

**알고리즘 과제 1**

**과목명 알고리즘**

**담당교수 김희철교수님**

**제출일 20210914**

**전공 컴퓨터전자시스템**

**학번 201904458**

**이름 이준용**

* **문제 1번**

1) 주식 투자를 즐기는 홍길동은 n일 동안 어떤 회사의 주식을 다음과 같이 매매한다: 주식 한 주를 한 번만 사서 이를 다음에 한 번 팔 수 있다. 홍길동은 n일 동안 주식 매매(한번 사서 한번 판다)를 하여 얻은 이익에 대하여, 얻을 수 있는 최대 이익과 얼마나 차이가 있는지를 알고 싶어한다. 그래서 이 기간 동안 얻을 수 있는 최대 이익을 계산하기로 하였다. 예를 들어, 7일 동안 주식 가격이 (30, 25, 50, 10, 20, 40)과 같을 때, 최대 이익은 네째 날 주식을 사서 마지막 날 팔면 이익이 30(=40–10)이다. n일 동안 주식 가격이 주어질 때, 얻을 수 있는 최대 이익을 구하는 프로그램을 작성하시오.

입력 형식

첫째 줄에는 n이 주어진다. n은 2 이상 100,000 이하의 정수이다. 둘째 줄에는 주식 가격(양의 정수)이 날짜 순서대로 n개 주어진다.

출력 형식

최대 이익을 첫 번째 줄에 출력한다. 두 번째 줄에 사는 가격과 파는 가격을 각각 출력한다. 최대 이익을 얻는 경우가 여러 개일 때는, 사는 가격이 최소인 경우 하나만 출력한다. 이득이 없는 경우는, –1만 출력한다.

* **알고리즘 및 자료구조**

이중배열을 이용했습니다. 마지막 출력단계에서 Price\_sorted[사고파는 주식의 인덱스][그의 해당하는 인덱스의 값]으로 배열을 만들었습니다.

최대이익은 최대값 인덱스가 더 크다는 if조건을 만족한다면

사는가격 – 파는가격 = 최대이익 이므로 최대이익 출력

최대값 인덱스가 더 작으면 최대이익이 없으므로 if 조건에서 -1 출력하게 만들었습니다.

* **시간복잡도 분석**

For 구문을 한번 만 돌고 조건에 만족한 값은 새로운 이중 배열의 저장하고 마지막에 출력하는 과정이므로

입력 n에 따라서 결정되므로 시간 복잡도는 O(N)입니다.

* **느낀 점**

처음에는 리스트 변수 하나만 사용해서 문제를 해결하려고 했지만 최대이익을 출력했지만 사는 가격과 파는 가격의 인덱스를 저장해 가격을 출력하는 부분에서 막혔습니다. 하지만 파이썬의 이중배열을 이용해서 2개 파라미터인 가격의 인덱스와 그 인덱스에 해당되는 값을 나눠서 코딩하였습니다.

* **프로그램 코드**

a = int(input())  
b = list() # 후보 값 리스트  
stock = list(map(int, input().split())) # 주식 가격리스트  
  
upperlimit\_stock = stock[0]  
upperlimit\_index = 0  
underlimit\_stock = stock[0]  
underlimit\_index = 0  
  
for i in range(1, len(stock)):  
# for i in range(1, a):  
 n = stock[i]  
 if upperlimit\_stock < n: # n이 원래의 max값보다 클 경우에는 n을 max값으로 업데이트하고, 인덱스를 i로 업데이트함  
 upperlimit\_stock = n  
 upperlimit\_index = i  
 elif underlimit\_stock > n: # n가 원래 min값보다 작을 경우엔 리스트에 기존 값을 담고 다시 돌린다.  
 b.append((upperlimit\_stock - underlimit\_stock, upperlimit\_stock, underlimit\_stock))  
 underlimit\_stock = n  
 underlimit\_index = i  
 upperlimit\_stock = n  
 upperlimit\_index = i  
  
b.append((upperlimit\_stock - underlimit\_stock, upperlimit\_stock, underlimit\_stock))  
# for문이 끝나고 난 뒤의 최대 이익(price\_max - price\_min)과, 최대값과 최소값을 각각 넣는다.  
  
price\_sorted = sorted(b, key=lambda n: (-n[0], n[2]))  
# list에 앞에서부터 이익이 큰 값을 넣고, 이익이 같은 경우 최소값이 낮은 것부터 넣는다.  
  
if upperlimit\_index <= underlimit\_index: # 최대값 인덱스가 더 작을 경우. 이럴 땐 최대 이익이 없으므로 -1 출력  
 print(-1)  
else:  
 print(price\_sorted[0][1] - price\_sorted[0][2])  
 print(price\_sorted[0][2], price\_sorted[0][1])

* **문제 2번**

n일 동안의 주식 지수가 날짜 순서대로 주어져 있다. 연속하여 지수가 상승한 날 수의 최대값을 구하는 프로그램을 작성하시오. 예를 들어 15개의 주식 지수가 (26, 22, 10, 25, 27, 29, 45, 23, 24, 25, 40, 26, 37, 13, 24)라면 연속으로 지수가 상승한 날 수의 최대값은 4((10, 25, 27, 29, 45)의 길이 –1)이다.

입력 형식

첫째 줄에는 n이 주어진다. n은 2 이상 100,000 이하의 정수이다. 둘째 줄에는 주식 지수(양의 정수)가 날짜 순서대로 n개 주어진다.

출력 형식

연속하여 지수가 오른 날 수의 최대값을 출력한다.

* **알고리즘 및 자료구조**

변수 tmp와 upper\_limit를 0로 초기화합니다.

Input : n -> 첫째 줄 n , stock -> 주식지수 (양의 정수)

첫째 줄에 입력 받은 n값이 2부터 100,000 이하의 정수인지 확인합니다.

조건문을 통과한다면 for구문에서 i가 1부터 둘째 줄에 입력 받은 len(주식 지수)의 범위로

Tmp가 stock[i]가 stock[i-1]보다 클 경우에는 1을 더함. But 그렇지 않은 경우에는 0으로 만듦.

Tmp가 stock의 값들을 검사하면서 0이 된다면 그 직전의 최대값과 다음의 최대값 중에 max()을 이용하여 비교한 값을 upper\_limit에 선언

출력: upper\_limit // 종료

* **시간복잡도 분석**

1부터 n까지의 for구문안에 if 구문 stock[i] > stock[i-1]부분만 연산되므로 이부분도 1번 문제와 같이 입력 n에 따라서 시간 복잡도가 결정됩니다. O(N)입니다.

* **느낀 점**

1번 문제보다는 계산과정과 코딩과정의 드는 시간이 적게 걸렸습니다. 자료구조 시간에 단련된 어려운 과제 덕분인지 점점 빠르게 코딩하게 되는 것 같습니다. 스스로 생각하는 시간을 늘리려고 노력하고 있습니다.

* **프로그램 코드**

n = int(input()) # n일  
stock = list(map(int, input().split())) # 정해진 주식 지수를 list에 매핑합니다.  
  
tmp = 0  
upper\_limit = 0 # 주식 지수 최대값  
if 2 <= n <= 100000:  
 for i in range(1, n):   
# for i in range(1, len(stock)):  
 tmp = tmp + 1 if stock[i] > stock[i - 1] else 0 # tmp가 증가하다가 0이 되기 직전에 최대값이 stock에 저장됨  
 upper\_limit = max(upper\_limit, tmp) # tmp가 0이 된 후의 값과 그 전 값 사이에 최대값 비교 max 구함  
print(upper\_limit)

* **문제 3번**

오름차순으로 정렬된 n(2이상 100,000이하 정수)개의 정수(1,000,000,000이하 정수) 중 K(정수)와 가장 가까운 정수를 찾는 프로그램을 작성하시오. K와 가장 가까운 정수가 여러 개일 경우, 이들 중 큰 수를 출력하시오.

요구 조건: 이진탐색(binary search)을 이용해야 한다.

* **알고리즘 및 자료구조**

입력

n -> 오름차순으로 정렬된 n

num\_list -> n개의 정수리스트

k -> k(정수)

함수 solve()

입력받은 num\_list를 오름차순으로 정렬 .sort()이용

left -> 0 으로 초기화

변수 right -> 입력 받은 num\_list가 홀수 일 경우 len() -1 생각

While() -> 중간값을 정하고 왼쪽은 left, 오른쪽은 right으로 정함. Left가 right보다 클 때까지 진행

n값이 2와 100,000사이에 있는지 조건 확인

mid -> 배열의 중간에 있는 임의의 값으로 지정

조건: k가 리스트의 중간값과 같을 경우 리턴

K가 리스트의 중간값보다 클 경우 right = mid -1

K가 리스트의 중간값보다 작을 경우 left = mid -1

조건 : k가 입력받은 num\_list에 없을 경우

Low\_num -> k- num\_list[right] 의 절대값 -> 현재 right에 있는 인덱스들이 더 작으므로 작은 인덱스 쪽의 절대값을 low\_num으로 선언

High\_num -> num\_list[left] – k의 절대값 -> 현재 left에 있는 인덱스들이 더 크므로 큰 인덱스 쪽의 절대값을 high\_num으로 선언

조건 : low\_num이 high\_num보다 작을경우 -> result에 num\_list[right]

low\_num이 high\_num보다 클 경우 -> result에 num\_list[left]

low\_num이 high\_num과 서로 같을 경우 -> result에 num\_list[right+1]

result 리턴

solve 함수 ( n , num\_list, k) 출력 // 종료

* **시간복잡도 분석**

5 // 20 30 40 55 60 // 36 와 같이 입력이 주어졌을 때

중간값 : 40 -> 36은 40보다 작다 -> {20 , 30}

중간값 : 30 -> 36은 30보다 크다 -> 종료

N개의 크기 배열을 이진 탐색할 경우 N, N/2, N/8,…,1로 연산 됨.

그러므로 이진 탐색의 시간 복잡도는 O(logN)입니다.

* **느낀 점**

함수를 만들 때 K가 오름차순으로 정렬된 n개 정수 리스트에 k가 있는지와 없는지를 나누는 단계를 계획하는 부분에서 시간을 많이 뺏겼습니다. 특히 k가 리스트에 없을 경우에 절대값을 씌워서 계산하는 과정을 바로 떠올리지 못하였으나 이번에 기회로 연산을 유연하게 생각해야 되겠다는 것을 다짐하는 기회가 되었습니다.

* **프로그램 코드**

n = int(input()) # n  
num\_list = list(map(int, input().split())) # n개의 정수  
k = int(input()) # k  
  
  
def solve(n,num\_list,k):  
 num\_list.sort() # 정렬  
 left = 0  
 right = len(num\_list) - 1  
 while left <= right: # left가 right보다 클 때까지 진행하는 while문  
 if 2 <= n <= 100000:  
 mid = (left + right) // 2 # 배열의 중간에 있는 임의의 값  
 if k == num\_list[mid]: # 구하고자 하는 k가 리스트의 중간값과 같다면 k를 리턴하고 종료  
 return num\_list[mid]  
 elif k < num\_list[mid]:  
 right = mid - 1 # else와 elif 부분은 교재 이진탐색 참고함.  
 elif k > num\_list[mid]:  
 left = mid + 1  
 else:  
 break  
 # left가 right보다 커지면 종료  
 if k not in num\_list: # num\_list에 k가 없을 경우에  
 low\_num = abs(k - num\_list[right]) # 현재 right의 인덱스가 더 작으므로,  
 # 작은 인덱스쪽의 절대값을 k - num\_list[right]으로 줬고 이를 low\_num로 선언  
 high\_num = abs(num\_list[left] - k) # 현재 left의 인덱스가 더 크므로,  
 # 큰 인덱스쪽의 절대값을 num\_list[left] - k로 줬고 이를 high\_num로 선언  
 # print(low\_num, high\_num)  
 if low\_num < high\_num: # 더 작은 인덱스의 절대값이 더 큰 인덱스의 절대값보다 작을 때:  
 result = num\_list[right] # 결과값 result에 num\_list[right]을 부여함  
 elif low\_num > high\_num: # 더 작은 인덱스의 절대값이 더 큰 인덱스의 절대값보다 클 때:  
 result = num\_list[left] # 결과값 result에 num\_list[left]을 부여함  
 elif low\_num == high\_num: # 인덱스의 절대값이 서로 같을 때:  
 result = num\_list[right+1] # 결과값 result에 num\_list[right+1]를 부여함  
 return result # 최종 결과값 result를 반환합니다.  
  
  
print(solve(n, num\_list, k))