코벤져스's PICK

미션1

6-8팀 답안>>

1. 정렬 후 비교하기

```
#include <stdio.h>

#define LENGTH 5

void sort(int array[]); // 정렬을 위한 함수

int initial[LENGTH] = {3, 2, 4, 1, 5}; // 초깃값
int anagram[LENGTH] = {5, 4, 3, 2, 1}; // 애너그램으로 변환한 값

int main(void)
{
    // 초깃값과 애너그램으로 변환한 값을 정렬
    sort(initial);
    sort(anagram);

    for (int i = 0; i < LENGTH; i++)
    {
        if (initial[i] != anagram[i]) // 두개의 정렬된 배열을 비교
        {
            printf("False₩n"); // 하나라도 다른 숫자가 있는 경우 False
            return 0;
        }
    }
    printf("True₩n"); // 배열의 모든 수가 같은 경우 True
    return 0;
}

void sort(int array[])
{
    int temp; // 임시로 값을 저장해두기 위한 변수

    // 배열의 앞과 뒤 인덱스를 비교하여 작은 숫자 찾기
    for (int i = 0; i < LENGTH; i++)
    {
        for (int j = i + 1; j < LENGTH; j++)
        {
            if (array[i] > array[j])
            {
                temp = array[i];
                array[i] = array[j];
```

```
            array[j] = temp;
        }
    }
  }
}
```

대다수의 조원이 정렬을 이용해 문제를 풀었습니다.

초깃값과 애너그램으로 변환한 값이 담긴 배열을 둘 다 정렬한 다음, 같은 인덱스를 비교해서 만약 하나라도 다른 값이 나오는 경우에는 False를 리턴하는 방식으로 미션을 해결했습니다.

2. 정렬을 사용하지 않고 해결

```
#include <stdio.h>

#define LENGTH 5

int main(void)
{
  int initial[LENGTH] = {1, 2, 3, 4, 5}; // 초깃값
  int anagram[LENGTH] = {5, 4, 3, 2, 1}; // 애너그램으로 변환한 값

  int compare_array[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // 값을 0으로 초기화
  int number;

  for (size_t i = 0; i < LENGTH; i++)
  {
        // initial(초깃값)의 각 숫자를 배열의 인덱스로 설정하고, 1씩 더한다.
    number = initial[i];
    compare_array[number] += 1;
  }

  for (size_t i = 0; i < LENGTH; i++)
  {
        // anagram(애너그램으로 변환한 값)의 각 숫자를 배열의 인덱스로 설정하고, 1씩 뺀다.
    number = anagram[i];
    compare_array[number] -= 1;
  }

  for (int i = 0; i < 10; i++)
  {
        // 배열의 값이 0이 아닌 게 하나라도 있다면 False를 반환한다.
    if (compare_array[i] != 0)
    {
      printf("False₩n");
      return 0;
    }
  }
```

```
    printf("True₩n");
    return 0;
}
```

1번의 방법으로 정렬을 하게 되면 시간복잡도가 O(N^2)O(N
2
) 이 되기 때문에 이를 더 줄일 방법을 생각해봤습니다.

우선 숫자를 이용해서 애너그램을 찾는 프로그램이기 때문에 compare_array에는 0부터 9까지의 값을 담을 수 있도록 배열의 크기를 10으로 설정해두고, 모든 값을 0으로 초기화했습니다.

초깃값이 들어 있는 배열을 반복문을 통해 순회하며 각 숫자를 compare_array의 인덱스로 설정하고, 1씩 더했습니다. 예를 들어 들어 있는 값이 3이라면 compare_array[3]의 값을 1씩 더했습니다.

애너그램으로 변환한 값을 담은 배열도 순회하며 각 숫자를 compare_array의 인덱스로 설정했는데, 이번에는 1씩 빼는 방식을 사용했습니다.

위의 과정을 마친 compare_array의 모든 값이 0이면 initial 배열과 anagram 배열이 가지고 있는 숫자의 개수가 일치한다는 뜻이기 때문에 배열의 값이 0이 아닌 게 하나라도 있다면 False를 반환했습니다.

마지막으로 각 방법의 장단점을 생각해봤습니다.

첫 번째 방법

장점 : 메모리를 초기 배열 두 개와 임시로 값을 저장하기 위한 temp에만 사용하면 된다.

단점 : 시간복잡도가 O(N^2)O(N
2
)이다.

두 번째 방법

장점 : 시간복잡도가 O(N)O(N)이다.

단점 : 메모리를 초기 배열 두 개에 compare_array를 위한 메모리를 추가로 사용해야 하기 때문에 첫 번째 방법보다 메모리 공간을 조금 더 사용한다.

code sandbox : https://bit.ly/30mBMq

1-2팀 답안>>

```
#include <stdio.h>

int main()
{
    int arr[10] = {0, };
    int number1[5] = {1, 4, 2, 5, 8};
    int number2[5] = {2, 5, 4, 3, 1};

    for (int i = 0; i < 5; i++)
    {
        arr[number1[i]]++;
        arr[number2[i]]--;
    }

    for (int i = 0; i < 10; i++)
    {
        if (arr[i] != 0)
        {
            printf("False₩n");
            return 0;
        }
    }
    printf("True₩n");
    return 0;
}
```

12-6팀 답안>>

<Seul님이 프로그래밍한 코드를 스터디를 통해 보완한 최종본>

```c
#include <cs50.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int IfAnagram(const char* s1, const char* s2);
void BubbleSort(int* number1, int* number2);

int main(){
    char input1[10], input2[10];

    printf("기본 입력값: ");
    scanf("%s", input1);

    printf("비교할 입력값: ");
    scanf("%s", input2);

    printf("=== Test %s %s ===\n", s1, s2);
    if (IfAnagram(s1, s2)){
        printf("True\n");
    }
    else{
        printf("False\n");
    }

    return 0;
}
```

```
29 int IfAnagram(const char* s1, const char* s2){
30     int i, j, tmp1, tmp2;
31     int check;
32     int comp1[10], comp2[10];
33 
34     if (strlen(s1) != strlen(s2)){
35         return 0;
36     } // 비교 전 길이 확인
37 
38     tmp1 = atoi(s1);
39     tmp2 = atoi(s2);
40     for(i=0;i<strlen(s1);i++){
41         comp1[i] = tmp1 % 10;
42         comp2[i] = tmp2 % 10;
43         tmp1 = tmp1/10;
44         tmp2 = tmp2/10;
45     } // 문자열을 숫자로 정렬하기 위해 바꿔주는 부분
46 
47     BubbleSort(comp1, comp2);
48 
49     for (int i =0 ; i < strlen(s1) ; i++){
50         if (comp1[i]!=comp2[i]){
51             return 0;
52         }
53     }// 앞에서부터 정렬한 배열을 비교하고 다를 경우 바로 '0'리턴
54 
55     return 1; // 정렬한 배열이 모두 일치할 경우 '1' 리턴
56     
57 }
```

```
59 void BubbleSort(int* number1, int* number2){
60     int temp;
61 
62     for (int i = 0; i < 5; i++) {
63         for ( int j = 0; j < 5; j++) {
64             if (number1[j] > number1[j+1]){
65                 temp = number1[j];
66                 number1[j] = number1[j+1];
67                 number1[j+1] = temp;
68             } //샘플미션에서 진행한 bubble sort를 그래로 사용한 부분
69             if (number2[j] > number2[j+1]){
70                 temp = number2[j];
71                 number2[j] = number2[j+1];
72                 number2[j+1] = temp;
73             }
74         }
75     } // 비교할 배열들을 모두 정렬
76 }
```

비교할 입력값을 엔터로 구분해서 2개로 받고 문자열에 저장한다. int IfAnagram(const char* s1, const char* s2) 함수를 통해서 첫번째 입력값이 재배열하여 2번째 입력값이 나올 수 있 는지 확인합니다. 먼저, 입력값의 길이 비교를 통해 1차 비교를 합니다.

2차 비교에서, 문자열로 하나씩 루프를 돌면서 비교하는 방법과 숫자로 배열을 바꿔서 정렬한 다음 한 번에 비교하는 방법을 생각했습니다. 샘플미션을 고려해서 BubbleSort를 쉽게 사

용하기 위해 문자열을 숫자로 바꿔주는 작업을 먼저 한 다음 Sort를 통해 2개의 int형 배열을 비 교하였습니다.

<코드 - 직접 문자열 비교를 통해 애너그램 확인하기>

```
char buf[10] = "";
strcpy(buf, s1);
for (i = 0; s2[i]; i++)
{
    check=0;
    for (j = 0; buf[j]; j++)
    {
        if (s2[i] == buf[j])
        {
            strcpy(buf + j, buf + j + 1);
            check=1;
            break;
        }
    }
    if(check==0)
    {
        return 0;
    }
}
return 1;
```

strcpy를 사용해서 s1의 문자열을 기준으로 s2가 s1에 있는지 이중 for문으로 확인하였습니다.

〈출력 결과〉

```
$ ./a.out
기본 입력값: 1234
비교할 입력값: 4321
=== Test 1234 4321 ===
True
$ ./a.out
기본 입력값: 12345
비교할 입력값: 65432
=== Test 12345 65432 ===
False
```

미션2

3-9팀 답안>>

```
#include <stdio.h>

int main(void)
{
    int input[7] = {2, 1, 3, 5, 4, 3, 3};
    int number_of_friends = sizeof(input) / sizeof(int);

    // 오름차순 정렬: 버블 정렬
    int temp;
    for (int i = 0; i < number_of_friends; i++)
    {
        for (int j = 0; j < number_of_friends - i - 1; j++)
        {
            if (input[j] > input[j+1])
            {
                temp = input[j];
                input[j] = input[j + 1];
                input[j + 1] = temp;
            }
        }
    }

    // 중앙값 선택: 배열의 크기가 짝수일 때
    if (number_of_friends % 2 == 0)
    {
        int mid1 = input[number_of_friends / 2 - 1];
        int mid2 = input[number_of_friends / 2];

        // 중앙값 2개의 값이 같은 경우
        if (mid1 == mid2)
        {
            printf("출력값: %i₩n", mid1);
            return 0;
        }

        // 중앙값 2개의 값이 다른 경우
        else
        {
            printf("출력값: ");
            for (int i = 0; i <= mid2 - mid1; i++)
            {
                printf("%i ", mid1 + i);
            }
            printf("₩n");
        }
    }
```

```
    // 중앙값 선택: 배열의 크기가 홀수일 때
    else
    {
        int mid = input[(number_of_friends - 1) / 2];

        printf("출력값: %i₩n", mid);
    }
}
```

오름차순으로 정렬할 때 어떤 정렬 방법을 사용하느냐에 따라 시간복잡도(Big O)가 달라집니다.

병합 정렬로 코딩할 경우 시간복잡도(Big O)가 $n*log(n)$ 으로 가장 작고 답변처럼 버블 정렬로 코딩할 경우 시간복잡도(Big O)가 n^2
2
이 됩니다.

11-7 답안>>

```
#include<stdio.h>

int main(void)
{
   int friendHouse[4] = {1, 3, 4, 5};
   int temp;
   int n = sizeof(friendHouse)/sizeof(int);
   int average;


   for (int i = 0; i < n - 1 ; i++)
   {
      for (int j = 0; j < n - i - 1; j++)
      {
         if (friendHouse[j] > friendHouse[j+1])
         {
            temp = friendHouse[j];
            friendHouse[j] = friendHouse[j+1];
            friendHouse[j+1] = temp;
         }
      }
   }


   if(n == 0)
   {
      average = (friendHouse[(n/2)-1] + friendHouse[(n/2)]) / 2;
      printf("%i₩n",average);
   }
   else
   {
       printf("%i₩n",friendHouse[((1+n)/2)-1]);
   }

}
```

저희 팀은 친구들과 최단거리에 있는 집이 결국 '중앙값'임을 알게 되었고, 그래서 일단 버블정렬을 통해 배열을 정렬하고 중앙값을 찾았습니다. 이때, 입력값 배열의 크기가 홀수일 때와 짝수일 때를 나누어 결과값을 출력하였습니다.

미션3

14-9팀 답안>>

```
#include <stdio.h>
#define NUM_OF_PEOPLE 4
int timeOfPeople[NUM_OF_PEOPLE] = {1, 2, 5, 10};
int totalTime = 0;

void returnBridge(int fast)
{
   totalTime += timeOfPeople[fast];
   printf("%d₩n", timeOfPeople[fast]);
}

void crossBridge(int slow, int slower)
{
   totalTime += timeOfPeople[slower];
   printf("%d %d₩n", timeOfPeople[slow], timeOfPeople[slower]);
}

void bridge(int numOfPeople)
{
   if (numOfPeople == 1)
      returnBridge(0);
   else if(numOfPeople == 2)
      crossBridge(0, 1);
   else if(numOfPeople == 3)
   {
      crossBridge(0, 1);
      returnBridge(0);
      crossBridge(0, 2);
   }
   else
   {
      crossBridge(0, 1);
      returnBridge(0);
      crossBridge(numOfPeople - 2, numOfPeople - 1);
      returnBridge(1);
      bridge(numOfPeople - 2);
   }
}

int main(void)
{
   bridge(NUM_OF_PEOPLE);
   printf("Total Time: %d₩n", totalTime);
}
```

<이해를 돕기 위한 시각화 코드>

```
#include <stdio.h>
#define NUM_OF_PEOPLE 4
int timeOfPeople[NUM_OF_PEOPLE] = {1, 2, 5, 10};  // 각 사람들이 다리를 건너는 시간. 다리를 건너기 전의 상태를 보여주는 배열
int afterBridge[NUM_OF_PEOPLE] = {0, 0, 0, 0}; // 다리를 건넌 도착상태를 알려주는 배열
int totalTime = 0;

void printArray() // 함수 시작 전에 초기화되어 있는 배열 내용 출력
{
   printf("|");
   for (int i = 0 ; i< NUM_OF_PEOPLE; i++)
      printf("%d ", timeOfPeople[i]);
   printf("|----------------|");
   for (int i = 0; i < NUM_OF_PEOPLE; i++)
      printf("%d ", afterBridge[i]);
   printf("|₩n");
}

void printBefore()     //timeOfPeople 배열을 출력하는 함수.
{
   printf("|");
   for (int i = 0; i < NUM_OF_PEOPLE; i++)
      printf("%d ", timeOfPeople[i]);
   printf("|");
}

void printAfter()      //afterBridge 배열을 출력하는 함수
{
   printf("|");
   for (int i = 0; i < NUM_OF_PEOPLE; i++)
      printf("%d ", afterBridge[i]);
   printf("|₩nTotal Time : %d₩n", totalTime);
}

void returnBridge(int fast)                  // 한 사람이 다리를 건너는 함수 or 다리를 건넜다 돌아오는 함수
{
   totalTime += afterBridge[fast];           // 한 사람이 다리를 건너는 시간을 총 시간에 더하기
   timeOfPeople[fast] = afterBridge[fast];     // ex) 1, 2, 3 : {1, 2, 0} {0, 0, 3} -> {1, 2, 3} {0, 0, 3}
   afterBridge[fast] = 0;                    // ex) 1, 2, 3 : {1, 2, 0} {0, 0, 3} -> {1, 2, 3} {0, 0, 0}
   printBefore();
   printf("<-------%d-------", timeOfPeople[fast]);
   printAfter();
}

void crossBridge(int slow, int slower)          //두 사람이 다리를 건너는 함수. 둘이 다리를 건널 때는 느린 사람과 더 느린 사람이 존재하므로 변수 이름을 slow와 slower로 지정.
{
   totalTime += timeOfPeople[slower];          // 더 느린 사람의 시간을 총 걸린 시 간에 더함
```

```
    afterBridge[slow] = timeOfPeople[slow];     //다리를 건너 반대편에 도착한 상태  표시.
    afterBridge[slower] = timeOfPeople[slower]; //다리를 건너 반대편에 도착한 상태  표시
    timeOfPeople[slow] = 0;                     //다리를 건넜으니 원래 배열의 원소는 0으로 초기화.
    timeOfPeople[slower] = 0;                   //다리를 건넜으니 원래 배열의 원소는 0으로 초기화
    printBefore();
    printf("------%d %d------>", afterBridge[slow], afterBridge[slower]);
    printAfter();
}

void bridge(int numOfPeople)                    //다리를 건너는 전체 과정을 나타내는 함수
{
    if (numOfPeople == 1)                       //1명이 건널 때 = 다리를 건넜다가 다시 1명이 돌아올 때
        returnBridge(0);                        //배열의 인덱스로 사용되므로 1명이어도 인자로 0을 줌.
    else if(numOfPeople == 2)                   //2명이 다리를 건너는 경우, 그냥 같 이 건너면 됨.
        crossBridge(0, 1);
    else if(numOfPeople == 3)                   // 3명이 다리를 건너는 경우,
    {
        crossBridge(0, 1);                      // 1) 셋 중 빠른 2명이 먼너 건넘
        returnBridge(0);                        // 2) 먼저 건넌 2명 중 빠른 사람(셋 중 가장 빠른 사람)이 다시 다리를
건너옴
        crossBridge(0, 2);                      // 3) 가장 빠른 사람과 가장 느린 사 람이 함께 다리를 건넘
    }
    else                                        // 4명 이상이 다리를 건너는 경우
    {
        crossBridge(0, 1);                      // 1) 첫번째로 빠른 사람과  두 번째로 빠른 사람이 다리를 건넘
        returnBridge(0);                        // 2) 그 중 첫번째로 빠른 사람이 다시 다리를 건너 돌아옴
        crossBridge(numOfPeople-2, numOfPeople-1);  // 3) 제일 느린 사람과 그 다음으로 느린 사람이
다리를 건넘
        returnBridge(1);                        // 4) 다리를 건넜던 두 번째로 빠른 사람이 다시 다리를 건너옴
        bridge(numOfPeople-2);                  // 5) 4번의 상태가 완료되면, 상 황은 numOfPeople-2 수의
사람을 처음부터 옮기는 것과 동일해짐
    }
}

int main(void)
{
    printf("START TO CROSS BRIDGE₩n");
    printArray();
    bridge(NUM_OF_PEOPLE);
}
```

미션4

15-5팀 답안>>

```
#include <stdio.h>                  // 표준입출력 (printf, scanf) 함수 이용
#include <stdlib.h>                 // calloc, free 함수를 이용하여 배열을 동적으로 할당 및 해제
#include <stdbool.h>                // bool 자료형과 true, false 리터럴 사용

void cal_fall_depth(int *, int *, int); // 각 열에서 가장 많이 낙하하는 상자(최상단 상자)의 낙하거리 계산
int max_element(int *, int);            // 배열에서 최대의 요소를 구하여 반환하는 함수

int main() {
    int room_length = 1,            // 방의 가로 길이 N
        room_height = 1;            // 방의 세로 길이 M

    while (room_length < 2 || room_length > 100 || room_height < 2 || room_height > 100) {
                                    // 범위(2 ~ 100)내에 있을 때까지 N, M 값을 입력
        printf("Enter room length N and room height M (2 ~ 100), separated by a space: ");
        scanf("%d%d", &room_length, &room_height);
        while ((getchar()) != '₩n');    // 입력 버퍼 초기화
    }                               // (의도한 입력 수보다 많이 입력한 경우 다음 입력으로 저장 방지)

    bool invalid_input = true;      // 범위에서 벗어난 상자 높이값이 있는지 여부
    int i, *box_height = calloc(room_length, sizeof(int));
                                    // 각 열의 상자 높이를 저장할 배열을 동적으로 할당

    while (invalid_input) {             // N개의 모든 높이값이 범위(0 ~ M)에 맞을 때까지 입력
        printf("Enter box height values (0 ~ M) for N columns, with each value spaced: ");
        for (i = 0; i < room_length; i++)
            scanf("%d", &box_height[i]);
        while ((getchar()) != '₩n');

        invalid_input = false;
        for (i = 0; i < room_length; i++) {
            if (box_height[i] < 0 || box_height[i] > room_height) {
                invalid_input = true;
                break;
            }
        }
    }                               // (여기까지는 숫자들을 입력받는 부분)

    int *fall_depth = calloc(room_length, sizeof(int));
                                    // 각 열의 최상단 상자가 낙하하는 거리를 저장할 배열 초기화
                                    // (calloc 함수는 배열의 모든 요소를 0으로 초기화)
    cal_fall_depth(box_height, fall_depth, room_length);
                                    // 낙하거리 계산
    int max_depth = max_element(fall_depth, room_length);
                                    // 계산된 낙하거리 중 최댓값을 저장
    printf("The maximum drop distance is %d!₩n", max_depth);
```

```
                                    // 최대 낙하거리 출력
    free(box_height);               // 동적 할당 메모리 해제
    free(fall_depth);
}

void cal_fall_depth(int *heightV, int *depthV, int length) {
                                    // 각 열의 (최대) 낙하거리를 계산하는 함수의 정의
    for (int i = 0; i < length; i++)
        if (heightV[i] > max_element(heightV, i))
                                    // 왼쪽의 모든 열보다 더 높은 열들의 낙하거리만 필요하므로,
                                    // 나머지 열들은 낙하거리 미계산, 결과적으로 depthV[i] 값(0) 유지
            for (int j = i + 1; j < length; j++)
                if (heightV[j] < heightV[i])
                    depthV[i]++;        // i번째 열보다 상자가 더 낮게 쌓인 i번째 이후 열의 개수가
}                                   // 최종적으로 depthV[i]에 저장됨

int max_element(int *numV, int num) {
                                    // 배열 요소들의 최댓값을 구하는 함수의 정의
    if (num <= 0)
        return 0;

    int max = numV[0];
    for (int i = 1; i < num; i++)
        if (numV[i] > max)
            max = numV[i];
    return max;
}
```

[출력 결과]

Week4/

```
~/Week4/ $ ./MaxFallDepth
Enter room length N and room height M (2 ~ 100), separated by a space: 9 8 (test case)
Enter box height values (0 ~ M) for N columns, with each value spaced: 7 4 2 0 0 6 0 7 0
The maximum drop distance is 7!
~/Week4/ $ ./MaxFallDepth
Enter room length N and room height M (2 ~ 100), separated by a space: 5 5
Enter box height values (0 ~ M) for N columns, with each value spaced: 1 2 3 4 5
The maximum drop distance is 0!
~/Week4/ $ ./MaxFallDepth
Enter room length N and room height M (2 ~ 100), separated by a space: 5 5
Enter box height values (0 ~ M) for N columns, with each value spaced: 5 4 3 2 1
The maximum drop distance is 4!
~/Week4/ $ ./MaxFallDepth
Enter room length N and room height M (2 ~ 100), separated by a space: 10 3
Enter box height values (0 ~ M) for N columns, with each value spaced: 1 3 0 2 1 0 3 3 0 2
The maximum drop distance is 6!
~/Week4/ $
```

[시간복잡도]

0. 공통 기초지식
   - n개 요소의 최댓값 구하는 데 걸리는 시간: n
   - $M \times N$ 방에서 (n, m) 박스의 낙하거리 계산 시간: $N - n$

1. 우리 팀의 방법
   - n번째 열의 낙하거리 계산 시간: $n + (N - n) = N$
   - 최대 N개 열의 낙하거리 계산 시간: $N \times N = N^2$ (1)

```
1  void cal_fall_depth(int *heightV, int *depthV, int length) {
2      for (int i = 0; i < length; i++)
3          if (heightV[i] > max_element(heightV, i))        // 시간: i
4              for (int j = i + 1; j < length; j++)
5                  if (heightV[j] < heightV[i])
6                      depthV[i]++;                         // 시간: length - i
7  }
```

   - 최대 N개 열 낙하거리의 최댓값 검색 시간: N (2)
   - $O((1) + (2)) = O(N^2 + N) = O(N^2)$

2. 문제에서 제시된 방법 ($M \times N$ 방의 모든 박스에 대해 낙하거리를 계산후 비교)
   - n번째 열의 각 박스 낙하거리 계산 시간: $N - n$
   - n번째 열의 모든 박스 (최대 M개) 낙하거리 계산 시간: $M(N - n)$
   - 모든 열의 모든 박스 낙하거리 계산 시간: $\sum_{n=0}^{N} M(N-n) = M\sum_{n=0}^{N}(N-n) = M\sum_{n=0}^{N} n \approx \frac{1}{2}MN^2$ (1)
   - MN개 박스 낙하거리의 최댓값 검색 시간: MN (2)
   - $O((1) + (2)) = O(\frac{1}{2}MN^2 + MN) = O(MN^2)$

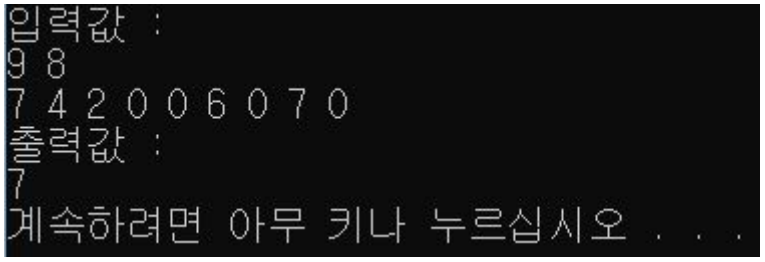
8-4팀 답안>>

```
#include <stdio.h>
#include <stdlib.h>

// Box 정보 구조체
typedef struct {
	int n; // 방의 가로 길이 n
	int m; // 세로 길이 m
	int *box_height; // box 높이를 저장할 배열
} Box;

// 합병 정렬 함수
void merge_sort(Box box, int p, int r);
void merge(Box box, int p, int q, int r);

// 최대 낙하거리 계산 함수
int fall_distance(Box box, Box box_sort);
int max_fall_distance(Box box);


int main(void)
{
	// 입력할 box의 높이
	// int box_input[9] = { 7, 4, 2, 0, 0, 6, 0, 7, 0 };
	int box_input[9] = { 3, 1, 1, 1, 1, 1, 1, 7, 6 };

	Box box;

	// 방의 가로 길이 n와 세로 길이 m 입력
	box.n = 9;
	box.m = 8;

	// 방의 가로 길이(n) 크기의 box 높이를 저장할 배열 선언
	box.box_height = malloc(sizeof(int) * box.n);


	// box 높이 입력
	for (int i = 0; i < box.n; i++) {
		box.box_height[i] = box_input[i];
	}


	// 입력값 출력
	printf("입력값 : ₩n");
	printf("%d %d₩n", box.n, box.m);
	for (int i = 0; i < box.n; i++)
		printf("%d ", box.box_height[i]);
	printf("₩n");
```

```
	// 출력값(최대 낙하거리 계산) 출력
	printf("출력값 : ₩n");
	printf("%d ₩n", max_fall_distance(box));

	free(box.box_height);

	return 0;
}

// 합병 정렬 함수
void merge_sort(Box box, int p, int r) {
	int q;
	if (p < r) {
		q = (p + r) / 2;
		merge_sort(box, p, q);
		merge_sort(box, q + 1, r);
		merge(box, p, q, r);
	}
}

void merge(Box box, int p, int q, int r) {
	int i = p, j = q + 1, k = p;
	int *tmp;

	tmp = malloc(sizeof(int) * box.n);

	while (i <= q && j <= r) {
		if (box.box_height[i] <= box.box_height[j])
			tmp[k++] = box.box_height[i++];
		else
			tmp[k++] = box.box_height[j++];
	}
	while (i <= q)
		tmp[k++] = box.box_height[i++];

	while (j <= r)
		tmp[k++] = box.box_height[j++];

	for (int a = p; a <= r; a++)
		box.box_height[a] = tmp[a];

	free(tmp);
}

// 최대 낙하거리 계산 함수
int fall_distance(Box box, Box box_sort) {

	int max_box = 0;

	for (int i = 0; i < box_sort.n; i++) {
```

```
                for (int j = i + 1; j < box_sort.n; j++) {
                        if (box.box_height[i] == box_sort.box_height[j]) {
                                if (j - i > max_box) {
                                        max_box = j - i;
                                        j = box_sort.n;

                                }
                        }
                }
                // printf("[ %d %d ]₩n", box.box_height[i], max_box);

        }

        // 최대 낙하거리 리턴
        return max_box;
}


int max_fall_distance(Box box) {
        // 모든 낙하거리를 구할 필요 없이 정렬하여 최대 낙하거리만을 구함

        // 정렬할 임의의 box 구조체 선언
        Box box_sort;
        box_sort.n = box.n;
        box_sort.m = box.m;
        box_sort.box_height = malloc(sizeof(int) * box_sort.n);

        for (int i = 0; i < box_sort.n; i++)
                box_sort.box_height[i] = box.box_height[i];

        // 합병 정렬
        merge_sort(box_sort, 0, box_sort.n - 1);

        /*
        // 정렬 후 배열
        for (int i = 0; i < box_sort.n; i++) {
                printf("%d ", box_sort.box_height[i]);
        }
        printf("₩n");
        */

        // box 낙하 최대 거리 저장 변수
        int distance = fall_distance(box, box_sort);

        free(box_sort.box_height);

        return distance;

}
```

실행 결과



시간복잡도(Big O) : O(n^2)O(n
2
)

코드 작성 과정
1) 직관적인 방법 : MxN내의 모든 box에 대해서 낙하거리를 계산한 뒤 정렬 알고리즘을 사용하여 최댓값 찾기

최대 M 높이의 박스가 있을 수 있고, 따라서 최대 MxN개의 박스가 존재할 수 있습니다.

2차원 배열로 표현되었다고 가정하고, 각각의 박스 높이에 대해 정렬을 수행한 후,

기존의 박스와 중력이 작용한 박스를 비교하여 낙하 거리를 구할 수 있습니다.

즉, 기존의 MxN개의 박스에 대해 중력이 작용한 MxN개의 박스 위치를 비교해야합니다.

M과 N의 값을 거의 비슷하다고 볼 때,

총 n^2개의 박스에 대해 하나의 박스 당 최대 n^2번 위치를 탐색해야하므로,

시간복잡도(Big O)는 O(n^4)O(n
4
) 입니다.

2) 개선된 방법 (코드에 적용)

(1) 합병 정렬

합병 정렬의 시간복잡도가 으로 좋은 성능에 기여할 수 있다고 생각하여 선정하였습니다.
- CS50 강의에서 합병 정렬에 대해 배울 수 있었으나 저를 포함하여 팀원 모두가 구현 과정에서 어려움을 겪어 검색을 통해 아래의 링크를 참고하였습니다.

https://gmlwjd9405.github.io/2018/05/08/algorithm-merge-sort.html

(2) 박스 높이에 따른 최대 낙하거리만 계산

제시된 상황 외에도 팀원들과 여러 상황을 시뮬레이션을 해보았고,

박스의 높이에 위치하는 박스의 낙하거리만으로도 최대 낙하거리를 구할 수 있었습니다.

따라서 2차원 배열을 사용하여 모든 박스를 표현하는 대신,

1차원 배열을 사용하여 박스의 높이만을 표현하였습니다.

낙하거리를 계산하기 위해 기존의 박스의 높이와 정렬된 박스의 높이가 같은 경우,

배열의 인덱스를 비교하였습니다.

임의의 상황 테스트
비교적 낮은 높이의 박스가 최대 낙하거리를 가지는 경우 (정렬 후 배열 출력)

5-1팀 답안>>

```
#include <stdio.h>
#include <string.h>

int getMatrix(int *nLow, int *nColumn, int *arrBoxStack); // 행 열 정의 및 상자들이 쌓여 있는 높이 받기
int setBoxInRoom(int nLow, int nColumn, int arrBoxStack[], char arrBoxRoom[][100]); // 방(2차원 배열)에 상자 배치
int setDropBox(int nLow, int nColumn, char arrBoxRoom[][100]); // 상자 낙하 거리 계산

int main(void)
{
int nLow, nColumn, i, j;
int arrBoxStack[100],nMaxDropBox;
char arrBoxRoom[100][100] = {" ", };

// 행 열 정의 및 상자들이 쌓여 있는 높이 받기
getMatrix(&nLow, &nColumn, arrBoxStack);
// 방(2차원 배열)에 상자 배치
setBoxInRoom(nLow, nColumn, arrBoxStack, arrBoxRoom);
// 상자 낙하 거리 계산
nMaxDropBox = setDropBox(nLow, nColumn, arrBoxRoom);

printf("최대 낙하한 상자는 %d칸 낙하 했습니다.₩n", nMaxDropBox);

}

// 행 열 정의 및 상자들이 쌓여 있는 높이 받기
int getMatrix(int *nLow, int *nColumn, int *arrBoxStack)
{
int x, y;
int i;

while(1)
{
printf("열(N)과 행(M)을 입력 하세요 ");
scanf("%d %d", &x, &y);
if(x >= 2 && x <= 100)
if(y >= 2 && y <= 100) break;

printf("₩n열과 행은 2-100 사이의 값이어야 합니다.₩n다시 입력 하십시오.₩n");
}

*nColumn = x;
*nLow = y;
printf("입력하신 열(N) %d 행(M) %d 입니다.₩n", *nColumn, *nLow);

//printf("총 %d 열에 쌓을 상자의 수를 입력 하세요₩n열의 높이는 %d을 넣을수 없습니다.₩n", x, y);
for(i = 0; i < x; i++)
{
```

```
scanf("%d", &arrBoxStack[i]);
if(arrBoxStack[i] > y || arrBoxStack[i] < 0) {
printf("잘못 입력하였습니다₩n다시 입력해 주세요₩n");
i = -1;
continue;
}
}

return 0;
}

// 방(2차원 배열에 상자 배치
int setBoxInRoom(int nLow, int nColumn, int arrBoxStack[], char arrBoxRoom[][100])
{
int i, j, cnt, nStart;

nStart = nLow-1;
for(j = 0; j < nLow; j++)
{
cnt = nLow-arrBoxStack[j] - 1;
// printf("******₩ncnt: %d. %d₩n", cnt, nLow);
for(i = nStart; i > cnt; i--)
{
arrBoxRoom[i][j] = '#';
// printf("^^^^^₩ni: %d, j: %d, arrBoxRoom[%d][%d]= %c₩n", i, j, i, j, arrBoxRoom[i][j]) ;
}
}

return 0;
}

// 상자 낙하 거리 계산
int setDropBox(int nLow, int nColumn, char arrBoxRoom[][100])
{
int nMaxDropBox, i, j,k;
int nDropCnt = 0,nDropCol;

for (i = 0 ; i < nLow; i++)
{
for(j = nColumn-1; j >= 0; j-- )
{
//printf("1- i=%d, j = %d, arrBoxRoom[i][j]=%c₩n", i, j, arrBoxRoom[i][j]);
if(arrBoxRoom[i][j] != '#') continue; // 상자가 있는지 확인

if(arrBoxRoom[i][j] == '#')
{
if(j == nColumn-1) continue; // 마지막 열에 상자 가 있음으로 움직이지 않음

for (k = j; k < nColumn; k++)
{
```

```
//printf("2- i=%d, j = %d, k = %d, arrBoxRoom[i][k]=%c, arrBoxRoom[i][k+1]=%c₩n", i, j, k,
arrBoxRoom[i][k], arrBoxRoom[i][k+1]);
if(arrBoxRoom[i][k+1] == '₩0')
{
if(k == nColumn-1) continue;
arrBoxRoom[i][k+1] = '#';
arrBoxRoom[i][k] = '₩0';
nDropCnt++;
printf("3- i=%d, j = %d, k = %d, arrBoxRoom[i][k]=%c, nDropCnt = %d₩n", i, j, k,
arrBoxRoom[i][k], nDropCnt);
}
else
break; // 다음에 문자가 있음으로 더이상 진행을 하지 않음
}
if(nMaxDropBox < nDropCnt)
{
nMaxDropBox = nDropCnt;
}
nDropCnt = 0;
}
}
}

return nMaxDropBox;
}
```