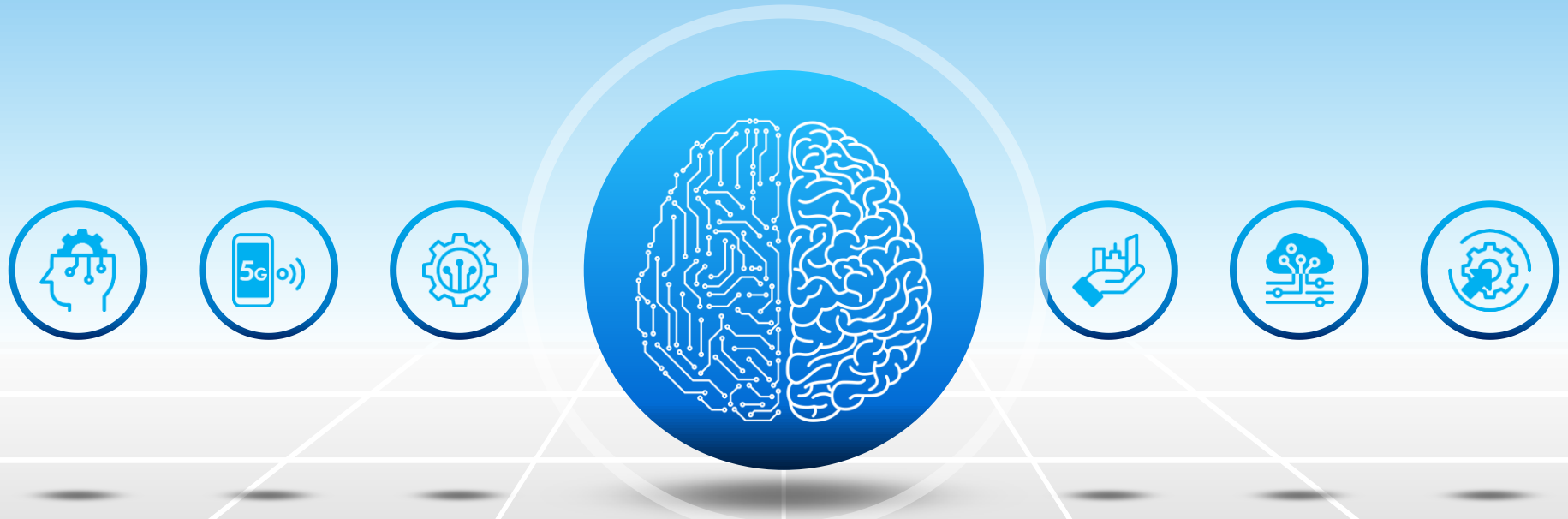
CIS3034 문제해결프로젝트

큐를 활용한 문제해결

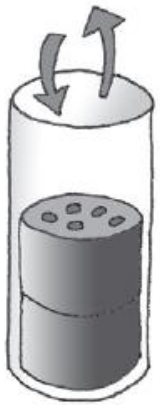


목 차
CONTENTS

큐의 이해

큐

- **큐(Queue)**
  - 스택과 비슷한 삽입과 삭제의 위치가 제한 되어있는 유한 순서 리스트
  - 줄, 혹은 줄을 서서 기다리는 것
  - 큐는 뒤에서는 삽입만 하고, 앞에서는 삭제만 할 수 있는 구조
    - 삽입한 순서대로 원소가 나열되어 가장 먼저 삽입(First-In)한 원소는 맨 앞에 있다가 가장 먼저 삭제(First-Out)됨
      ☞ **선입선출 구조** (FIFO, First-In-First-Out)

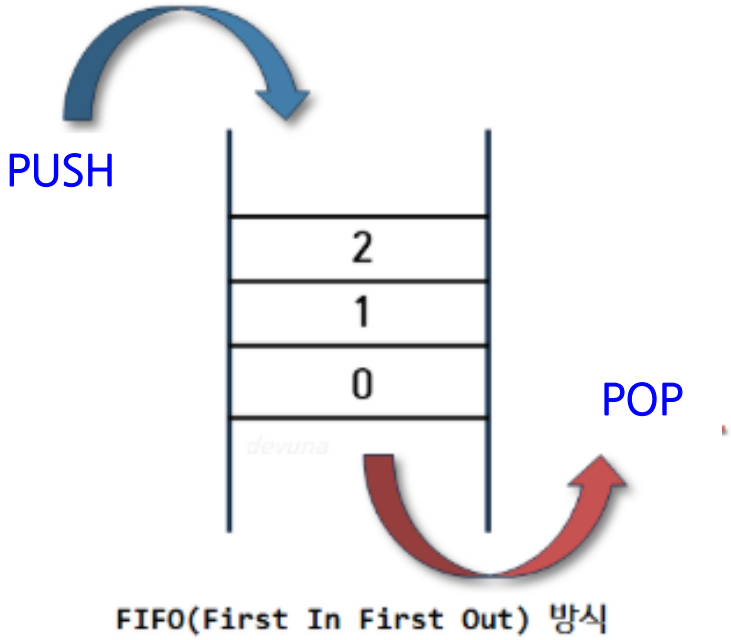


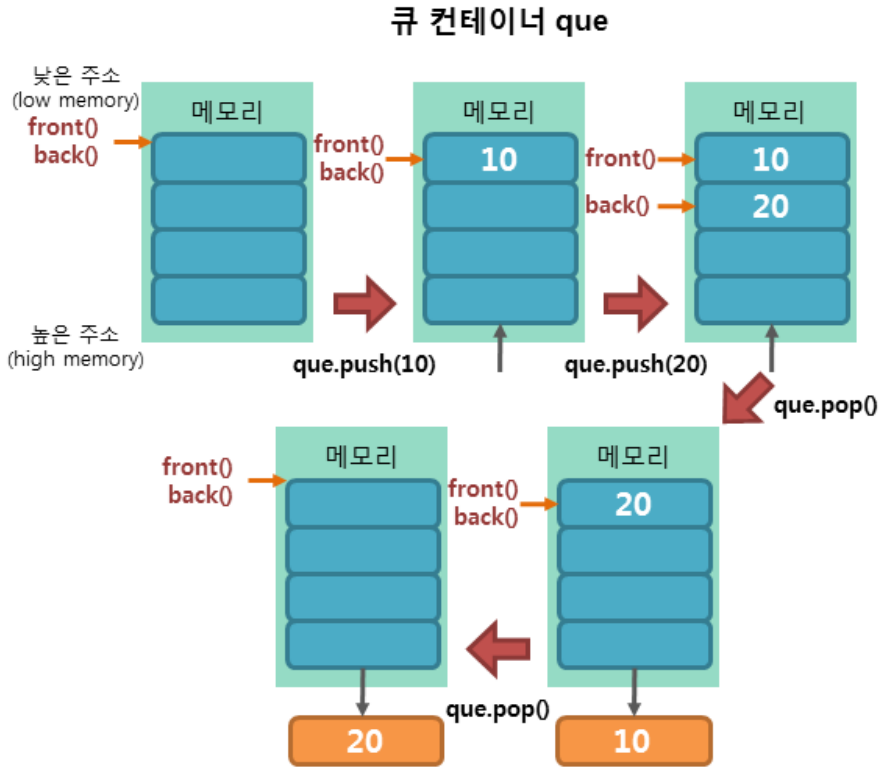
(a) 스택의 구조

(b) 큐의 구조

큐

- **큐의 사용 사례**
  - OS CPU의 연산 처리시 작업 대기
  - 프린터의 인쇄 대기열
  - 동영상 스트리밍 서비스에서 컨텐츠 버퍼링시
  - 은행 업무





큐

- **큐의 사용 사례**
  - 헤더파일 추가

    #include <queue>

  - 표준 네임스페이스 사용하면 편리

    using namespace std;

  - 큐 변수 선언

    queue<datatype> 변수명;

    queue<char> que;

멤버 함수	
empty()	큐가 비어 있으면 true를, 비어 있지 않으면 false를 반환함.
size()	큐 요소의 총 개수를 반환함.
front()	큐의 맨 앞에 있는(제일 먼저 저장된) 요소에 대한 참조를 반환함.
back()	큐의 맨 뒤에 있는(제일 나중에 저장된) 요소에 대한 참조를 반환함.
push()	큐의 맨 뒤에 요소를 삽입함.
pop()	큐의 맨 앞의 요소를 삭제함.

- **큐의 사용 사례**

```
int n = 20;  // 20개의 피보나치 수열을 출력함.
queue<int> que;
que.push(0); // 초깃값인 0과 1을 저장함.
que.push(1);

// 피보나치 수열
for(int i = 2; i < n; i++)
{
    int temp = que.front();
    cout << temp << " ";
    que.pop();
    que.push(temp + que.front());
}
```

멤버 함수
empty()
size()
front()
back()
push()
pop()

실행 결과

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

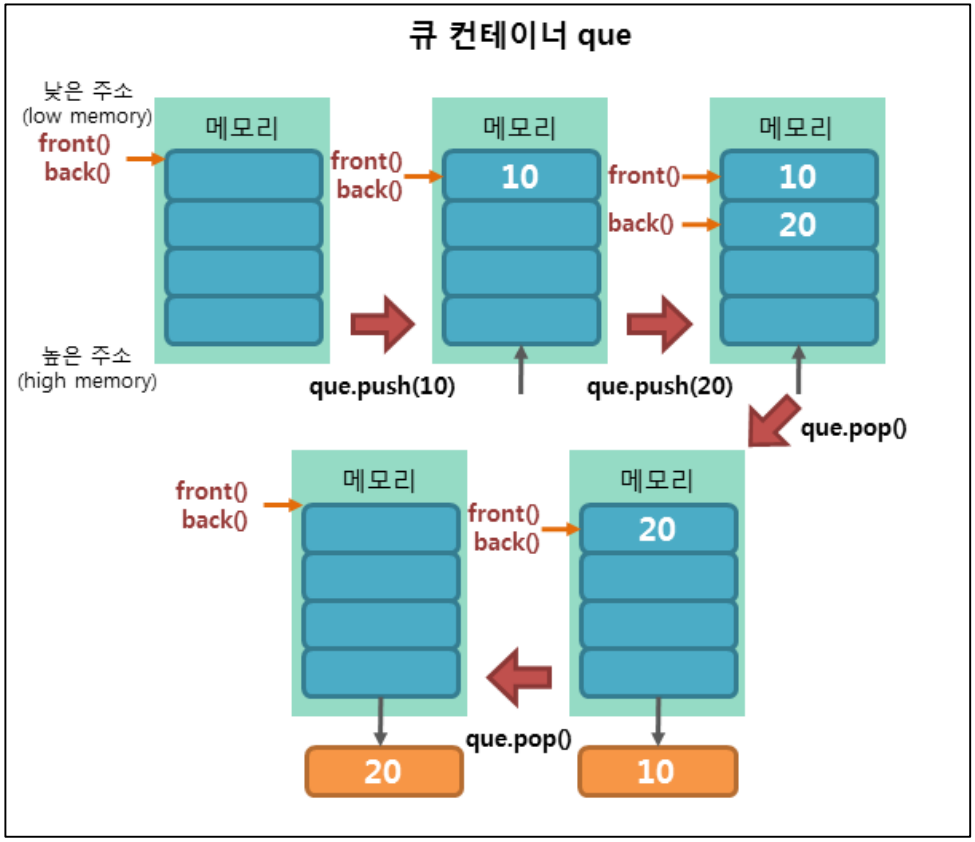
우선순위 큐

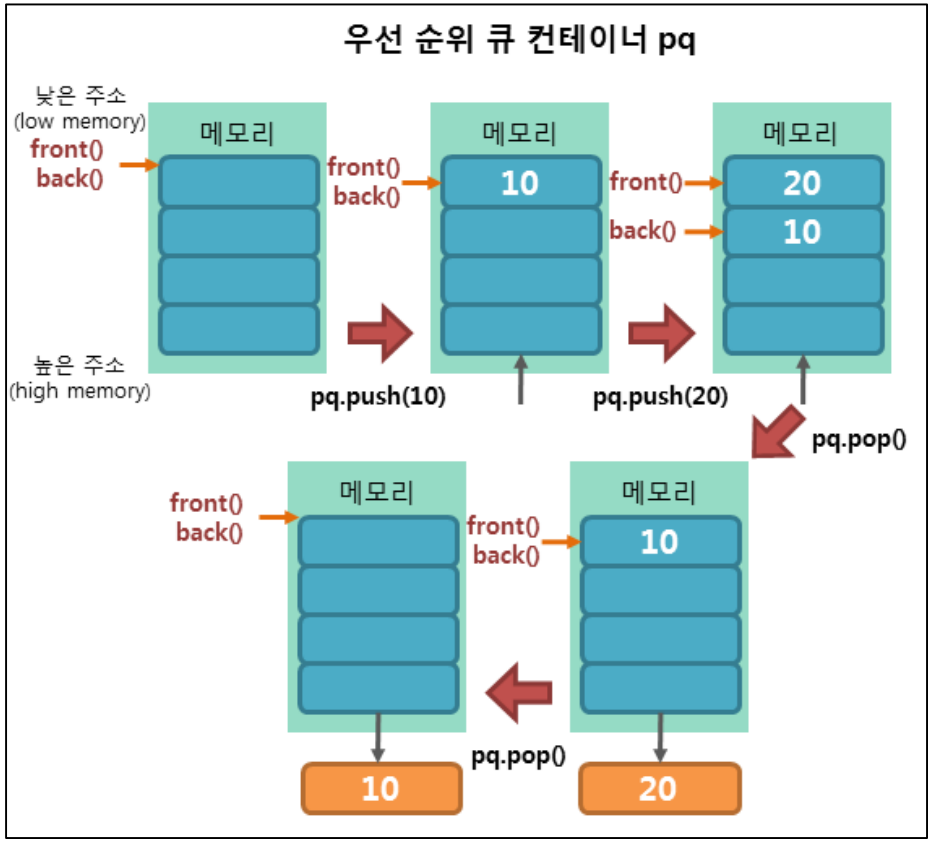
우선순위 큐

- **환자 치료의 예**
  - 큐(queue): 먼저 온 사람을 먼저 치료
  - 스택(stack): 나중 온 사람을 먼저 치료
  - 우선순위 큐(priority_queue): 위급한 사람을 먼저 치료

우선순위 큐

- **큐 vs. 우선순위 큐**
  - 큐: deque 클래스 기반
  - 우선순위 큐: vector 클래스 기반





우선순위 큐

- **우선순위 큐의 사용 사례**
  - 헤더파일 추가

    #include <queue>

  - 표준 네임스페이스 사용하면 편리

    using namespace std;

  - 우선순위 큐 변수 선언

    priority_queue<datatype> 변수명;

    priority_queue<int> que; //내림차순으로 저장

    priority_queue<datatype, container, compare> 변수명;

    container : vector<datatype>, compare : 비교함수 클래스

우선순위 큐

- Max Heap (최대 힙) : Top 부터 내림차순으로 정렬
- Min Heap (최소 힙): Top 부터 오름차순으로 정렬

```cpp
#include <iostream>
#include <functional>
#include <queue>

using namespace std;

int main(){

    // priority_queue
    priority_queue< int, vector<int>, less<int> > pq;
    // or priority_queue<int> pq;


    // push(element)
    pq.push(5);
    pq.push(2);
    pq.push(8);
    pq.push(9);
    pq.push(1);
    pq.push(14);
```

```cpp
#include <iostream>
#include <functional>
#include <queue>

using namespace std;

int main(){

    // priority_queue
    priority_queue< int, vector<int>, greater<int> > pq;


    // push(element)
    pq.push(5);
    pq.push(2);
    pq.push(8);
    pq.push(9);
    pq.push(1);
    pq.push(14);
```

우선순위 큐

- 정의한 구조체 및 비교함수 이용

```cpp
#include <iostream>
#include <queue>

using namespace std;

struct Custom{

    int x;
    int y;
    int value;
    Custom(int value) : x(0), y(0), value(value) {
    }

};


// 오름차순 정렬
struct cmp{
    bool operator()(Custom t, Custom u){
        return t.value > u.value;
    }
};
```

```cpp
int main(){

    // priority_queue
    priority_queue< Custom, vector<Custom>,  cmp > pq;


    // push(element)
    pq.push(Custom(5));
    pq.push(Custom(2));
    pq.push(Custom(8));
    pq.push(Custom(9));
    pq.push(Custom(1));
    pq.push(Custom(14));
```

우선순위 큐

- **우선순위 큐의 사용 사례**

```
priority_queue<int> pq;
pq.push(10);
pq.push(20);
pq.push(100);
pq.push(3);

// 우선순위 큐의 모든 요소를 인출
while(!pq.empty())
{
    cout << pq.top() << " ";
    pq.pop();
}
```

멤버 함수	
empty()	
size()	
top()	
push()	
pop()	Top을 삭제

실행 결과

```
100 20 10 3
```

큐 예제 풀이

문제6D. 조세퍼스의 문제

N명의 사람이 원 모양으로 둥글게 앉아있으며, 각각 시계방향으로 1~N번의 번호를 부여받는다고 하자. 아래와 같은 규칙으로 사람들을 제외해 나간다.

- 가장 처음에는 1번 사람이 지목된다.
- 이후 시계방향으로 다음 사람이 지목된다.
- 이를 반복하다가 M번째로 지목받은 사람을 제외한다. 그 이후 제외 된 사람의 다음 사람부터 위의 과정을 반복해 나간다.

예를 들어서 $N=7$이고 $M=3$인 경우를 생각해보자.

- 초기에는 $\{1, 2, 3, 4, 5, 6, 7\}$처럼 사람들이 앉아있다.
- 3번째로 지목되는 3번을 제외하여 $\{1, 2, 4, 5, 6, 7\}$이 남으며 4번부터 게임을 다시 시작한다.
- 차례로 4, 5, 6번 사람이 지목받아 3번째로 지목받은 6번이 게임에서 제외되므로 $\{1, 2, 4, 5, 7\}$의 사람이 남아 게임을 다시 시작한다.
- 차례로 7, 1, 2번 사람이 지목받아 3번째로 지목받은 2번이 게임에서 제외되므로 $\{1, 4, 5, 7\}$의 사람이 남아 게임을 다시 시작한다.
- ...

위와 같은 과정을 반복한다고 하자. N과 M이 주어졌을 때 각 번호의 사람들이 게임에서 제외되는 순서대로 출력하는 프로그램을 작성하시오.

문제6D. 조세퍼스의 문제

입력 형식

첫 줄에는 테스트케이스의 수를 나타내는 10이하의 자연수 T가 주어진다. 이후 총 T개에 대한 입력이 차례로 주어진다.

각 테스트케이스의 입력은 한 줄로 `N M`형식으로 주어진다.

- N은 사람의 수를 나타내는 5,000이하의 자연수다.
- M은 사람을 제외해 나갈 간격을 나타내는 5,000이하의 자연수다.

출력 조건

각 테스트케이스별로 한 줄에 정답을 출력한다.

- N명의 사람들의 번호를 제외 된 순서대로 출력한다.
- 각 번호는 공백으로 구분한다.

입력

```
2
4 2
7 3
```

출력

```
2 4 3 1
3 6 2 7 5 1 4
```

문제6D. 조세퍼스의 문제

```cpp
#include <iostream>
#include <vector>
#include <queue>

using namespace std;


class Player{
public:
  int index;

  Player(int index){
    this->index = index;
  }
};

vector<Player> getDeadPlayersList(int n, int m, const vector<Player>& players){
  // 현재 게임에서 제외된 플레이어들의 리스트
  vector<Player> deadPlayers;

  // 아직 게임에서 제외되지 않는 플레이어들의 리스트
  queue<Player> playerQueue;

  return deadPlayers;
}
```

문제6D. 조세퍼스의 문제

```
void process(int caseIndex) {
  int n, m;
  cin >> n >> m;

  vector<Player> players;
  for(int i = 0; i < n; i++) {
    players.push_back(Player(i + 1));
  }

  vector<Player> deadPlayers = getDeadPlayersList(n, m, players);

  for(int i = 0; i < n; i ++) {
    if(i > 0) {
      cout << " ";
    }

    Player p = deadPlayers[i];
    cout << p.index;
  }

  cout << endl;
}

int main() {
  int caseSize;
  cin >> caseSize;

  for (int caseIndex = 1; caseIndex <= caseSize; caseIndex += 1) {
    process(caseIndex);
  }
}
```

문제6G. 불안정 지역

강 하나를 따라서 N개의 도시가 일렬로 위치해 있다. 각 도시는 서쪽에서 동쪽으로 1에서 N번으로 차례로 번호를 부여하여 구분한다. 도시는 각각 독립적인 경제 기반과 정책을 바탕으로 성장하였기 때문에 경제적 성장 정도가 다를 수 밖에 없다. 사회공학자인 지수는 연속한 K개의 도시에 대하여 소득이 가장 높은 도시와 가장 낮은 도시의 소득 차가 일정이상이 되면 이 K개의 도시가 속한 영역을 불안전 지역이라고 정의한다. 불안전 지역에서는 지역 주민간 상대적 박탈감으로 인해 많은 사회적 문제가 발생할 수 있으므로 이러한 지역을 최대한 빨리 발견하여 사회 안정화를 위한 조치를 취하고자 한다.

예를 들어서 $N=7, K=3$이고 각 도시의 소득 수준이 차례로 $\{10, 2, 5, 3, 7, 9, 1\}$이라고 하자. 이 중 1~3번 도시가 포함된 영역과 5~7번 도시가 포함된 영역은 최고 소득과 최저 소득의 차가 8이므로 문제의 정답이 된다. 반면에 2~4번 도시가 포함된 영역에서는 최고와 최저 소득 차가 3이므로 앞의 두 영역에 비해서 작다.

N개의 도시에 대한 소득 수준이 입력으로 주어질 때 동서로 연속한 K개의 도시를 포함하는 영역들 중 가장 큰 소득차를 가지는 영역의 소득차이를 계산하는 프로그램을 작성하시오.

문제6G. 불안정 지역

입력 형식

첫 줄에는 테스트케이스의 수를 나타내는 10이하의 자연수 T가 주어진다. 이후 총 T개의 테스트케이스에 대한 입력이 차례로 주어진다.

각 테스트케이스의 첫 줄에는 두 개의 자연수가 공백으로 구분되어 `N K`형식으로 주어진다.

- N은 도시의 수를 나타내는 20만이하의 자연수다.
- K는 한 조사 영역이 포함하는 연속한 도시의 수를 나타내는 N이하의 자연수다.

각 테스트케이스의 두 번째 줄에는 각 도시의 소득 수준을 나타내는 N개의 자연수가 차례로 공백으로 구분되어 주어진다.

- 1번 도시부터 N번 도시까지의 소득 수준이 순서대로 주어진다.
- 입력되는 숫자는 모두 10억이하의 자연수이며 각각 공백으로 구분되어 있다.

출력 형식

각 테스트케이스에 대한 정답을 한 줄씩 출력한다.

- 연속한 K개의 도시로 이루어진 영역들 중 가장 큰 소득차를 가지는 영역의 소득차를 출력한다.

문제6G. 불안정 지역

입력

```
1
7 3
10 2 5 3 7 9 1
```

출력

```
8
```

문제6G. 불안정 지역

```cpp
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

class City {
public:
  int index;      // 도시의 인덱스
  int income;     // 해당 도시의 소득

  City(int index, int income){
    this->index = index;
    this->income = income;
  }

  bool operator < (const City& o) const {
    return this->income < o.income;
  }
  bool operator > (const City& o) const {
    return this->income > o.income;
  }
};
```

문제6G. 불안정 지역

```
int getMaximumRangeDifference(int n, int k, const vector<City>& cities) {
  int answer = 0;

  // 소득이 가장 작은 도시부터 pop되는 우선순위 큐
  priority_queue<City, vector<City>, greater<City>> rangeMinimum;

  // 소득이 가장 높은 도시부터 pop되는 우선순위 큐
  priority_queue<City> rangeMaximum;


  return answer;
}

void process(int caseIndex) {
  int n, k;
  cin >> n >> k;
  vector<City> cities;

  for(int i = 0 ; i < n ; i += 1){
    int income;
    cin >> income;
    cities.push_back(City(i, income));
  }

  int answer = getMaximumRangeDifference(n, k, cities);

  cout << answer << endl;
}

int main() {
  int caseSize;
  cin >> caseSize;

  for (int caseIndex = 1; caseIndex <= caseSize; caseIndex += 1) {
    process(caseIndex);
  }
}
```