|코딩테스트 스터디 참고자료 (6/15)|

6/15 코딩테스트 스터디

최소공배수와 최대공약수

최대공약수

```
function gcd(a, b) {
  if (!b) {
      return a;
  }

  return gcd(b, a % b);
}
```

ex1)
gcd(6, 12) => gcd(12, 6) => gcd(6, 0) => 6

ex2)
gcd(3, 19) => gcd(19, 3) => gcd(3, 1) => gcd(1,0) => 1

TIP
gcd(큰수, 작은수) =>gcd(작은수, 큰수%작은수) --- => gcd(n, 0)
=> n이 최대공약수

https://school.programmers.co.kr/learn/courses/30/lessons/12940

최소공배수와 최대공약수

최소공배수

$$a * b = GCD(a, b) * LCM(a, b)$$

```
function lcm(a, b) {
  return (a * b) / gcd(a, b);
}
```

gcd(4, 24 % 4) -> gcd(4, 0)

ex1)

lcm(4, 24) = (4 * 24) / gcd(4, 24) = 96 / 4 = 24

gcd(7, 15 % 7) -> gcd(7, 1)

gcd(1, 7 % 1) -> gcd(1, 0)

ex2)

lcm(7, 15) = (7 * 15) / gcd(7, 15) = 105 / 1 = 105

https://school.programmers.co.kr/learn/courses/30/lessons/12940

소수판별

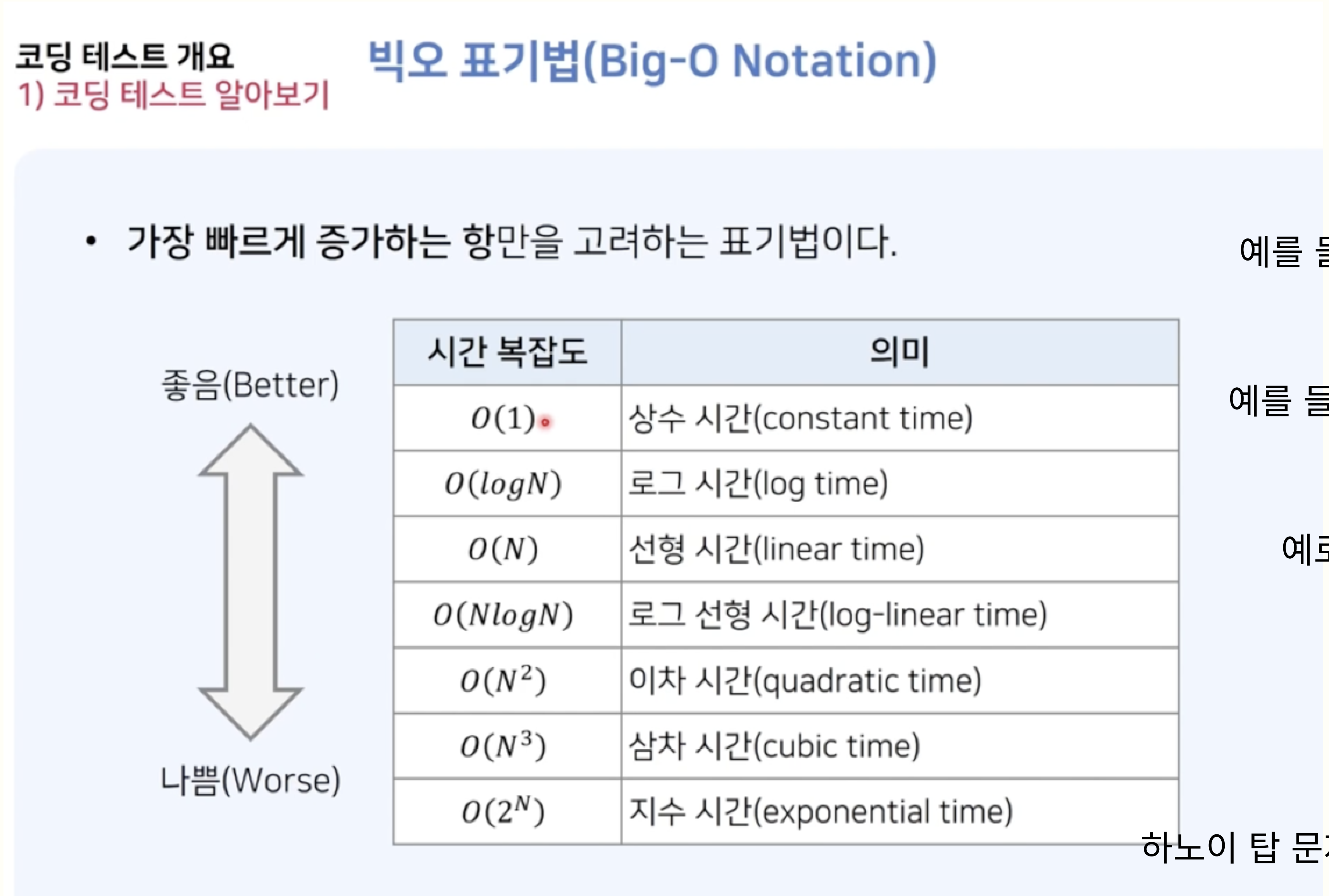
```
function isPrime(num) {
  for(let i = 2, sqrt = Math.sqrt(num); i <= sqrt; i++)
      if(num % i === 0) return false;
  return num > 1;
}
```

소수를 판별하기 위해선 주어진 수(num)를 2부터 해당 수(num)의 제곱근(Math.sqrt(num))까지의 숫자로 나누어보면 된다.

만약 해당 수(num)이 소수가 아니라면, 어떤 숫자로든지 나누어지기 때문에 나누는 숫자는 해당 수(num)의 약수 중 하나이다.
그리고 약수는 해당 수(num)의 제곱근(Math.sqrt(num)) 이하의 범위에 반드시 존재한다.

https://school.programmers.co.kr/learn/courses/30/lessons/12921

시간복잡도



코딩 테스트 개요
1) 코딩 테스트 알아보기

빅오 표기법(Big-O Notation)

- **가장 빠르게 증가하는 항**만을 고려하는 표기법이다.

좋음(Better)

나쁨(Worse)

시간 복잡도	의미
$O(1)$	상수 시간(constant time)
$O(logN)$	로그 시간(log time)
$O(N)$	선형 시간(linear time)
$O(NlogN)$	로그 선형 시간(log-linear time)
$O(N^2)$	이차 시간(quadratic time)
$O(N^3)$	삼차 시간(cubic time)
$O(2^N)$	지수 시간(exponential time)

상수 시간 (O(1)): 입력 크기에 관계없이 일정한 실행 시간을 가지는 알고리즘.
예를 들어, 배열에서 인덱스로 요소에 접근하는 경우

선형 시간 (O(n)): 입력 크기에 비례하여 실행 시간이 선형적으로 증가하는 알고리즘.
예를 들어, 배열의 모든 요소를 한 번씩 방문하는 경우가 이에 해당(단일 for문, map, reduce, filter 메소드 등)

이차 시간 (O(n^2)): 입력 크기의 제곱에 비례하여 실행 시간이 증가하는 알고리즘..
예를 들어, 2차원 배열을 순회하면서 모든 요소를 방문하는 경우가 이에 해당(선택 정렬, 삽입 정렬, 버블 정렬 등)

로그 시간 (O(log n)): 입력 크기의 로그에 비례하여 실행 시간이 증가하는 알고리즘.
예로, 이진 검색 알고리즘이 있다. 이 알고리즘은 입력을 절반씩 나누어 탐색하기 때문에 매우 효율적이다.

선형로그 시간 (O(n log n)): 선형 시간과 로그 시간의 조합의 알고리즘
예로, 병합 정렬, 퀵 정렬이 있다.

지수 시간 (O(2^n)): 입력 크기에 대해 2의 지수 함수로 실행 시간이 증가하는 알고리즘.
하노이 탑 문제와 같은 재귀적인 알고리즘이 이에 해당하지만, 큰 입력에 대해 매우 느려지므로 효율적인 방법을 찾아야 한다.

https://www.youtube.com/watch?v=BEVnxbxBqi8

재귀함수 관련 글
https://data-marketing-bk.tistory.com/27

피보나치 - 기본 재귀호출 (시간복잡도(O(n^2)))

0과 1로 시작하고 n번째 피보나치 수는 바로 직전의
두 피보나치 수의 합
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

```
function fibonacci(n) {
  if(n <= 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2
);
}
```

fibonacci(3)

이 경우에는 재귀적으로 다음과 같이 호출된다.

fibonacci(3) = fibonacci(2) + fibonacci(1)

fibonacci(2) = fibonacci(1) + fibonacci(0)

fibonacci(1)과 fibonacci(0)은 각각 1과 0을 반환

fibonacci(2) = 1 + 0 = 1

fibonacci(3)은 fibonacci(2) + fibonacci(1) = 1 + 1 = 2

https://school.programmers.co.kr/learn/courses/30/lessons/12945

피보나치 - 기본 재귀호출 (시간복잡도(O(n^2)))

0과 1로 시작하고 n번째 피보나치 수는 바로 직전의
두 피보나치 수의 합

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ……

```
function fibonacci(n) {
  if(n <= 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2
);
}
```

fibonacci(5)

fibonacci(5) = fibonacci(4) + fibonacci(3)

fibonacci(4) = fibonacci(3) + fibonacci(2)

fibonacci(3) = fibonacci(2) + fibonacci(1)

fibonacci(2) = fibonacci(1) + fibonacci(0) => 1 + 0 = 1

fibonacci(3) = fibonacci(2) + fibonacci(1) => 1 + 1 = 2

fibonacci(4) = fibonacci(3) + fibonacci(2) => 2 + 1 = 3

따라서 ibonacci(5)= fibonacci(4) + fibonacci(3) => 3 + 2 = 5

https://school.programmers.co.kr/learn/courses/30/lessons/12945

피보나치 - memoization (시간복잡도 O(n))

```
let memo = [];
function fibonacci(n) {
  if (n <= 1) {
    return n;
  } else if (!memo[n]) {
    memo[n] = fibonacci(n - 1) + fibonacci(n - 2);
  }
  return memo[n];
}
```

fibonacci(5) = fibonacci(4) + fibonacci(3)
memo 배열에는 아직 어떤 값도 저장되어 있지 않으므로 fibonacci(4)와 fibonacci(3)을 계산하기 위해 재귀 호출

fibonacci(4) = fibonacci(3) + fibonacci(2)
memo 배열에는 아직 값이 저장되어 있지 않으므로 fibonacci(3)과 fibonacci(2)를 계산하기 위해 재귀 호출

fibonacci(3) = fibonacci(2) + fibonacci(1)
memo 배열에 값이 없으므로 fibonacci(2)와 fibonacci(1)를 계산하기 위해 재귀 호출

fibonacci(2)는 fibonacci(1) + fibonacci(0)
fibonacci(1)과 fibonacci(0)은 각각 1과 0을 반환하므로, 이를 계산하여 fibonacci(2)의 결과인 1을 memo[2]에 저장

이제 fibonacci(3)을 계산할 수 있습니다. fibonacci(3)은 fibonacci(2) + fibonacci(1)를 반환해야 하는데,
fibonacci(2)는 이미 memo[2]에서 1이라는 값을 찾을 수 있다. 따라서 fibonacci(3)의 결과인 2를 memo[3]에 저장

이제 fibonacci(4)을 계산할 수 있다. fibonacci(4)는 fibonacci(3) + fibonacci(2)를 반환해야 하는데,
fibonacci(3)과 fibonacci(2)는 이미 memo 배열에서 찾을 수 있다. 따라서 fibonacci(4)의 결과인 3을 memo[4]에 저장

마지막으로 fibonacci(5)를 계산
fibonacci(5)는 fibonacci(4) + fibonacci(3)를 반환해야 하는데, fibonacci(4)와 fibonacci(3)의 값은 memo 배열에 있다.
따라서 fibonacci(5)의 결과인 5를 memo[5]에 저장

https://school.programmers.co.kr/learn/courses/30/lessons/12945