_list[j]
52                     del friend_list[j]
53                 elif b in friend_list[j] and check==1:
54                     check=2
55                     count+=len(friend_list[j])
56                     friend_list[c]=friend_list[c]+friend_list[j]
57                     del friend_list[j]
58                     bcheck=True
59
60             if check<2:
61                 if check==1 and acheck:
62                     count=2
63                     friend_list[c].append(b)
64                 elif check==1 and bcheck:
65                     count=2
66                     friend_list[c].append(a)
67                 elif check==0:
68                     friend_list.append([a,b])
69                     count=2
70     print(count)

```

(4195) 내가 푼방식

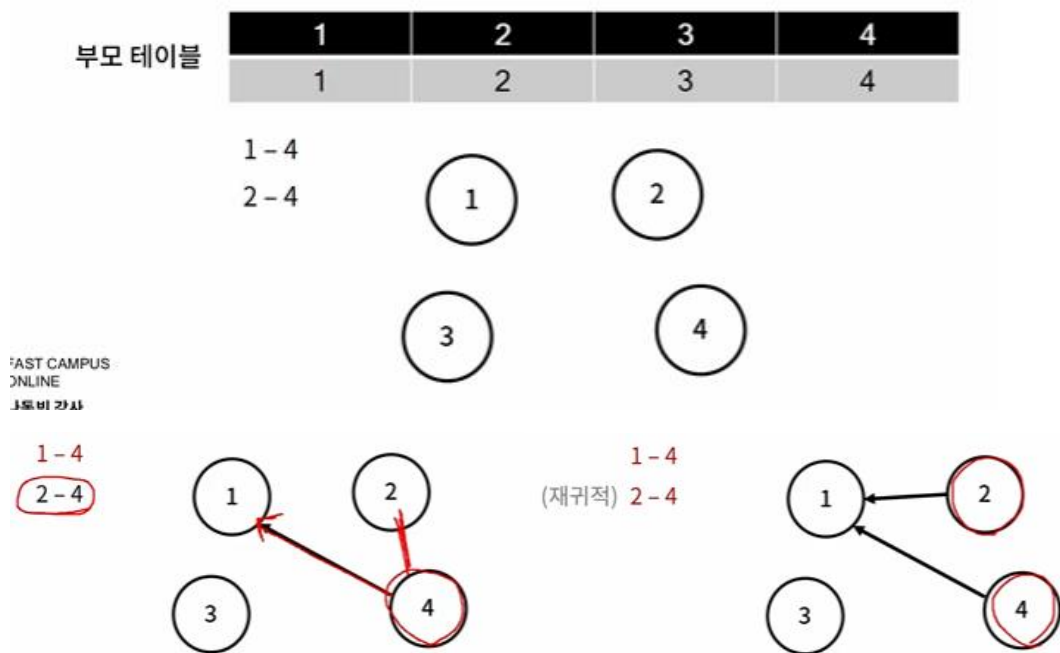
리스트를 일일이 하나하나 찾아가면서 풀기 때문에 시간초과가 날수 밖에 없다.

1. 해시를 활용한 Union-Find 알고리즘을 이용해 문제를 풀 수 있습니다.
2. Python에서는 dictionary 자료형을 해시처럼 사용할 수 있습니다.

Union-Find 합집합 찾기

합 집합 찾기(Union-Find) 알고리즘

- 원소들의 연결 여부를 확인하는 알고리즘입니다.
- (그림 가정) 더 작은 원소를 부모로 삼도록 설정



실질적으로는 왼쪽이 아니라 오른쪽으로 된다.(재귀적으로 생각해보면)

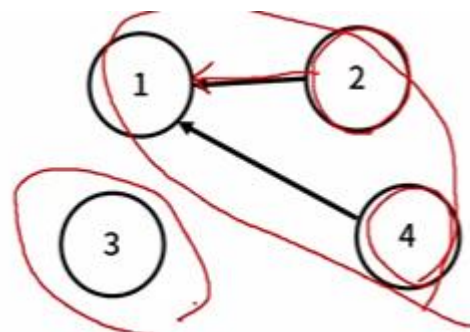
항상 큰거에서 작은거를 가르키도록 하기 때문에



가르키는것이 같으면 같은 집합으로 본다.

$\{1, 2, 4\}, \{3\}$

결과: {1, 2, 4}, {3}



합 집합 찾기(Union-Find) 알고리즘

```
def find(x):
    if x == parent[x]:
        return x
    else:
        p = find(parent[x])
        parent[x] = p
        return parent[x]

def union(x, y):
    x = find(x)
    y = find(y)

    parent[y] = x
```

```
parent = []

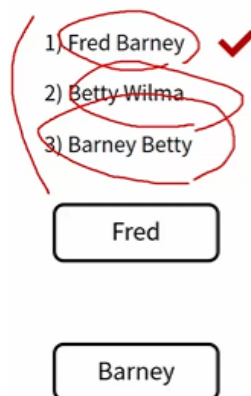
for i in range(0, 5):
    parent.append(i)

union(1, 4)
union(2, 4)

for i in range(1, len(parent)):
    print(find(i), end=' ')
```

find 는 그 부모를 재귀적으로 찾아서 그 부모(제일작은값)를 가져오는것이다

오른쪽값의 부모를 x로 설정하겠다 이렇게 해도된다. (위에서는 설명을 위해 작은값을 부모로 한
다고 한것이다.)



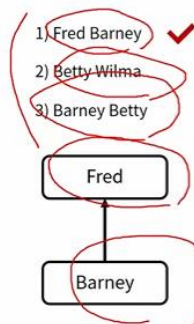
[부모 테이블]

Fred	→	Fred
Barney	→	Barney

[네트워크 크기]

Fred	1
Barney	1

I 문제 풀이 핵심 아이디어



출력: 2

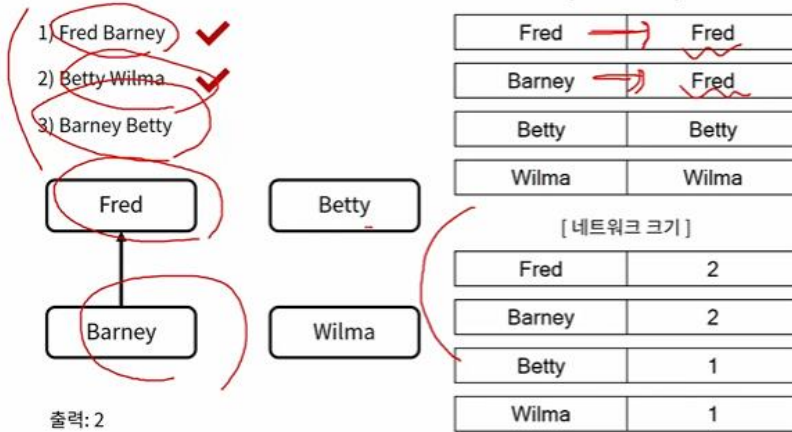
[부모 테이블]

Fred	→	Fred
Barney	→	Fred

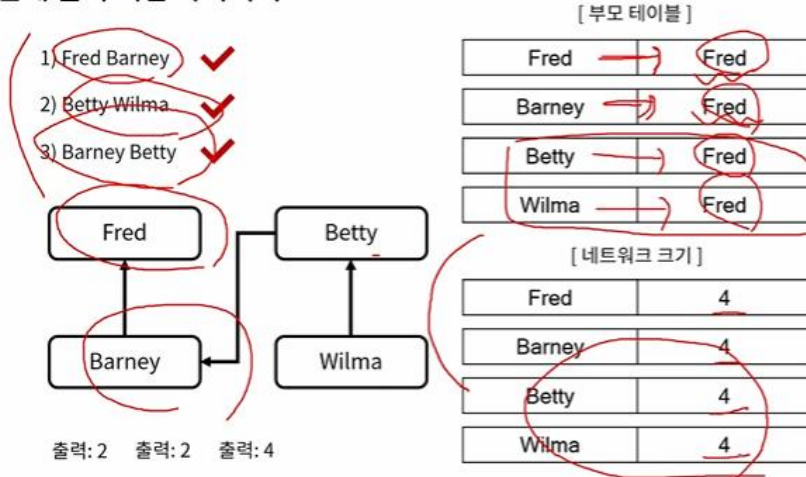
[네트워크 크기]

Fred	2
Barney	2

I 문제 풀이 핵심 아이디어



I 문제 풀이 핵심 아이디어



```
def find(x):
    if x == parent[x]:
        return x
    else:
        p = find(parent[x])
        parent[x] = p
        return parent[x]

def union(x, y):
    x = find(x)
    y = find(y)
    if x != y:
        parent[y] = x
        number[x] += number[y]
```

```
test_case = int(input())

for _ in range(test_case):
    parent = dict()
    number = dict()

    f = int(input())

    for _ in range(f):
        x, y = input().split(' ')

        if x not in parent:
            parent[x] = x
            number[x] = 1
        if y not in parent:
            parent[y] = y
            number[y] = 1

        union(x, y)

    print(number[find(x)])
```

꼬리 물기라고 생각 오른쪽것이 왼쪽거를 꼬리를 문다고 생각

기존 y 에 꼬리가 달려있으면 그것도 다가지고 x를 무는것이다.

여기서 x , y 는 자기가 가지고 있는 요소들중에 제일 머리쪽으로 이동해야한다. 머리로 무는거니까

머리에는 꼬리가 여러개가 달릴수도 있다 여러곳이 물수도 있으니까

(10989)

파이썬은 1초에 2000만번 연산 가능 기본정렬 알고리즘(파이썬 기본 정렬라이브러리 sorted ,sort)은 $n \log n$ 정도 가능하니까

n 이 100만 까지 가능하다. 하지만 이문제는 n이 1000만 가지이므로 기본정렬알고리즘으로는 구하지 못한다.

데이터개수가 많은 대신에 데이터범위가 좁다는 특징이 있다 이런 특징이 있을때 쓸수있는 알고리즘인 계수정렬이 있다.

계수 정렬(Counting Sort) 알고리즘

인덱스 번호 그자체를 숫자라고 생각하는 알고리즘이다.

- 배열의 인덱스를 특정한 데이터의 값으로 여기는 정렬 방법입니다.
- 배열의 크기는 데이터의 범위를 포함할 수 있도록 설정합니다.
- 데이터가 등장한 횟수를 셉니다.

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

- 배열의 크기는 데이터의 범위를 포함할 수 있도록 설정합니다.
- 데이터가 등장한 횟수를 셉니다.

<u>0</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>
0	0	0	0	0	0	0	0	0	0

[3, 3, 1]

3이 2번나왔으니까 인덱스 3 자리에 2를 넣는다. 인덱스 7자리에는 1

예시 데이터: 759031629148052

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

예시 데이터: 759031629148052

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	1	0	0

예시 데이터: 759031629148052

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	0	1	0	0

예시 데이터: 759031629148052

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	0	1	0	1

⋮

예시 데이터: 759031629148052

0	1	2	3	4	5	6	7	8	9
2	2	1	1	1	2	1	1	1	2


예시 데이터: 759031629148052

0	1	2	3	4	5	6	7	8	9
2	2	2	1	1	2	1	1	1	2

계수 정렬(Counting Sort) 알고리즘

예시 데이터: 759031629148052

0	1	2	3	4	5	6	7	8	9
2	2	2	1	1	2	1	1	1	2

 정렬 결과: 001122345567899

유의사항: 데이터의 개수가 많을 때 파이썬에서는 `sys.stdin.readline()`를 사용해야 합니다.

`input()` 보다 애가 더빠르다.

pypy3 는 python3보다 메모리는 더잡아먹지만 시간이 더빠르다

이문제는 메모리제한이 걸려있으므로 python3로 제출하기.

```
import sys

n = int(sys.stdin.readline())
array = [0] * 10001

for i in range(n):
    data = int(sys.stdin.readline())
    array[data] += 1

for i in range(10001):
    if array[i] != 0:
        for j in range(array[i]):
            print(i)
```

[0]*10001 을해주면 [0,0,0,0, ... 0] 이게 10001개 생긴다.

피보나치수열(2747)

앞에계산이 반복될때 전에 데이터를 저장해두고 계산한다.(동적계획법 DP)

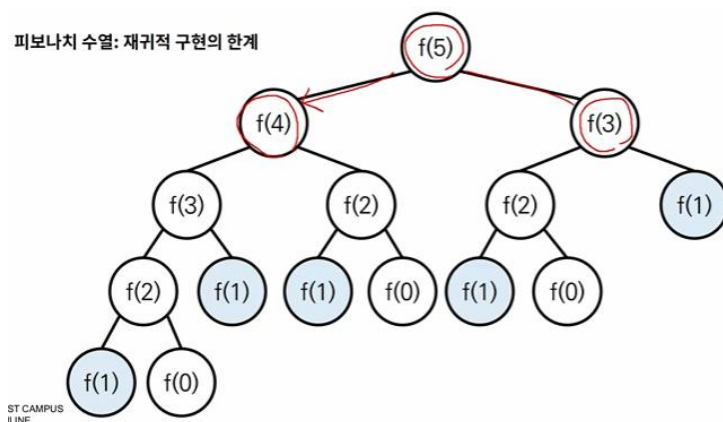
실패 소스코드

```
def fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fibonacci(n - 1) + fibonacci(n - 2)

print(fibonacci(int(input())))
```

이렇게 하면 실패한다.

피보나치 수열: 재귀적 구현의 한계



동일한 값들을 계산을 계속 반복하기 때문이다. 동일한 값을 한번 계산해 두면은 그값을 저장해두면은 더 빠르게 계산할수 있다.

(위에서부터 내려오면 엄청나게 오래 걸린다.)

(아래서부터 올라가자 !)

(상향식접근법)

```

11 def fibo(data):
12     cache=[0]*(data+1)
13     cache[0]=0
14     cache[1]=1
15     for i in range(2,data+1):
16         cache[i]=cache[i-2]+cache[i-1]
17     return cache[data]
18 n=int(input())
19 print(fibo(n))

```

왼쪽 방법은 상향식 접근방법

동적계획법으로 풀것이다.

아래방법은 나동빈이 쓴 방법

재귀함수가 아닌 단순 반복문으로
풀것

```

n = int(input())

a, b = 0, 1

while n > 0:
    a, b = b, a + b
    n -= 1

print(a)

```

```

1 a,b=1,2
2 a=b
3 b=a+b
4 # a,b=b,a+b
5 print(a,b)

```

2 4

```

1 a,b=1,2
2 # a=b
3 # b=a+b
4 a,b=b,a+b
5 print(a,b)

```

2 3

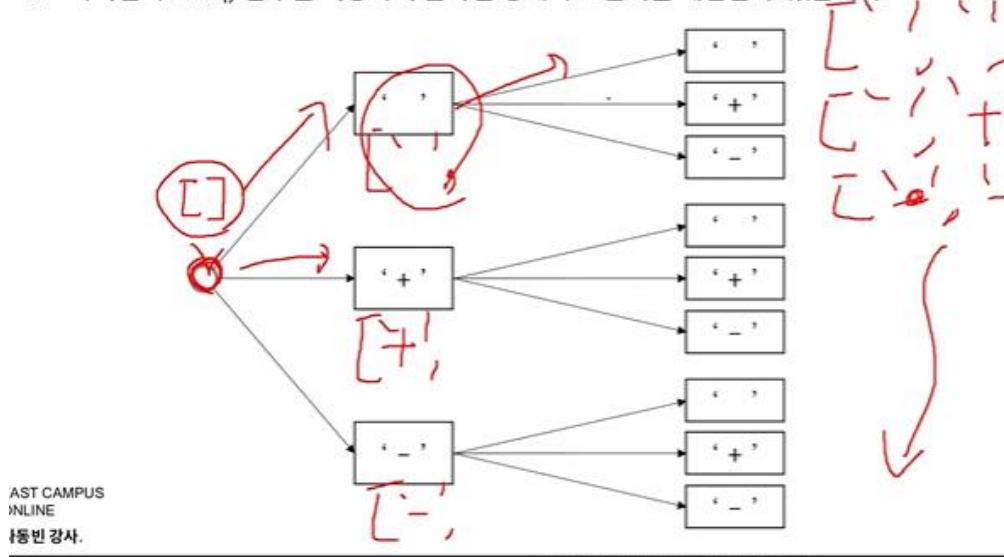
왼쪽과 달리 오른쪽처럼 한줄에 다쓰면 그값이 변경되기 전에 값들이 각각 적용된다.

(7490)

1. 자연수 N의 범위($3 \leq N \leq 9$)가 매우 한정적이므로 완전 탐색으로 문제를 해결할 수 있습니다.
2. 수의 리스트와 연산자 리스트를 분리하여 모든 경우의 수를 계산합니다.

N의 범위가 작으면은 완전 탐색으로 가능하다

1. 가능한 모든 경우를 고려하여 연산자 리스트를 만드는 것이 관건입니다. (재귀 함수 이용)
2. 파이썬의 `eval()` 함수를 이용하여 문자열 형태의 표현식을 계산할 수 있습니다.



`eval` ==> 문자열을 그냥 계산해준다. `eval("1+2+3")` 하면 6 이 나오게된다.

1	<code>print(eval("1+2+3"))</code>	6
2	<code>print(eval("12*3"))</code>	36

```
import copy

def recursive(array, n):
    if len(array) == n:
        operators_list.append(copy.deepcopy(array))
        return
    array.append(' ')
    recursive(array, n)
    array.pop()

    array.append('+')
    recursive(array, n)
    array.pop()

    array.append('-')
    recursive(array, n)
    array.pop()
```

`copy.deepcopy(array)`를 이용하면 `array`에 있는 내용이 그대로 `copy` 되서 `operators_list`에 담기게 된다.

반드시 `deepcopy`를 해줘야한다. 안그러면 재귀반복을 하면서 `array`를 공유하기때문에 `array`가 하나씩 사라지면 마지막에는 `[]` 빈공간만 남게 된다.


```

test_case = int(input())

for _ in range(test_case):
    operators_list = []
    n = int(input())
    recursive([], n - 1)

    integers = [i for i in range(1, n + 1)]

    for operators in operators_list:
        string = ""
        for i in range(n - 1):
            string += str(integers[i]) + operators[i]
        string += str(integers[-1])
        if eval(string.replace(" ", "")) == 0:
            print(string)
    print()

```

```

1 operator_list=[]
2
3 def recursive(array,n):
4     if len(array)==n:
5         operator_list.append(array)
6         return
7     array.append(" ")
8     recursive(array,n)
9     array.pop()
10
11     array.append("+")
12     recursive(array, n)
13     array.pop()
14
15     array.append("-")
16     recursive(array, n)
17     array.pop()
18
19 recursive([],3)
20 print(operator_list)

```

recursive(array, n) 에서 어떤 리스트에 데이터를 추가할때는 함수가 최종적으로 끝나는 시점의 데이터를 집어넣는다 . 즉 함수가 끝난 시점의 array 값을 append한다(append만 한경우)

copy.deepcopy() 를 하면 그순간의 값을 집어넣어 준다. 최종적인 array 가 무슨값이든 상관없이.

copy.deepcopy(array) 를 안한경우는 아래와 같이 나온다.

```
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
```

```

19 recursive([1],3)
20 print(operator_list)

```

```
[[1], [1], [1], [1], [1], [1], [1], [1], [1]]
```

```

1  b=[20]
2
3  def cal2(data):
4      if len(data)==3:
5          b.append(data)
6          return
7          data.append(3)
8          cal2(data)
9          print(data)
10     if len(data)>1:
11         del data[-1]
12
13     a=[]
14     cal2(a)
15     print(a)
16     # a 값도 함수에 영향을 받아 최종적인 값으로 변하게 된다.
17     print(b)
18     # []가 추가 된것이 아닌 cal2 함수의 (data)==a 의 최종값 [3] 이 추가 되었다.

```

[3, 3, 3] append 를 하면 len(data)==3 인순간의 [3,3,3] 이 아닌 함수가 끝나는 시점의
 [3, 3] 데이터인 data=[3] 이 추가된다.
 [3]
 [3]
 [20, [3]]

```

20  import copy
21  b=[20]
22
23  def cal2(data):
24      if len(data)==3:
25          b.append(copy.deepcopy(data))
26          return
27          data.append(3)
28          cal2(data)
29          print(data)
30          if len(data)>1:
31              del data[-1]
32
33  a=[]
34  cal2(a)
35  print(a)
36
37  print(b)

```

[3, 3, 3]
 [3, 3]
 [3]
 [3]
 [20, [3, 3, 3]]

나머지 값은 다 똑같지만 append 되는 데이터값이 [3,3,3]으로 그 시점의 값을 추가한다. 마지막 시점의 data 값이 아닌

리스트는 따로 리턴을 하지않아도 함수안에서 실행된 결과값이 저장되어있다. 하지만 리스트가 아닌 그냥 요소 값은 따로 리턴을 하지 않으면 값이 저장되지 않는다. 리턴하고 그 함수를 따로

변수로 지정해줘야지 값이 저장된다.

```
38 b=[]
39 def cal3(data):
40     if data==3:
41         b.append(data)
42         return
43     data+=1
44     cal3(data)
45     if data>1:
46         data-=1
47     print(data)
48     a=data
49     return a
50
51 a=0
52 cal3(a)
53 print("-----")
54 print(a)
55 print(b)
```

2

1

1

0

[3]

또한 리스트가 아닌경우는 copy.deepcopy()를
하지 않더라도 그 요소값이 그대로 저장된다.

```
38 b=[]
39 def cal3(data):
40     if data==3:
41         b.append(data)
42         return
43     data+=1
44     cal3(data)
45     if data>1:
46         data-=1
47     print(data)
48     a=data
49     return a
50
51 a=0
52 cal3(a)
53 print("-----")
54 print(a)
55 print(b)
56 print(cal3(a))
57
```

2

1

1

0

[3]

2

1

1

1

a는 변하지 않는다.

함수를 따로 변수로 지정해줘야한다.

<pre> 1 b=[20] 2 3 def cal2(data): 4 if data==[3,3,3]: 5 b.append(data) 6 return 7 data.append(3) 8 9 cal2(data) 10 print(data) 11 if len(data)>1: 12 del data[-1] 13 14 a=[] 15 cal2(a) 16 print(a) 17 # a 값도 함수에 영향을 받아 최종적인 값으로 변하게 된다. 18 print(b) 19 # [] 가 추가 된것이 아닌 cal2 함수의 (data)==a 의 최종값 [3] 이 추가 되었다. </pre>	<pre> [3, 3, 3] [3, 3] [3] [3] [20, [3]] </pre> <p>data==[3,3,3] 일때라 고 해도 리스트는 똑 같은 결과가 나온다 항상 함수 마지막에 나오는 최종적인 값이 append 된다.</p>
--	---

그 시점의 값을 append 하고 싶으면 항상 copy.deepcopy(리스트) 이용하기!

- 어떤 중복을 허락한 여러개를 데이터를 이용해서 리스트를 만드는방법

지정된데이터 (a,b,c)를 가지고 중복을 허락해서 3개 또는 4개 의 모든 배열을 만들어봐라.

다음과 같이 이용할수 있다!

<pre> 1 import copy 2 3 arr=[] 4 5 def plu(data,n): 6 if len(data)==n: 7 arr.append(copy.deepcopy(data)) 8 return 9 data.append('a') 10 plu(data,n) 11 data.pop() 12 13 data.append('b') 14 plu(data,n) 15 data.pop() 16 17 data.append('c') 18 plu(data,n) 19 data.pop() 20 21 plu([],3) 22 print(arr) 23 arr=[] 24 plu([],4) 25 print(arr) </pre>	<pre> [('a', 'a', 'a'), ('a', 'a', 'b'), ('a', 'a', 'c'), ('a', 'b', 'a'), ('a', 'b', 'b'), ('a', 'b', 'c'), ('a', 'c', 'a'), ('a', 'c', 'b'), ('a', 'c', 'c'), ('b', 'a', 'a'), ('b', 'a', 'b'), ('b', 'a', 'c'), ('b', 'b', 'a'), ('b', 'b', 'b'), ('b', 'b', 'c'), ('b', 'c', 'a'), ('b', 'c', 'b'), ('b', 'c', 'c'), ('c', 'a', 'a'), ('c', 'a', 'b'), ('c', 'a', 'c'), ('c', 'b', 'a'), ('c', 'b', 'b'), ('c', 'b', 'c'), ('c', 'c', 'a'), ('c', 'c', 'b'), ('c', 'c', 'c')] </pre>
--	--

(7490) 내가 다시한번 쪽 풀어써본 코드

```
1 import copy
2
3 def recursive(array, n):
4     if len(array) == n:
5         buho.append(copy.deepcopy(array))
6         return
7
8     array.append(" ")
9     recursive(array, n)
10    array.pop()
11
12    array.append("+")
13    recursive(array, n)
14    array.pop()
15
16    array.append("-")
17    recursive(array, n)
18    array.pop()
19
20    t=int(input())
21    for _ in range(t):
22        n=int(input())
23        arr=[x for x in range(1,n+1)]
24        buho=[]
25        recursive([],n-1)
26
27        for i in range(len(buho)):
28            string=""
29            for j in range(len(buho[i])):
30                string+=str(arr[j])+buho[i][j]
31            string+=str(arr[-1])
32            if eval(string.replace(" ",""))==0:
33                print(string)
34        print("")
```

주어지는 n 이 많으면 `input()` 말고 `sys.stdin.readline()`을 사용하자 ! (2751)

첫째 줄에 수의 개수 $N(1 \leq N \leq 1,000,000)$

나 같은 정수이다. 수는 중복되지 않는다.

- 데이터의 개수가 최대 1,000,000개입니다.
- 시간 복잡도 $O(N \log N)$ 의 정렬 알고리즘을 이용해야 합니다.
- 고급 정렬 알고리즘 (병합 정렬, 퀵 정렬, 힙 정렬 등) 을 이용하여 문제를 해결할 수 있습니다.
- 혹은 파이썬의 기본 정렬 라이브러리를 이용하여 문제를 풀 수 있습니다.
- 메모리가 허용된다면, 되도록 Python 3보다는 PyPy 3를 선택하여 코드를 제출합니다.

$\log_2 N \Rightarrow \text{㉠}$

$2^{\text{㉠}} = 1,000$

2의 10제곱이 1024 이므로 약 1000이라고 했을때

1000 곱하기 1000 이 1000000 백만 이므로 약 2의 20제곱 이 백만이다. 그러므로 $\log N$ 이 20이 되도록 해야한다.

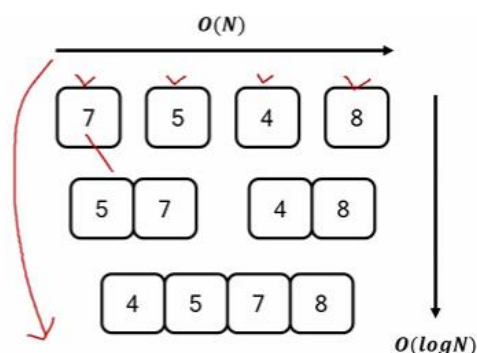
$n \log n$ 이면 백만 곱하기 20 이니까 2000만 1초에 약 2000 만번 수행하니까 제한시간이 2초면 $n \log n$ 의 알고리즘을 사용해야한다.

정렬 알고리즘 중에 $n \log n$ 인 정렬알고리즘은 병합정렬 퀵정렬 힙정렬 등이 있다. 하지만 퀵정렬 은 최악의 경우가 n^2 이 될수도 있으므로 병합정렬과 힙정렬을 이용해서 푸는것이 좋다.

웬만하면 시험문제에서 만나면 기본정렬라이브러리 `sort` 를 사용한다. 그리고 메모리가 허용되면 `pypy3`로 사용하기 .

병합 정렬(Merge Sort) 알고리즘

- 분할 정복 (Divide & Conquer) 방식을 이용합니다.
- 절반씩 합치면서 정렬하면, 전체 리스트가 정렬됩니다.
- 시간 복잡도 $O(N \log N)$ 을 보장합니다.



사실상 병합정렬 쓰나 `sorted` 로 쓰나 시간 차이는 없다 그냥 파이썬 기본 라이브러리를 사용하 자 시간초과나면 `input()` 대신에 `sys.stdin.readline()` 사용하기 시험에선 그냥 기본정렬라이브러리 사용하기 ! 또는 `pypy3` 로 제출하기 !

