

**알고리즘 과제 2**

**과목명 알고리즘**

**담당교수 김희철교수님**

**제출일 20211008**

**전공 컴퓨터전자시스템**

**학번 201904458**

**이름 이준용**

* **문제 1번**

n(1이상 500000 이하 정수)개의 정수들에 대하여 두 수 차이의 최솟값을 출력하는 프로그램을 작성하시오. O(n2) 시간 알고리즘을 이용

입력 예 출력 예

10

1 10 20 7 15 30 40 50 32 5 2

* **알고리즘 및 자료구조**

선택 정렬은 정렬된 값을 배열의 맨 앞부터 하나씩 채워나가게 됩니다. 따라서, 뒤에 있는 index로 갈수록 비교 범위가 하나씩 점점 줄어드는 특성을 가지고 있습니다. (index 0에서는 0부터 n-1까지 비교해야 되지만, index n-1에서는 남은 숫자가 하나만 남아서 비교가 필요 없음)

입력 배열이 이미 정렬되어 있건 말건 관계없이 동일한 연산량을 가지고 있기 때문에 최적화 여자가 적어서 다른 O(N^2) 대비해도 성능이 떨어지는 편입니다.

* **시간복잡도 분석**

선택 정렬은 별도의 추가 공간을 사용하지 않고 주어진 배열이 차지하고 있는 공간 내에서 값들의 위치만 바꾸기 때문에 O(1)의 공간 복잡도를 가집니다.

시간 복잡도는 우선 루프문을 통해 모든 인덱스에 접근해야 하기 때문에 기본적으로 O(N)을 시간을 소모하며, 하나의 루프에서는 현재 인덱스의 값과 다른 인덱스의 값들과 비교하여 최소값을 찾은 후 현재 인덱스에 있는 값과 상호 자리 교대를(swap)해야 해야 하기 때문에 O(N)을 시간이 필요하게 됩니다. 따라서 선택 정렬은 총 O(N^2)의 시간 복잡도를 가지는 정렬 알고리즘입니다.

* **느낀 점**

O(n^2) 시간 알고리즘을 선택 정렬을 이용해서 두 수 차이의 최솟값을 출력하는 프로그램을 코딩하면서 선택 정렬로 정렬되는 과정을 다시 공부하게 되었습니다.

* **프로그램 코드**
* # 1.제목:한국외대 알고리즘 과제  
  # 2.날짜:20210929  
  # 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 2번**

2) n(1이상 500000 이하 정수)개의 정수들에 대하여 두 수 차이의 최솟값을 출력하는 프로그램을 작성하시오. **정렬을 이용하고, 정렬 알고리즘은 병합정렬을 사용한다.**

입력 예 출력 예

10

-1 10 20 7 15 30 40 50 32 5 2

* **알고리즘 및 자료구조**

재귀를 이용해서 병합 정렬을 구현할 수 있습니다. 먼저 배열을 더 이상 나눌 수 없을 때까지 (원소가 하나만 남을 때까지) 최대한 분할 후에, 다시 병합하면서 점점 큰 배열을 만들어 나가면 됩니다. 따라서 이 재귀 알고리즘의 기저 조건은 입력 배열의 크기가 2보다 작을 때이며, 이 조건에 해당할 때는 배열을 그대로 반환하면 됩니다.

* **시간복잡도 분석**

전반적인 반복의 수는 점점 절반으로 줄어들 기 때문에 O(logN) 시간이 필요하며, 각 패스에서 병합할 때 모든 값들을 비교해야 하므로 O(N) 시간이 소모됩니다.

Solution()함수에서 O(n)이 소모됩니다. 시간 복잡도 식은 nlogn + n 이므로 따라서 총 시간 복잡도는 O(NlogN) 입니다.

두 개의 배열을 병합할 때 병합 결과를 담아 놓을 배열이 추가로 필요합니다. 따라서 공간 복잡도는 O(N) 입니다.

* **느낀 점**

병합 정렬을 이용해서 두 수의 차이의 최솟값을 출력하는 프로그램을 코딩하면서 병합 정렬의 정렬되는 과정과 위의 1번 문제에서 선택 정렬의 과정을 되새기면서 코딩 최솟값 문제는 시간이 적게 드는 정렬 알고리즘을 선택해서 풀어내는 것의 중요함을 느꼈고, 나중에 이런 유형의 두 수의 차이 최솟값 구하는 문제가 나올 경우에 정렬 알고리즘을 이용한 해결을 좀 더 빠르게 머리에 떠올릴 수 있을 것 같습니다.

* **프로그램 코드**
* # 1.제목:한국외대 알고리즘 과제  
  # 2.날짜:20210929  
  # 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 3번**

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.**

입력 예

6 // n

3 10 50 15 25 14 // n개의 수

5 // d

출력 예

3 // 10 14 15

* **알고리즘 및 자료구조**

먼저 리스트에서 pivot값을 랜덤으로 뽑기 위해 **random.randint(0, len(A)-1)**인 설정한 범위내의 균일 분포의 정수 난수 1개 생성하는 것으로 정했고, pivot 값보다 작은 값(smaller), 동일한 값(middle) 그리고 큰 값(greater)을 담아둘 3개의 리스트를 생성합니다. 그리고 반복문을 통해 각 값을 pivot과 비교 후에 해당하는 리스트에 추가시킵니다. 그 다음 작은 값과 큰 값을 담고 있는 배열을 대상으로 퀵 정렬 함수를 재귀적으로 호출합니다. 마지막으로 재귀 호출의 결과를 다시 크기 순으로 합치면 정렬된 리스트를 얻을 수 있습니다.

* **시간복잡도 분석**

쿽 정렬의 성능은 어떻게 pivot 값을 선택 선택하느냐에 크게 달라질 수 있습니다. 이상적인 경우에는 pivot 값을 기준으로 동일한 개수의 작은 값들과 큰 값들이 분할되어 병합 정렬과 마찬가지로 O(NlogN)의 시간 복잡도를 가지게 됩니다.

하지만 pivot 값을 기준으로 분할했을 때 값들이 한 편으로 크게 치우치게 되면, 퀵 정렬은 성능은 저하되게 되며, 최악의 경우 한 편으로만 모든 값이 몰리게 되어 O(N^2)의 시간 복잡도를 보이게 됩니다.

따라서 상용 코드에서는 중앙값(median)에 가까운 pivot 값을 선택할 수 있는 섬세한 전략이 요구되며, 배열의 첫 값과 중앙값 그리고 마지막 값 중에 크기가 중간인 값을 사용하는 방법이 많이 사용됩니다.

퀵 정렬은 공간 복잡도는 구현 방법에 따라 달라질 수 있는데, 입력 배열이 차지하는 메모리만을 사용하는 in-place sorting 방식으로 구현을 사용할 경우, O(1)의 공간 복잡도를 가진 코드의 구현이 가능합니다.

Solution()함수에서 O(N)이 소모되므로 시간 복잡도 식은 N + NlogN입니다. 그러므로 시간 복잡도는 O(N)입니다.

* **느낀 점**

처음에는 문제 이해를 잘 못해서 무엇이 틀린 것인지 고민했지만, 수요일 금요일 과제 질문 시간에 질문하고 교수님께서 친절히 설명해주신 덕분에 이해해서 풀어냈습니다. 마지막 테스트케이스는 Timeout인데 어떻게 하면 좀 더 간단하고 좋은 알고리즘과 함수를 구현할 수 있을까 계속 고민했습니다.

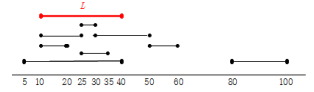
Solution()함수에서 O(N^2)을 O(N)으로 구현했습니다. 마지막 12번 테스트케이스를 통과했습니다.

* **프로그램 코드**
* 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 4번**

4) 집과 사무실을 통근하는 𝑛명의 사람들이 있다. 각 사람의 집과 사무실은 수평선 상에 있는 서로 다른 곳에 위치하고 있다. 임의의 두 사람 𝐴, 𝐵에 대하여, 𝐴의 집 혹은 사무실의 위치가 𝐵의 집 혹은 사무실의 위치와 같을 수 있다. 통근을 하는 사람들의 편의를 위하여 일직선 상의 어떤 두 점을 잇는 철로를 건설하여, 기차를 운행하려고 한다. 제한된 예산 때문에, 철로의 길이는 𝑑로 정해져 있다. 집과 사무실의 위치 모두 철로 선분에 포함되는 사람들의 수가 최대가 되도록, 철로 선분을 정하고자 한다.

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.

아래 그림에 있는 8 명의 집과 사무실 위치의 예를 고려해 보자.



여기서 이고, , , , , , , , 이고, 이다. 이 예에서 위치 10과 40 사이의 빨간색 선분 이 가장 많은 사람들의 집과 사무실 위치를 모두 포함하는 길이 30인 선분 중 하나이다. 답은 4이다.

입력

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.

출력

길이 인 선분에 집과 사무실 위치가 모두 포함되는 사람들의 최대 수를 출력한다.

입력 예 1 출력 예 1

8 4

5 40

35 25

10 20

10 25

30 50

50 60

30 25

80 100

30

입력 예 2 출력 예 2

4 0

20 80

70 30

35 65

40 60

10

* **알고리즘 및 자료구조**

prioriryQueue()보다는 heapq()을 이용했습니다. 데이터를 정렬된 상태로 저장하기 위해서 사용하기 위해서 입니다. heapq모듈은 이진 트리 기반의 최소 힙 자료구조를 제공합니다. min heap을 사용하면 원소들이 항상 정렬된 상태로 추가되고 삭제되며, min heap에서 가장 작은 값은 언제나 인덱스 0. 즉, 이진 트리의 루트자리에 위치합니다.

Heapq()모듈을 이용한 힙 정렬은 힙 자료구조의 성질을 이용한 대표적인 정렬 알고리즘인데 road를 추가 하고 삭제하는 과정을 힙 정렬 알고리즘에서 응용해서 구현했습니다.

import heapq

def heap\_sort(nums):

heap = []

for num in nums:

heapq.heappush(heap, num)

sorted\_nums = []

while heap:

sorted\_nums.append(heapq.heappop(heap))

return sorted\_nums

1) 먼저 각 사무실, 집 정보를 roads에 저장할 때 사무실과 집의 거리가 d보다 크다면 d가 포함될 수 없으므로 저장하지 않는다. d보다 작거나 같다면 좌표정보를 오름차순으로 정렬해 저장합니다.

2) 철로의 시작점을 가장 작은 것부터 시작할 수 있도록 raods를 본인의 원소 중 큰 원소를 기준으로 오름차순 정렬합니다. (앞서 말한 것처럼 더 멀리있는 좌표값에서 왼쪽으로 철로를 깔 것이기 때문에 더 멀리있는 좌표값 (= 큰 원소)가 시작점이 됩니다.)  
roads.sort(key=lambda x:x[1])

3) 철로의 시작점을 가장 작은 것부터 순회하면서 차례대로 힙에 넣어주게 됩니다. 이때 힙에 존재하는 가장 작은 값이 철로의 끝점안에 있는지 확인해 철로 내에 있지 않다면 힙에서 제거합니다(heappop).  
while hp[0][0] < road[1] - d:  
heapq.heappop(hp)

* **시간복잡도 분석**

정렬 + 우선순위 큐(최소힙)사용으로 간단히 요약하면은 n개의 선분들을 오른쪽 끝점을 기준으로 오름차순 정렬합니다. 이후에 각 i번째 선분의 오른쪽 끝 점을 철로의 오른쪽 끝점을 잡고 길이 내부에 포함되는 선분들의 최대 개수를 계산하는 것입니다. For 구문을 돌면서 삽입연산하는 부분 O(n), 철도 선분의 길이와 비교하는 부분 O(logn)이므로

시간복잡도는 O(nlogn)입니다.

* **느낀 점**

일단 문제를 처음에 해석하는데 힙 문제들은 어떻게 사용할지를 떠올리는게 쉽지 않은 것 같습니다. 특히 2차원을 이용한 함수 문제는 많이 해결을 못해봐서 좀 더 시간이 오래 걸렸던 것 같습니다. 저는 힙 졍렬 알고리즘에서 힌트를 얻어서 road에 집, 사무실 집어넣고 삭제하고 하는 과정을 빠르게 구현할 수 있었습니다.

* **프로그램 코드**
* # 1.제목:한국외대 알고리즘 과제  
  # 2.날짜:20210929  
  # 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 not hp:  
   heapq.heappush(hp, road)  
   else:   
   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)  
   result = max(result, len(hp))  
  print(result)