2021 여름학기 동국대학교 SW역량강화캠프

2일차. 완전탐색

▶ 저의 GitHub 계정 비밀번호는 8자리입니다.

▶ 만약 이 비밀번호를 알아내고자 한다면 어떠한 방법이 있을까요?

▶ 가장 무식하면서 확실한 방법

비밀번호로 가능한 모든 길이 8짜리 문자열들을 다 시도해본다.

▶ 암호학의 무차별 대입 공격 (Brute Force Attack)

● 완전탐색

▶ 가능한 모든 경우의 수를 고려하는 가장 기본적인 탐색 알고리즘

▶ 가장 무식하지만 가장 확실한 방법. 답을 찾을 확률 100%.

▶ 주로 반복문, DFS(깊이 우선 탐색), BFS(너비 우선 탐색)을 사용

● 콜라배달(2188)

문제

북극곰 윌리는 요즘 콜라공장에서 콜라를 배달하고 있다. 윌리는 지금 콜라가게에 콜라를 정확하게 N병을 배달해야 한다. 콜라는 반드시 상자에 가득 담겨서 옮겨져야 하며, 상자는 콜라를 3병 담을 수 있는 상자와 5병 담을 수 있는 상자의 두 종류가 있다.

윌리는 상자가 많으면 배달하기 불편하기 때문에, 최대한 적은 수의 상자를 들고 가려고 한다. 예를 들어, 18병의 콜라를 배달해야 할 때, 3병이 든 상자를 6개를 가져가도 되지만, 5병이 든 상자 3개와 3병이 든 상자 1개를 배달하면, 총 4개의 상자로 더 적은 개수의 상자를 배달할 수 있다.

윌리가 콜라를 정확하게 N병 배달해야 할 때, 상자를 최소 몇 개 가져가면 되는지 그 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N이 주어진다. ($3 \le N \le 5000$)

출력

윌리가 배달하는 상자의 최소 개수를 출력한다. 만약, 정확하게 N병을 만들 수 없다면 -1을 출력한다

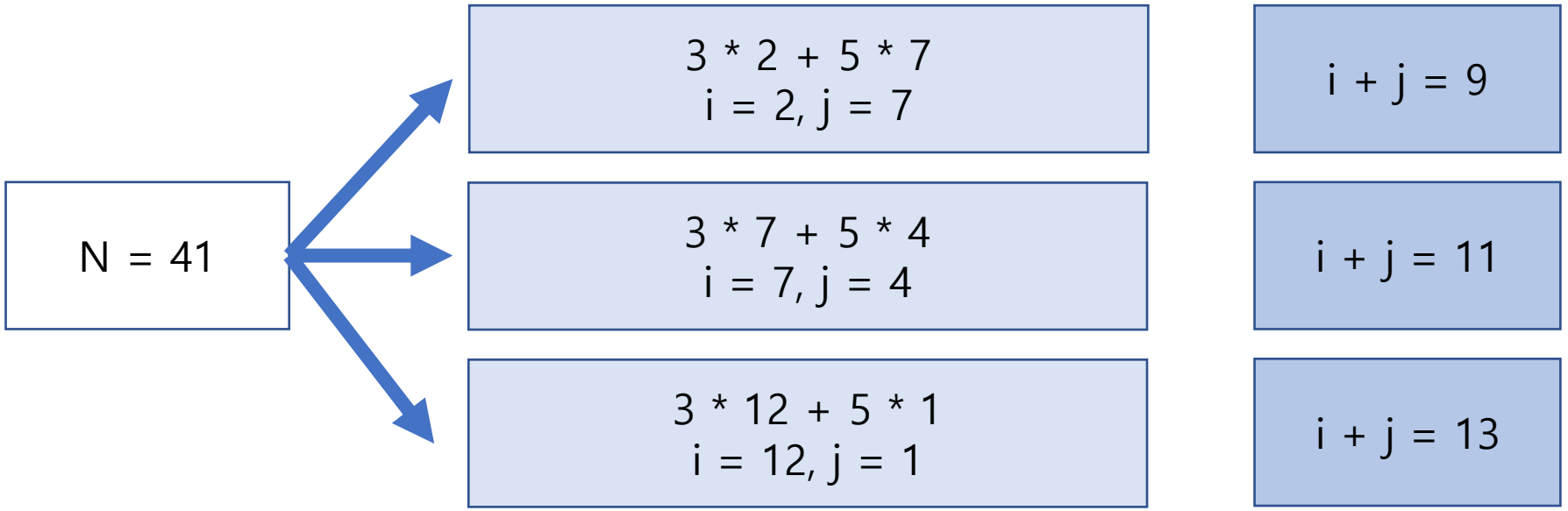
● 완전탐색을 통한 해결

▶ 3병 상자를 몇 개 , 5병 상자를 몇 개 이용할 때 정답이 나올까

▶ 3병 상자를 i 개, 5병 상자를 j개 이용했을 때, 총 콜라의 병 수가 N이 되는 i와 j를 반복문을 이용하여 찾는다.

▶ 총 상자 수(i+j)의 최소값을 출력하면 된다.

● 완전탐색을 통한 해결



핵심 코드

```
int Answer = -1;
for (int i = 0; i * 3 <= N; i++) { // 3병짜리 상자를 i개 고르고
    for (int j = 0; j * 5 <= N; j++) { // 5병짜리 상자를 j개 고른다.
        int Total = i * 3 + j * 5; // 총 콜라의 병 수 = Total
        if (Total == N) { // 만약 3병짜리 상자를 i개 고르고, 5병짜리 상자를 j개 고르는 것이 답의 후보가 된다면
            int Box = i + j; // 총 상자수는 i+j
            if (Answer == -1 || Answer > Box) { // 지금까지 구한 최소 상자수 Answer보다 Box가 작다면(또는 Answer이 아직 갱신되지 않았다면)
                Answer = Box;
            }
        }
    }
}
```

● 완전 탐색 문제 유형 – 조합/순열

▶ 주어진 N개 중 조건에 맞게 몇 개를 고르는 문제

▶ 고르는 방법의 순서가 상관 없으면 조합, 상관 있으면 순열

▶ 특정 값의 최대/최소를 묻거나, 고르는 방법의 경우의 수를 묻는 형태로 많이 출제

▶ 주로 DFS를 이용하여 풀이

▶ But, 고르는 개수가 정해져 있고, 그 수가 적다면 반복문으로도 해결 가능

●3장으로 하는 블랙잭(2123)

문제

카지노에서 제일 인기 있는 게임 블랙잭의 규칙은 상당히 쉽다. 카드의 합이 21을 넘지 않는 한도 내에서, 카드의 합을 최대한 크게 만드는 게임이다. 블랙잭은 카지노마다 다양한 규정이 있다.

남국 최고의 블랙잭 고수 윌리는 새로운 블랙잭 규칙을 만들어 존과 함께 게임하려고 한다.

윌리 버젼의 블랙잭에서 각 카드에는 양의 정수가 쓰여 있다. 그 다음, 딜러는 N장의 카드를 **모두 숫자가 보이도록 바닥에 놓는다. **그런 후에 딜러는 숫자 M을 크게 외친다.

이제 플레이어는 제한된 시간 안에 N장의 카드 중에서 3장의 카드를 골라야 한다. 블랙잭 변형 게임이기 때문에, 플레이어가 고른 카드의 합은 M을 넘지 않으면서 M과 최대한 가깝게 만들어야 한다.

N장의 카드에 써져 있는 숫자가 주어졌을 때, M을 넘지 않으면서 M에 최대한 가까운 카드 3장의 합을 구해 출력하시오.

입력

첫째 줄에 카드의 개수 N($3 \le N \le 100$)과 M($10 \le M \le 300,000$)이 주어진다. 둘째 줄에는 카드에 쓰여 있는 수가 주어지며, 이 값은 100,000을 넘지 않는다.

합이 M을 넘지 않는 카드 3장을 찾을 수 있는 경우만 입력으로 주어진다.

출력

첫째 줄에 M을 넘지 않으면서 M에 최대한 가까운 카드 3장의 합을 출력한다.

●반복문을 이용한 완전탐색 풀이

▶ 고르는 개수가 3개로 정해져 있고, 그 개수가 3개로 적기 때문에 반복문으로 풀이

▶ i번째, j번째, k번째 카드를 고르는 경우를 (i,j,k)로 가능한 모든 경우의 수를 탐색

▶ (3,2,1) 과 (1,2,3)은 같은 선택 방법

▶ (2,2,2) 와 같은 선택은 막아야 한다.

●반복문을 이용한 완전탐색 풀이

▶ 같은 선택방법을 여러 번 탐색하는 것을 막기 위해, i ≤ j ≤ k 를 가정한다.

▶ 같은 카드를 여러 번 뽑는 것을 막기 위해, i < j < k 로 진행한다.

(i,j,k)를 탐색하는 삼중 반복문을 돌릴 때 j를 i+1부터 탐색, k를 j+1부터 탐색

핵심 코드

```
int Answer = 0;
for (int i = 1; i <= N; i++) {
    for (int j = i + 1; j <= N; j++) { // j는 i+1부터 탐색
        for (int k = j + 1; k <= N; k++) { // k는 j+1부터 탐색
            int Total = arr[i] + arr[j] + arr[k]; // 총 카드의 합은 Total
            if (Total <= M && Total > Answer) { // Total이 M 이하이고, 지금까지 구한 최대값보다 크다면
                Answer = Total; // Answer를 Total로 갱신
            }
        }
    }
}
```

●시간 복잡도

▶ 모든 문제를 완전 탐색으로 해결할 수 있을까?

▶ 제한된 시간 내로 프로그램이 답을 반환해야 한다.

▶ 내 코드는 시간 내로 답을 반환할 수 있을까?

▶ 시간 복잡도 : 프로그램이 대략 몇 번의 연산을 수행하는지 표현하는 방법

● Big O 표기법

▶ n번 연산을 통해 답 반환 → O(n)

▶ n^2 번 연산을 통해 답 반환 → O(n^2)

▶ n*m + k번 연산을 통해 답 반환 → O(nm + k)

▶ 2n번 연산을 통해 답 반환 → O(2n) → O(n)

▶ 상수가 클 때에는 예외적으로 표기하기도 함. Ex) O(|S| * n) , O(n^2 / 64)

● 시간 복잡도 활용법

▶ 실행하는 연산 수 1억 당 1초로 환산

▶ N = 5000일 때,

O(N^3) 알고리즘은 5000^3 = 1250억, 약 1250초 소요 예상

O(N^2) 알고리즘은 5000^2 = 0.25억, 약 0.25초 소요 예상

▶ 실행하는 연산이 얼마나 복잡한지/간단한지에 따라 10배까지 차이 날 수 있다.

●삼각화단 만들기(2951)

문제

주어진 화단 둘레의 길이를 이용하여 삼각형 모양의 화단을 만들려고 한다. 이 때 만들어진 삼각형 화단 둘레의 길이는 반드시 주어진 화단 둘레의 길이와 같아야 한 다. 또한, 화단 둘레의 길이와 각 변의 길이는 자연수이다. 예를 들어, 만들고자 하는 화단 둘레의 길이가 9m라고 하면,

- 한 변의 길이가 1m, 두 변의 길이가 4m인 화단
- 한 변의 길이가 2m, 다른 변의 길이가 3m, 나머지 변의 길이가 4m인 화단 • 세 변의 길이가 모두 3m인 3가지 경우의 화단을 만들 수 있다.

화단 둘레의 길이를 입력받아서 만들 수 있는 서로 다른 화단의 수를 구하는 프로 그램을 작성하시오.

입력

화단의 길이 n이 주어진다.(단, 1 <= n <= 100)

출력

출력내용은 입력받은 n으로 만들 수 있는 서로 다른 화단의 수를 출력한다.

●반복문을 이용한 완전탐색 풀이 $O(N^3)$

▶ 삼각형의 세 변 i, j, k로 가능한 모든 경우의 수를 탐색

▶ 같은 삼각형을 여러 번 세는 것을 막기 위해 $i \leq j \leq k$ 를 만족하는 삼각형만 탐색

▶ 만족해야 하는 조건:

1. 세 변의 길이의 합 = N
2. 가장 긴 변의 길이 〈 나머지 두 변의 길이의 합

핵심 코드 $O(N^3)$

```
int count = 0;
for (int i = 1; i <= N; i++) {
    for (int j = i; j <= N; j++) {
        for (int k = j; k <= N; k++) {
            if (i + j + k == N && k < i + j)
                count++;
        }
    }
}
```

●반복문을 이용한 완전탐색 풀이 $O(N^2)$

▶ i + j + k 가 N이 되는 k를 반복문을 이용해서 찾지 않아도 된다.

▶ k = N − i − j 로 계산

▶ 만족해야 하는 조건:

1. $i \le j \le k$
2. 가장 긴 변의 길이 < 나머지 두 변의 길이의 합

핵심 코드 $O(N^2)$

```
int count = 0;
for (int i = 1; i <= N; i++) {
    for (int j = i; j <= N; j++) {
        int k = N - i - j;
        if (j <= k && k < i + j)
            count++;
    }
}
```

● 정렬

▶ 일정 기준에 따라서 원소들을 순서대로 나열하는 과정

▶ 버블 정렬, 선택 정렬, 삽입 정렬, 힙 정렬, 병합 정렬, 퀵 정렬, 계수 정렬, 기수 정렬

▶ Java에서는 Array.sort() 나 Collection.sort() 와 같은 정렬 알고리즘이 내장

▶ 버블 정렬, 삽입 정렬, 병합 정렬

● 버블 정렬

▶ 인접한 두 원소끼리 비교하여 정렬하는 알고리즘

▶ 한 번 선형 탐색 할 때마다, 마지막 원소를 고정시킬 수 있다.

▶ (N-1)번 선형 탐색하여 정렬

▶ 시간복잡도 $O(N^2)$

- 버블 정렬

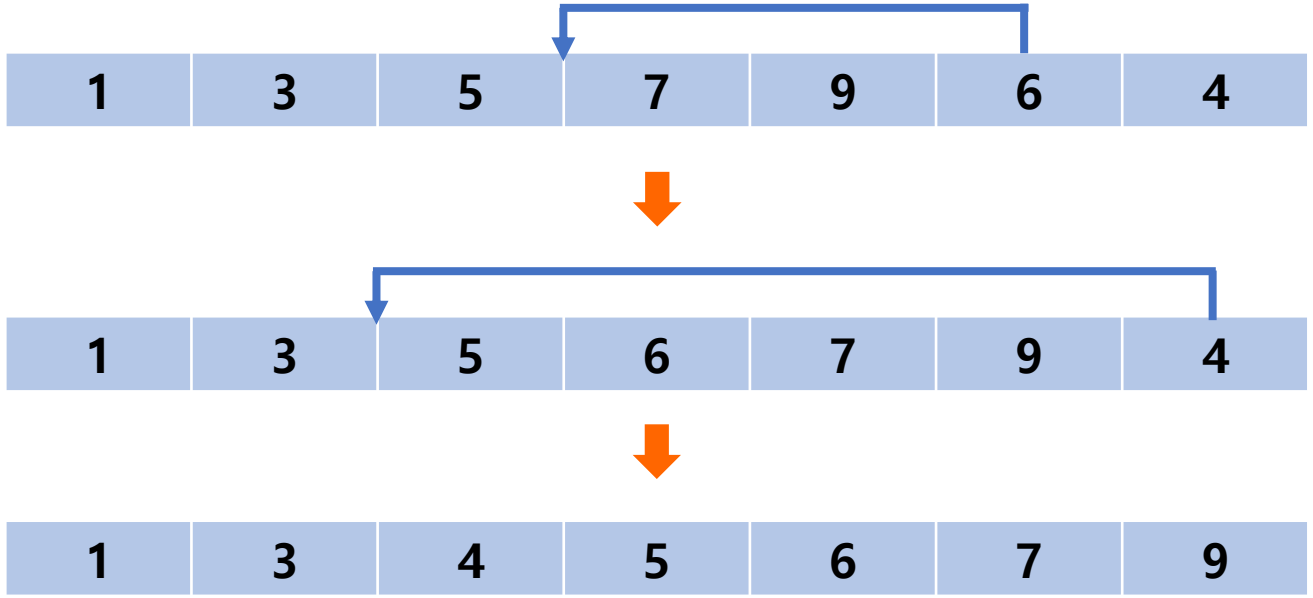
6 5 3 1 8 7 2 4

버블 정렬 코드 $O(N^2)$

```
for(int i=0; i < N-1; i++){
    for(int j=1; j<N-i; j++){
        if(arr[j] < arr[j-1]) {
            int tmp = arr[j];
            arr[j] = arr[j-1];
            arr[j-1] = tmp;
        }
    }
}
```

● 삽입 정렬

▶ 원소를 정렬된 배열에서 위치를 찾아 삽입하여 정렬하는 알고리즘



▶ 시간복잡도 $O(N^2)$

● 삽입 정렬

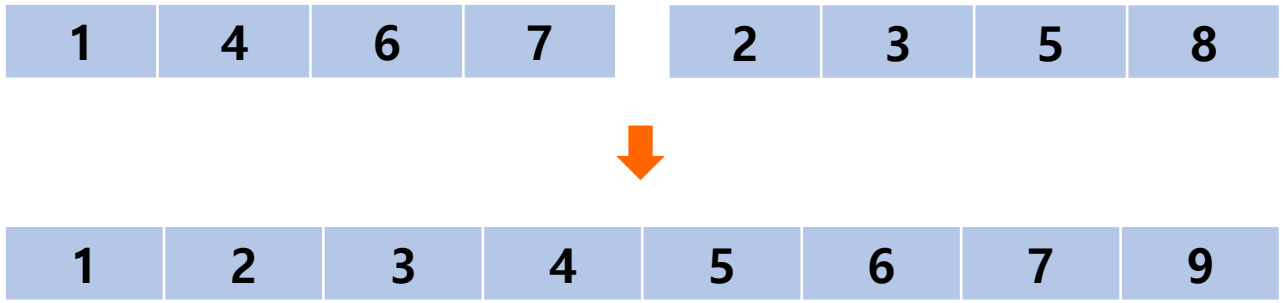
6 5 3 1 8 7 2 4

삽입 정렬 코드 $O(N^2)$

```
for(int i=1; i<N; i++) {
    int target = arr[i]; // arr[i]의 위치 정하기
    int j = i; // 초기 위치 j
    while(j>=1 && target < arr[j-1]) { // j의 한 칸 왼쪽이 target보다 크다면
        arr[j] = arr[j-1]; // arr[j-1]을 오른쪽으로 한 칸 옮기고
        j--; // target의 위치를 한 칸 왼쪽으로
    }
    arr[j] = target;
}
```

● 병합 정렬

▶ 배열을 2등분하여 각각 정렬하고, 정렬된 두 배열을 병합하는 것을 재귀적으로 진행하여 배열을 정렬하는 알고리즘



● 병합 정렬

6 5 3 1 8 7 2 4

● 병합 정렬

▶ 구간의 길이가 8이라면 병합 Layer = 3 (1 → 2 → 4 → 8)

▶ 구간의 길이가 16이라면 병합 Layer = 4 (1 → 2 → 4 → 8 → 16)

▶ 시간복잡도 = O (Layer 수 * N) = O(NlgN)

▶ 컴퓨터 과학에서 lgN = $\log_2 N$

병합 정렬 코드 O(NlgN)

```
public static void mergesort(int start, int end) {
    if (start == end)
        return;
    int mid = (start + end) / 2;
    mergesort(start, mid); // 왼쪽 절반 정렬
    mergesort(mid + 1, end); // 오른쪽 절반 정렬
    //임시 배열 tmp에 arr[start~end]을 정렬한 결과를 저장
    int L = start; // 왼쪽 분할의 첫 인덱스
    int R = mid + 1; // 오른쪽 분할의 첫 인덱스
    for (int i = start; i <= end; i++) {
        if (L > mid) // 왼쪽 분할에 남은 원소가 없는 경우
            tmp[i] = arr[R++];
        else if (R > end) // 오른쪽 분할에 남은 원소가 없는 경우
            tmp[i] = arr[L++];
        else if (arr[L] < arr[R]) // 왼쪽 분할의 원소가 더 작은 경우
            tmp[i] = arr[L++];
        else // 오른쪽 분할의 원소가 더 작은 경우
            tmp[i] = arr[R++];
    }
    for (int i = start; i <= end; i++) { // 임시로 저장해둔 tmp의 값들을 arr로 복사
        arr[i] = tmp[i];
    }
}
```