알고리즘과 코딩테스트 준비

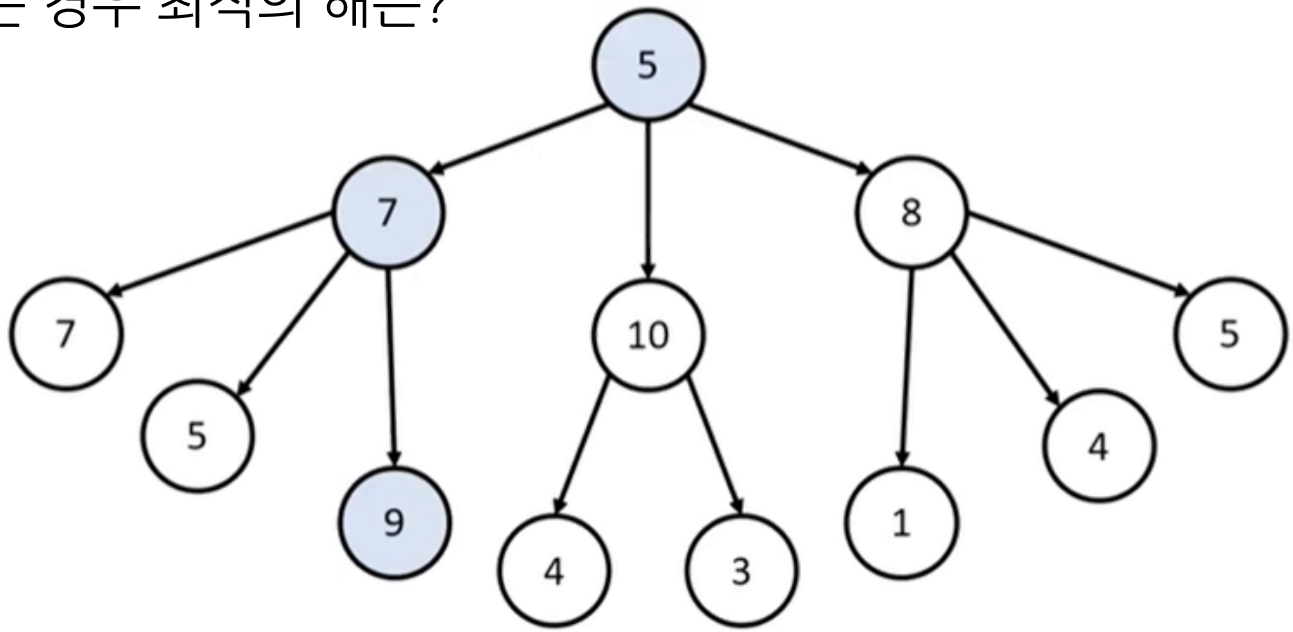
2022/1/13~1/19

윤형기 (hky@openwith.net)

CHAPTER 03 그리디

1 당장 좋은 것만 선택하는 그리디

- 개요
  - 현재 상태에서 지금 당장 좋은 것만 고르는 방법
  - 정당성 분석이 중요 (단, 코딩테스트에서는 …)
- 문제상황:
  - root node에서 시작해서 거쳐가는 node의 값을 최대로 만들고자 하는 경우 최적의 해는?



- 예제 3-1 거스름 돈

당신은 음식점의 계산을 도와주는 점원이다. 카운터에는 거스름돈으로 사용할 500원, 100원, 50원, 10원짜리 동전이 무한히 존재한다고 가정한다. 손님에게 거슬러 줘야 할 돈이 N원일 때 거슬러 줘야 할 동전의 최소 개수를 구하라. 단, 거슬러 줘야 할 돈 N은 항상 10의 배수이다.

- 그리디 알고리즘의 정당성
  - 동전의 큰 단위가 항상 작은 단위의 배수
- 거스름 돈의 시간 복잡도는 O(K)

- 3-1.py 답안 예시

```
n = 1260
count = 0

# 큰 단위의 화폐부터 차례대로 확인하기
coin_types = [500, 100, 50, 10]

for coin in coin_types:
    count += n // coin # 해당 화폐로 거슬러 줄 수 있는 동전의 개수 세기
    n %= coin

print(count)
```

- 3-1.cpp

```cpp
#include <bits/stdc++.h>

using namespace std;

int n = 1260;
int cnt = 0;
int coinTypes[4] = {500, 100, 50, 10};

int main() {
    for (int i = 0; i < 4; i++) {
        int coin = coinTypes[i];
        cnt += n / coin;
        n %= coin;
    }
    cout << cnt << '₩n';
}
```

2 [실전 문제] 큰 수의 법칙

실습과제

난이도 ●○○ | 풀이 시간 30분 | 시간 제한 1초 | 메모리 제한 128MB | 기출 2019 국가 교육기관 코딩 테스트

'큰 수의 법칙'은 일반적으로 통계 분야에서 다루어지는 내용이지만 동빈이는 본인만의 방식으로 다르게 사용하고 있다. 동빈이의 큰 수의 법칙은 다양한 수로 이루어진 배열이 있을 때 주어진 수들을 M번 더하여 가장 큰 수를 만드는 법칙이다. 단, 배열의 특정한 인덱스(번호)에 해당하는 수가 연속해서 K번을 초과하여 더해질 수 없는 것이 이 법칙의 특징이다.

예를 들어 순서대로 2, 4, 5, 4, 6으로 이루어진 배열이 있을 때 M이 8이고, K가 3이라고 가정하자. 이 경우 특정한 인덱스의 수가 연속해서 세 번까지만 더해질 수 있으므로 큰 수의 법칙에 따른 결과는 6 + 6 + 6 + 5 + 6 + 6 + 6 + 5인 46이 된다.

단, 서로 다른 인덱스에 해당하는 수가 같은 경우에도 서로 다른 것으로 간주한다. 예를 들어 순서대로 3, 4, 3, 4, 3으로 이루어진 배열이 있을 때 M이 7이고, K가 2라고 가정하자. 이 경우 두 번째 원소에 해당하는 4와 네 번째 원소에 해당하는 4를 번갈아 두 번씩 더하는 것이 가능하다. 결과적으로 4 + 4 + 4 + 4 + 4 + 4 + 4인 28이 도출된다.

배열의 크기 N, 숫자가 더해지는 횟수 M, 그리고 K가 주어질 때 동빈이의 큰 수의 법칙에 따른 결과를 출력하시오.

입력 조건

- 첫째 줄에 N(2 ≤ N ≤ 1,000), M(1 ≤ M ≤ 10,000), K(1 ≤ K ≤ 10,000)의 자연수가 주어지며, 각 자연수는 공백으로 구분한다.
- 둘째 줄에 N개의 자연수가 주어진다. 각 자연수는 공백으로 구분한다. 단, 각각의 자연수는 1 이상 10,000 이하의 수로 주어진다.
- 입력으로 주어지는 K는 항상 M보다 작거나 같다.

출력 조건

- 첫째 줄에 동빈이의 큰 수의 법칙에 따라 더해진 답을 출력한다.

입력 예시

```
5 8 3
2 4 5 4 6
```

출력 예시

```
46
```

- Python
- C++

Greedy 보충: 수강과목 스케줄링

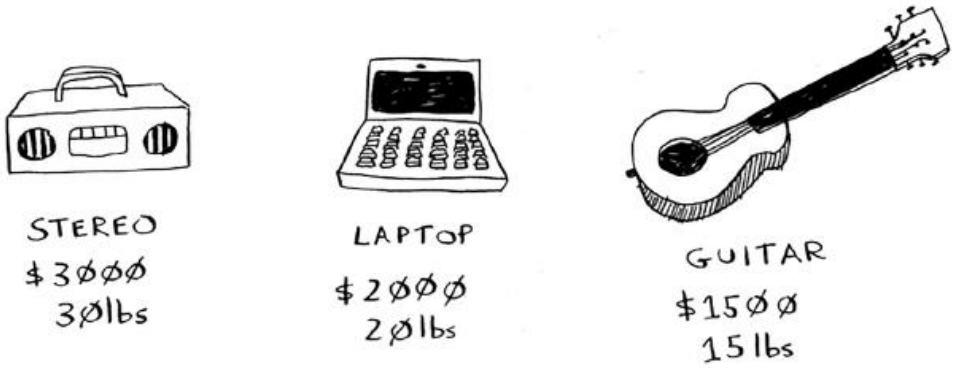
- 목적: 가급적 많은 과목을 수강하고자 함

과목	시작시간	종료시간
Art	09:00	10:00
English	09:30	10:30
Math	10:00	11:00
CS	10:30	11:30
Music	11:00	12:00

- 방법:
  - 가장 먼저 시작하는 과목 > 이후 과목 중 가장 빨리 끝나는 것
  - Art > Math > Music
- 즉, 각 단계에서 locally optimal solution을 선택

Knapsack 문제

- 목적: 배낭에 최대한 많은 value를 담고자 함



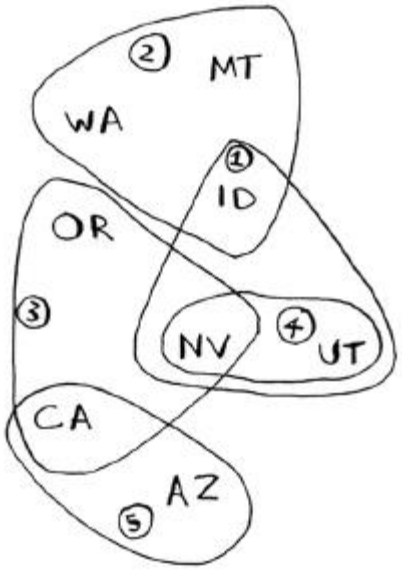
- 방법
  - 가장 비싼 것부터 차례로 …
  - But, 최적해가 되지 못함
- 즉, optimal solution은 아니지만 그에 근접함에 만족
- 연습문제: 가구회사에서 전국 각지에 가구 배송?

실습과제

Set-Covering 문제

- 목적: 라디오 프로그램이 전국 모든 곳에 전송되도록 함
  - 미국 50개 주라고 가정

Radio station	장소
K-one	ID, NV, UT
K-two	WA, ID, MT
K-three	OR, NV, CA
K-four	NV, UT
K-five	CA, AZ
…	…



- 방법
  - 모든 경우의 수 (조합)을 나열한 후 모든 곳을 cover하는 곳을 찾음
  - 그런데 문제: O(2^n)

- Greedy algorithm에 의한 approximation
  - 판단기준: How fast? How close to the optimal solution?
  - 힌트

```
# You pass an array in, and it gets converted to a set.
states_needed = set(["mt", "wa", "or", "id", "nv", "ut", "ca", "az"])

stations = {}
stations["kone"] = set(["id", "nv", "ut"])
stations["ktwo"] = set(["wa", "id", "mt"])
stations["kthree"] = set(["or", "nv", "ca"])
stations["kfour"] = set(["nv", "ut"])
stations["kfive"] = set(["ca", "az"])
```

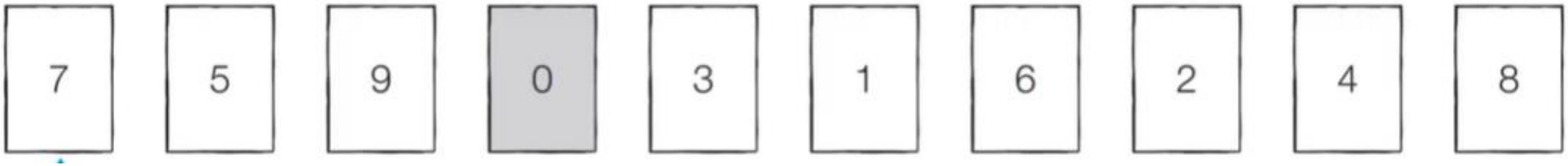
- Python
- C++

Chapter 06 정렬

- 1 기준에 따라 데이터를 정렬
- 2 [실전 문제] 위에서 아래로
- 3 [실전 문제] 성적이 낮은 순서로 학생 출력하기
- 4 [실전 문제] 두 배열의 원소 교체

1 기준에 따라 데이터를 정렬

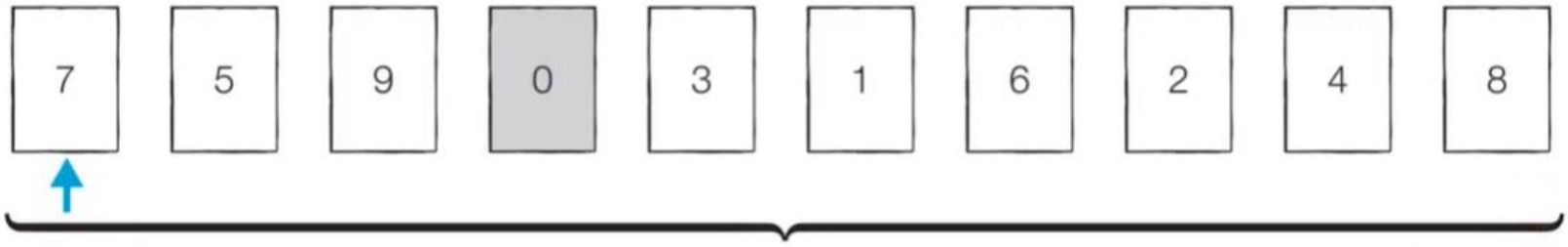
- 정렬 알고리즘 개요



https://www.youtube.com/watch?v=7zuGmKfUt7s&list=PLqM7alHXFySHrGIxeBOo4-mKO4H8j2knW&index=7

선택 정렬 (Selection Sort)

- 매번 가장 작은 것을 찾아서 바꾸면서 Sorted area와 Unsorted area를 구분해 나감



- 선택 정렬 시간 복잡도
  - N번만큼 가장 작은 수를 찾아서 맨 앞으로 보냄.
  - 전체 연산회수 = N + (N-1) + (N-2) + … + 2 ==> $O(N^2)$
- 단점: 다소 느리다

- 6-1.py 선택정렬 소스코드

```
array = [7, 5, 9, 0, 3, 1, 6, 2, 4, 8]

for i in range(len(array)):
    min_index = i # 가장 작은 원소의 인덱스
    for j in range(i + 1, len(array)):
        if array[min_index] > array[j]:
            min_index = j
    array[i], array[min_index] = array[min_index], array[i] # 스와프

print(array)
```

- 6-1.cpp

```
#include <bits/stdc++.h>

using namespace std;

int n = 10;
int arr[10] = {7, 5, 9, 0, 3, 1, 6, 2, 4, 8};

int main(void) {
    for (int i = 0; i < n; i++) {
        int min_index = i; // 가장 작은 원소의 인덱스
        for (int j = i + 1; j < n; j++) {
            if (arr[min_index] > arr[j]) {
                min_index = j;
            }
        }
        swap(arr[i], arr[min_index]); // 스와프
    }
    for(int i = 0; i < n; i++) {
        cout << arr[i] << ' ';
    }
}
```

- 6-2.py Python Swap 소스코드

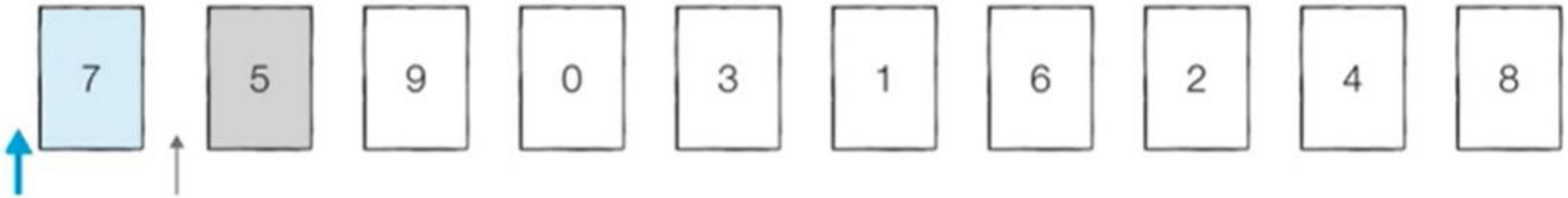
```python
# 0 인덱스와 1 인덱스의 원소 교체하기
array = [3, 5]
array[0], array[1] = array[1], array[0]

print(array)
```

- 6-2.cpp

```cpp
#include <bits/stdc++.h>

using namespace std;

int arr[2] = {3, 5};

int main(void) {
    swap(arr[0], arr[1]);
    cout << arr[0] << ' ' << arr[1] << '₩n';
}
```

삽입 정렬

- (첫 번째 데이터가 이미 정렬되어 있다고 판단하고) 두 번째 데이터부터 시작 - 처리되지 않은 데이터를 하나씩 골라 적절한 위치에 삽입



- 삽입정렬의 시간 복잡도
  - $O(N^2)$

- 6-3.py 삽입정렬 소스코드

```
array = [7, 5, 9, 0, 3, 1, 6, 2, 4, 8]

for i in range(1, len(array)):
    for j in range(i, 0, -1): # 인덱스 i부터 1까지 1씩 감소하며 반복하는 문법
        if array[j] < array[j - 1]: # 한 칸씩 왼쪽으로 이동
            array[j], array[j - 1] = array[j - 1], array[j]
        else: # 자기보다 작은 데이터를 만나면 그 위치에서 멈춤
            break

print(array)
```

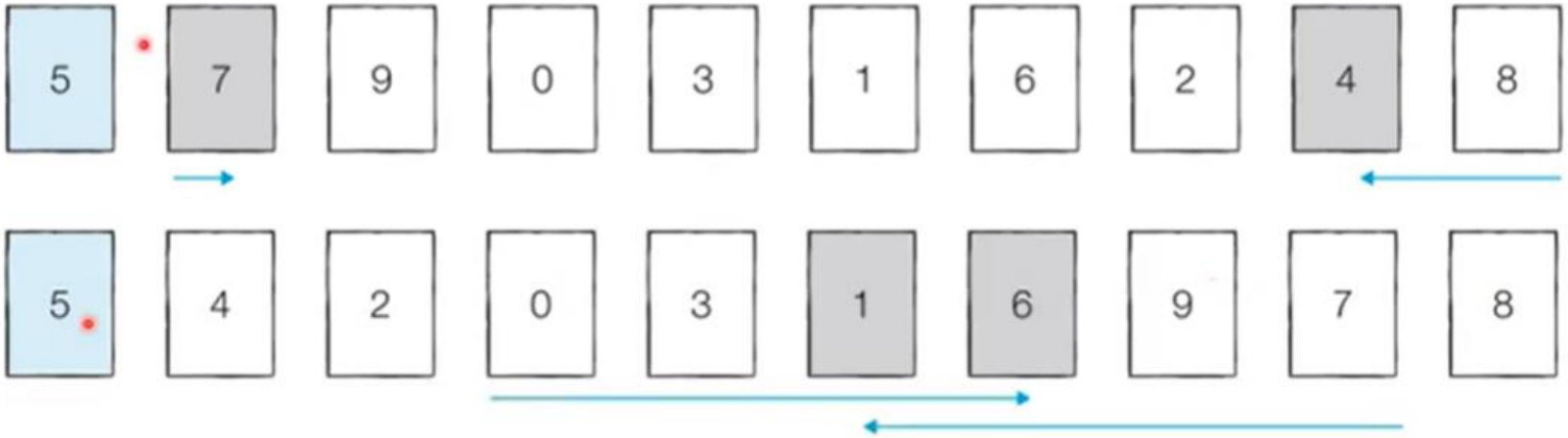
- 6-3.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

int n = 10;
int arr[10] = {7, 5, 9, 0, 3, 1, 6, 2, 4, 8};

int main(void) {
    for (int i = 1; i < n; i++) {
        // 인덱스 i부터 1까지 감소하며 반복하는 문법
        for (int j = i; j > 0; j--) {
            // 한 칸씩 왼쪽으로 이동
            if (arr[j] < arr[j - 1]) {
                swap(arr[j], arr[j - 1]);
            }
            // 자기보다 작은 데이터를 만나면 그 위치에서 멈춤
            else break;
        }
    }
    for(int i = 0; i < n; i++) {
        cout << arr[i] << ' ';
    }
}
```

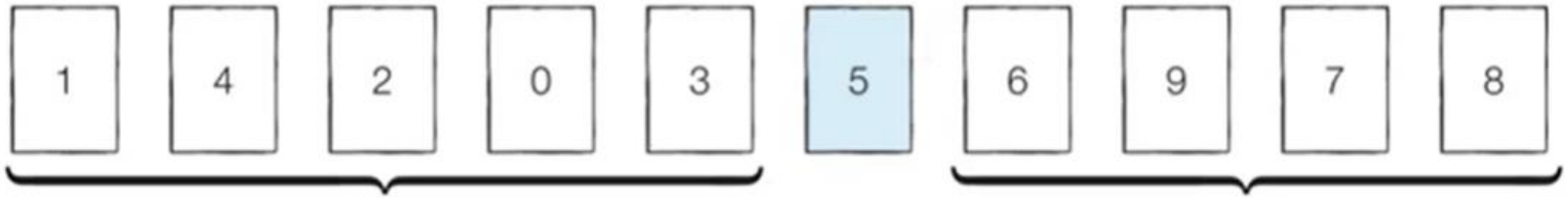
퀵 정렬 (Quick Sort)

- pivot element를 선택하여 array을 2개로 나눔 - pivot 기준 왼쪽 elements와 오른쪽 elements
  - 첫 번째 또는 마지막 데이터를 기준데이터 (pivot)로 설정
- 분할정복 방식 (recursive)

- Step 0: 현재의 pivot값은 '5'
  - 왼쪽부터 '5'보다 큰 데이터를, 오른쪽으로부터 '5'보다 작은 값을 선택



- 분할완료: Division을 통해 pivot 기준으로 데이터묶음을 나누는 것



- 퀵 정렬의 시간 복잡도
  - 평균: O(N log N)
  - 최악: $O(N^2)$

- 6-4.py 퀵 정렬 소스코드

```
array = [5, 7, 9, 0, 3, 1, 6, 2, 4, 8]

def quick_sort(array, start, end):
    if start >= end: # 원소가 1개인 경우 종료
        return
    pivot = start # 피벗은 첫 번째 원소
    left = start + 1
    right = end
    while(left <= right):
        # 피벗보다 큰 데이터를 찾을 때까지 반복
        while(left <= end and array[left] <= array[pivot]):
            left += 1
        # 피벗보다 작은 데이터를 찾을 때까지 반복
        while(right > start and array[right] >= array[pivot]):
            right -= 1
        if(left > right): # 엇갈렸다면 작은 데이터와 피벗을 교체
            array[right], array[pivot] = array[pivot], array[right]
        else: # 엇갈리지 않았다면 작은 데이터와 큰 데이터를 교체
            array[left], array[right] = array[right], array[left]
    # 분할 이후 왼쪽 부분과 오른쪽 부분에서 각각 정렬 수행
    quick_sort(array, start, right - 1)
    quick_sort(array, right + 1, end)

quick_sort(array, 0, len(array) - 1)
print(array)
```

- 6-5.py Python의 장점을 살린 퀵 정렬 소스코드

```
array = [5, 7, 9, 0, 3, 1, 6, 2, 4, 8]

def quick_sort(array):
    # 리스트가 하나 이하의 원소만을 담고 있다면 종료
    if len(array) <= 1:
        return array

    pivot = array[0] # 피벗은 첫 번째 원소
    tail = array[1:] # 피벗을 제외한 리스트

    left_side = [x for x in tail if x <= pivot] # 분할된 왼쪽 부분
    right_side = [x for x in tail if x > pivot] # 분할된 오른쪽 부분

    # 분할 이후 왼쪽 부분과 오른쪽 부분에서 각각 정렬을 수행하고, 전체 리스트를 반환
    return quick_sort(left_side) + [pivot] + quick_sort(right_side)

print(quick_sort(array))
```

- 6-4.cpp

```
#include <bits/stdc++.h>
using namespace std;

int n = 10;
int arr[10] = {7, 5, 9, 0, 3, 1, 6, 2, 4, 8};

void quickSort(int* arr, int start, int end) {
    if (start >= end) return; // 원소가 1개인 경우 종료
    int pivot = start; // 피벗은 첫 번째 원소
    int left = start + 1;
    int right = end;
    while (left <= right) {
        while (left <= end && arr[left] <= arr[pivot])
left++;
        // 피벗보다 작은 데이터를 찾을 때까지 반복
        while (right > start && arr[right] >= arr[pivot])
right--;
        // 엇갈렸다면 작은 데이터와 피벗을 교체
        if (left > right) swap(arr[pivot], arr[right]);
        // 엇갈리지 않았다면 작은 데이터와 큰 데이터를
교체
        else swap(arr[left], arr[right]);
    }
    // 분할 이후 왼쪽 부분과 오른쪽 부분에서
    // 각각 정렬 수행
    quickSort(arr, start, right - 1);
    quickSort(arr, right + 1, end);
}

int main(void) {
    quickSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++) {
        cout << arr[i] << ' ';
    }
}
```

계수 정렬 (Count Sort)

- 데이터 크기 범위가 제한되어 정수 형태로 표현할 수 있을 때 사용 가능
- 데이터 개수가 N, 데이터 (양수) 중 최대값이 K일 때 최악의 경우에도 O(N+K)를 보장

– 계수 정렬의 시간 복잡도

– 계수 정렬의 공간 복잡도

- **[Step 0]** 가장 작은 데이터부터 가장 큰 데이터까지의 범위가 모두 담길 수 있도록 리스트를 생성합니다.
- **정렬할 데이터**: 7 5 9 0 3 1 6 2 9 1 4 8 0 5 2

인덱스	0	1	2	3	4	5	6	7	8	9
개수(Count)	0	0	0	0	0	0	0	0	0	0

인덱스	0	1	2	3	4	5	6	7	8	9
개수(Count)	2	2	2	1	1	2	1	1	1	2

- 6-6.py 계수 정렬 소스코드

```
# 모든 원소의 값이 0보다 크거나 같다고 가정
array = [7, 5, 9, 0, 3, 1, 6, 2, 9, 1, 4, 8, 0, 5, 2]
# 모든 범위를 포함하는 리스트 선언 (모든 값은 0으로 초기화)
count = [0] * (max(array) + 1)

for i in range(len(array)):
    count[array[i]] += 1 # 각 데이터에 해당하는 인덱스의 값 증가

for i in range(len(count)): # 리스트에 기록된 정렬 정보 확인
    for j in range(count[i]):
        print(i, end=' ') # 띄어쓰기를 구분으로 등장한 횟수만큼 인덱스 출력
```

- 6-6.cpp

```
#include <bits/stdc++.h>
#define MAX_VALUE 9

using namespace std;

int n = 15;
// 모든 원소의 값이 0보다 크거나 같다고 가정
int arr[15] = {7, 5, 9, 0, 3, 1, 6, 2, 9, 1, 4, 8, 0, 5, 2};
// 모든 범위를 포함하는 배열 선언(모든 값은 0으로 초기화)
int cnt[MAX_VALUE + 1];

int main(void) {
    for (int i = 0; i < n; i++) {
        cnt[arr[i]] += 1; // 각 데이터에 해당하는 인덱스의 값 증가
    }
    for (int i = 0; i <= MAX_VALUE; i++) { // 배열에 기록된 정렬 정보 확인
        for (int j = 0; j < cnt[i]; j++) {
            cout << i << ' '; // 띄어쓰기를 기준으로 등장한 횟수만큼 인덱스 출력
        }
    }
}
```

- 비교정리

정렬 알고리즘	평균시간 복잡도	공간 복잡도	특징
선택정렬	$O(N^2)$	O(N)	아이디어가 간단
삽입정렬	$O(N^2)$	O(N)	데이터가 거의 정렬되어 있을 때 빠름
퀵 정렬	O(N log N)	O(N)	빠름
계수 정렬	O(N+K)	O(N+K)	빠름 (단, 데이터 크기가 한정될 것)

  - 대부분의 프로그래밍 언어에서 지원하는 sort library는 최악의 경우도 O (N log N)을 보장

- 파이썬의 정렬 라이브러리
  - 6-7.py sorted 소스코드

```
array = [7, 5, 9, 0, 3, 1, 6, 2, 4, 8]

result = sorted(array)
print(result)
```

  - 6-8.py sort 소스코드

```
array = [7, 5, 9, 0, 3, 1, 6, 2, 4, 8]

array.sort()
print(array)
```

  - 6-9.py 정렬 라이브러리에서 key를 활용한 소스코드

```
array = [('바나나', 2), ('사과', 5), ('당근', 3)]

def setting(data):
    return data[1]

result = sorted(array, key=setting)
print(result)
```

- 6-7.cpp

```
#include <bits/stdc++.h>

using namespace std;

int n = 10;
int arr[10] = {7, 5, 9, 0, 3, 1, 6, 2, 4, 8};

int main(void) {
    sort(arr, arr + n);
    for (int i = 0; i < n; i++) {
        cout << arr[i] << ' ';
    }
}
```

- 6-9.cpp

```
#include <bits/stdc++.h>

using namespace std;

class Fruit {
public:
    string name;
    int score;
    Fruit(string name, int score) {
        this->name = name;
        this->score = score;
    }
    // 정렬 기준은 '점수가 낮은 순서'
    bool operator <(Fruit &other) {
        return this->score < other.score;
    }
};
```

```
int main(void) {
    int n = 3;
    Fruit fruits[] = {
        Fruit("바나나", 2),
        Fruit("사과", 5),
        Fruit("당근", 3)
    };
    sort(fruits, fruits + n);
    for(int i = 0; i < n; i++) {
        cout << '(' << fruits[i].name << ','
<< fruits[i].score << ')' << ' ';
    }
}
```

실습과제

2 [실전 문제] 위에서 아래로

난이도 ●○○ | 풀이 시간 15분 | 시간 제한 1초 | 메모리 제한 128MB | 기출 T 기업 코딩 테스트

하나의 수열에는 다양한 수가 존재한다. 이러한 수는 크기에 상관없이 나열되어 있다. 이 수를 큰 수부터 작은 수의 순서로 정렬해야 한다. 수열을 내림차순으로 정렬하는 프로그램을 만드시오.

입력 조건

- 첫째 줄에 수열에 속해 있는 수의 개수 N이 주어진다. ($1 \leq N \leq 500$)
- 둘째 줄부터 N + 1번째 줄까지 N개의 수가 입력된다. 수의 범위는 1 이상 100,000 이하의 자연수이다.

출력 조건

- 입력으로 주어진 수열이 내림차순으로 정렬된 결과를 공백으로 구분하여 출력한다. 동일한 수의 순서는 자유롭게 출력해도 괜찮다.

입력 예시

```
3
15
27
12
```

출력 예시

```
27 15 12
```

- Python
- C++

실습과제

3 [실전 문제] 성적 낮은 순서로 학생 출력하기

난이도 ●○○ | 풀이 시간 20분 | 시간 제한 1초 | 메모리 제한 128MB | 기출 D 기업 프로그래밍 콘테스트 예선

N명의 학생 정보가 있다. 학생 정보는 학생의 이름과 학생의 성적으로 구분된다. 각 학생의 이름과 성적 정보가 주어졌을 때 성적이 낮은 순서대로 학생의 이름을 출력하는 프로그램을 작성하시오.

입력 조건

- 첫 번째 줄에 학생의 수 N이 입력된다. ($1 \leq N \leq 100,000$)
- 두 번째 줄부터 N + 1번째 줄에는 학생의 이름을 나타내는 문자열 A와 학생의 성적을 나타내는 정수 B가 공백으로 구분되어 입력된다. 문자열 A의 길이와 학생의 성적은 100 이하의 자연수이다.

출력 조건

- 모든 학생의 이름을 성적이 낮은 순서대로 출력한다. 성적이 동일한 학생들의 순서는 자유롭게 출력해도 괜찮다.

입력 예시

```
2
홍길동 95
이순신 77
```

출력 예시

```
이순신 홍길동
```

- Python
- C++

과제 실습

실습과제

난이도 ●○○ | 풀이 시간 20분 | 시간 제한 1초 | 메모리 제한 256MB | 기출 핵심 유형
링크 https://www.acmicpc.net/problem/10825

도현이네 반 학생 N명의 이름과 국어, 영어, 수학 점수가 주어집니다. 이때, 다음과 같은 조건으로 학생의 성적을 정렬하는 프로그램을 작성하세요.

1. 국어 점수가 감소하는 순서로
2. 국어 점수가 같으면 영어 점수가 증가하는 순서로
3. 국어 점수와 영어 점수가 같으면 수학 점수가 감소하는 순서로
4. 모든 점수가 같으면 이름이 사전 순으로 증가하는 순서로 (단, 아스키코드에서 대문자는 소문자보다 작으므로 사전 순으로 앞에 옵니다.)

입력 조건
- 첫째 줄에 도현이네 반의 학생 수 N(1 ≤ N ≤ 100,000)이 주어집니다.
- 둘째 줄부터 한 줄에 하나씩 각 학생의 이름, 국어, 영어, 수학 점수가 공백으로 구분해 주어집니다.
- 점수는 1보다 크거나 같고, 100보다 작거나 같은 자연수입니다.
- 이름은 알파벳 대소문자로 이루어진 문자열이고, 길이는 10자리를 넘지 않습니다.

출력 조건
- 문제에 나와 있는 정렬 기준으로 정렬한 후 첫째 줄부터 N개의 줄에 걸쳐 각 학생의 이름을 출력합니다.

입력 예시

```
12
Junkyu 50 60 100
Sangkeun 80 60 50
Sunyoung 80 70 100
Soong 50 60 90
Haebin 50 60 100
Kangsoo 60 80 100
Donghyuk 80 60 100
Sei 70 70 70
Wonseob 70 70 90
Sanghyun 70 70 80
nsj 80 80 80
Taewhan 50 60 90
```

출력 예시

```
Donghyuk
Sangkeun
Sunyoung
nsj
Wonseob
Sanghyun
Sei
Kangsoo
Haebin
Junkyu
Soong
Taewhan
```

구현 (보충)

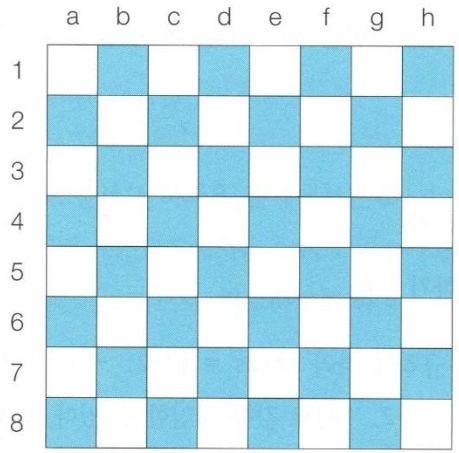
2 [실전 문제] 왕실의 나이트

난이도 ●○○ | 풀이 시간 20분 | 시간 제한 1초 | 메모리 제한 128MB

행복 왕국의 왕실 정원은 체스판과 같은 8 × 8 좌표 평면이다. 왕실 정원의 특정한 한 칸에 나이트가 서 있다. 나이트는 매우 충성스러운 신하로서 매일 무술을 연마한다.

나이트는 말을 타고 있기 때문에 이동을 할 때는 L자 형태로만 이동할 수 있으며 정원 밖으로는 나갈 수 없다. 나이트는 특정한 위치에서 다음과 같은 2가지 경우로 이동할 수 있다.

1. 수평으로 두 칸 이동한 뒤에 수직으로 한 칸 이동하기
2. 수직으로 두 칸 이동한 뒤에 수평으로 한 칸 이동하기



이처럼 8 × 8 좌표 평면상에서 나이트의 위치가 주어졌을 때 나이트가 이동할 수 있는 경우의 수를 출력하는 프로그램을 작성하시오. 이때 왕실의 정원에서 행 위치를 표현할 때는 1부터 8로 표현하며, 열 위치를 표현할 때는 a부터 h로 표현한다.

예를 들어 만약 나이트가 a1에 있을 때 이동할 수 있는 경우의 수는 다음 2가지이다. a1의 위치는 좌표 평면에서 구석의 위치에 해당하며 나이트는 정원의 밖으로는 나갈 수 없기 때문이다.

1. 오른쪽으로 두 칸 이동 후 아래로 한 칸 이동하기(c2)
2. 아래로 두 칸 이동 후 오른쪽으로 한 칸 이동하기(b3)

또 다른 예로 나이트가 c2에 위치해 있다면 나이트가 이동할 수 있는 경우의 수는 6가지이다. 이건 직접 계산해보시오.

입력 조건
- 첫째 줄에 8 × 8 좌표 평면상에서 현재 나이트가 위치한 곳의 좌표를 나타내는 두 문자로 구성된 문자열이 입력된다. 입력 문자는 a1처럼 열과 행으로 이뤄진다.

출력 조건
- 첫째 줄에 나이트가 이동할 수 있는 경우의 수를 출력하시오.

입력 예시

```
a1
```

출력 예시

```
2
```