ㅇㄱㄹㅈ

시간복잡도

알고리즘의 선능 평가를 위해 사용하는 것

특정한 크기의 입력에 대하여 알고리즘의 수행 시간 분석

빅오 표기법

알고리즘의 성능을 수학적으로 표현한 것입니다.

빠르게 증가하는 항만을 표기 하는 방법으로

예를 들어 연산 횟수가 $3N^3 + 5N^2$+ 1,000,00 인 알고리즘이 있다면

빅오 표기법으로는 $O(N^3)$ 으로 표기 할수있다.

알고리즘의 시간 복잡도는 O(N)으로, 데이터의 개수 N에 비례한다는 것을 나타냅니다.

```
num=[1,2,3,4,5];
sum = 0

for i in num:
 sum += i

print(sum)
```

수행 시간은 데이터의 개수 N에 비례할것을 예측 할수 있음

시간 복잡도 : O(N)

2중 반복문의 경우

```
num=[1,2,3,4,5];
```

```
for i in num:
    for j in num:
        dd=i*j
        print(dd)
```

간단히 계산하면 i*j 가 반복되는 횟수는 num에 입력된 갯수의 제곱만큼 반복하게 됩니다.

이럴 경우 시간 복잡도는 $O(N^2)$

빅오 표기법	복잡성	속도
O(1)	상수	1
O(log N)	로그	2
O(N)	선형	3
O(N log N)	로그선형	4
$O(n^2)$	이차	5
$O(2^n)$	지수	6

1. **O(1) - 일정한 시간**: 일정한 시간 복잡도를 갖는 작업은 일반적으로 입력 데이터의 크기에 관계없이 빠른 것으로 간주됩니다. 이는 입력 데이터의 크기가 커져도 연산에 소요되는 시간이 늘어나지 않는다는 의미이다. 예를 들어 인덱스를 통해 배열의 요소에 액세스하거나 기본 산술 연산을 수행하는 것이 포함됩니다.
2. **O(log n) - 로그 시간**: 로그 시간 복잡도가 있는 작업은 특히 대규모 데이터 세트의 경우 빠른 것으로 간주됩니다. 입력 데이터의 크기가 커짐에 따라 작업에 소요되는 시간도 늘어나지만 선형 또는 2차 시간 복잡도에 비해 훨씬 느린 속도입니다. 예로는 정렬된 배열의 이진 검색이나 특정 효율적인 트리 기반 알고리즘이 있습니다.
3. **O(n) - 선형 시간**: 선형 시간 복잡도가 있는 작업은 일반적으로 적당히 빠른 것으로 간주되지만 소요 시간은 입력 데이터의 크기에 따라 선형적으로 늘어납니다. 대규모 데이터 세트의 경우 선형 시간 알고리즘이 느려질 수 있습니다. 예를 들어 배열이나 연결 목록을 반복하여 요소를 검색하는 것이 포함됩니다.
4. **O(n log n) - 선형 시간**: 선형 시간 복잡도가 있는 작업은 일반적으로 대규모 데이터 세트의 경우 빠른 것으로 간주되지만 로그 시간 복잡도보다 느립니다. 병합 정렬, 퀵 정렬과 같은 일반적인 정렬 알고리즘은 O(n log n)의 시간 복잡도를 갖습니다.
5. $O(n^2)$ **- 2차 시간**: 2차 시간 복잡도가 있는 작업은 중간 규모에서 대규모 데이터 세트에 대해 느린 것으로 간주됩니다. 소요 시간은 입력 데이터의 크기에 따라 2차적으로 증

가하므로 대규모 데이터 세트에는 비효율적입니다. 예로는 특정 중첩 루프나 버블 정렬과 같은 비효율적인 정렬 알고리즘이 있습니다.

6. $O(2^n)$ **- 지수 시간**: 지수 시간 복잡성이 있는 작업은 특히 적당한 크기의 데이터 세트의 경우 매우 느린 것으로 간주됩니다. 입력 데이터의 크기에 따라 소요 시간이 기하급수적으로 증가하므로 대부분의 실제 시나리오에서는 실용적이지 않습니다. 예를 들면 메모가 없는 특정 재귀 알고리즘이 있습니다.

일반적인 cpu의 경우 연산 횟수가 5억이 넘어가는 경우

C언어 기준 통산 1 ~ 3 초

Pyton 기준 통상 5 ~ 15 초 가량 시간 소요 됨

만약 $O(N^3)$ 의 알고리즘을 설계한 경우, N의 값이 5,000이 넘는다면

간단하게 계산을 하면 125,000,000,000 이 나오고 파이썬이 1초에 약 5천만번을 처리할 수 있다고 가정하면 2500초가 걸린다고 예측이 가능하다.

문제를 풀때 수행시간 요구사항을 확인 하는 것이 중요

시간 제한이 1초인 문제를 푼다면 일반적으로

(파이썬이 1초에 2000만번 연산이 가능하다 가정하였을 경우)

N의 범위가 500인 경우: 시간 복잡도가 $O(N^3)$인 알고리즘을 설계하면 문제를 풀 수 있습니다.

N의 범위가 2,000인 경우: 시간 복잡도가 $O(N^2)$인 알고리즘을 설계하면 문제를 풀 수 있습니다.

N의 범위가 100,000인 경우: 시간 복잡도가 $O(NlogN)$인 알고리즘을 설계

N의 범위가 100,000인 경우: 시간 복잡도가 $O(NlogN)$인 알고리즘을 설계

N의 범위가 10,000,000인 경우: 시간 복잡도가 $O(N)$인 알고리즘을 설계

그리디 (탐욕법)은 최적의 해를 구하는 알고리즘 중 하나로, 각 단계에서 항상 최적의 선택을 하는 방식으로 문제를 해결합니다. 이 방식은 항상 최적의 결과를 보장하지는 않지만, 계산 시간을 크게 줄일 수 있어 많은 문제에서 효율적인 해결책을 제공합니다.

근사해를 구할 때 쓰이면 가장 좋으나, 최적해를 구할 때에는 역추적을 해야만 한다.

아래의 트리 구조에서 가장 최고의 합을 찾는 문제가 있다면

```
        5
      / | \
     /  |  \
   7    9    8
  /|    /\   |\
 1 3    4 5  6 8
```

일반적으로는 8 과 8을 선택해서 16을 얻겠지만

그리디적인 방법으로 보면 현재 단계에서 가장 최적을 선택해서

9와 5 를 선택한다.

이와 같은 방법이 그리디 알고리즘이라고 한다.

<문제>

점원이 카운터에서 거스름돈으로 사용할 500원 100원 50원 10원 짜리 동전이 무한히 존재한다는 가정하에 손님에게 거슬러야 할 돈이 N원 일때 거슬러주어야 할 동전의 최소 개수를 구하려면?

(단 N원은 항상 10의 배수)

입력 1260

출력 6

```
n = 1260
cut = 0
mm = [500,100,50,10]

for i in mm:

    cut += n // i

    n %= i
```

```
print(cut)
```

<문제>

한개의 회의실이 있고 사용하고자 하는 N 개의 회의에 대하여 회의실 사용표를 만들려고 합니다.

각 회의 I에 대해 시작시간과 끝나는 시간이 주어져 있고, 각 회의기 겹치지 않게 하면서 회의실을

사용할수 있는 회의의 최대 개수를 찾아봅시다.

(단 회의는 시작되면 중단없고 , 회의가 끝남과 동시에 다음 회의시 시작 되며 , 회의의 시작 시작과 끝나는 시같은 같을수도 있다 . 이경우 시작하자마자 끝나는 것으로 간주한다.)

첫째 줄에 회의의 수 N(1 ≤ N ≤ 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 231-1보다 작거나 같은 자연수 또는 0이다.

```
입력
11
1 4
3 5
```

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

출력
4
```

https://wikidocs.net/206266

<문제>

상근이는 요즘 설탕공장에서 설탕을 배달하고 있다. 상근이는 지금 사탕가게에 설탕을 정확하게 N킬로그램을 배달해야 한다. 설탕공장에서 만드는 설탕은 봉지에 담겨져 있다. 봉지는 3킬로그램 봉지와 5킬로그램 봉지가 있다.

상근이는 귀찮기 때문에, 최대한 적은 봉지를 들고 가려고 한다. 예를 들어, 18킬로그램 설탕을 배달해야 할 때, 3킬로그램 봉지 6개를 가져가도 되지만, 5킬로그램 3개와 3킬로그램 1개를 배달하면, 더 적은 개수의 봉지를 배달할 수 있다.

상근이가 설탕을 정확하게 N킬로그램 배달해야 할 때, 봉지 몇 개를 가져가면 되는지 그 수를 구하는 프로그램을 작성하시오.

첫째 줄에 N이 주어진다. (3 ≤ N ≤ 5000)

상근이가 배달하는 봉지의 최소 개수를 출력한다. 만약, 정확하게 N킬로그램을 만들 수 없다면 -1을 출력한다.

입력 18 출력 4

입력 4 출력 -1

입력 6 출력 2

입력 9 출력 3

입력 11 출력 3

```
sugar = int(input())
cnt=0
while sugar >= 0 :
    if sugar % 5 == 0:
        cnt += sugar//5
        print(cnt)
        break
    sugar -= 3
    cnt+=1
else :
    print(-1)
```