# 문제 1-1  
# 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 예  
# 첫째 줄에는 n이 주어진다. n은 2 이상 100,000 이하의 정수이다. 둘째 줄에는 주식 가격(양의 정수)이 날짜 순서대로 n개 주어진다.  
  
# 출력 예  
# 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.  
  
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])  
'''  
15 -> 20?  
15  
6 5 10 5 7 9 25 1 4 10 6 23 5 3 12 2 4 24 14 10  
'''  
'''  
입력받은 수를 그대로 for 구문 돌아서 최소값을 찾기 때문에 O(n)  
'''  
'''  
10  
20 19 18 17 16 10 9 8 7 6   
'''

# 문제 1-2  
# 2) 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 예  
# 첫째 줄에는 n이 주어진다. n은 2 이상 100,000 이하의 정수이다. 둘째 줄에는 주식 지수(양의 정수)가 날짜 순서대로 n개 주어진다.  
  
# 출력 예  
# 연속하여 지수가 오른 날 수의 최대값을 출력한다.  
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)  
  
'''  
15  
26 22 10 25 27 29 45 23 24 25 40 26 37 13 24  
'''  
  
'''  
10   
10 20 30 40 50 60 70 80 90 100  
'''  
  
'''  
10   
100 90 80 70 60 50 40 30 20 10  
'''

# 문제 1-3) 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.  
  
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 같거나 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:  
 result = num\_list[right+1]  
 return result # 최종 결과값 result를 반환합니다.  
  
  
print(solve(n, num\_list, k))  
  
'''  
5  
20 30 40 55 60  
36  
'''  
'''  
중간값 : 40 -> 작다 -> {20,30}  
중간값 : 30 -> 크다 -> 종료  
N개의 크기 배열을 이진 탐색하면 N, N/2, N/8,...,1으로 실행 됨.  
이진 탐색의 시간 복잡도는 O(logN)입니다.  
  
'''  
'''  
5  
20 30 40 55 60  
35  
'''

# 문제 1-1  
# 1) n(1이상 500000 이하 정수)개의 정수들에 대하여  
# 두 수 차이의 최솟값을 출력하는 프로그램을 작성하시오. O(n2) 시간 알고리즘을 이용  
n = int(input()) # n  
n\_list = list(map(int, input().split())) # n 개의 정수  
def selection\_sort(arr): # O(n^2)알고리즘인 선택 정렬 알고리즘을 이용  
 for i in range(len(arr) - 1): # 최소값의 index와 현재 index에 있는 값을 swap함.  
 min\_idx = i  
 for j in range(i + 1, len(arr)): # 현재 index부터 마지막 index까지  
 if arr[j] < arr[min\_idx]: # 최소값의 인덱스를 찾아냄  
 min\_idx = j # 각 index에 대해서 최소값을 찾기 위해 대소비교는 여러번  
 # 일어나지만 swap은 한번만 일어남.  
 arr[i], arr[min\_idx] = arr[min\_idx], arr[i]  
def solution(n, n\_list):  
 selection\_sort(n\_list) # n 개의 정수를 위의 선택 정렬 알고리즘으로 선택 정렬 시킴.  
 num = list() # 최소값을 출력하기위한 list  
 for i in range(1, len(n\_list)): # n개 정수를 하나씩 검사  
 j = n -1  
 while j > i:  
 if(n\_list[j] > n\_list[i]):  
 num.append(n\_list[j] - n\_list[i]) # 두수의 차이를 num list에 append  
 j -= 1 # j를 1씩 줄여가면서 n\_list의 모든 i와 검사  
 print(min(num)) # 최소값 출력  
solution(n, n\_list)

# 문제 1-2  
# 2) n(1이상 500000 이하 정수)개의 정수들에 대하여  
# 두 수 차이의 최솟값을 출력하는 프로그램을 작성하시오.  
# 정렬을 이용하고, 정렬 알고리즘은 병합정렬을 사용한다.  
  
n = int(input()) # n  
n\_list = list(map(int, input().split())) # n 개의 정수  
def merge\_sort(arr): # 재귀를 이용해서 병합 정렬 구현  
 if len(arr) < 2: # 재귀 알고리즘의 기저 조건  
 return arr # 입력 배열 의 크기가 2보다 작을 때  
 # 이 조건에 해당 되면 배열을 그대로 반환  
 mid = len(arr) // 2  
 left\_arr = merge\_sort(arr[:mid]) # mid원소 기준 왼쪽 배열 재배치  
 right\_arr = merge\_sort(arr[mid:]) # mid 원소 오른쪽 배열 재배치  
  
 merged\_arr = []  
 l = h = 0  
 while l < len(left\_arr) and h < len(right\_arr): # 원소 1개로 쪼개고 난 후 합칠 때는  
 if left\_arr[l] < right\_arr[h]: # 작은 숫자가 앞에 큰 숫자를 뒤에 위치  
 merged\_arr.append(left\_arr[l])  
 l += 1  
 else: # 큰 숫자가 앞에 작은 숫자가 뒤에 위치  
 merged\_arr.append(right\_arr[h])  
 h += 1  
 merged\_arr += left\_arr[l:]  
 merged\_arr += right\_arr[h:]  
 return merged\_arr  
  
def solution(n, n\_list):  
 new\_n\_list = merge\_sort(n\_list) # n개의 정수를 병합정렬함  
 num = list()  
 for i in range(0, n-1):  
 if(new\_n\_list[i+1] >= new\_n\_list[i]):  
 if(len(num)==0): # num리스트에 아무것도 없을때 그냥 대입  
 num.append(new\_n\_list[i+1] - new\_n\_list[i])  
 else: # num리스트가 none이 아닐 경우  
 x = new\_n\_list[i+1] - new\_n\_list[i]  
 if(num[0] >x): # num에 추가된 차이값보다 x가 작을 경우 append  
 num.append(x)  
 print(min(num)) # num리스트에서 최솟값 출력  
  
solution(n, n\_list)

# 문제 1-3  
#3) n개의 수(109이하 –109이상 정수)들과 d(0이상 정수)가 주어져 있다.  
# 이들 n개의 수 중에서 다음 조건을 만족하는 수들을 선택하려고 한다: 조건)  
# 모든 두 수의 차이가 d이하이다. 이렇게 선택할 수 있는 수들의 최대 개수를 구하는  
# 프로그램을 작성하시오.  
#제약 조건:  
#(i) 정렬을 이용한다.  
#(ii) 정렬 알고리즘은 퀵 정렬을 사용하고, 피봇은 random하게 선택한다.  
n = int(input()) # n  
n\_list = list(map(int, input().split())) # n 개의 정수  
d = int(input())  
  
def quick\_sort(A): # 퀵 정렬 알고리즘  
 if(len(A)>1): # 정렬할때 받아오는 list에 적어도 한개의 원소가 있어야함.  
 import random # random함수 import  
 pivot = A[random.randint(0, len(A)-1)] # random.randint 균일분포의 정수 난수 1개 생성  
 greater = [i for i in A if i > pivot] # pivot보다 큰 값  
 smaller = [i for i in A if i < pivot] # pivot보다 작은 값  
 middle = [i for i in A if i == pivot] # pivot과 동일한 값  
 return quick\_sort(smaller) + middle + quick\_sort(greater) # 다시 크기 순으로 리스트 합  
 else:  
 return A # list의 원소가 없을 경우 반환  
  
def solution(n, n\_list, d):  
 new\_n\_list = quick\_sort(n\_list) # n\_list을 quicksort함  
 x = 0  
 result = 0 # 결과값 출력  
 for i in range(0, n): # 입력 받은 n개 만큼 for 구문  
 if abs(new\_n\_list[x] - new\_n\_list[i]) <=d: # [x] 와 [i]의 차이의 절대 값이 0보다 작을경우  
 result = len(new\_n\_list[x:i+1]) # result 변수에 길이 저장  
 else:  
 x+=1 # 두 수의 차이가 d보다 클 경우  
 print(result) # 결과 출력  
  
solution(n, n\_list, d)

# 문제 1-4  
# 양의 정수 𝑑와 𝑛 개의 정수 쌍, (), ,이 주어져 있다. 여기서 와 는 각각 사람 𝑖의 집과 사무실의 위치이다.  
# 길이 𝑑의 모든 선분 𝐿에 대하여, 집과 사무실의 위치가 모두 𝐿에 포함되는 사람들의 최대 수를 구하는  
# 프로그램을 작성하시오.  
  
# 첫 번째 줄에 사람 수 이 주어진다. 다음 개 각 줄에 정수 쌍 ()가 주어진다.  
# 여기서 와 는 -100,000,000 이상 100,000,000 이하의 서로 다른 두 정수이다.  
# 마지막 줄에 철로의 길이를 나타내는 정수 가 주어진다.  
  
import sys  
import heapq  
# 각 경로를 순회하면서 집, 사무실중 좌표값이 큰 점에서 왼쪽으로 d 길이의 철로를 깔았을 때  
# 철로를 깔았을 때 철로에 몇 개의 경로가 포함되는지를 확인하는 방식  
n = int(sys.stdin.readline())  
road\_info = [] # 입력받은 철도 저장하는 철도 정보  
result = 0 # 결과값  
hp = [] # heap  
for \_ in range(n):  
 road = list(map(int, sys.stdin.readline().split())) # 입력받은 순서대로 enter를 기준으로 좌표선분 mapping  
 road\_info.append(road) # road\_info 배열에 좌표 선분 append  
  
d = int(sys.stdin.readline()) # 길이 d   
roads = [] # 오름 차순으로 정렬된 road를 저장함  
for road in road\_info:  
 house, office = road # 각 사무실, 집 정보를 road에 저장할 때  
 if abs(house - office) <= d: # 사무실과 집의 거리가 d 보다 크다면 포함될 수 없으므로 저장x  
 road = sorted(road) # d보다 작거나 같다면 좌표정보를 오름차순으로 정렬  
 roads.append(road)  
roads.sort(key=lambda x:x[1]) # 철로의 시작점을 가장 작은 것부터 시작할 수 있도록 road을  
 # 입력된 원소 중 큰 원소를 기준으로 오름차순 정렬  
  
for road in roads: # 철로의 시작점을 가장 작은 것부터 순회하면서 차례대로 힙에 저장  
 if hp:  
 while hp[0][0] < road[1] - d: # 힙에 존재하는 가장 작은 값이 철로의 끝점안에 있는지 확인  
 heapq.heappop(hp) # 철로 내에 있지 않다면 힙에서 pop  
 if not hp:  
 break  
 heapq.heappush(hp, road)  
   
 else: # if hp  
 heapq.heappush(hp, road)  
 result = max(result, len(hp))  
  
print(result)