그리디

그리디 알고리즘

탐욕법으로 자주 소개되는 알고리즘이다. 여기서 탐욕적이라는 말은 **현재 상황에서 지금 당장 좋은 것만 고르는 방법**을 의미한다. 매 순간 가장 좋아 보이는 것을 선택하며, 현재의 선택이 나중에 미칠 영향은 고려하지 않는다.

보통 문제에서 '가장 큰 순서대로', '가장 작은 순서대로'와 같은 기준을 제시해주며 대체로 이 기준은 정렬 알고리즘을 사용했을 때 만족시킬 수 있으므로 정렬 알고리즘과 짝을 이뤄 출제된다.

💡 Q1. 거스름돈

문제

당신은 음식점의 계산을 도와주는 점원이다. 거스름돈으로 사용할 500, 100, 50, 10원짜리 동전이 무한히 존재한다고 가정한다. 손님에게 거슬로 줘야 할 돈이 N원일 때 거슬러 줘야 할 동전의 최소 개수를 구하라.

해설

'가장 큰 화폐 단위'부터 돈을 거슬러 주면 된다. 예를 들어 N = 1260이라면 500원 2개, 100원 2개, 50원 1개, 10원 1개로 총 6개가 된다.

코드

```
N = int(input())
coin_list = [500, 100, 50, 10]
count = 0
```

```
for coin in coin_list:
  count += N // coin
  N %= coin

print(count)
```

💡 정당성

대부분의 문제는 그리디 알고리즘으로 풀 수 없다. 하지만 거스름돈 문제처럼 **탐욕적으로 문제에 접근했을 때 정확한 답을 찾을 수 있다는 보장이 있으면** 매우 효과적이고 직관적이다.

그리디로 문제를 풀겠다고 마음먹었다면 그 해법이 정당한지 검토해야 한다. 거스름돈 문제를 그리디로 풀 수 있었던 이유는 가지고 있는 동전 중에 큰 단위가 항상 작은 단위의 배수이기 때문이다. 만약 큰 단위가 작은 단위의 배수가 아니라면 그리디로 문제를 해결할 수 없다. 만약 240원을 거슬러 줘야 하고 동전이 100원, 80원, 10원 이렇게만 있다면 80원 3개를 주는 것이 가장 좋은 방법이다. 그리디로 풀면 100원 2개, 10원 4개로 총 6개의 동전이 필요하다.

어떤 문제를 만났을 때, 바로 유형을 파악하기 어렵다면 그리디 알고리즘을 떠올리고 그리디로 문제를 풀 수 있는지 고민해봐야 한다.

💡 Q2. 큰 수의 법칙

문제

다양한 수로 이루어진 배열이 있을 대 주어진 수들을 M번 더하여 가장 큰 수를 만들어야 한다. 단, 배열의 특정 인덱스에 해당하는 수가 연속해서 K번을 초과하여 더해질 수 없다.

예를 들어 [2, 4, 5, 4, 6], M = 8, K = 3 이라면 6+6+6+5+6+6+6+5 = 46이 된다. 단, 서로 다른 인덱스에 해당하는 수가 같은 경우에도 서로 다른 것으로 간주한다. 예를 들어 [3, 4, 3, 4, 3], M = 7, K = 2 라고 하면 4+4+4+4+4+4+4 = 28이다.

배열 크기 N, 숫자가 더해지는 횟수 M, 연속 반복 가능 횟수 K, 배열이 주어질 때 결과를 출력하라.

해설

정렬 후, 가장 큰 수와 두 번째로 큰 수만 이용해 문제를 푼다.

코드

1. 내가 생각한 코드

   가장 큰 수를 가장 많이 사용하면 되기 때문에 M % K 번 만큼 두 번째로 큰 수를 더하고, 남은 횟수만큼 최대값을 더해주면 된다고 생각했다.

   그러나 만약 [2, 4, 5 4, 6], M = 11, K = 4 이면 아래 계산으로는 6+6+6+6+5+6+6+6+6+5+5가 된다. 하지만 맞는 계산은 6+6+6+6+5+6+6+6+6+5+6 이 되어야 한다.

   여기서 생각을 해 보면 M // K * K 만큼 최대값을 더하는 생각은 틀리지 않았으나 마지막으로 남은 부분에 대한 계산이 추가되면 될 것 같다.

   이제 남은 부분에 대한 계산을 어떻게 할까라는 문제가 있는데 이는 M % K 를 다시 M // K 로 나눈 나머지 값을 횟수로 생각해서 maxCnt에 더해주면 될 것 같다.

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

data.sort()

maxVal = data[n - 1]
secondMaxVal = data[n - 2]
```

```
result = 0

result += maxVal * (m - (m % k)) + secondMaxVal * (m % k)

print(result)
```

2. 수정 코드

   여기까지 작성 후 정답 코드를 보니… k로 나눈 몫으로 계산하면 오차가 나올 수 있는 가능성이 생긴다. m의 값이 충분히 크다면 수열 반복이 정답보다 더 많이 나올 수 밖에 없다. 예를 들어 입력이 5 10003 4 [2 4 5 4 6] 이라면 본인 코드의 답은 60018이 나온다. 하지만 옳게 된 정답은 58018 이다.

   값의 차이가 나는 이유는 명확하다. 책의 정답은 수열의 반복을 두 번째로 큰 수까지 고려해서 반복 횟수를 정하지만 본인은 두 번째로 큰 수까지 고려하지 않았다.

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

data.sort()

maxVal = data[n - 1]
secondMaxVal = data[n - 2]

maxCnt = m // k * k # 횟수 나누기 반복 가능의 몫 * 반복 가능
maxCnt += (m % k) % (m // k) # 나머지 나누기 수열이 반복된 횟수의 몫

result = 0

result += maxVal * maxCnt + secondMaxVal * (m - maxCnt)

print(result)
```

3. 정답 코드

   위의 오류들을 모두 해결한다. 수열의 반복만큼을 maxCnt로 설정한 후, 나머지만큼 maxCnt에 더해준다. 여기서 나머지만큼 maxCnt를 더해주는 이유를 생각해보자.

   5, 10, 3, [2, 4, 5, 4, 6] 이라면 [6, 6, 6, 5] 가 반복되어 더해진다. 나열해보면 6+6+6+5+6+6+6+5까지가 m // (k + 1) * k 만큼의 반복이다. 이후에 나올 수는 당연히

6이고 두 번째로 가장 작은 수는 다시 나올 수 없기 때문에(나오는 순간 반복 횟수 + 1) 나머지만큼 최대값을 더해준다.

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

data.sort()

maxVal = data[n - 1]
secondMaxVal = data[n - 2]

# 최대값이 더해지는 횟수
cnt = m // (k + 1) * k # 수열의 반복만큼
cnt += m % (k + 1) # 반복 수열 횟수의 나머지

result = 0

result += cnt * first # 위에서 구한 횟수만큼 최대값 더하기
result += (m - cnt) * second # 여긴 당연히 남은 횟수

print(result)
```

💡 Q3. 숫자 카드 게임

문제

여러 개의 숫자 카드 중 가장 높은 숫자가 쓰인 카드 한 장을 뽑는 게임이다.

Rule)

1. 숫자가 쓰인 카드들이 N X M 형태로 놓여 있다. (N = 행, M = 열)
2. 먼저 뽑고자 하는 카드가 포함되어 있는 행을 선택한다.
3. 선택된 행에 포함된 카드 중 가장 숫자가 낮은 카드를 뽑는다.
4. 따라서 처음에 카드를 골라낼 행을 선택할 때, 이후에 해당 행에서 가장 숫자가 낮은 카드를 뽑을 것을 고려하여 최종적으로 가장 높은 숫자의 카드를 뽑을 수 있도록 전략을 세운다.

예를 들어 3 X 3 형태로 아래와 같이 카드가 놓여있다.

[3, 1, 2]

[4, 1, 4]

[2, 2, 2]

1, 2 행에서 카드를 뽑을 경우 1이 나오고 3행에서 카드를 뽑으면 2가 나오기 때문에 답은 2이다.

해설

각 행의 최소값을 리스트에 저장하고 그 중 최대값을 출력한다.

내가 생각한 코드

코드 적으로 문제가 없어 보인다. 단, min, max 함수는 O(N) 이므로 반복문까지 생각하면 O(N^2)의 시간복잡도를 가진다.

```
n, m = map(int, input().split())

result = []

for i in range(n):
  data = list(map(int, input().split()))
  result.append(min(data))

print(max(result))
```

정답 코드

```
n, m = map(int, input().split())

result = 0

for i in range(n):
  data = list(map(int, input().split()))
  min_value = min(data)
```

```
  result = max(result, min_value)

print(result)
```

코드를 보니 시간복잡도 상으로도 문제가 없다. 정답 코드 또한 O(N^2) 이기 때문에 그냥 아무렇게나 풀어도 되겠다.

💡 Q4. 1이 될 때까지

문제

어떠한 수 N이 1이 될 때까지 다음의 두 과정 중 하나를 반복적으로 선택 수행한다. 단, 두 번째 연산은 N이 K로 나누어 떨어질때만 가능하다.

1. N에서 1을 뺀다.
2. N을 K로 나눈다.

예를 들어, N = 17, K = 4라면 1. 1번, 2. 2번을 하면 N이 1이 되고 총 횟수는 3이 된다.

해설

N이 K로 나누어 떨어지면 2번을 아니면 1번을 N이 1이 될 때까지 반복한다.

내가 생각한 코드

```
n, k = map(int, input().split())

result = 0

while n != 1:
  if n % k != 0:
    n -= 1
    result += 1
```

```
  else:
    n //= k
    result += 1

print(result)
```

정답 코드

는 오히려 더 복잡해 보이니 놔두겠다.