1) 검색 알고리즘

선형검색

배열의 인덱스를 처음부터 끝까지 하나씩 증가시키면서 찾는 방법

```
For i from 0 to n–1
    If i'th element is 50
        Return true
Return false
```

이진검색

정렬된 배열에서 중간 인덱스부터 찾아서 그 값보다 찾을 값이 작으면 왼쪽, 크면 오른쪽을 검색하는 법

```
If no items
    Return false
If middle item is 50
    Return true
Else if 50 < middle item
    Search left half
Else if 50 > middle item
    Search right half
```

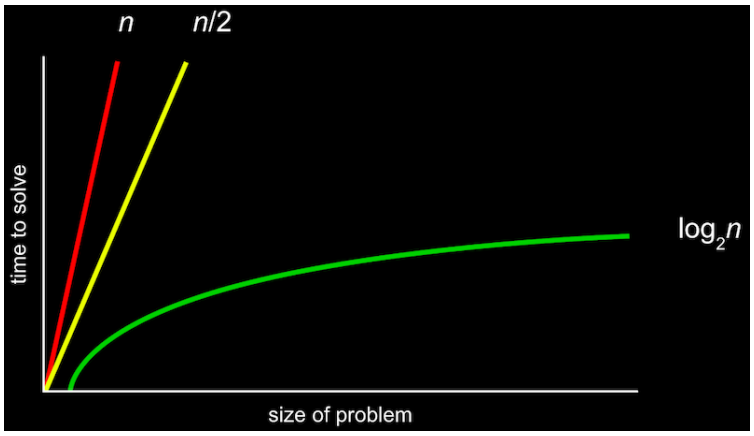
2) 알고리즘 표기법

알고리즘 실행 시간 표기법

처리하는 데이터가 많아질수록 알고리즘의 실행 시간이 중요하다.

실행 시간을 표기하는 방법 : 실행 시간의 상한 - Big O 실행 시간의 하한 - Big Ω

실행 시간의 상한 (Big O)



O(n^2)
O(n log n)
O(n) - 선형 검색
O(log n) - 이진 검색
O(1)

실행 시간의 하한 (Big Ω)

Ω(n^2)
Ω(n log n)
Ω(n) - 배열 안에 존재하는 값의 개수 세기
Ω(log n)
Ω(1) - 선형 검색, 이진 검색 (운이 좋은 경우 한 방)

실행 시간의 상한이 낮은 알고리즘이 하한이 낮은 알고리즘 보다 더 좋다고 생각한다.

3) 선형 검색

선형검색의 특징

선형 검색 알고리즘은 정확하나 비효율적인 검색 방법

자료가 정렬되지 않은 경우 유용

선형검색을 하는 방법

선형 검색을 이용해서 EMMA 의 번호 찾기

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

int main(void)
{
    string names[] = {"EMMA", "RODRIGO", "BRIAN", "DAVID"};
    string numbers[] = {"617–555–0100", "617–555–0101", "617–555–0102", "617–555–0103"};

    for (int i = 0; i < 4; i++)
    {
        if (strcmp(names[i], "EMMA") == 0)
        {
            printf("Found %s₩n", numbers[i]);
            return 0;
        }
    }
    printf("Not found₩n");
    return 1;
}
```

하지만 이 방법은 names 와 numbers 가 같은 인덱스를 가져야 한다는 한계가 있다.

구조체

구조체를 이용하면 위와 같은 한계를 극복 할 수 있다.

```
#include <cs50.h>
```

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

typedef struct
{
    string name;
    string number;
}
person;

int main(void)
{
    person people[4];

    people[0].name = "EMMA";
    people[0].number = "617–555–0100";
    people[1].name = "RODRIGO";
    people[1].number = "617–555–0101";
    people[2].name = "BRIAN";
    people[2].number = "617–555–0102";
    people[3].name = "DAVID";
    people[3].number = "617–555–0103";

    // EMMA 검색
    for (int i = 0; i < 4; i++)
    {
        if (strcmp(people[i].name, "EMMA") == 0)
        {
            printf("Found %s₩n", people[i].number);
            return 0;
        }
    }
    printf("Not found₩n");
    return 1;
}
```

구조체는 int float char 와 비슷한 일을 한다.

예를 들어, int i[3]; i[0]=1; i[1]=2; i[2]=3; 처럼, person people[4]; people[0]="EMMA"; people[1]="RODRIGO";...

4) 버블 정렬

버블정렬 이란?

두개의 인접한 자료 값을 비교하여 위치를 교환하여 정렬하는 방식

ex) 7 3 2 6 -> 3 7 2 6 -> 3 2 7 6 -> 3 2 6 7 -> 2 3 6 7

```
Repeat n–1 times

    For i from 0 to n–2

        If i'th and i+1'th elements out of order

            Swap them
```

n 개의 값이 주어졌을 때, 각 루프는 각각 n-1 번, n-2 번 반복된다.

그래서 (n-1)*(n-2) = n^2-3n+2(n−1)*(n−2)=n^2 −3n+2 번의 비교 및 교환이 필요

정렬이 빨리 되었더라도 루프를 돌면서 비교해야하므로 정렬
시간의 상한은 O(n^2) 하한은 Ω(n^2) 이다.

배열의 크기가 작을 경우 버블 정렬을 써도 좋다.

5) 선택 정렬

선택 정렬 이란?

배열 안에서 가장 작은(큰)수를 골라 첫 번째 위치(마지막 위치)와 교환하는 정렬 방법

선택 정렬은 교환 횟수는 최소화하지만 비교 횟수는 증가한다.

63852417 -> 13852467 -> 12853467 -> 12358467 -> 12348567 -> 12345867 ->12345687 -> 12345678

```
For i from 0 to n–1

    Find smallest item between i'th item and last item

    Swap smallest item with i'th item
```

두 번의 루프를 돈다 (바깥 : 순서대로 방문, 안쪽 : 가장 작은 값을 찾는다.)

소요시간의 상한은 O(n^2) 하한은 Ω(n^2)로 버블 정렬과 동일하다.

6) 정렬 알고리즘의 실행 시간

지금까지 배웠던 정렬 알고리즘의 실행 시간

실행시간의 상한	실행시간의 하한
O(n^2): 선택 정렬, 버블 정렬	Ω(n^2): 선택 정렬, 버블 정렬
O(n log n)	Ω(n log n)
O(n): 선형 검색	Ω(n)
O(log n): 이진 검색	Ω(log n)
O(1)	Ω(1): 선형 검색, 이진 검색

버블 정렬의 실행 시간을 단축시키는 방법

```
Repeat until no swaps //교환이 없을 때까지 반복

    For i from 0 to n–2

        If i'th and i+1'th elements out of order
```

```
                Swap them
```

정렬이 완료되어 더이상 교환이 일어나지 않으면 반복을 멈추게 만든다.

그러면,

버블 정렬의 실행시간의 하한이 내려간다.

Ω(n^2): 선택 정렬
Ω(n log n)
Ω(n): 버블 정렬
Ω(log n)
Ω(1): 선형 검색, 이진 검색

7) 재귀

재귀란?

함수를 함수 내부에서 재사용 하여 동일 작업을 피하는 방법

중첩루프로 피라미드 출력

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

void draw(int h);

int main(void)
{
    // 사용자로부터 피라미드의 높이를 입력 받아 저장
    int height = get_int("Height: ");

    // 피라미드 그리기
    draw(height);
}

void draw(int h)
{
    // 높이가 h 인 피라미드 그리기
```

```
    for (int i = 1; i <= h; i++)
    {
        for (int j = 1; j <= i; j++)
        {
            printf("#");
        }
        printf("₩n");
    }
}
```

#

##

###

####

라는 결과가 나옴

여기서 바깥 쪽 루프는 안 쪽 루프에서 수행하는 내용을 반복하도록 하는 것일 뿐이다.

재귀로 피라미드 출력

바깥 쪽 루프를 없앤 draw 함수를 만들고, 이를 '재귀적으로' 호출해 피라미드를 만들 수 있다.

즉, draw 함수 안에서 draw 함수를 호출 하는 것

```
#include <cs50.h>
#include <stdio.h>

void draw(int h);

int main(void)
{
    int height = get_int("Height: ");
```

```
    draw(height);
}

void draw(int h)
{
    // 높이가 0 이라면 (그릴 필요가 없다면)
    if (h == 0)
    {
        return;
    }

    // 높이가 h-1 인 피라미드 그리기
    draw(h - 1);

    // 피라미드에서 폭이 h 인 한 층 그리기
    for (int i = 0; i < h; i++)
    {
        printf("#");
    }
    printf("₩n");
}
```

h 라는 높이를 받는다 -> h-1 높이로 draw 함수 호출 -> 그 후 h 만큼의 #을 출력

여기서 내부적으로 호출된 draw 함수를 따라가다 보면 **h = 0** 인 상황이 오는데,

그 때 아무것도 출력 하지 않도록 하는 조건문을 추가한다.

8) 병합 정렬

병합 정렬 이란?

원소가 한 개가 될 때까지 계속해서 반으로 나누다가 다시 합쳐나가며 정렬을 하는 방식

7 | 4 | 5 | 2 | 6 | 3 | 8 | 1

4 7 | 2 5 | 3 6 | 1 8

2 4 5 7 | 1 3 6 8

1 2 3 4 5 6 7 8

병합 정렬 실행 시간

병합 정렬 실행 시간의 상한 O(n log n)

병합 정렬 실행 시간의 하한 Ω(n log n)

실행시간의 상한	**실행시간의 하한**
O(n^2): 선택 정렬, 버블 정렬	Ω(n^2): 선택 정렬
O(n log n) 병합 정렬	Ω(n log n) 병합 정렬
O(n): 선형 검색	Ω(n): 버블 정렬
O(log n): 이진 검색	Ω(log n)
O(1)	Ω(1): 선형 검색, 이진 검색

"알고리즘을 만들때 우리의 목표는 정확하게 만드는 것 뿐만 아니라 잘 설계하는 것"- David Malan

https://www.edwith.org/boostcourse-cs-050/joinLectures/41488