프로그래머스 - 소인수분해

태그	Array.from(new Set(arr)) [...new Set(array)] arr.filter((item index) => arr.indexOf(item) === index)

자바스크립트
소인수분해
문제 설명
소인수분해란 어떤 수를 소수들의 곱으로 표현하는 것입니다. 예를 들어 12를 소인수 분해하면 2 * 2 * 3 으로 나타낼 수 있습니다. 따라서 12의 소인수는 2와 3입니다. 자연수 n이 매개변수로 주어질 때 n의 소인수를 오름차순으로 담은 배열을 return하도록 solution 함수를 완성해주세요.

제한사항
2 ≤ n ≤ 10,000

입출력 예 #1
n: 12 ,result: [2, 3]
12를 소인수분해하면 2 * 2 * 3 입니다. 따라서 [2, 3]을 return합니다.
입출력 예 #2
n: 17 ,result: [17]
17은 소수입니다. 따라서 [17]을 return 해야 합니다.
입출력 예 #3
n: 420 ,result: [2, 3, 5, 7]
420을 소인수분해하면 2 * 2 * 3 * 5 * 7 입니다. 따라서 [2, 3, 5, 7]을 return합니다.

```
function solution(n) {
    const result = [];
     for (let i = 2; i <= n; i++) {
       while (n % i === 0) {
         result.push(i);
         n /= i;

       }
     }
     return result.filter((item, index) => result.indexOf(item) === index);


   }
```

이 함수는 2에서 시작하여(2가 가장 작은 소수이므로) n까지 반복하여 작동합니다. 각 숫자 i에 대해 n이 더 이상 i로 나누어지지 않을 때까지 n을 i로 반복해서 나눕니다. 나눗셈은 n = n / i와 동일한 나눗셈 할당 연산자 /=를 사용하여 수행됩니다. n이 더 이상 i로 나누어지지 않으면 루프가 중단되고 다음 숫자 i + 1이 시도됩니다.
이 프로세스는 i가 n보다 클 때까지 계속되며, 이 시점에서 함수는 소인수 배열을 반환합니다.

JavaScript에서는 다음과 같은 방법으로 배열에서 중복 요소를 제거할 수 있습니다.

새로운 Set 객체를 생성한 후, 기존 배열을 순회하며 Set에 추가합니다. Set은 자동으로 중복 요소를 제거하기 때문입니다. 그런 다음 Array.from()을 사용하여 Set 객체를 다시 배열로 변환합니다.

```
function removeDuplicates(arr) {
  return Array.from(new Set(arr));
}
```

filter()를 사용하여 중복 요소를 제거할 수 있습니다. 이 방법은 중복을 제거하기 위해 추가적인 자료구조(예: Set)를 사용하지 않기 때문에 일반적으로 좀 더 빠르게 작동합니다.

```
 function removeDuplicates(arr) {
  return arr.filter((item, index) => arr.indexOf(item) === index);
}
```

스프레드 연산자를 사용하는 방법:
배열의 스프레드 연산자(...)를 사용하면 중복을 제거할 수 있습니다. 예를 들어:

```
const array = [1, 2, 3, 3, 4, 5, 5];
const uniqueArray = [...new Set(array)];
console.log(uniqueArray); // [1, 2, 3, 4, 5]
```

```
import java.util.*;
class Solution {
    public int[] solution(int n) {
       List<Integer> list = new ArrayList<>();
        int div = 2;
        while(n > 1){
            if(n%div==0){
                if(!list.contains(div))
                    list.add(div);
                n /= div;
                continue;
            }
            div++;
        }
        int[] answer = new int[list.size()];
        for(int i=0; i<answer.length; i++)
            answer[i] = list.get(i);
        return answer;
    }
}
```

List 인터페이스의 구현체인 ArrayList를 생성합니다. ArrayList는 순서가 있는 요소의 컬렉션이고, 중복을 허용합니다.
정수 div를 2로 초기화합니다. 이 값은 소인수분해 과정에서 사용할 수 있는 수의 범위를 제한합니다.
while문을 사용하여 n이 1보다 클 때까지 반복합니다.
n이 div의 배수인지 확인합니다. n이 div의 배수이면 list에 div가 포함되어 있지 않다면 list에 div를 추가합니다. 그리고 n을 div로 나누고, while문의 처음으로 이동합니다.
div의 값을 1 증가시킵니다.
while문이 종료되면, list의 크기만큼의 정수 배열인 answer를 생성합니다.
for문을 사용하여 answer 배열에 list의 요소를 할당합니다.
answer 배열을 반환합니다.

```
 import java.util.LinkedHashSet;

class Solution {
    public int[] solution(int n) {
        LinkedHashSet<Integer> primeNumbers = new LinkedHashSet<>();

        int i = 2;
        while (n != 0 && i <= n) {
            if (n % i == 0) {
                primeNumbers.add(i);
                n /= i;
            } else {
                i++;
            }
        }

//        System.out.println(primeNumbers);

        return primeNumbers.stream().mapToInt(Integer::intValue).toArray();
    }
}
```

LinkedHashSet 인터페이스의 구현체인 LinkedHashSet을 생성합니다. LinkedHashSet은 순서가 있는 요소의 컬렉션이고, 중복을 허용하지 않습니다.
정수 i를 2로 초기화합니다. 이 값은 소인수분해 과정에서 사용할 수 있는 수의 범위를 제한합니다.
while문을 사용하여 n이 0이 아니고, i가 n보다 작거나 같을 때까지 반복합니다.
n이 i의 배수인지 확인합니다. n이 i의 배수이면 primeNumbers 컬렉션에 i를 추가합니다. 그리고 n을 i로 나누고, while문의 처음으로 이동합니다.
i의 값을 1 증가시킵니다.
while문이 종료되면, primeNumbers 컬렉션을 스트림으로 변환한 후 int 타입으로 변환하고, int 배열로 변환하여 반환합니다.