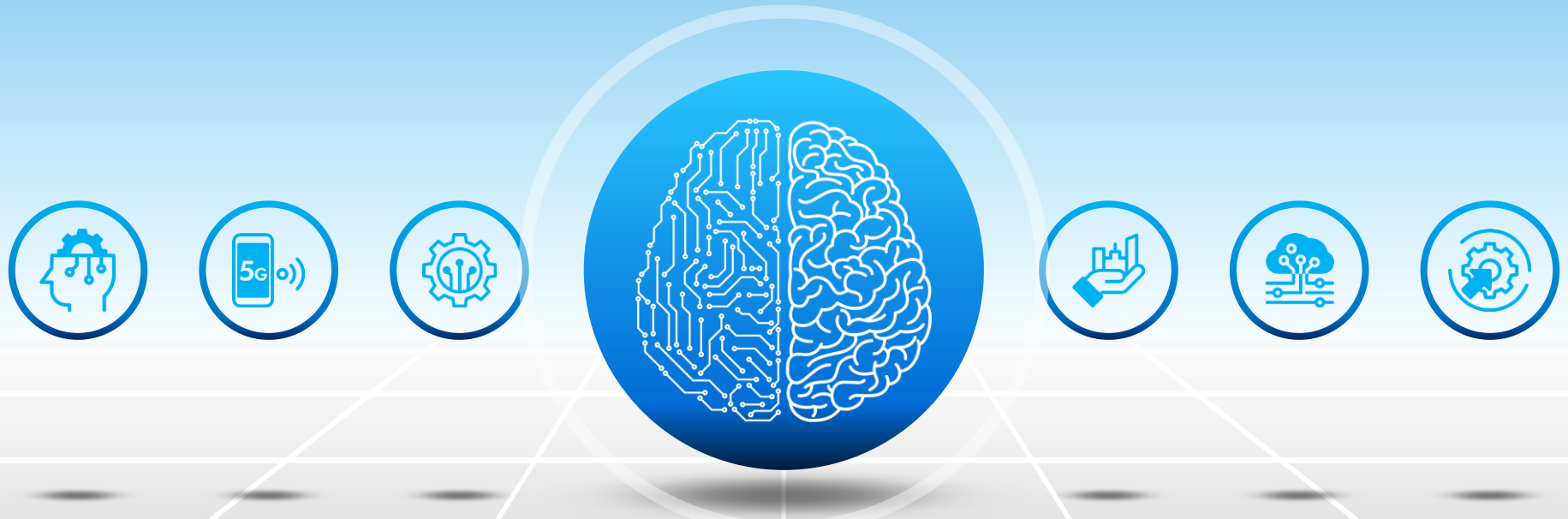
CIS3034 문제해결프로젝트

배열활용과 테크닉



목 차
CONTENTS

```
int getElementTypeCount(int data[], int n) {
	int countType = 0;
	for (int i = 0; i < n; i++) {
		if (i == 0 || data[i - 1] != data[i]) {
			countType += 1;
		}
	}

	return countType;
}
```

다양성.cpp

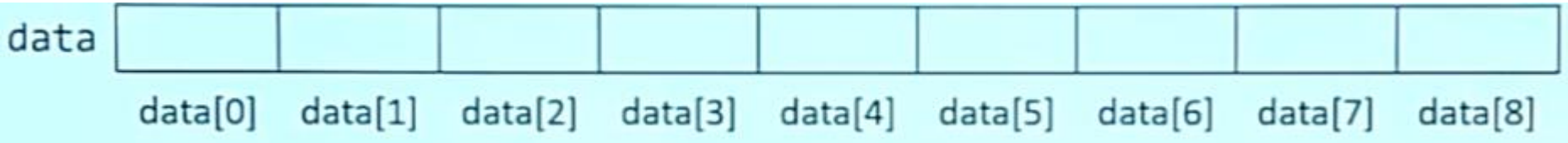
```
bool isOrdered(int data[], int n) {
	int count = 0;

	for (int i = 0; i < n-1; i++) {
		if (data[i] > data[i + 1]) {
			count += 1;
			break;
		}
	}

	if (count >= 1) {
		return false;
	} else {
		return true;
	}
}
```

오름차순인가.cpp

배열과 테크닉

배열

- 배열은 하나의 이름과 번호로 여러 저장공간을 접근할 수 있는 구조이다.
- 각 원소는 0부터 번호로 접근할 수 있으며 접근 및 수정에 걸리는 시간복잡도는 $O(1)$이다.



배열의 단점

- **배열은 아래의 단점을 유의하여 사용해야 함**
  - 인덱스는 항상 0부터 연속적으로 부여됨.
  - 기본적으로 고정적인 길이를 갖도록 미리 선언 (메모리 낭비 요인)
    - ArrayList, vector, list 등으로 어느 정도 개선 가능
  - 배열 중간에 원소를 추가, 삭제, 크기변경하는 것은 O(N)의 시간 복잡도를 갖는다. N은 배열의 전체 길이임.
  - 선언과 초기화도 길이 N에 비례한 시간이 걸림.
  - 선언한 길이만큼 메모리를 차지함.
  - C++의 경우 지역 배열을 크게 잡으면 StackOverflow 발생

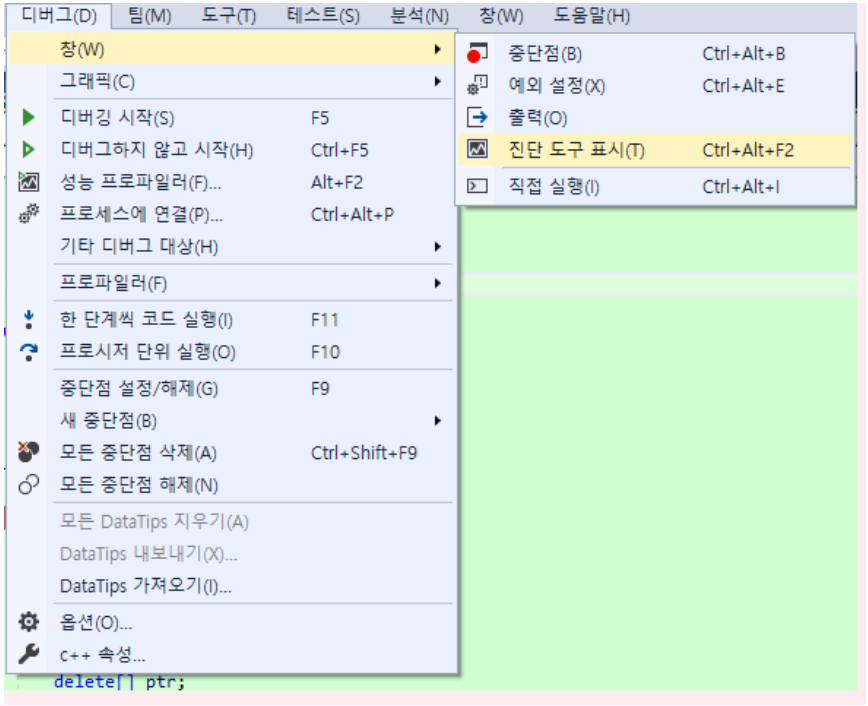
- **이후에 배울 자료구조 방법은 이러한 배열의 단점을 개선하기 위해 등장**

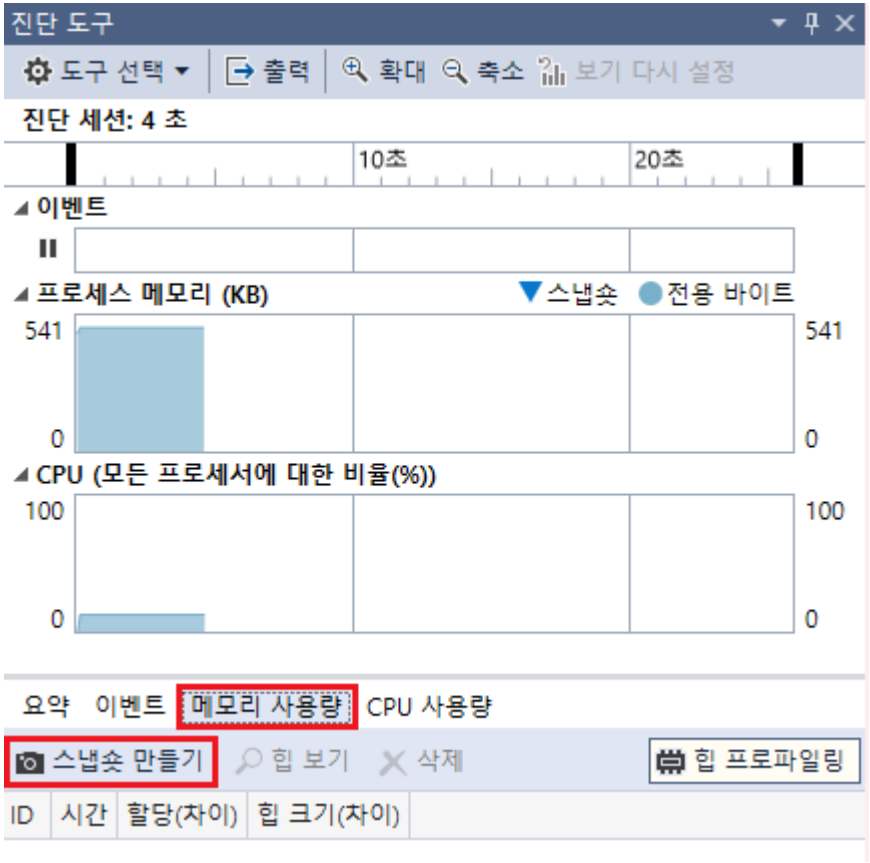
공간과 시간

- 배열이나 변수와 같은 공간에 필요한 값을 저장해두거나 미리 계산해 두면 다양한 방법으로 연산량을 줄이는 것이 가능함.
- 하지만 배열은 공간을 많이 차지하므로 사용할 수 있는 크기가 제한적이다.
- 메모리 제한 이내에서 최대한 연산량을 줄일 수 있는 방향으로 배열과 자료구조를 활용해야 한다.

메모리 진단

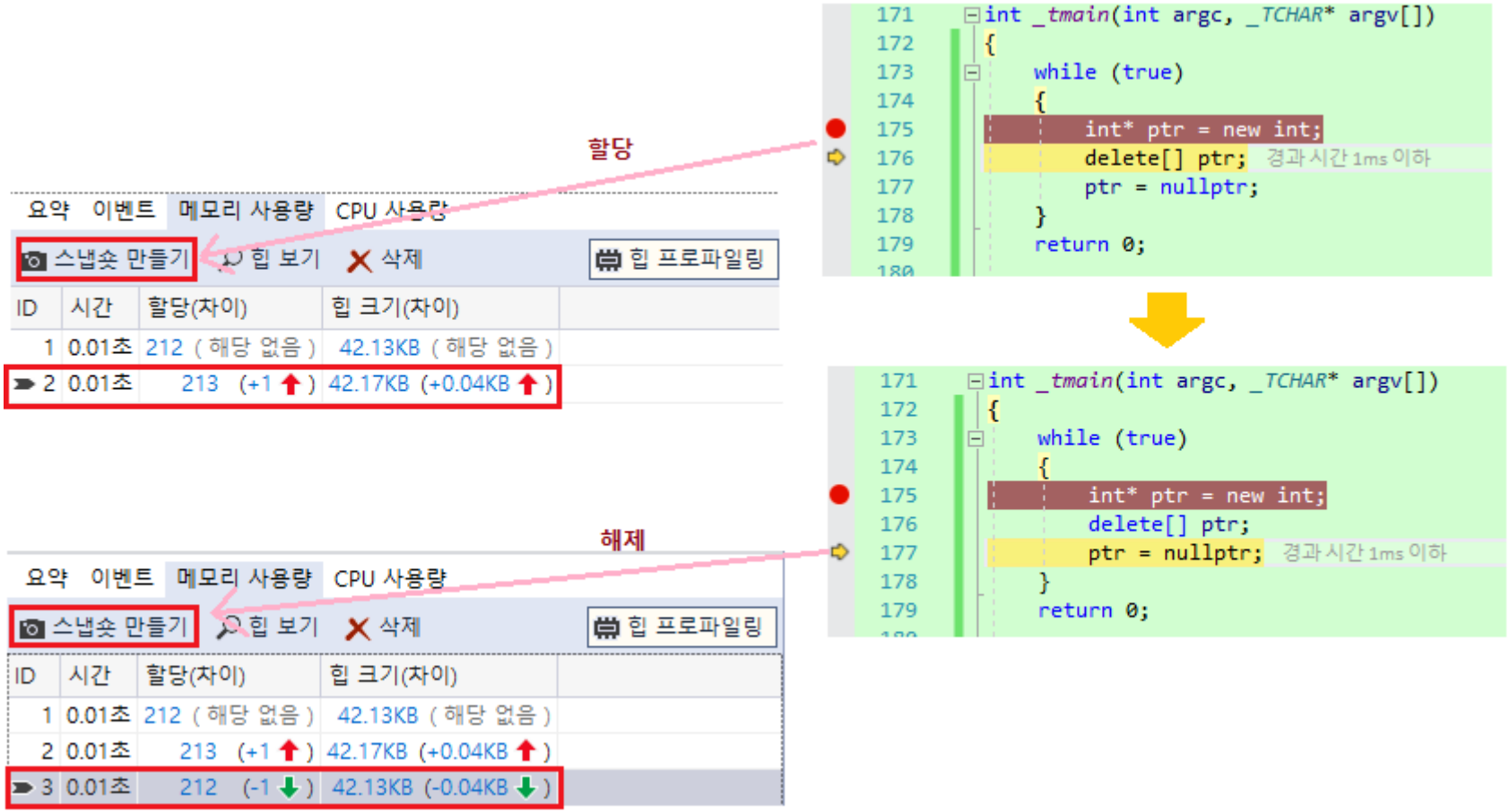
- Visual Studio **진단도구 활용 (디버깅 모드)**
  - 디버그 → 창(windows) → 진단도구 표시





메모리 진단

- Visual Studio 진단도구 활용 (디버깅 모드)
  - 디버그 → 창(windows) → 진단도구 표시



다양한 테크닉

- **전처리**(Preprocessing)
  - 필요한 값들을 미리 모두 구해 두거나, 데이터를 미리 전부 가공해두는 기법
  - 문제에서 사용될 방식에 따라 값들을 계산, 정렬, 가공하는 것

- **메모이제이션**(Memoization)
  - 중복 계산이 많이 일어날 것 같은 값들을 한 번 계산한 이후에 배열과 같은 공간에 저장해두고, 추가적인 계산이 필요할 때 다시 계산하지 않고 저장된 값을 재활용하는 기법

- **인덱싱**(Indexing)
  - 불규칙적인 데이터에 특정 기준으로 번호를 매기는 작업
  - 이렇게 붙여진 번호로 배열의 각 공간을 활용할 수 있음.

공간에 의미 부여하기

- 배열의 각 칸을 단순히 “몇 번째 칸”이 아니라, 해당 칸의 인덱스를 기준으로 다양한 의미를 붙여서 활용할 수 있어야 한다.
- 처리방법에 따라 각 칸의 의미는 계속 변함.

```
int data[10000];
for(int i=0;i<n;i++)
    scanf("%d", &data[i]);
//data[i] := i번째로 입력받은 데이터

sort(data, data+n);
//data[i] := 전체 값 중 i번째로 작은 값
```

공간에 의미 부여하기

- 미리 공간에 의미를 부여하고 해당 의미가 성립하도록 처리할 수 있다.

```
int data[10000];
int cnt[1000]; //cnt[i] := i가 등장한 횟수
for(int i=0;i<n;i++)
{
    scanf("%d", &data[i]);
    cnt[data[i]] += 1;
}
//각 cnt[i]는 i가 입력된 횟수만큼
//1씩 증가했을 것이다.
//그러므로 의미에 부합
```

예제 풀이

문제3A-전화번호

용돈이 필요해진 지수는 한 설문조사 기관에서 아르바이트를 하게 되었다. 지수는 수 많은 사람들에게 전화를 통해 설문조사를 하는 일을 맡게 되었는데, 어느 날 휴식시간에 문득 사람들이 전화번호 뒷자리로 가장 많이 사용하는 번호는 무엇인지 궁금해졌다. 하지만 지수는 보유한 전화번호가 너무 많고, 전화번호의 뒷자리는 0000~9999의 총 1만가지의 종류가 있기에 눈으로 세는 것은 도저히 불가능할 것이라는 판단을 내렸다. 지수를 위하여 사람들의 전화번호 뒷자리 중 가장 많이 사용되는 번호를 찾아주자.

입력 형식

가장 첫 줄에 전화번호의 수 N이 1이상 10만 이하의 자연수로 주어진다.

그 후 총 N줄에 걸쳐서 공백없이 4개의 자연수로 구성 된 전화번호 뒷자리가 한 줄에 하나씩 주어진다. 전화번호 뒷자리는 0000~9999사이 중 하나이다.

출력 형식

주어진 전화번호 뒷자리들 중 가장 많이 등장한 번호를 출력한다. 0을 포함하여 공백없는 네 자리 숫자로 출력해야한다.

단, 등장한 횟수가 같은 번호가 두 개 이상인 경우 가장 사전순으로 빠른 숫자를 출력한다.

문제3A-전화번호

입/출력 예시

␣ : 공백 ↲ : 줄바꿈 ↹ : 탭

예시 1

입력

```
3↲
1234↲
4321↲
1234
```

출력

```
1234
```

예시 2

입력

```
2↲
4321↲
1234
```

출력

```
1234
```

문제3A-전화번호

```cpp
#include <cstdio>

using namespace std;

const int MAX_TABLE_LENGTH = 10000;

/**
* data[0] ~ data[n-1]에 등장한 번호들에 대한 빈도수 테이블을 채우는 함수
* @param data
* @param n
* @param table table[x] := data배열에서 x가 등장한 횟수
*/
void fillFrequencyTable(int data[], int n, int table[]) {

}

/** cnt 배열이 해당 의미를 만족하도록 각 값을 채워보자 (초기값에 주의) **
* data[0] ~ data[n-1]사이에서 가장 많이 등장한 번호를 반환하는 함수
* @param data
* @param n
* @return 가장 많이 등장한 번호. 여러개인 경우 사전순으로 가장 빠른 번호.
*/
int getFrequentNumber(int data[], int n) {
  // 0000~9999 중에서 가장 많이 등장한 번호를 구해보자
  int frequent_number = 0;


  return frequent_number;
}
```

```cpp
int main() {
  int n;

  scanf("%d", &n);
  int* data = new int[n];

  for (int i = 0; i < n; ++i) {
    scanf("%d", &data[i]);
  }

  int answer = getFrequentNumber(data, n);

  // 네 자리로 출력
  printf("%04d", answer);

  delete[] data;

  return 0;
}
```

문제3B-페인트

한 아이돌 그룹은 2집 앨범 발매 기념 콘서트를 열기로 하였다. 이 콘서트의 기획을 맞게 된 당신은 2집 앨범의 컨셉트에 맞게 콘서트장을 알록달록하게 꾸미고자 한다. 하지만 디자인 감각이 전혀 없던 당신은 어떻게하면 예쁘고 불규칙적으로 알록달록하게 색상을 배치할 수 있을지 고민을 하고있다. 당신은 콘서트장에 한 줄로 배치 된 좌석들 중 연속된 몇 칸을 골라 한 가지 색으로 색칠하고, 또 다른 연속된 몇 칸을 골라 색칠하는 과정을 반복하기로 하였다.

처음에 모든 좌석은 하얀색(0번 색상)으로 색칠되어 있으며, 당신은 연속 된 몇 칸의 좌석을 골라서 한 가지 색으로 칠할 수 있다. 당신은 0번부터 99번까지 총 100가지 색상을 사용할 수 있다. 0번은 항상 흰색이라고 가정한다. 그러므로 가장 처음에 모든 좌석은 0번 색상으로 색칠되어 있다. 색깔과 색칠할 연속 된 좌석들을 선택하면 해당 좌석들은 모두 같은 색으로 칠해진다. 의자를 칠하는 페인트들은 순식간에 마르므로 두 가지 이상의 색상이 섞이는 일은 벌어지지 않는다.

예를 들어서 좌석이 1번부터 10번까지 있다고 하자. 2번부터 5번까지의 좌석을 빨간색으로 색칠하고, 그 후 4번부터 7번까지의 좌석을 파란색으로 색칠하면 결과적으로 2~3번 좌석은 빨간색, 4~7번 좌석은 파란색이 된다. 그리고 나머지 좌석들은 흰색이 된다.

당신은 이렇게 불규칙적으로 좌석들을 색칠한 후, 가장 많은 좌석이 가진 색과 반대로 가장 적은 좌석이 가진 색의 번호를 찾고자 한다. 단, 마지막에 아무 좌석에도 칠해져 있지 않은 색상은 제외한다. 두 색의 번호를 찾을 수 있는 프로그램을 작성해보자.

문제3B-페인트

입력 형식

첫 줄에는 좌석의 수 N과 색칠을 할 방법의 수 M이 차례로 주어진다. N과 M은 모두 1,000 이하의 자연수이다.

이후 M줄에 걸쳐서 공백으로 구분된 세 정수가 주어진다. 모든 규칙은 수행되는 순서대로 주어진다. 즉, 먼저 입력된 색칠 규칙이 먼저 적용된다.

- 첫 번째 수자는 색칠할 가장 왼쪽(번호가 작은) 좌석의 번호이다.
- 두 번째 숫자는 색칠할 가장 오른쪽(번호가 큰) 좌석의 번호이다.
- 세 번째 숫자는 좌석에 칠할 색깔의 번호이다.

단, 좌석의 번호는 1과 N사이의 자연수이며. 색깔의 번호는 0이상 99이하의 정수이다.

출력 형식

첫 번째 줄에 가장 많은 좌석이 칠해진 색의 번호를 출력한다.

두 번째 줄에 가장 적은 좌석이 칠해진 색의 번호를 출력한다.

조건을 만족하는 색이 두 개 이상이라면 번호가 작은 색을 출력한다.

문제3B-페인트

입/출력 예시

␣ : 공백 ↲ : 줄바꿈 ⇥ : 탭

예시 1

입력

```
10␣2↲
2␣5␣1↲
4␣7␣2
```

출력

```
0↲
1
```

예시 2

입력

```
10␣2↲
1␣9␣5↲
2␣10␣5
```

출력

```
5↲
5
```

문제3B-페인트

```cpp
#include <cstdio>
#include <vector>

using namespace std;

const int MAX_SEAT_NUMBER = 1000;
const int MAX_COLOR_NUMBER = 100;

//좌석들을 한 번 색칠하는 이벤트에 대한 정보
class Painting{
public:
  int left;
  int right;
  int color;

  Painting() { }

  Painting(int left, int right, int color) {
    this->left = left;
    this->right = right;
    this->color = color;
  }
};


/**
* data[0] ~ data[n-1]에 등장한 번호들에 대한 빈도수 테이블을 채우는
* @param data
* @param n
* @param table table[x] := data배열에서 x가 등장한 횟수
*/
void fillFrequencyTable(int data[], int n, int table[]) {

}
```

```cpp
void solve(int n, int m, const Painting* paintings)
{
  int* seats = new int[n];
  for (int i = 0; i < n; i++)
  { // 처음 좌석은 전부 0번 색으로 칠해져 있다.
    seats[i] = 0;
  }

  int minColor = seats[0]; //가장 적게 등장한 색상
  int maxColor = seats[0]; //가장 많이 등장한 색상

  printf("%d\n", maxColor);
  printf("%d\n", minColor);

  delete[] seats;
}

int main() {
  int n, m;
  scanf("%d %d", &n, &m);

  // paintings[i] := i번째에 일어난 색칠 이벤트의 정보
  Painting* paintings = new Painting[n];

  for (int i = 0; i < m; ++i) {
    scanf("%d", &paintings[i].left);
    scanf("%d", &paintings[i].right);
    scanf("%d", &paintings[i].color);

    // 좌석의 번호는 1번부터 시작하므로, 0 ~ (n-1)범위로 맞추기 위하여 1씩 빼준다
    paintings[i].left -= 1;
    paintings[i].right -= 1;
  }

  // 문제의 정답을 구한다
  solve(n, m, paintings);

  return 0;
}
```

문제3C-응모

어느 한 아이돌 그룹의 팬 클럽 *'아재리눅스'*의 회원들은 앨범과 기념품 구매에 돈을 아끼지 않기로 유명하다. 이 아이돌 그룹의 소속사는 이번 새 2집 리패키지 앨범을 구매한 회원들을 대상으로 팬 사인회를 진행하기로 결정하였다. 예상과 같이 새 앨범은 출시와 동시에 엄청난 수량이 판매되었으며, 팬 사인회에서 수용할 수 있는 구매자의 수를 아득히 넘어섰다.

결국 모든 회원을 팬 사인회에 초청할 수 없었기에 판매 된 앨범의 시리얼 번호를 통하여 추첨을 하기로 하였다. 하지만 앨범을 구매한 모든 사람이 팬 사인회 참가가 가능한 것은 아니므로 참가를 희망하는 팬들에게 본인이 구매한 앨범의 시리얼 넘버를 통해 응모할 수 있도록 하였다.

하지만 너무나도 아이돌을 만나고 싶어하는 몇몇 삼촌 팬이 같은 시리얼 번호로 여러 번 응모했다는 사실이 밝혀지면서, 이 아이돌의 매니저인 당신은 중복 응모한 시리얼 넘버를 추첨 대상에서 제외하기로 결정하였다. 하지만 응모한 사람이 너무 많은 나머지 도저히 눈과 손으로는 중복을 제거할 수 없었다. 하지만 현재는 매니저이지만, 왕년에 컴퓨터공학과를 졸업한 당신은 프로그램을 작성해 이 문제를 해결하고자 한다. 응모한 모든 시리얼 번호가 주어질 때, 이 시리얼 번호들 중 두 번 이상 응모한 번호를 모두 제거하여 오름차순으로 출력하는 프로그램을 작성해보자.

입력 형식

첫 줄에 데이터의 수 N이 1이상 10만 이하의 자연수로 공백없이 주어진다.

한 줄에 응모한 시리얼 번호들이 공백으로 구분된 1이상 10만 이하의 자연수로 주어진다.

출력 형식

응모한 모든 시리얼 번호 중 두 번이상 응모한 번호를 제외하고 나머지 번호들을 공백으로 구분하여 한 줄에 오름차순으로 출력한다. 최소 하나의 시리얼 번호는 응모 조건을 만족한다.

문제3C-응모

```cpp
#include <cstdio>
#include <vector>
#include <algorithm>

using namespace std;

const int MAX_SERIAL_NUMBER = 100000;

/**
* data[0] ~ data[n-1]에 등장한 시리얼 번호들에 대한 빈도수 테이블을 채우는 함수
* @param data
* @param n
* @param table  table[x] := data배열에서 x가 등장한 횟수
*/
void fillFrequencyTable(const vector<int> & data, int n, vector<int> &table) {

}


/**
* data[0] ~ data[n-1]에서 중복이 존재하지 않는 원소들을 반환한다.
* 단, 각 원소들은 오름차순으로 정렬되어 있어야 한다.
* @param data  data[0] ~ data[n-1]에는 10만 이하의 자연수다.
* @param n
* @return
*/
vector<int> getUniqueElements(const vector<int>& data, int n) {

  vector<int> ret; //유일한 원소들 배열


  //오름차순 순서로 추가했기 때문에 ret에 대한 정렬은 불필요하다.
  return ret;
}
```

문제3C-응모

```
int main() {
  int n;
  scanf("%d", &n);

  vector<int> data = vector<int>(n);
  for (int i = 0; i < n; ++i) {
    scanf("%d", &data[i]);
  }

  const vector<int> answer = getUniqueElements(data, n);

  // 각 원소들을 출력한다.
  for (int i = 0; i < answer.size(); ++i) {
    if (i > 0) { //첫 번째 원소가 아니라면 앞에 공백을 하나 추가한다.
      printf(" ");
    }
    printf("%d", answer[i]);
  }

  return 0;
}
```

문제3D-피보나치

피보나치 수열이란 0, 1부터 시작해서 이 전의 두 값을 더해 새로운 값을 만들어나가며 전개되는 수열이다.

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

피보나치 수열은 규칙이 간단하고 수학적으로 다양한 특징이 있기에 자주 사용되는 수열 중 하나이다. 하지만 수열이 진행될 수록 숫자가 급격히 커지기 때문에 손으로 계산하기에는 쉽지 않은 수열이다.

피보나치 수열과 8자리 전화번호의 관련성을 연구하고 있던 지애는 손으로 하던 계산을 프로그램으로 대체하기 위하여 당신에게 도움을 요청했다. 지애를 도와주기 위하여 어떤 수 N이 주어졌을 때, N번째 피보나치 수의 마지막 8자리 숫자를 정수형태로 출력하는 프로그램을 작성해주자.

예를 들어서 실제 수열의 값이 1,000,283,859이라면 283859만 출력하면 된다.

입력 형식

첫 줄에는 **테스트케이스의 수 T가 1이상 10만 이하**의 자연수로 주어진다.

각 테스트케이스의 입력은 한 줄로 구성되며, 1이상 100만 이하의 자연수 N이 공백없이 주어진다.

출력 형식

각 테스트케이스별로 한 줄에 정답을 출력한다. 피보나치 수열의 N번째 숫자를 공백없이 정수형태로 출력한다. **단, 숫자 가장 앞의 0들은 생략한다.**

단, 피보나치 수열의 1번째 숫자는 0이고 2번째 숫자는 1이다.

문제3D-피보나치

예시 1

입력

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

출력

```
0
1
1
2
3
5
```

예시 2

입력

```
5
999996
999997
999998
999999
1000000
```

출력

```
94312505
31921872
26234377
58156249
84390626
```

문제3D-피보나치

```cpp
#include <cstdio>
#include <vector>

using namespace std;


const int MAX_N = 1000000;
vector<int> FIBONACCI_TABLE;

/**
* 피보나치 수열의 1~n번째 항을 배열에 저장하여 반환해주는
* * 단, 항의 가장 뒤 8자리만을 저장한다.
* @param n
* @return fibo[i] := 피보나치 수열의 i번째 항
*/
vector<int> makeFibonacciTable(int n) {
  const int MOD = 100000000;

  // 피보나치 배열을 미리 선언해준다.
  vector<int> ret(n + 1);


  return ret;
}
```

```cpp
/**
* 피보나치 수열의 n번째 항을 반환하는 함수
* 단, 항의 가장 뒤 8자리만을 반환한다.
* @param n
* @return
*/
int getFibo(int n) {

}

int main() {
  FIBONACCI_TABLE = makeFibonacciTable(MAX_N);

  int caseSize;
  scanf("%d", &caseSize);

  for (int caseIndex = 1; caseIndex <= caseSize; ++caseIndex) {
    int n;
    scanf("%d", &n);

    // 피보나치 수열의 n번째 항을 계산하여 출력한다.
    int answer = getFibo(n);
    printf("%d\n", answer);
  }

  // 불필요한 배열은 가능하면 할당 해제해주는 버릇을 들이자.
  FIBONACCI_TABLE.clear();

  return 0;
}
```

실습 과제

- **5주차 4개 문제해결 프로그램 작성**
  - 전화번호
  - 페인트
  - 응모
  - 피보나치

- **실습과제 게시판에 업로드할 것**
  - 4개의 cpp 파일
  - 파일명은 전화번호.cpp, 페인트.cpp, 응모.cpp, 피보나치.cpp로 할 것
  - Cpp 파일에 코드를 설명할 수 있는 본인만의 주석을 상세히 작성할 것 (타인이 코드를 봤을 때도 쉽게 이해할 수 있도록 항상 주석을 상세히 작성하는 습관을 가집시다! 사회에 나가면 여럿이 공동으로 소스코드를 공유하고 프로그래밍을 통해 공동의 코드를 업데이트하는 경우가 대부분입니다. 주석이 없으면 심지어 본인이 구현한 코드도 시간이 좀 지나서 다시 보면 잘 모릅니다.)