자료구조

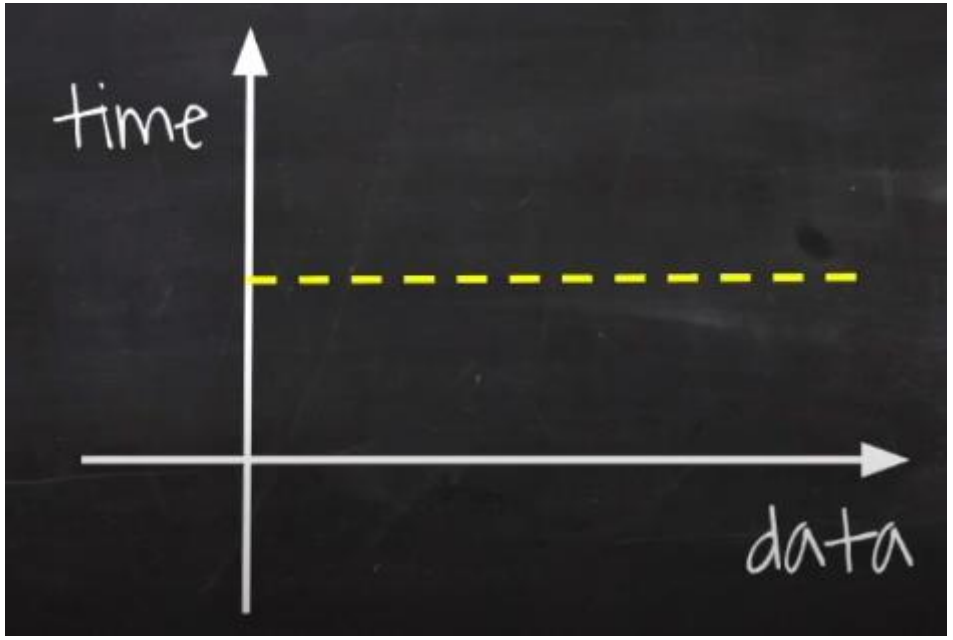
빅오(Big-O)표기법

(참고 : https://www.youtube.com/watch?v=6Iq5iMCVsXA)

O(1) : 일정한 속도로 결과가 도출되는 것

```
F(int[] n){
   return (n[0] == 0)? true:false;
}
```

해당 함수와 같은 경우를 말한다.
n의 개수와 상관없이
인덱스 0번째의 값을 비교하여 리턴하기에
일정한 결과가 나온다.

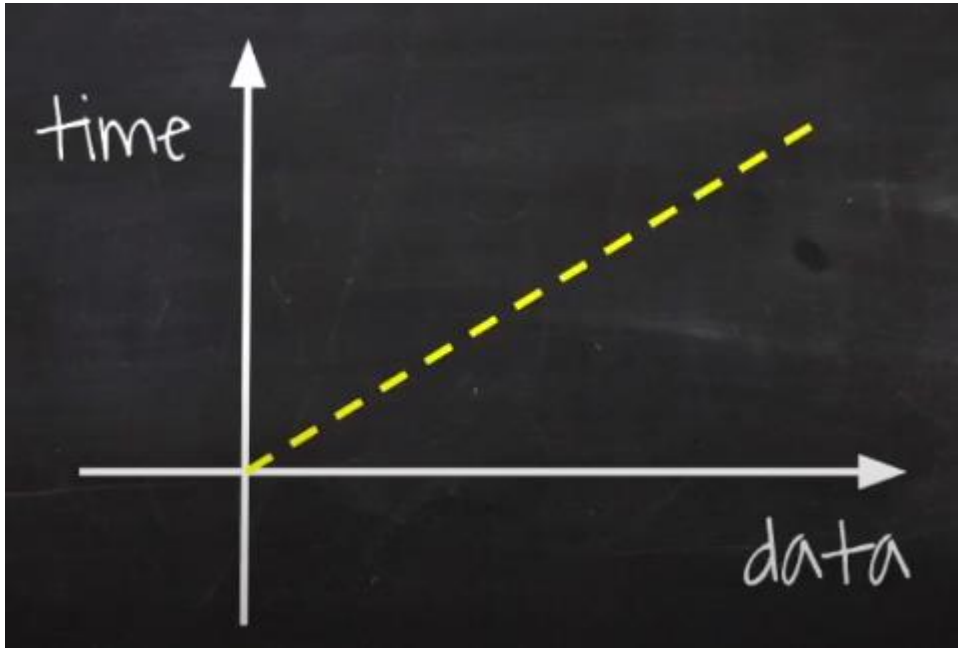


그래프로 보면 위와 같다.

O(n) : 처리되는 양에 따라 처리 시간이 바뀜

```
F(int[] n) {
   for i = 0 to n.length
      print i
}
```

해당 함수와 같은 경우를 말한다.
n의 개수가 늘어나면 늘어날수록
결과시간 또한 늘어난다.

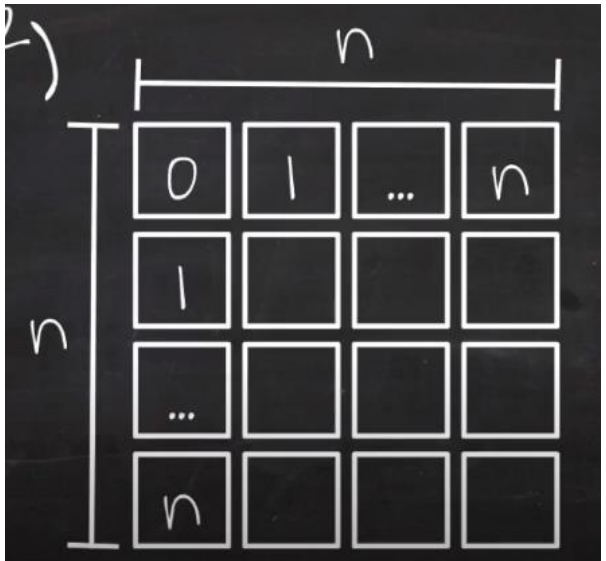


그래프로 보면 위와 같다.

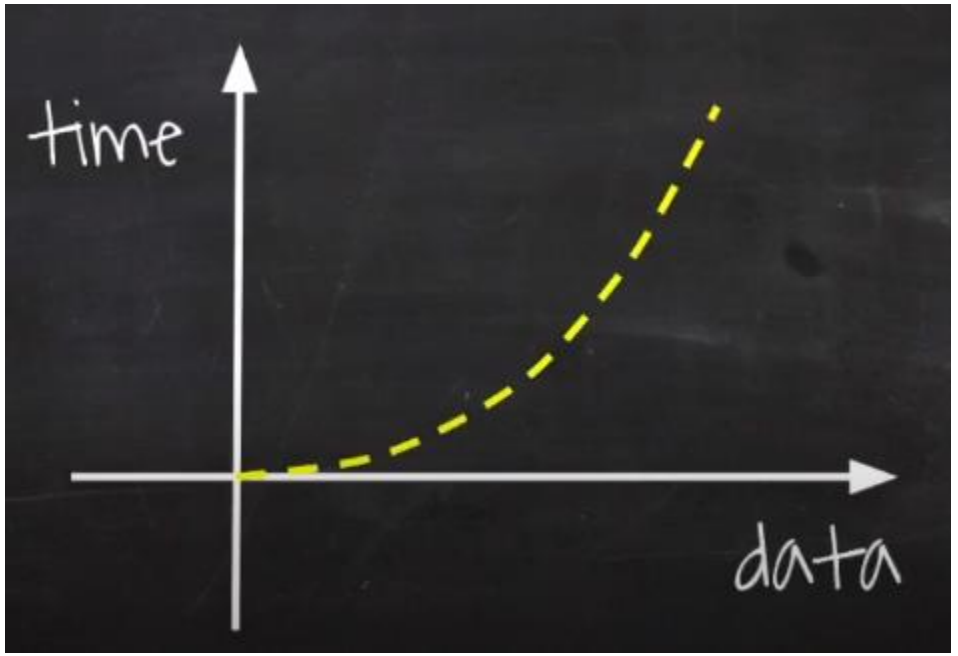
O(n^2) : n의 가로, 세로 길이만큼의 처리시간이 걸린다.

```
F(int[] n) {
   for i = 0 to n.length
      for j = 0 to n.length
         print i + j;
}
```

해당 함수와 같은 경우를 말한다.
n이 두 번 돌기에



왼쪽과 같은 그림의
모양이 된다.

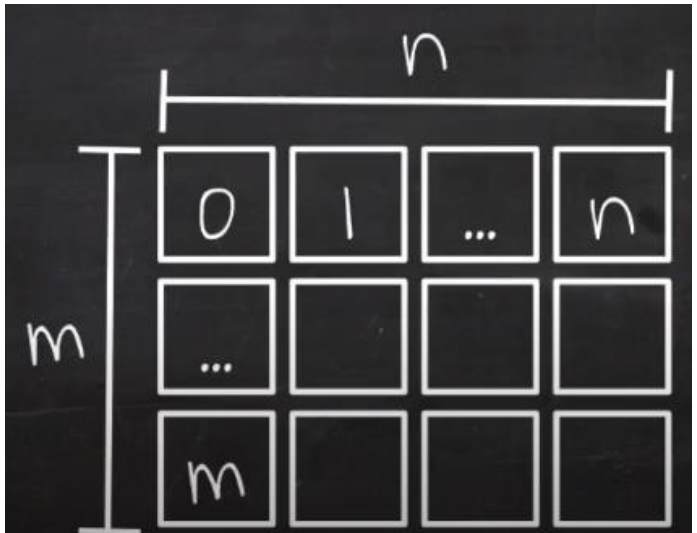


그래프로 보면 위와 같다.

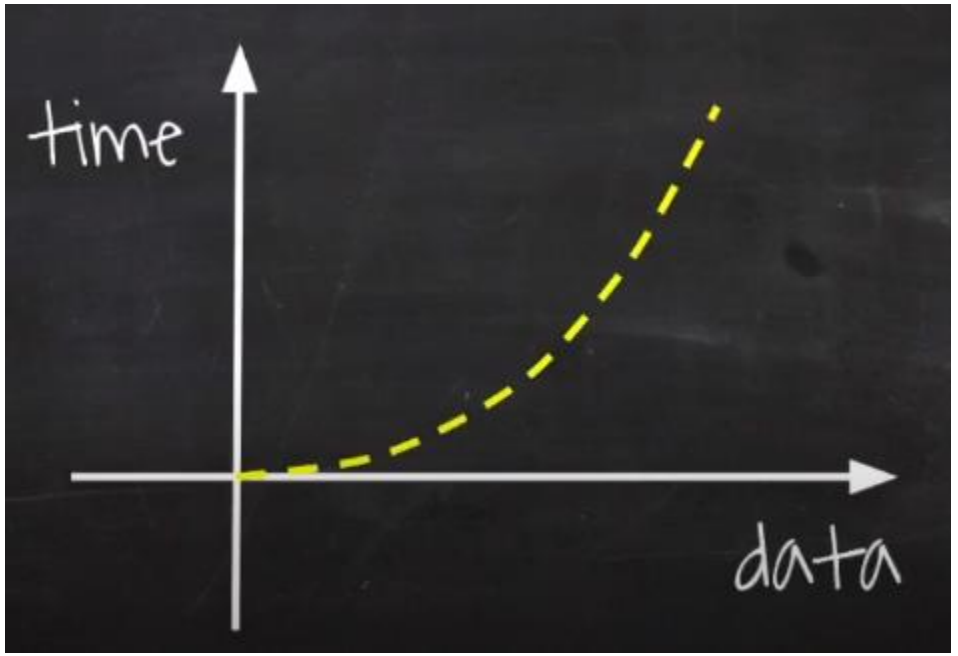
O(nm) : O(n^2)과 비슷하지만 n을 두 번 돌리는게 아닌 n,m으로 나눠서 돌린다.

```
F(int[] n, int[] m) {
   for i = 0 to n.length
      for j = 0 to m.length
         print i + j;
}
```

해당 함수와 같은 경우를 말한다.
n과 m으로 돈다.



왼쪽 그림과 같은
모양을 가진다.
n^2처럼 일정하지 않고
m에 따라
처리되는 양이 결정된다.

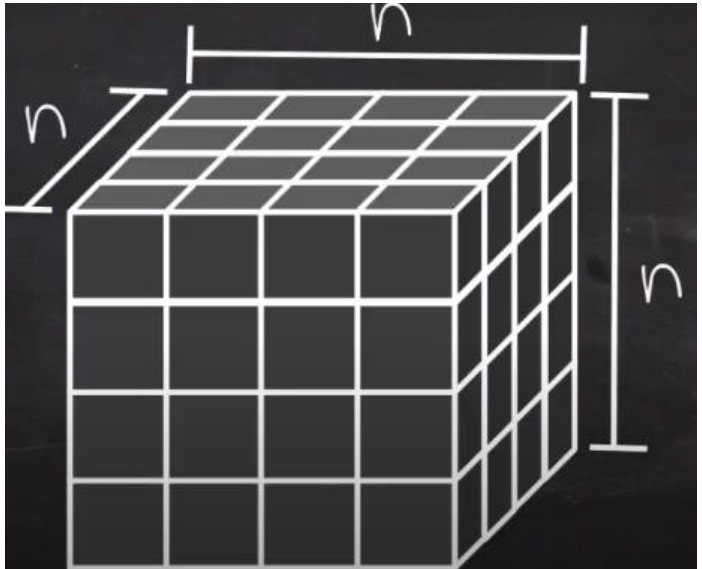


그래프로 보면 위와 같다.

O(n^3) : n을 세 번 돌리므로 양이 커질수록 처리시간이 급격하게 늘어난다.

```
F(int[] n) {
   for i = 0 to n.length
      for j = 0 to n.length
         for k = 0 to n.length
            print i + j + k;
}
```

해당 함수와 같은 경우를 말한다.
n을 세 번 돌린다.



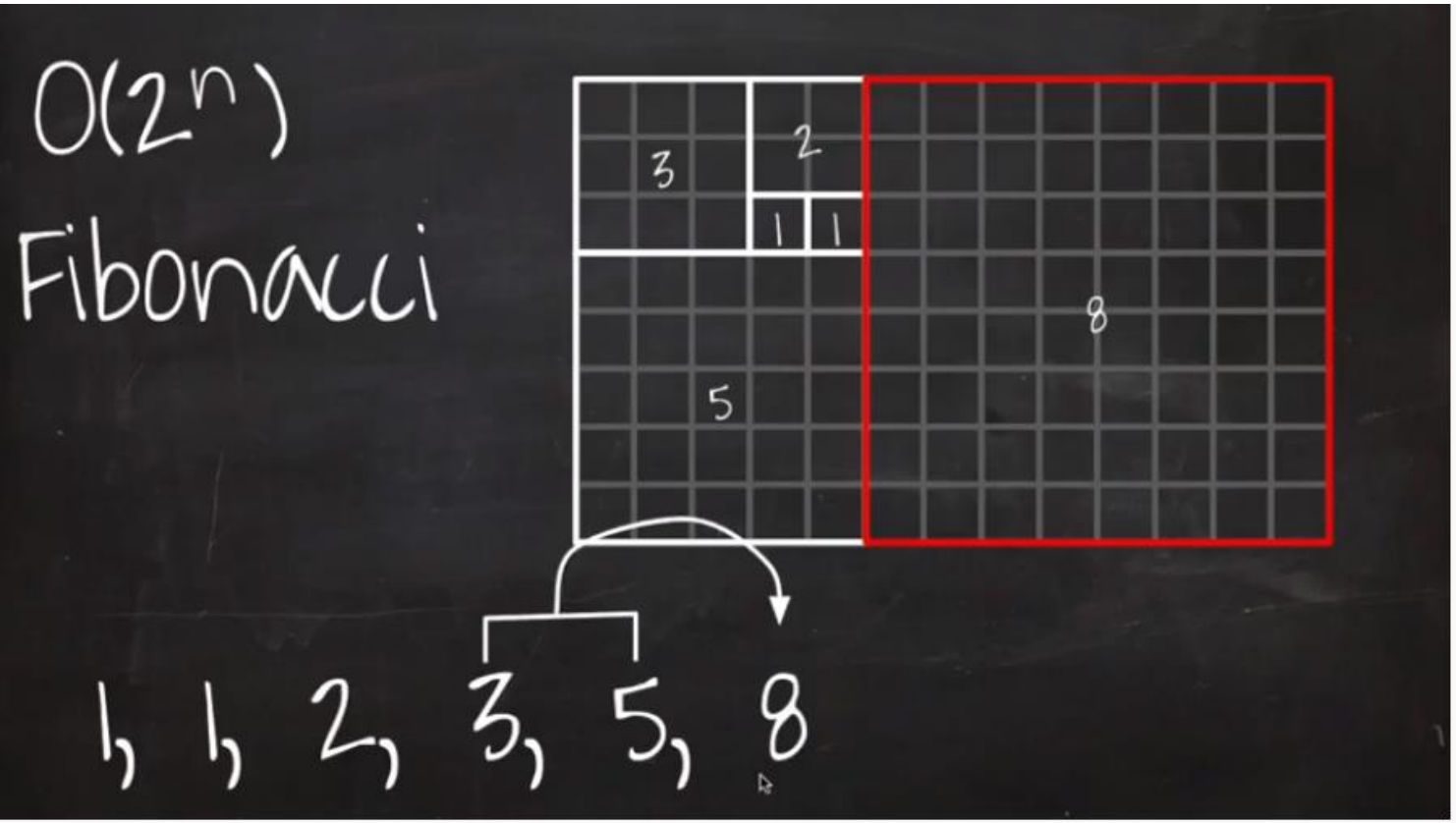
왼쪽 그림과 같은
모양을 가진다.

그래프로 보면 n^2와 비슷하지만
높이가 추가됨으로 처리 시간은 급격하게
올라가는 것을 볼 수 있다.

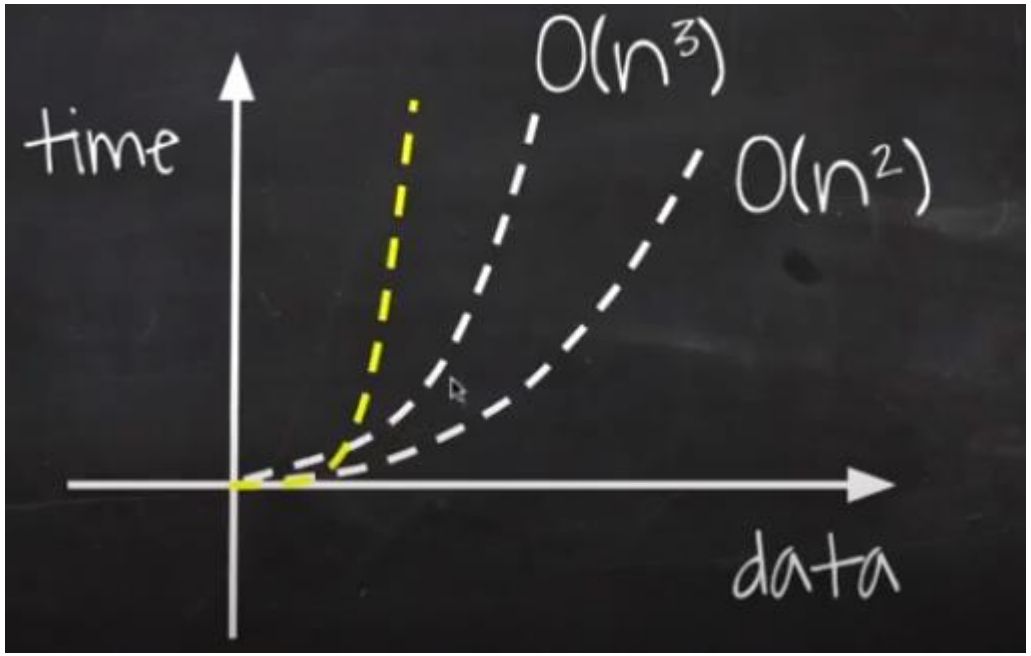
O(2^n) : 이전 값과 그 이전 값을 더하여 현재 값을 만든다. 더하여 증가하기에 처리시간 또한 급격하게 늘어난다.

```
F(n, r) {
  if (n <= 0) return 0;
  else if (n == 1) return r[n] = 1;
  return r[n] = F(n - 1, r) + F(n - 2, r);
}
```

해당 함수는 대표적인 피보나치 수열을 재귀함수로 구성한 것인데,
리턴 값을 보면 알 수 있듯이 이전 값과 그 이전 값을 더하여 리턴한다.

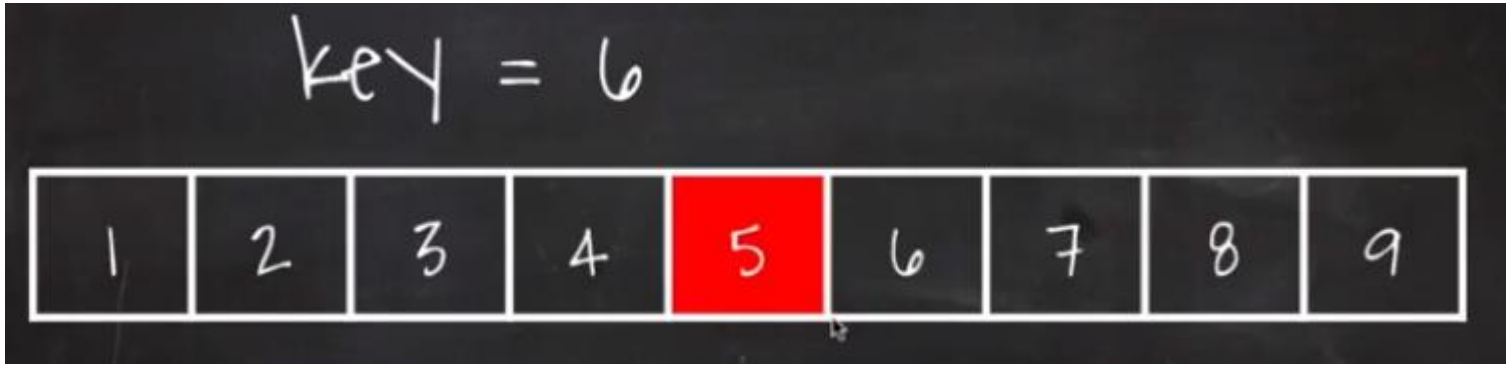


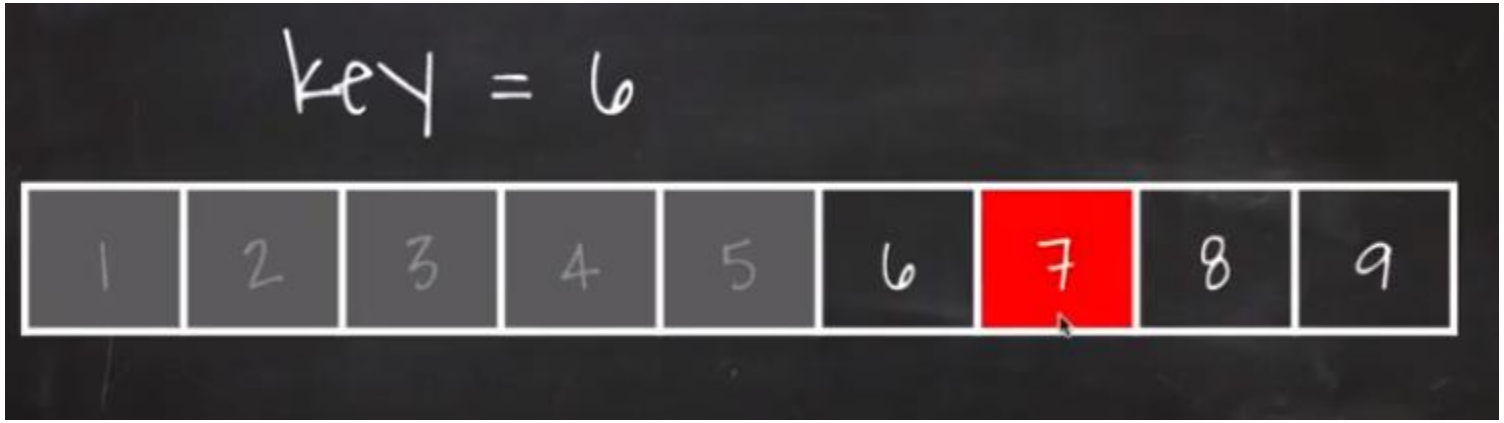
위와 같은 모양을 가진다.



해당 그래프와 같다.

O(log n) : 먼저 데이터에서 중간 값을 찾고 중간 값과 찾는 값을 비교하여 작으면 오른쪽으로 날리고 크면 왼쪽을 날려버리는 식으로 찾는다.
필요 없는 부분을 비교하지 않아도 됨으로 처리 시간이 줄어든다.

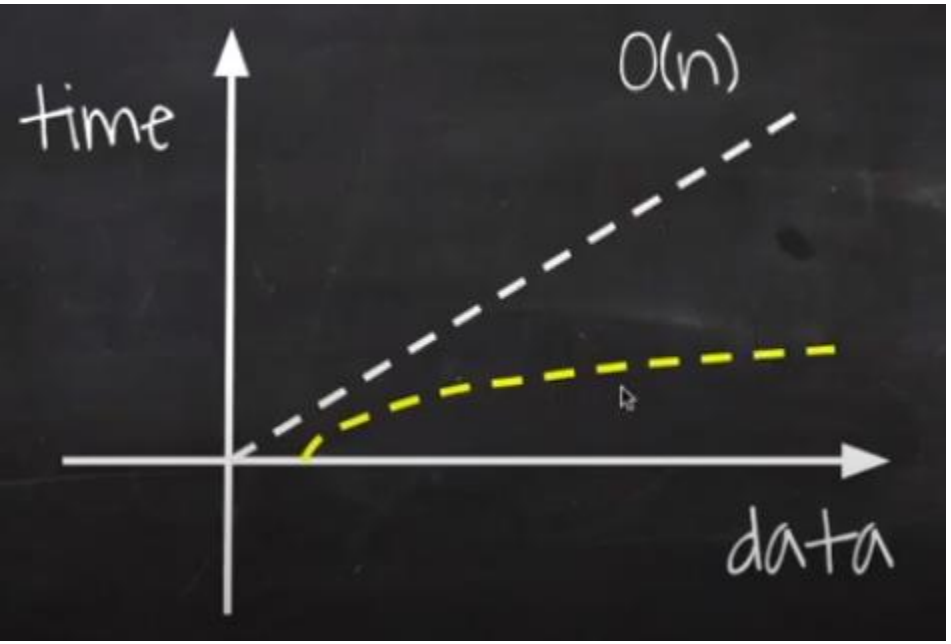




위와 같이 중간 값과 키 값을 비교하여 절반 씩 뚝뚝 버리기에
효율적으로 키 값을 찾는다. (대표적인 binary search)

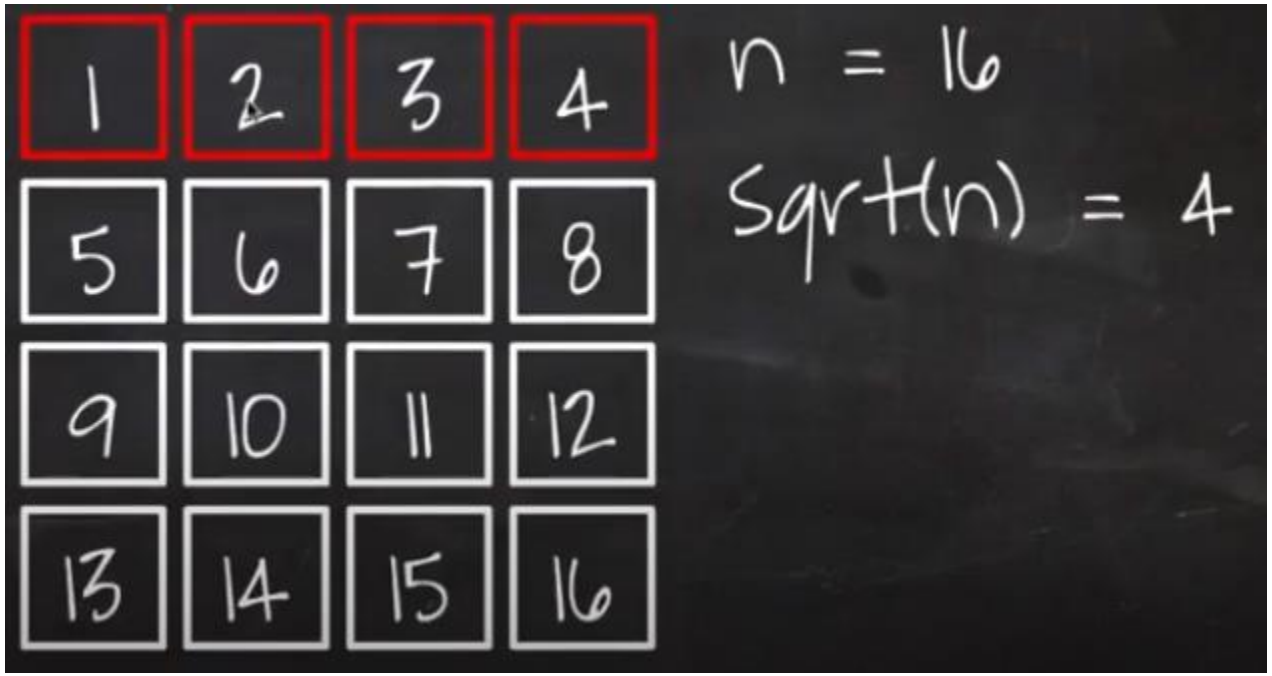
```
F(k,arr,s,e) {
   if (s > e) return -1;
   m = (s + e) / 2;
   if (arr[m] == k) return m;
   else if (arr[m] > k) return F(k,arr,s,m-1);
   else return F(k,arr,m+1,e);
}
```

위와 같은 함수는 재귀함수를 이용하여 binary search를 구현한 결과이다.



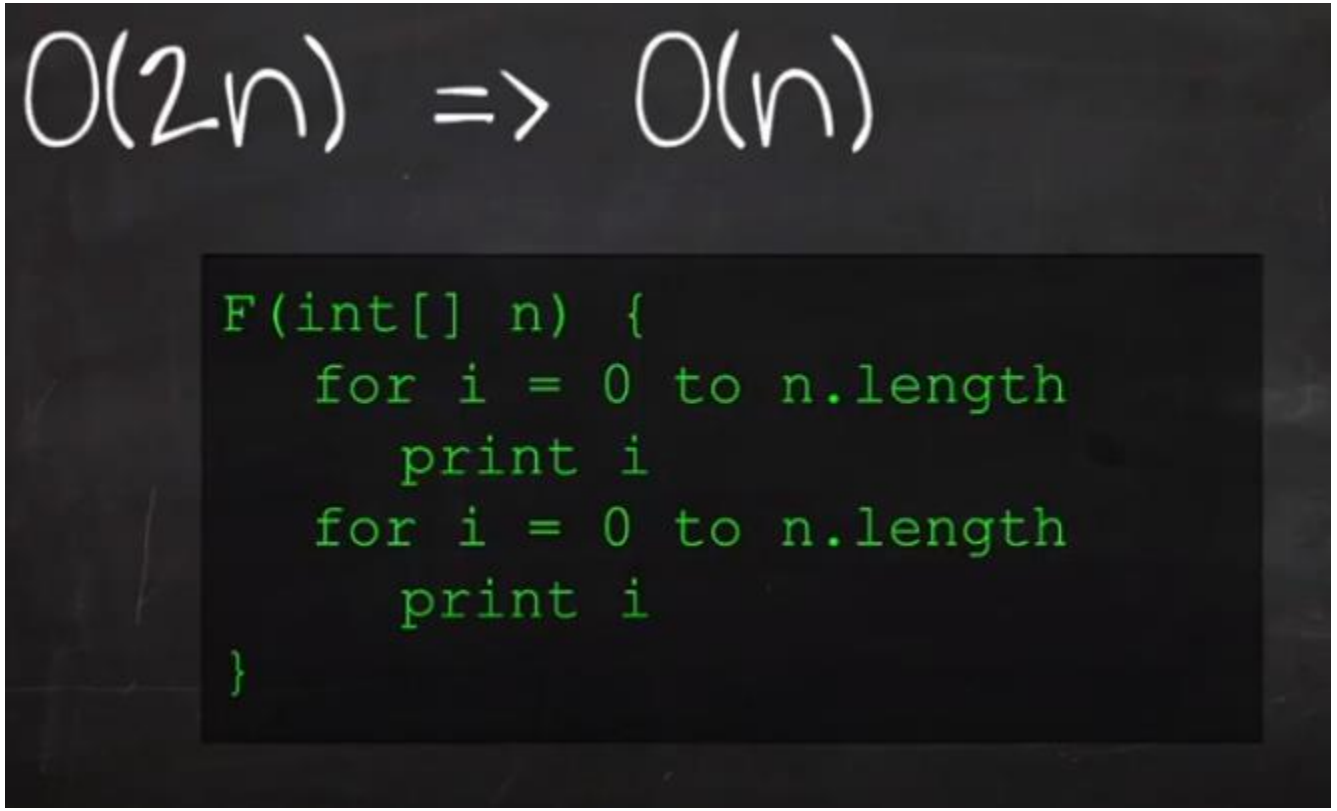
그래프로 나타내면 왼쪽과 같이 나타낼 수 있으며, 데이터가 증가해도 그래프가 크게 증가하지 않는다는 것을 볼 수 있다.

O(sqrt(n)) : 제곱근을 이용한 것으로, 만약 16개의 데이터가 있다면 루트를 씌워 4로 만들고 해당 값 만큼을 처리한다.



왼쪽과 같은 모양을 가진다.

빅 오 표기법에서는 상수는 과감하게 버린다.



왼쪽과 같은 경우인데,
버리는 이유는 실제 알고리즘의 러닝타임을 재기위해 만든 것이 아니라 장기적으로 데이터가 증가함에 따른 처리시간의 증가율을 보기 위해 나온 것이기 때문이다. 즉, 상수는 변하지 않는 값(증가하지 않는 값)이기에 증가율에 영향을 미치지 않는다.