광주 인공지능 사관학교

인공지능을 위한 알고리즘

LIKE LION

휴식을 위하여

날씨 좋은 휴일,

인근 공원에서 독서를 하기로 했지만 공원에서는 공사가 진행되고 있었습니다.

공사 현장 근처는 소음이 크기 때문에 휴식과 독서에 적합하지 않습니다.

공사 현장은 공원에서 오직 한 군데이고, 그 위치를 (a, b)라고 합니다. 공사현장에서 R 만큼의 거리 미만은 소음이 크기 때문에 독서에 적합하지 않습니다.

또한 공원에는 휴식과 독서에 적합한 그늘이 N개 존재합니다. 각각의 그늘 위치는 (x_i, y_i) 입니다. (i번째)

이상의 정보에서 각 그늘이 독서에 적합한지 (공사 현장에서 R 이상 떨어진 그늘인지) 판별하는 코드를 작성하시오.

- 위치 (x, y)가 공사 현장에서 R 이상 떨어져있다는 조건

$$(x-a)^2+(y-b)^2 \geq R^2$$

- **각 입력 값의 범위**

$0 \leqq a \leqq 100$

$0 \leqq b \leqq 100$

$1 \leqq R \leqq 100$

$1 \leqq N \leqq 1000$

$0 \leqq x_i \leqq 100$

$0 \leqq y_i \leqq 100$

- 입력은 아래와 같은 형식으로 들어온다.

a b R # 공사 현장의 x 좌표 (a), y 좌표 (b), 공사장 소음 거리 R
N # 그늘의 수
x_1 y_1 # 그늘 1의 x 좌표 (x_1), y 좌표 (y_1)
x_2 y_2 # 그늘 2의 x 좌표 (x_2), y 좌표 (y_2)
…
x_N y_N # 그늘 N의 x 좌표 (x_N), y 좌표 (y_N)

- 입력 예제 1

20 10 10
3
25 10
20 15
70 70

- 출력 예제 1

noisy
noisy
silent

- 입력 예제 2

50 50 100
4
0 0
0 100
100 0
100 0

- 출력 예제 2

noisy
noisy
noisy
noisy

```python
def check_sound_status(a,b,r,x,y):
    if (x-a)**2 + (y-b)**2 >= r**2:
        return "silent"
    return "noisy"

a, b, r = map(int, input().split())
n = int(input())

for _ in range(n):
    x, y = map(int, input().split())
    print(check_sound_status(a,b,r,x,y))
```

약수 구하기

- 입력받은 숫자 N의 모든 약수를 출력하시오.
- 입력 예제 1

100

- 출력 예제 1

1 2 4 5 10 20 25 50 100

```python
num = int(input())
divisor_list = []

for i in range(1, num+1):
    if num % i == 0:
        divisor_list.append(i)

print(divisor_list)
```

```python
# import math

num = int(input())
# sqrt_num = int(math.sqrt(num))
sqrt_num = int(num ** (1/2))

front_divisor_list = []
rear_divisor_list = []

for i in range(1, sqrt_num+1):
    if num % i == 0:
        front_divisor_list.append(i)
        if i != int(num/i):
            rear_divisor_list.append(int(num/i))

print(front_divisor_list + rear_divisor_list[::-1])
```

최대값, 최소값 구하기

- 입력받은 정수값들 중 최대값과 최소값 순서대로 출력하시오.
- 단, 파이썬 내장 함수 max, min 및 기타 numpy, pandas 등의 라이브러리는 쓸 수 없다.

- 입력 예제 1

1 2 100 3 30

- 출력 예제 1

100 1

```python
num_list = list(map(int, input().split()))

max_num = num_list[0]
min_num = num_list[0]

for num in num_list:
    if max_num < num:
        max_num = num

    if min_num > num:
        min_num = num

print(max_num)
print(min_num)

# print(max(num_list))
# print(min(num_list))
```

- 표기법과 개념
  - 시간복잡도
  - 공간복잡도
  - 점근 표기법 (Big-O)

시간복잡도

- the time complexity is the computational complexity that describes the amount of time it takes to run an algorithm.

점근 표기법 (Big-O)

- 함수의 증가 양상을 다른 함수와의 비교로 표현하는 수론과 해석학의 방법이다.
- **알고리즘의 복잡도**를 단순화할 때나 무한급수의 뒷부분을 간소화할 때 쓰인다

점근 표기법 (Big-O)

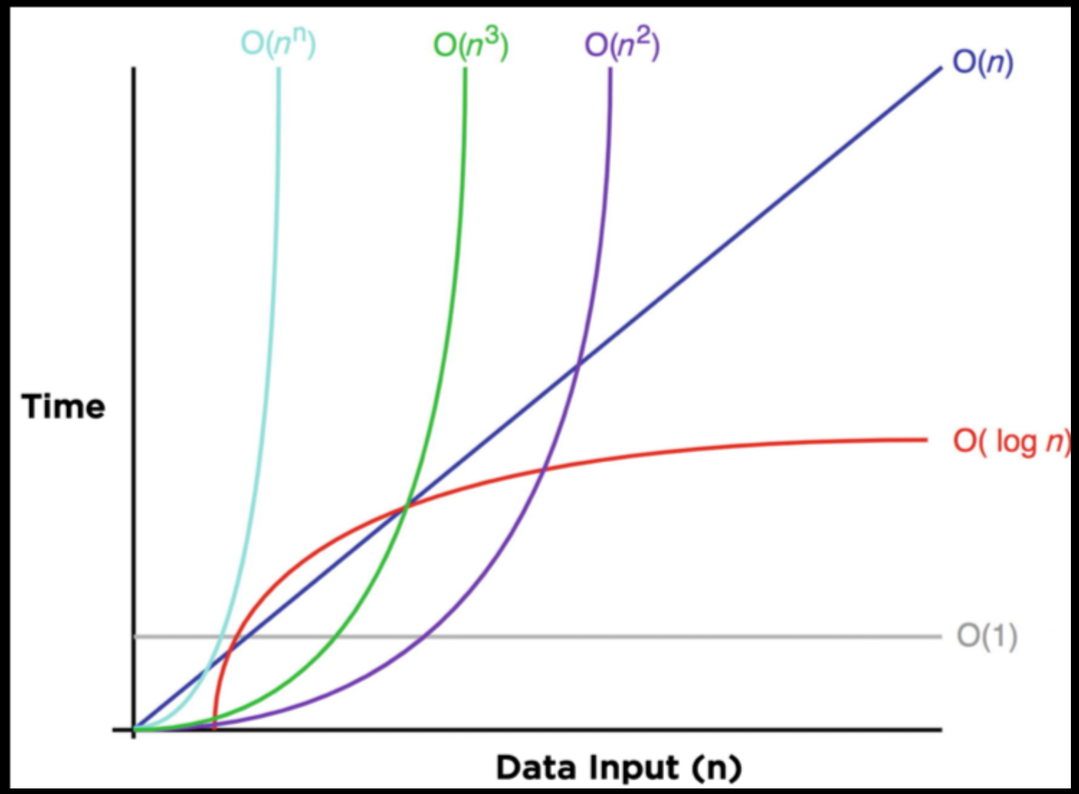
- 표현할 복잡도의 최대 변수에 해당하는 항을 제외하고 나머지 아래 차수의 항 및 모든 항의 계수를 제외시키는 방법.

```python
num = int(input())
divisor_list = []

for i in range(1, num+1):
    if num % i == 0:
        divisor_list.append(i)

print(divisor_list)
```

시간 복잡도	n					
	1	**2**	**4**	**8**	**16**	**32**
1	1	1	1	1	1	1
log N	0	1	2	3	4	5
N	1	2	4	8	16	32
N log N	0	2	8	24	64	160
N^2	1	4	16	64	256	1024
N^3	1	8	64	512	4096	32768
N^N	2	4	16	256	65536	4294967296



- $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$

```python
# import math

num = int(input())
# sqrt_num = int(math.sqrt(num))
sqrt_num = int(num ** (1/2))

front_divisor_list = []
rear_divisor_list = []

for i in range(1, sqrt_num+1):
    if num % i == 0:
        front_divisor_list.append(i)
        if i != int(num/i):
            rear_divisor_list.append(int(num/i))

print(front_divisor_list + rear_divisor_list[::-1])
```

Square root 값의 시간복잡도 비교는?

그래프로 확인해보자

- O(1) < O(log n) **< O(n^1/2) <** O(n) < O(n log n) < O(n^2) < O(n^3) < O(2^n)

```python
num_list = list(map(int, input().split()))

max_num = num_list[0]
min_num = num_list[0]

for num in num_list:
    if max_num < num:
        max_num = num

    if min_num > num:
        min_num = num

print(max_num)
print(min_num)

# print(max(num_list))
# print(min(num_list))
```

```python
def check_sound_status(a,b,r,x,y):
    if (x-a)**2 + (y-b)**2 >= r**2:
        return "silent"
    return "noisy"

a, b, r = map(int, input().split())
n = int(input())

for _ in range(n):
    x, y = map(int, input().split())
    print(check_sound_status(a,b,r,x,y))
```

공간복잡도는?

공간복잡도

- The space complexity of an algorithm or a computer program is the amount of memory space required to solve an instance of the computational problem as a function of characteristics of the input. It is the memory required by an algorithm to execute a program and produce output.

Operation	Example	Big-O	
Index	l[i]	O(1)	
Store	l[i] = 0	O(1)	
Length	len(l)	O(1)	
Append	l.append(5)	O(1)	
Pop	l.pop()	O(1)	
Clear	l.clear()	O(1)	l = []
Slice	l[a:b]	O(b-a)	l[:] : O(len(l)-0) = O(N)
Extend	l.extend(…)	O(len(…))	
Construction	list(…)	O(len(…))	
check ==, !=	l1 == l2	O(N)	
Insert	l .insert(i, v)	O(N)	
Delete	del l[i]	O(N)	
Containment	x in/not in l	O(N)	
Copy	l.copy()	O(N)	l[:] - O(N)
Remove	l.remove(…)	O(N)	
Pop	l.pop(i)	O(N)	l.pop(0) - O(N)
Extreme value	min(l)/max(l)	O(N)	
Reverse	l.reverse()	O(N)	
Iteration	for v in l:	O(N)	
Sort	l.sort()	O(N Log N)	
Multiply	k*l	O(k N)	[0,1,2,3,4] * 5 » O(N**2)

Operation	Example	Big-O	
Index	d[k]	O(1)	
Store	d[k] = v	O(1)	
Length	len(d)	O(1)	
Delete	del d[k]	O(1)	
get/setdefault	d.method	O(1)	
Pop	d.pop(k)	O(1)	
Pop item	d.popitem()	O(1)	
Clear	d.clear()	O(1)	s = {} or = dict()
View	d.keys()	O(1)	= d.values()
Construction	dict(…)	O(len(…))	
Iteration	for k in d:	O(N)	

재귀에서 다시 살펴보자.