**Advanced Coding Practice HW Problem #2**

**20191599 송경호**

**A) Largest N numbers**

**A-1) 입력 형식 정의**

모두 다른 값을 가진 N x N matrix에서 가장 큰 수부터 N번째 큰 수 까지를 찾아야 하는 문제이다. 즉 입력 값은 N x N matrix가 된다. 예시는 다음과 같다.

|  |
| --- |
| 5  1 5 9 13 25  2 6 11 16 27  3 7 14 18 28  4 8 15 21 30  10 11 20 23 50 |

**A-2) 문제해결**

* ***N\*N 2차원 배열을 1차원 배열로 변환한다***
* ***Sort를 수행한다***
* ***N개의 수를 출력한다***

**A-3) 시간, 공간복잡도 및 고려사항**

기본적으로 N\*N개의 입력을 저장해야 하므로 O(N2)의 공간복잡도를 갖는다. 추가적인 공간은 필요로 하지 않는다.

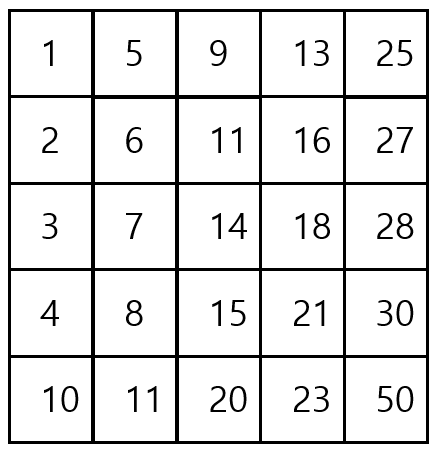
최적화된 정렬방식을 수행한다고 했을 때, 정렬의 시간복잡도는 O(입력값log입력값)이므로 시간복잡도는 O(N2logN)이다.

이외에도 다른 방식을 고려했으나 시간복잡도는 O(N2logN)로 동일한 반면 오히려 공간복잡도를 더 많이 차지하는 등의 문제가 발생하여 해당 알고리즘을 택했다.

**B) Position of K**

**B-1) 규칙 확인**

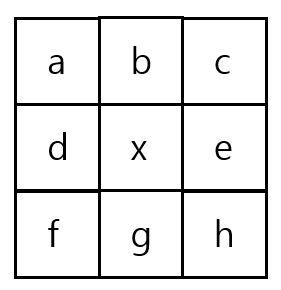
주어지는 입력값은 N x N의 matrix이며 row와 column이 non-decreasing order이다. N=5일 때의 예시는 다음과 같다.

 **사각형이(가) 표시된 사진

자동 생성된 설명**

여기서 row와 column이 non-decreasing order이므로 a < d < f < g < h는 반드시 성립한다. a < b < c < e < h 역시 반드시 성립한다.

**B-2) 문제 해결**

****

* 현재위치는 x이다 (최초 현재위치는 가장 우측 위의 값이다). 만약 x가 찾으려는 값이라면 반복을 종료 아니라면 다음 단계로 넘어간다.
* 목표숫자를 a,d,f,g,h와 비교한다. 이때 앞서 말했듯이, a<d<f<g<h가 성립하므로 목표숫자를 target이라고 할 때
  + target < d의 경우 x = a
  + d < target < f의 경우 x = d
  + f < target < g의 경우 x = f
  + g < target < h의 경우 x = g
  + h < target 의 경우 x = h

규칙을 적용한다.

해당 과정을 반복하며 목표숫자를 찾아 나간다. 위의 예시에서 찾는 숫자가 8인 경우 다음과 같은 경로로 8을 찾아낸다.

텍스트, 번호, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

**B-3) 시간복잡도, 공간복잡도 및 고려사항**

기본적으로 N\*N matrix를 저장해야 하므로 공간복잡도는 O(N2)이며 추가적인 공간은 필요로 하지 않는다.

시간복잡도는 최악을 고려했을 때 matrix의 변을 타고 움직이는 경우이다. 위의 예시에서 25에서 출발하여 10을 찾아야 하는 경우 숫자의 배치상 최악의 경우 10(2N)번의 움직임이 발생할 수 있다. 따라서 시간복잡도는 최악의 경우 O(N)이 된다.

해당 알고리즘 사용시 최단거리를 찾아낼 수는 없다. 그러나 앞서 말한 a<d<f<g<h임을 이용하여 탐욕적으로 가장 가까운 숫자로 움직인다. 또한 만약 최북서쪽 지점이나 최남동쪽 지점에서 출발하게 되면 정확한 방향을 특정할 수 없다. 따라서 해당 알고리즘이 가장 적은 경우를 탐색하면서 목표 숫자를 찾아내는 알고리즘이라고 판단했다.