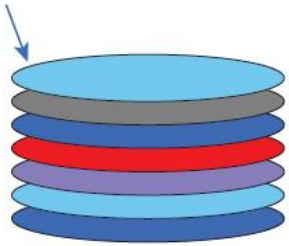
# 1장 자료구조와 알고리즘

# 자료구조란 ?




**프로그램에서 자료를 정리하여 보관하는 구조**

# 일상생활과 자료구조의 비교

〈표 1-1〉 일상생활과 자료구조의 유사성

| 일상생활에서의 예 | 해당하는 자료구조 |
|---|---|
| 그릇을 쌓아서 보관하는 것 | 스택 |
| 마트 계산대의 줄 | 큐 |
| 버킷 리스트 | 리스트 |
| 영어사전 | 사전 |
| 지도 | 그래프 |
| 컴퓨터의 디렉토리 구조 | 트리 |

# 자료구조와 알고리즘

- 프로그램 = 자료구조 + 알고리즘

자료구조

scores[]

| 80 | 70 | 90 | ··· | 30 |
|---|---|---|---|---|

알고리즘

```
largest←scores[0]
for i←1 to N-1 do
  if scores[i]>largest
     then largest←scores[i]
return largest
```

- 시험성적을 읽어 들여 최고성적 알아보기?

```
#define MAX_ELEMENTS 100
int scores[MAX_ELEMENTS]; // 자료구조
int get_max_score(int n)     // 학생의 숫자는 n
{
        int i, largest;
        largest = scores[0]; // 알고리즘
        for (i = 1; i<n; i++) {
                if (scores[i] > largest) {
                        largest = scores[i];
                }
        }
        return largest;
}
```

```c
int main(void)
{
    int i, score;
    for (i = 0; i < MAX_ELEMENTS; i++) {
        printf("%d 번째 점수를 입력하세요 >>", (i + 1));
        scanf_s("%d", &score);
        scores[i] = score;
    }

    for (i = 0; i < MAX_ELEMENTS; i++) {
        printf("%d 번째의 점수는 %d₩n", (i+1), scores[i]);
    }
    printf("최고점수는 %d 점 입니다. ₩n", get_max_score());

    return 0;
}
```

```
int get_max_score() {
    int i, largest;
    largest = scores[0];
    for (i = 1; i < MAX_ELEMENTS; i++)
    {
        if (scores[i] > largest) {
            largest = scores[i];
        }
    }
    return largest;
}
```

# 5명의 성적을 입력받아 평균 구하기

```
#include <stdio.h>
#define STUDENTS 5

int main(void)
{
	int scores[STUDENTS];
	int sum = 0;
	int i, average;

	for (i = 0; i < STUDENTS; i++)
	{
		printf("학생들의 성적을 입력하시오: ");
		scanf("%d", &scores[i]);
	}
	for (i = 0; i < STUDENTS; i++)
		sum += scores[i];

	average = sum / STUDENTS;
	printf("성적 평균= %d\n", average);

	return 0;
}
```

# 배열에 난수로 채우기

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

int main(void)
{
	int i;
	int scores[SIZE];

	for(i = 0; i < SIZE; i++)
		scores[i] = rand() % 100;

	for(i = 0; i < SIZE; i++)
		printf("scores[%d]=%d\n", i, scores[i]);

	return 0;
}
```

# 문제 1

- 100 개의 정수를 저장할 수 있는 배열을 선언하고 정수를 차례로 입력 받다가 0 이 입력되면 0 을 제외하고 그 때까지 입력된 정수를 가장 나중에 입력된 정수부터 차례대로 출력하는 프로그램을 작성하시오.

## 입력 예

```
3 5 10 55 0
```

## 출력 예

```
55 10 5 3
```

# 문제2

- 10개의 정수를 입력받아 배열에 저장한 후 짝수 번째 입력된 값의 합과 홀수 번째 입력된 값의 평균을 출력하는 프로그램을 작성하시오.

  평균은 반올림하여 소수첫째자리까지 출력한다..

## 입력 예

```
95 100 88 65 76 89 58 93 77 99
```

## 출력 예

```
sum : 446
avg : 78.8
```

# 문제 3

- 10개의 문자를 입력받아 마지막으로 입력받은 문자부터 첫 번째 입력받은 문자까지 차례로 출력하는 프로그램을 작성하시오.

## 입력 예

```
A E C X Y Z c b z e
```

## 출력 예

```
e z b c Z Y X C E A
```

# 문제 4

- 6개의 문자배열을 만들고 {'J', 'U', 'N', 'G', 'O', 'L'} 으로 초기화 한 후 문자 한 개를 입력받아 배열에서의 위치를 출력하는 프로그램을 작성하시오.
- 첫 번째 위치는 0번이며 배열에 없는 문자가 입력되면 "none" 라는 메시지를 출력하고 끝내는 프로그램을 작성하시오.

입력 예

```
L
```

출력 예

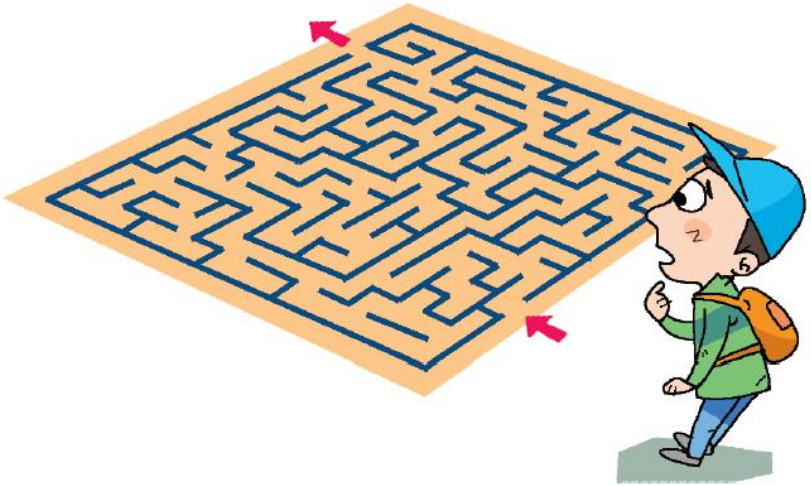
```
5
```

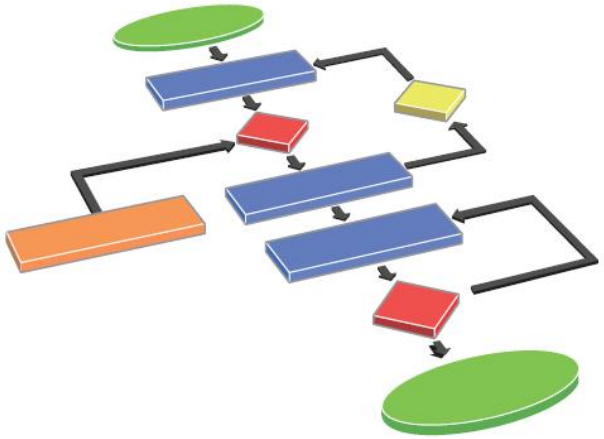

- **알고리즘(algorithm):** 컴퓨터로 문제를 풀기 위한 단계적인 절차

- 알고리즘의 조건
  - 입력 : 0개 이상의 입력이 존재하여야 한다.
  - 출력 : 1개 이상의 출력이 존재하여야 한다.
  - 명백성 : 각 명령어의 의미는 모호하지 않고 명확해야 한다.
  - 유한성 : 한정된 수의 단계 후에는 반드시 종료되어야 한다.
  - 유효성 : 각 명령어들은 실행 가능한 연산이여야 한다.

# 알고리즘의 기술 방법

- 영어나 한국어와 같은 자연어
- 흐름도(flow chart)
- 의사 코드(pseudo-code)
- 프로그래밍 언어





# 자연어로 표기된 알고리즘

- 인간이 읽기가 쉽다.
- 그러나 자연어의 단어들을 정확하게 정의하지 않으면 의미 전달이 모호해질 우려가 있다.

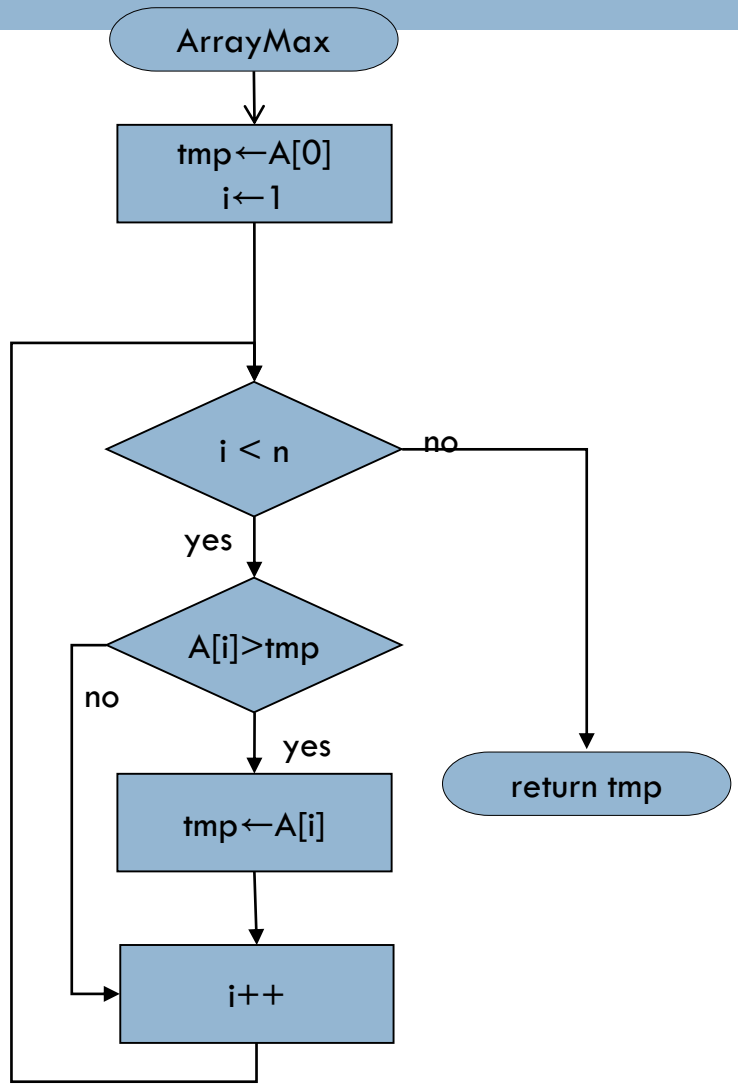
(예) 배열에서 최대값 찾기 알고리즘

**ArrayMax(list, n)**

1. 배열 list의 첫번째 요소를 변수 **tmp**에 복사
2. 배열 **list**의 다음 요소들을 차례대로 **tmp**와 비교하면 더 크면 **tmp**로 복사
3. 배열 **list**의 모든 요소를 비교했으면 **tmp**를 반환

# 흐름도로 표기된 알고리즘

- 직관적이고 이해하기 쉬운 알고리즘 기술 방법
- 그러나 복잡한 알고리즘의 경우, 상당히 복잡해짐.



# 유사코드로 표현된 알고리즘

- 알고리즘 기술에 가장 많이 사용
- 프로그램을 구현할 때의 여러 가지 문제들을 감출 수 있다.

  즉 알고리즘의 핵심적인 내용에만 집중할 수 있다.

```
ArrayMax(list, N):
    largest←list[0]
    for i←1 to N-1 do
        if list[i]>largest
            then largest←list[i]
    return largest
```

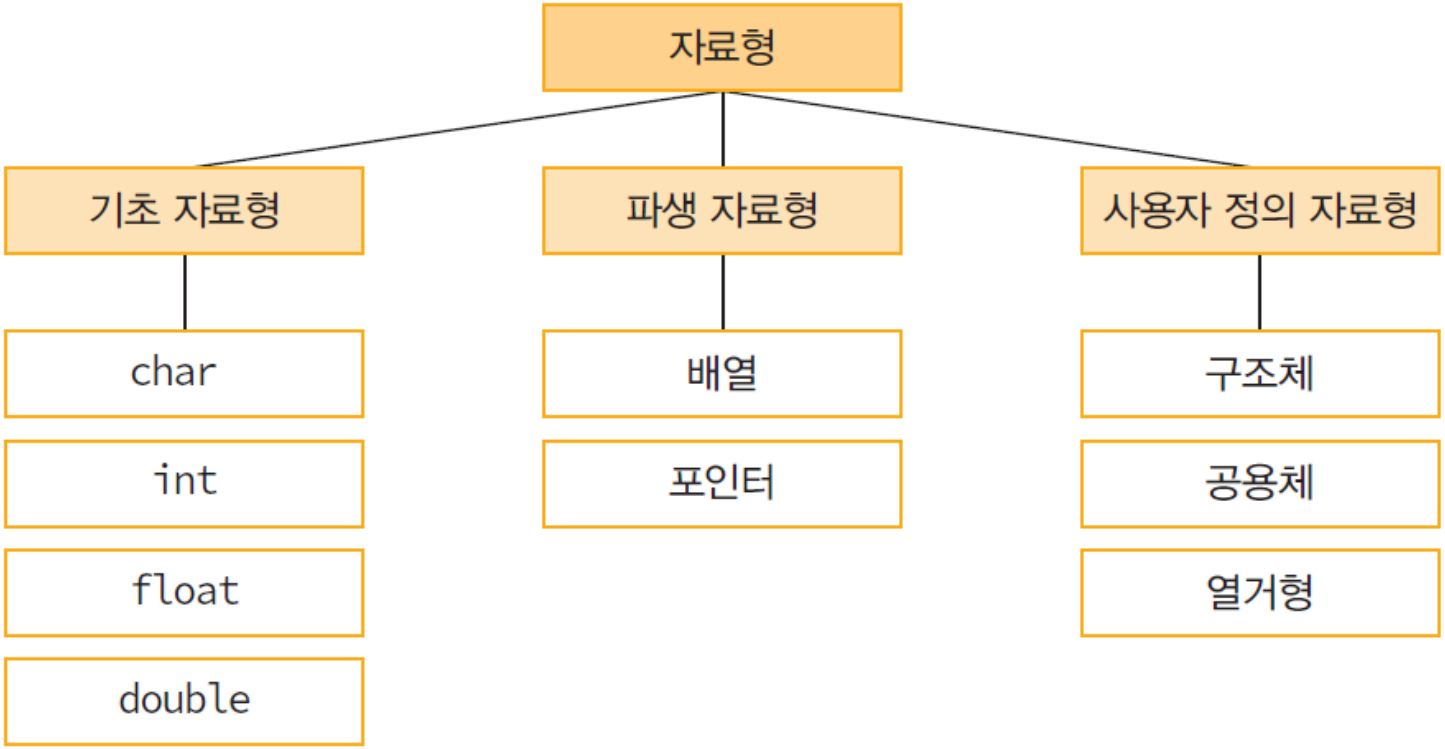
# C로 표현된 알고리즘

- 알고리즘의 가장 정확한 기술이 가능
- 반면 실제 구현 시, 많은 구체적인 사항들이 알고리즘의 핵심적인 내용에 대한 이해를 방해할 수 있다.

```
#define MAX_ELEMENTS 100
int score[MAX_ELEMENTS];
int find_max_score(int n)
{
        int i, tmp;
        tmp=score[0];
        for(i=1;i<n;i++){
                if( score[i] > tmp ){
                        tmp = score[i];
                }
        }
        return tmp;
}
```

# 자료형

- 자료형(data type): “데이터의 종류”
- 정수, 실수, 문자열 등이 기초적인 자료형의 예

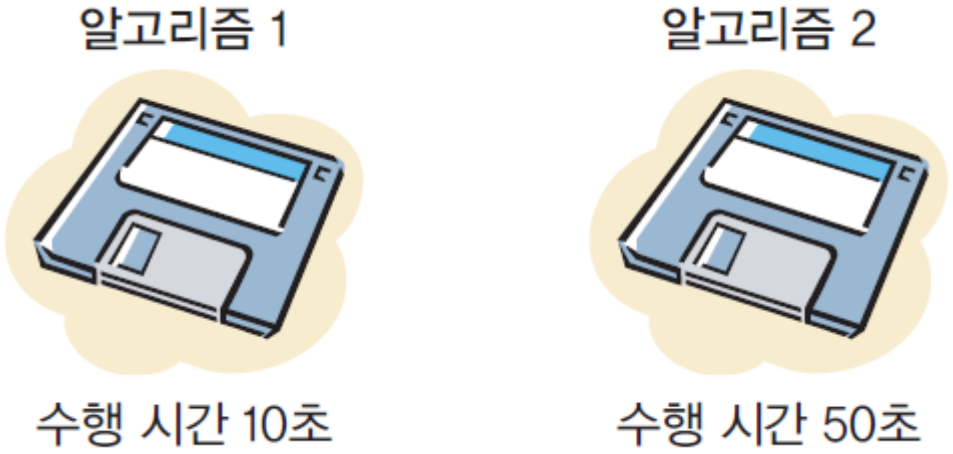


# 알고리즘의 성능분석

- 알고리즘의 성능 분석 기법
  - **수행 시간 측정**
    - 두개의 알고리즘의 실제 수행 시간을 측정하는 것
    - 실제로 구현하는 것이 필요
    - 동일한 하드웨어를 사용하여야 함
  - **알고리즘의 복잡도 분석**
    - 직접 구현하지 않고서도 수행 시간을 분석하는 것
    - 알고리즘이 수행하는 연산의 횟수를 측정하여 비교
    - 일반적으로 연산의 횟수는 n의 함수

# 수행시간측정

- 알고리즘을 프로그래밍 언어로 작성하여 실제 컴퓨터상에서 실행시킨 다음, 그 수행시간을 측정



# 수행시간 측정 2가지 방법

| 방법 #1 | 방법 #2 |
| --- | --- |
| #include <time.h><br><br>start = clock();<br>...<br>stop = clock();<br>double duration = (double)(stop - start) /<br>CLOCKS_PER_SEC; | #include <time.h><br><br>start = time(NULL);<br>...<br>stop = time(NULL);<br>double duration = (double) difftime(stop,<br>start); |
| • Cup시간을 계산한다<br>• Clock() 함수는 시스템 시각을 CLOCKS_PER_SEC단위로 반환한다.<br>• 초단위 출력을 위해 CLOCKS_PER-SEC으로 나누어 준다. | • Time()함수는 초단위로 측정한다.<br>• Difftime()함수는 두 수의 차이를 초단위로 반환한다. |

```c
int main(void)
{
    clock_t start, stop;
    double duration;
    start = clock();

    for (int i = 0; i < 100000000; i++) {}

    stop = clock();
    duration = clock();
    printf("start : %f₩n", (double)start);
    printf("stop : %f₩n", (double)stop);
    duration = ((double)stop - (double)start) / CLOCKS_PER_SEC;
    printf("수행시간은 %f 초입니다.₩n", duration);
    return 0;

}
/*
clock() : time.h에 들어있는 함수로 프로그램에 의해 프로세서가
        소비된 시간을 반환하는 함수입니다.프로세서가 측정한 프로그램 실행시간이라 볼 수 있다.
clock_t : clock ticks의 자료를 담고 있는 자료형으로 clock()의 반환형입니다.
CLOCKS_PER_SEC : 초당 clock ticks의 수를 나타낸 매크로로 시스템에 따라 기본 값이 다르다
*/
```

# 알고리즘 복잡도 분석

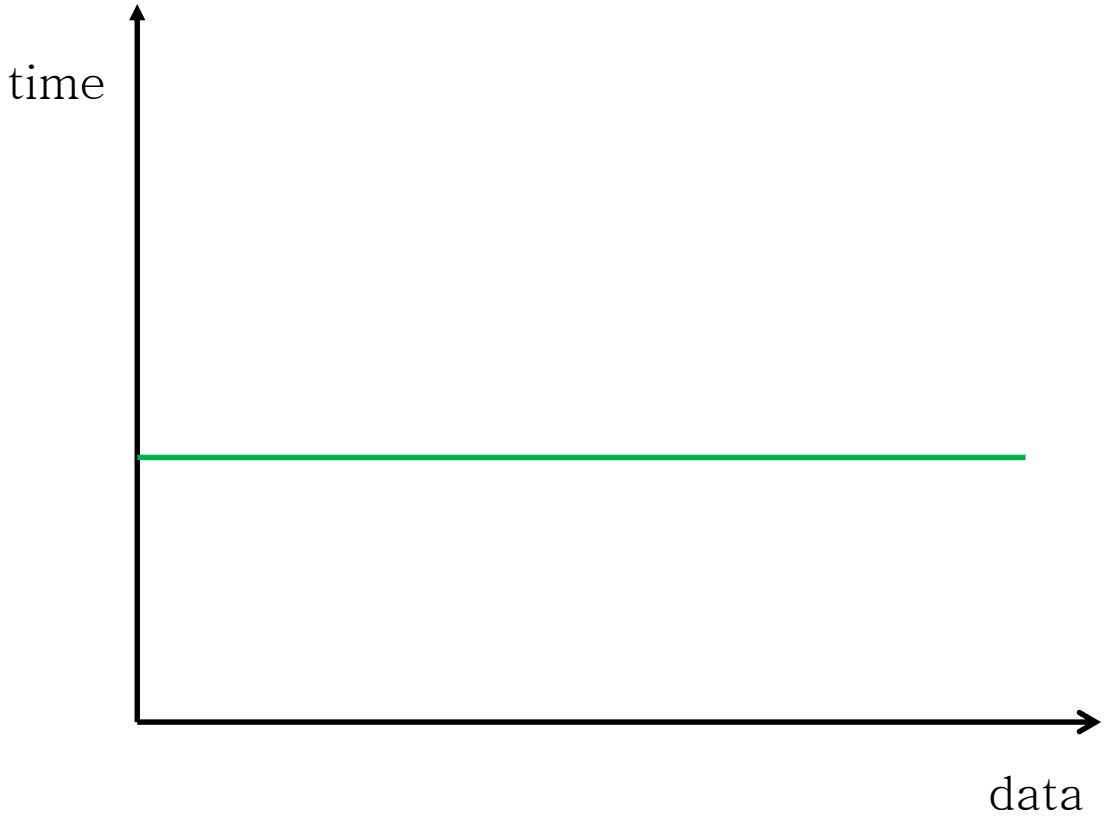
- 시간 복잡도는 알고리즘을 이루고 있는 연산들이 몇 번이나 수행되는지를 숫자로 표시
- 직접 수행해 보지 않고 알고리즘 분석기법들을 통해서 속도를 예측해 볼 수 있다.
- 알고리즘 수행 시간 분석을 시간 복잡도(time complexity)라 한다.
- 빅오표기법 으로 복잡도를 이해해 보기로 한다.
  - 알고리즘 성능을 수학적으로 표기해 준다
  - 시간, 공간 복잡도 표기가 가능하다.

# 빅오 표기법의 종류

- $O(1)$: 상수형
- $O(\log n)$: 로그형
- $O(n)$: 선형
- $O(n \log n)$: 선형로그형
- $O(n^2)$: 2차형
- $O(n^3)$: 3차형
- $O(2^n)$: 지수형
- $O(n!)$: 팩토리얼형

# O(1) : 상수형

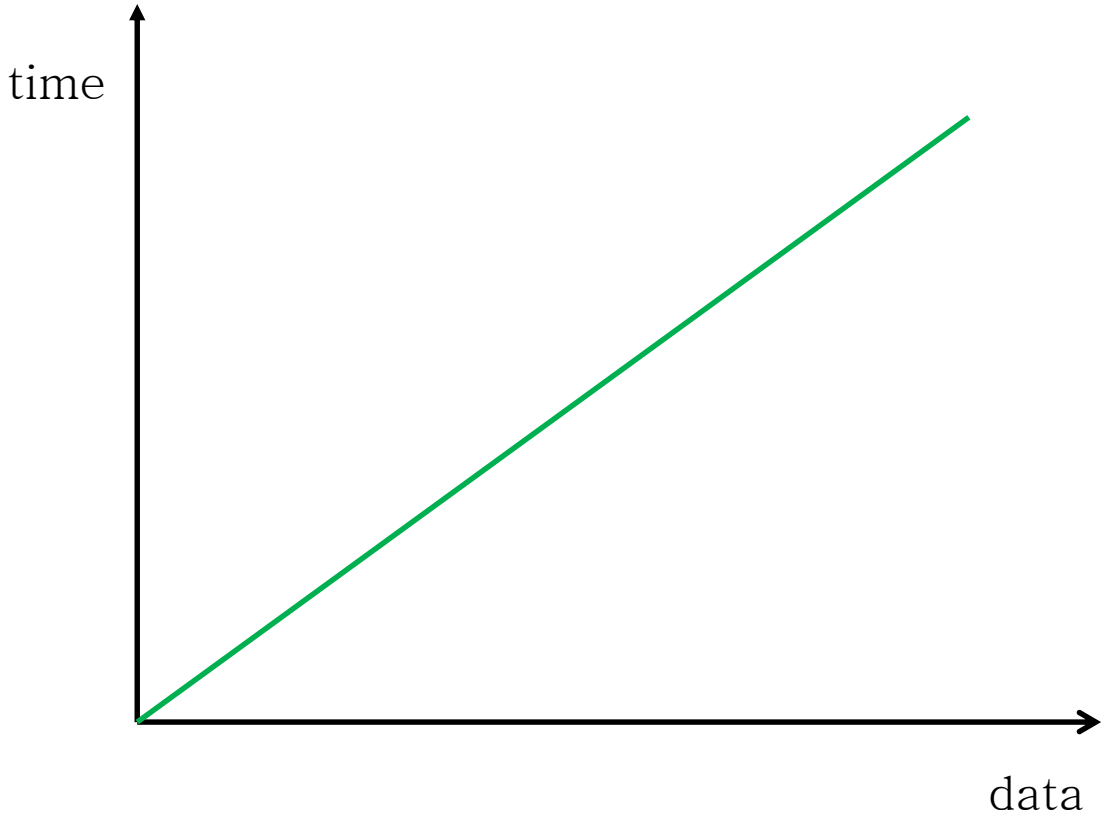
- 입력 data 의 크기 상관없이 항상 일정한 시간소요



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

bool my_f(int arr2[], int n)
{
    return (arr2[0] == 0) ? true : false;
}

int main(void)
{
    int i ;
    int arr[5];
    bool result;

    for (i = 0; i < 5; i++)
    {
        arr[i] = i;
    }

    result = my_f(arr, 5);

    printf("결과는 %d arr[0]=%d", result, arr[0]);

    return 0;
}
```

# O(n) : 선형

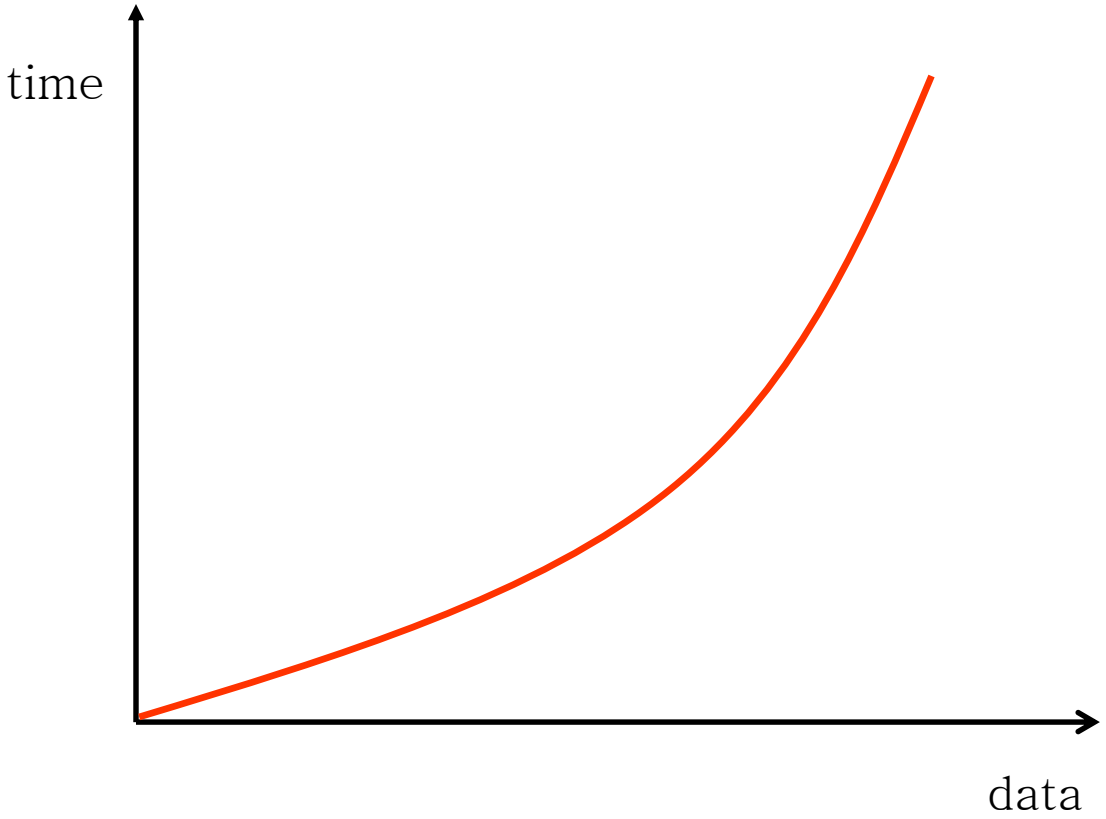
- 입력 data 의 크기에 비례하여 처리시간 결정



```c
void my_f(int arr2[], int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        printf("arr2[%d] = %d₩n", i, arr2[i]);
    }
}

int main(void)
{
    int i;
    int arr[5];

    for (i = 0; i < 5; i++)
    {
        arr[i] = i;
    }

    my_f(arr, 5);

    return 0;
}
```

# $O(n^2)$ : 2차형

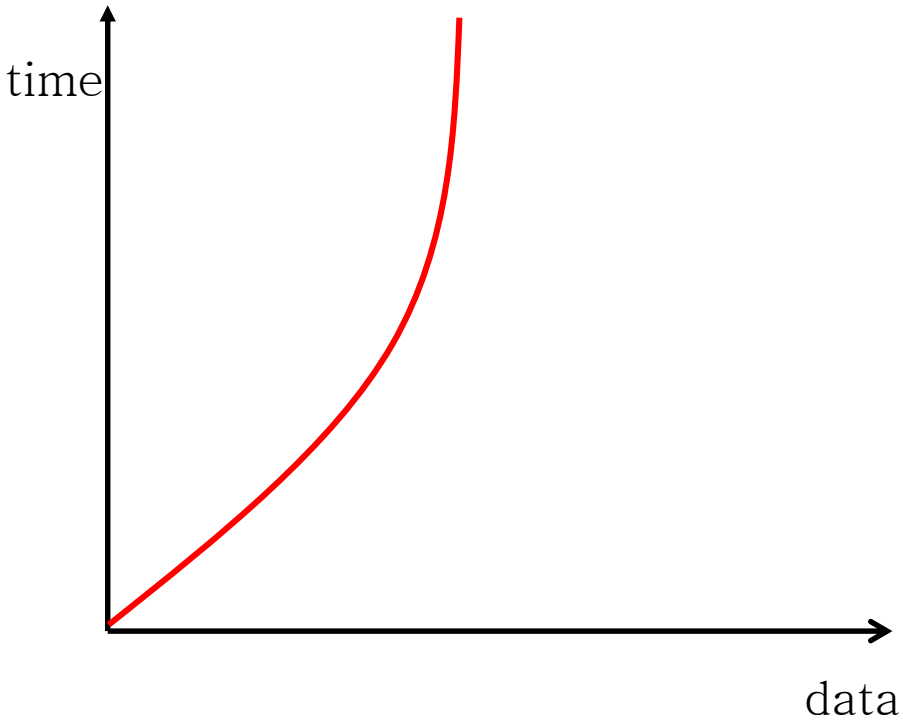
- 2차원 표같은 처리/ data만큼 가로*세로 처리한다.

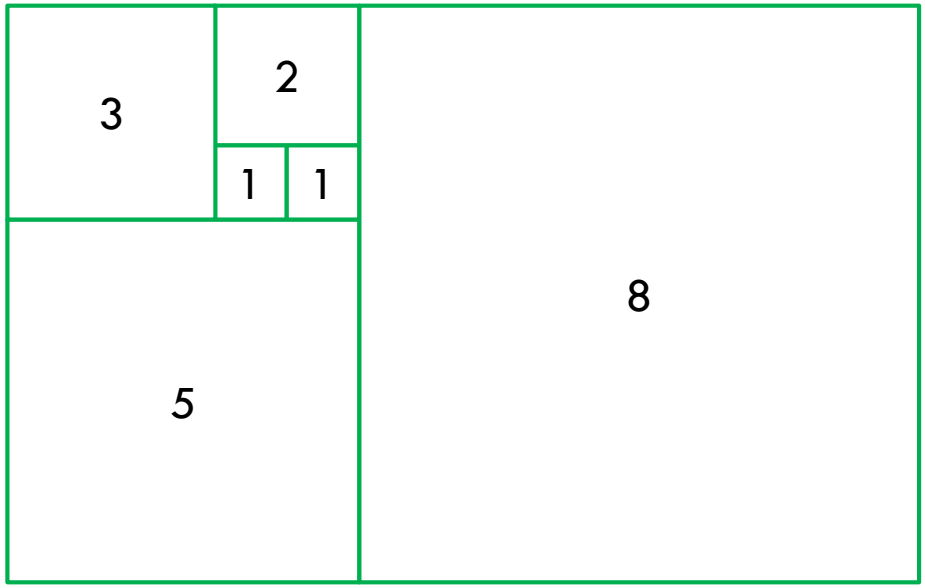


```c
void my_f(int arr2[], int n)
{
    int i, k;
    for (i = 0; i < n; i++)
    {
        for (k = 0; k < n; k++) {
            printf("i=%d,k=%d ---> %d ₩t", i, k, i+k);
        }
        printf("₩n");
    }
}

int main(void)
{
    int i;
    int arr[5];

    for (i = 0; i < 5; i++)
    {
        arr[i] = i;
    }

    my_f(arr, 5);

    return 0;
}
```

# $O(2^n)$ : 지수형

- 1부터 정사각형 으로 늘려가는 모양
- 예) Fibonacci 수열




1,1,2,3,5,8…

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

int my_f(int n)
{
    if (n <= 0) return 0;
    else if (n == 1) return 1;
    else return (my_f(n - 1) + my_f(n - 2));

}

int main(void)
{
    int n,i;

    printf("피보나치수열을 구할 수를 입력하시오 >>");
    scanf_s("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("%d , ", my_f(i));
    }
    return 0;
}
```

# 재귀함수 예제1

## 문제

20 이하의 자연수 N을 입력받아 재귀함수를 이용해서 문자열 "recursive"를 N번 출력하는 프로그램을 작성하시오.

## 입력 예

```
3
```

## 출력 예

```
recursive
recursive
recursive
```

# 재귀함수 예제2

## 문제

자연수 N을 입력받아 재귀함수를 이용하여 N부터 1까지 차례대로 출력하는 프로그램을 작성하시오.
N은 50이하의 자연수이다.

## 입력 예

```
5
```

## 출력 예

```
5 4 3 2 1
```

# 재귀함수 예제3

## 문제

100 이하의 자연수 N을 입력받아 재귀함수를 이용하여 1부터 N까지의 합을 구하는 프로그램을 작성하시오.

## 입력 예

```
100
```

## 출력 예

```
5050
```

# 재귀함수 예제4

## 문제

자연수 N을 입력받아 1부터 N까지 출력을 하되 n-1번째 값은 n번째 값을 2로 나눈 몫이 되도록 하는 프로그램을 작성하시오.

## 입력 예

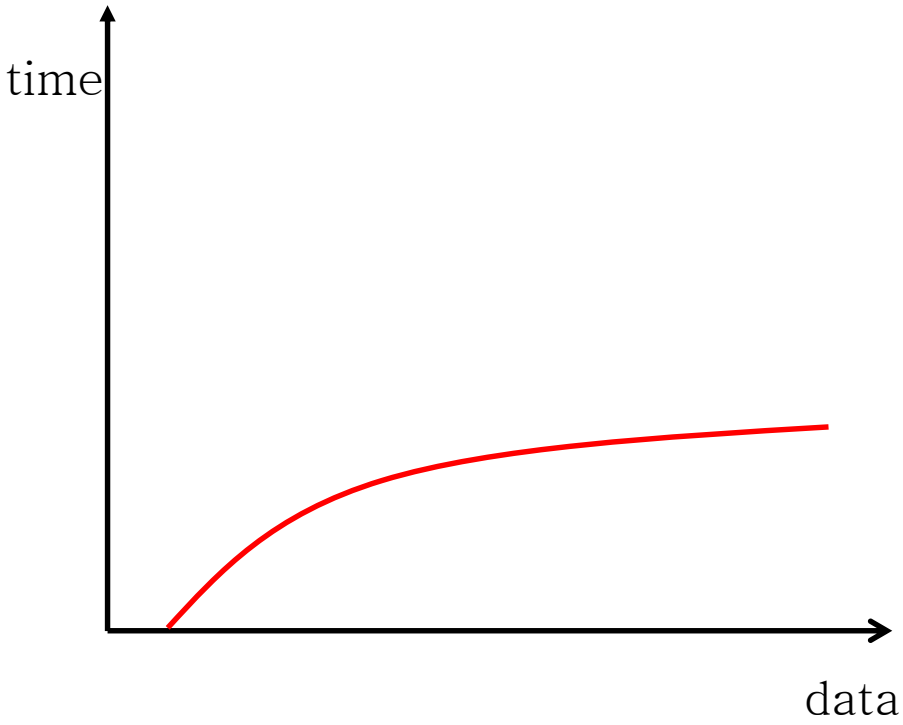
```
100
```

## 출력 예

```
1 3 6 12 25 50 100
```

# O(log n) : 로그형

- 한번처리 후 data의 절반만큼 으로 처리
- data가 증가해도 실행시간이 크게 증가하지 않는다
- 예) Binary Search (2진검색)




1 2 3 4 5 6 7 8 9 10

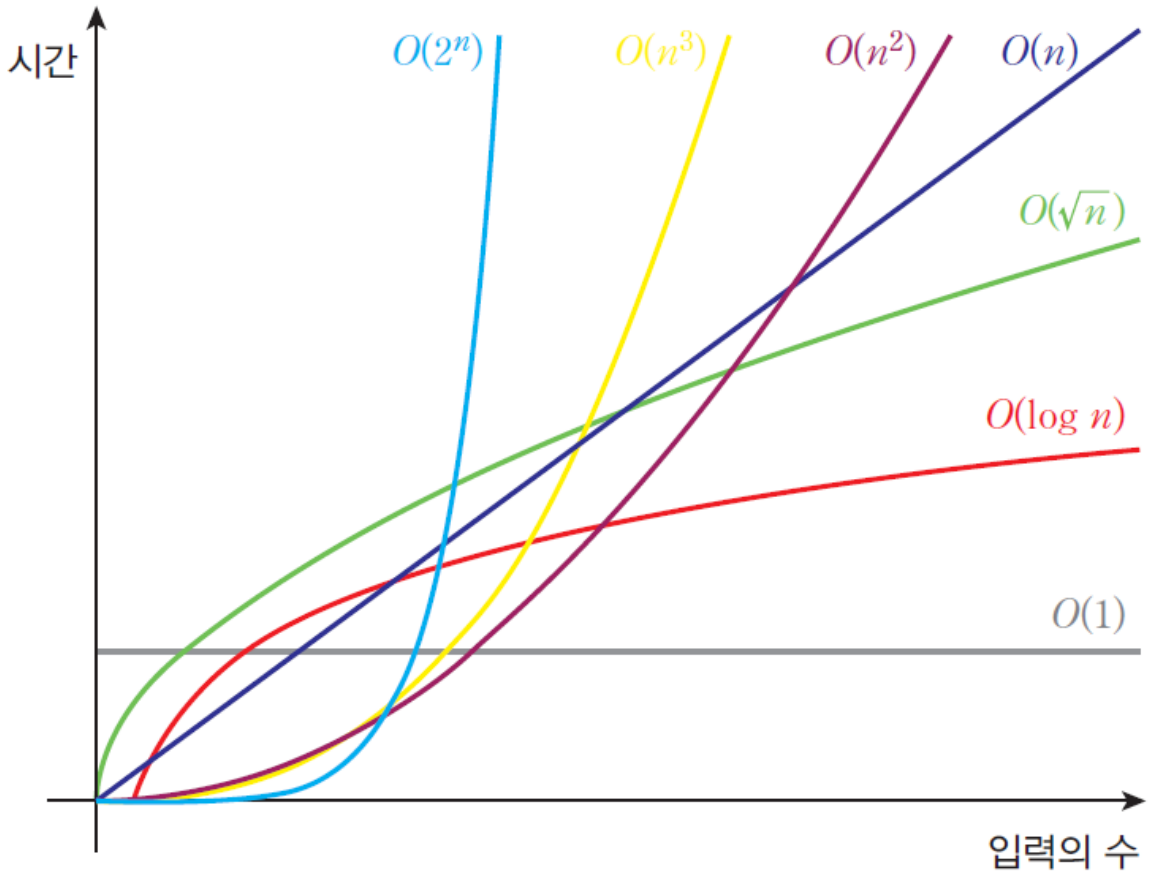
찾는값 : 6

```c
#include<stdio.h>

// ...

int main(void)
{
    int n, i,  mid;
    int arr[10];
    int start = 0, end = 9;

    for (i = 0; i < 10; i++)
    {
        arr[i] = i+1;
        printf("arr[%d] = %d ₩n", i,arr[i]);
    }

    printf("찾고자 하는 수를 입력 하시오 >>");
    scanf_s("%d", &n);

    while (true)
    {
        mid = (start + end) / 2;
        if (arr[mid] == n)
        {
            printf("찾는 수의 위치는 %d ", mid);
            break;
        }
        else if (arr[mid] > n)  end = mid - 1;
        else start = mid + 1;
    }

    return 0;
}
```

```c
#include<stdio.h>

int arr[10];
int start = 0, end = 9;

void my_f(int n)
{
    int mid = (start + end) / 2;
    if (arr[mid] == n)
    {
        printf("찾는 수의 위치는 %d ", mid);
    }
    else if (arr[mid] > n)
    {
        end = mid - 1;
        my_f(n);
    }
    else
    {
        start = mid + 1;
        my_f(n);
    }
}
```

```c
int main(void)
{
    int n, i,  mid;

    for (i = 0; i < 10; i++)
    {
        arr[i] = i+1;
        printf("arr[%d] = %d ₩n", i,arr[i]);
    }

    printf("찾고자 하는 수를 입력 하시오 >>");
    scanf_s("%d", &n);
    my_f(n);
    return 0;
}
```

# 빅오표기법



# 빅오 표기법이외의 표기법

- 빅오메가 표기법
  - 모든 $n \geq n_0$에 대하여 $|f(n)| \geq c|g(n)|$을 만족하는 2개의 상수 c와 $n_0$가 존재하면 $f(n)=\Omega(g(n))$이다.
  - 빅오메가는 함수의 하한을 표시한다.
  - (예) $n \geq 5$ 이면 $2n+1 < 10n$ 이므로 $n = \Omega(n)$

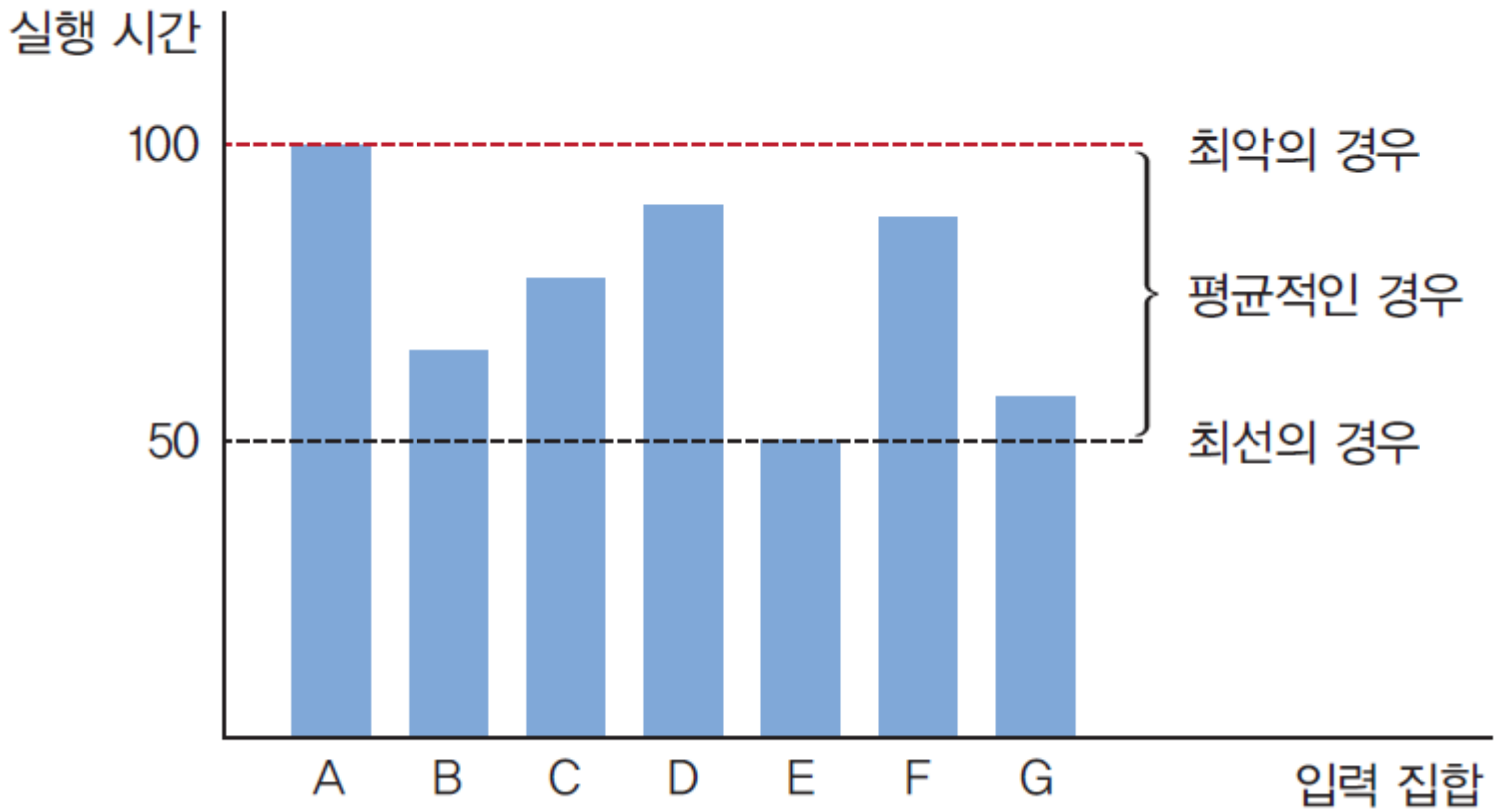
# 빅오 표기법이외의 표기법

- 빅세타 표기법
  - 모든 $n \geq n_0$에 대하여 $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$을 만족하는 3개의 상수 $c_1$, $c_2$와 $n_0$가 존재하면 $f(n)=\theta(g(n))$이다.
  - 빅세타는 함수의 하한인 동시에 상한을 표시한다.
  - $f(n)=O(g(n))$이면서 $f(n)= \Omega(g(n))$이면 $f(n)= \theta(n)$이다.
  - (예) $n \geq 1$이면 $n \leq 2n+1 \leq 3n$이므로 $2n+1 = \theta(n)$

# 최선, 평균, 최악의 경우

- 알고리즘의 수행시간은 입력 자료 집합에 따라 다를 수 있다.

- **최선의 경우(best case):** 수행 시간이 가장 빠른 경우
- **평균의 경우(average case):** 수행시간이 평균적인 경우
- **최악의 경우(worst case):** 수행 시간이 가장 늦은 경우


실행 시간
100
50
최악의 경우
평균적인 경우
최선의 경우
A
B
C
D
E
F
G
입력 집합

# (예) 최선, 평균, 최악의 경우

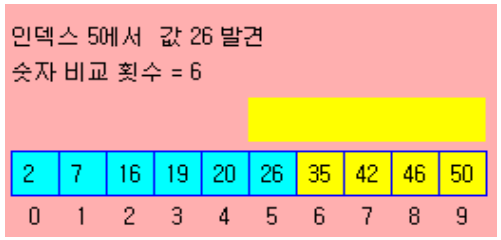
- (예) 순차탐색
- **최선의 경우**: 찾고자 하는 숫자가 맨 앞에 있는 경우
  ∴ O(1)


- **최악의 경우**: 찾고자 하는 숫자가 맨 뒤에 있는 경우
  ∴ O(n)


- **평균적인 경우**: 각 요소들이 균일하게 탐색된다고 가정하면

$$(1+2+\cdots+n)/n=(n+1)/2$$

∴ O(n)



# 최선, 평균, 최악의 경우

- 최선의 경우: 의미가 없는 경우가 많다.
- 평균적인 경우: 계산하기가 상당히 어려움.
- 최악의 경우: 가장 널리 사용된다. 계산하기 쉽고 응용에 따라서 중요한 의미를 가질 수도 있다.
  - (예) 비행기 관제업무, 게임, 로보틱스

# 문제: 순차탐색 프로그램 작성하기

- 위 data 그대로 이용
- 찾는값 5 입력시 비교횟수 1출력
- 찾는값 43 입력시 비교횟수 10출력
- 찾는값 29 입력시 비교횟수 6출력

```c
#include<stdio.h>

int arr[10] = { 5,9,10,17,21,29,33,37,38,43 };


int main(void)
{
    int n, i, cnt=0;

    printf("찾고자 하는 수를 입력 하시오 >>");
    scanf_s("%d", &n);
    for (i = 0; i < 10; i++)
    {
        cnt++;
        if (arr[i] == n)
        {
            printf("비교횟수 = %d ", cnt);
            break;
        }
    }
    return 0;
}
```