[Java] 코딩테스트에서 알고리즘 구현에 자주 사용되는 함수 50선!

작성 : Zeromini

디스코드 채널(취업폭격기 제로미니 IT 취업 공부방) : https://discord.gg/yDWrPjqv

1. `System.out.println()`: 콘솔에 출력할 때 사용합니다.

```
System.out.println("Hello, World!");
```

2. `Arrays.sort()`: 배열을 정렬할 때 사용합니다.

```
int[] arr = {3, 2, 1};
Arrays.sort(arr);  // arr: {1, 2, 3}
```

3. `Arrays.binarySearch()`: 정렬된 배열에서 원소를 이진 탐색합니다.

```
int[] arr = {1, 2, 3};
int index = Arrays.binarySearch(arr, 2);  // index: 1
```

4. `Collections.sort()`: 컬렉션을 정렬합니다.

```
ArrayList<Integer> list = new ArrayList<>(Arrays.asList(3, 2, 1));
Collections.sort(list);  // list: {1, 2, 3}
```

5. `Math.max()`: 두 수 중에서 큰 수를 반환합니다.

```
int maxNum = Math.max(1, 2);  // maxNum: 2
```

6. `Math.min()`: 두 수 중에서 작은 수를 반환합니다.

```
int minNum = Math.min(1, 2);  // minNum: 1
```

7. `Math.pow()`: 거듭제곱을 계산합니다.

```
double result = Math.pow(2, 3);  // result: 8.0
```

8. `Math.sqrt()`: 제곱근을 계산합니다.

```
double result = Math.sqrt(4);  // result: 2.0
```

9. `Math.abs()`: 절대값을 계산합니다.

```
int absVal = Math.abs(-1);  // absVal: 1
```

10. `String.valueOf()`: 다른 타입을 문자열로 변환합니다.

```
String str = String.valueOf(123);  // str: "123"
```

11. `String.charAt()`: 문자열에서 특정 위치의 문자를 가져옵니다.

```
String str = "Hello, World!";
char ch = str.charAt(7);  // ch: 'W'
```

12. `String.substring()`: 문자열에서 하위 문자열을 가져옵니다.

```
String str = "Hello, World!";
String sub = str.substring(7, 12);  // sub: "World"
```

13. `String.split()`: 문자열을 특정 구분자로 분리하여 배열로 변환합니다.

```
String str = "Hello, World!";
String[] arr = str.split(", ");  // arr: {"Hello", "World!"}
```

14. `Integer.parseInt()`: 문자열을 정수로 변환합니다.

```
String str = "123";
int num = Integer.parseInt(str);  // num: 123
```

15. `Integer.toString()`: 정수를 문자열로 변환합니다.

```
int num = 123;
String str = Integer.toString(num);  // str: "123"
```

16. `List.add()`: 리스트에 요소를 추가합니다.

```
List<Integer> list = new ArrayList<>();
list.add(1);  // list: {1}
```

17. `List.get()`: 리스트의 특정 위치에 있는 요소를 반환합니다.

```
List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3));
int num = list.get(1);  // num: 2
```

18. `List.remove()`: 리스트에서 특정 위치의 요소를 제거합니다.

```
List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3));
list.remove(1);  // list: {1, 3}
```

19. `List.size()`: 리스트의 크기를 반환합니다.

```
List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3));
int size = list.size();  // size: 3
```

20. `HashMap.put()`: 해시맵에 키-값 쌍을 추가합니다.

```
HashMap<Integer, String> map = new HashMap<>();
map.put(1, "one");  // map: {1="one"}
```

21. `HashMap.get()`: 해시맵에서 키에 해당하는 값을 가져옵니다.

```
HashMap<Integer, String> map = new HashMap<>();
map.put(1, "one");
String val = map.get(1);  // val: "one"
```

22. `HashMap.remove()`: 해시맵에서 키에 해당하는 키-값 쌍을 제거합니다.

```
HashMap<Integer, String> map = new HashMap<>();
map.put(1, "one");
map.remove(1);  // map: {}
```

23. `HashMap.containsKey()`: 해시맵이 특정 키를 포함하고 있는지 확인합니다.

```
HashMap<Integer, String> map = new HashMap<>();
map.put(1, "one");
boolean contains = map.containsKey(1);  // contains: true
```

24. `HashMap.values()`: 해시맵에 있는 모든 값을 컬렉션으로 반환합니다.

```
HashMap<Integer, String> map = new HashMap<>();
map.put(1, "one");
Collection<String> values = map.values();  // values: {"one"}
```

25. `HashSet.add()`: 해시셋에 요소를 추가합니다.

```
HashSet<Integer> set = new HashSet<>();
set.add(1);  // set: {1}
```

26. `HashSet.contains()`: 해시셋이 특정 요소를 포함하고 있는지 확인합니다.

```
HashSet<Integer> set = new HashSet<>();
set.add(1);
boolean contains = set.contains(1);  // contains: true
```

27. `HashSet.remove()`: 해시셋에서 요소를 제거합니다.

```
HashSet<Integer> set = new HashSet<>();
set.add(1);
set.remove(1);  // set: {}
```

28. `HashSet.size()`: 해시셋의 크기를 반환합니다.

```
HashSet<Integer> set = new HashSet<>();
set.add(1);
int size = set.size();  // size: 1
```

29. `Queue.poll()`: 큐에서 가장 앞에 있는 요소를 제거하고 반환합니다.

```
Queue<Integer> queue = new LinkedList<>();
queue.add(1);
int num = queue.poll();  // num: 1
```

30. `Queue.offer()`: 큐에 요소를 추가합니다.

```
Queue<Integer> queue = new LinkedList<>();
queue.offer(1);  // queue: {1}
```

31. `Queue.peek()`: 큐에서 가장 앞에 있는 요소를 반환합니다(제거하지 않습니다).

```
Queue<Integer> queue = new LinkedList<>();
queue.add(1);
int num = queue.peek();  // num: 1
```

32. `Stack.push()`: 스택에 요소를 추가합니다.

```
Stack<Integer> stack = new Stack<>();
stack.push(1);  // stack: {1}
```

33. `Stack.pop()`: 스택에서 가장 위에 있는 요소를 제거하고 반환합니다.

```
Stack<Integer> stack = new Stack<>();
stack.push(1);
int num = stack.pop();  // num: 1
```

34. `Stack.peek()`: 스택에서 가장 위에 있는 요소를 반환합니다(제거하지 않습니다).

```
Stack<Integer> stack = new Stack<>();
stack.push(1);
int num = stack.peek();  // num: 1
```

35. `PriorityQueue.poll()`: 우선순위 큐에서 가장 앞에 있는 요소를 제거하고 반환합니다.

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
pq.add(3);
pq.add(1);
pq.add(2);
int num = pq.poll();  // num: 1
```

36. `PriorityQueue.offer()`: 우선순위 큐에 요소를 추가합니다.

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
pq.offer(1);  // pq: {1}
```

37. `PriorityQueue.peek()`: 우선순위 큐에서 가장 앞에 있는 요소를 반환합니다(제거하지 않습니다).

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
pq.add(1);
int num = pq.peek();  // num: 1
```

38. `StringBuilder.append()`: 문자열 빌더에 문자열을 추가합니다.

```
StringBuilder sb = new StringBuilder();
sb.append("Hello, ");  // sb: "Hello, "
sb.append("World!");  // sb: "Hello, World!"
```

39. `StringBuilder.toString()`: 문자열 빌더를 문자열로 변환합니다.

```
StringBuilder sb = new StringBuilder("Hello, World!");
String str = sb.toString();  // str: "Hello, World!"
```

40. `Character.isDigit()`: 특정 문자가 숫자인지 확인합니다.

```
boolean isDigit = Character.isDigit('1');  // isDigit: true
```

41. `Character.isLetter()`: 특정 문자가 알파벳인지 확인합니다.

```
boolean isLetter = Character.isLetter('a');  // isLetter: true
```

42. `Character.isUpperCase()`, `Character.isLowerCase()`: 특정 문자가 대문자 또는 소문자인지 확인합니다.

```
boolean isUpper = Character.isUpperCase('A');  // isUpper: true
boolean isLower = Character.isLowerCase('a');  // isLower: true
```

43. `Character.toUpperCase()`, `Character.toLowerCase()`: 문자를 대문자 또는 소문자로 변환합니다.

```
char upper = Character.toUpperCase('a');  // upper: 'A'
char lower = Character.toLowerCase('A');  // lower: 'a'
```

44. `String.equals()`: 두 문자열이 같은지 비교합니다.

```
String str1 = "Hello";
String str2 = "World";
boolean equals = str1.equals(str2);  // equals: false
```

45. `String.equalsIgnoreCase()`: 대소문자를 무시하고 두 문자열이 같은지 비교합니다.

```
String str1 = "Hello";
String str2 = "hello";
boolean equals = str1.equalsIgnoreCase(str2);  // equals: true
```

46. `Arrays.fill()`: 배열의 모든 요소를 특정 값으로 채웁니다.

```
int[] arr = new int[5];
Arrays.fill(arr, 1);  // arr: {1, 1, 1, 1, 1}
```

47. `Collections.fill()`: 컬렉션의 모든 요소를 특정 값으로 채웁니다.

```
List<Integer> list = new ArrayList<>(Arrays.asList(0, 0, 0, 0, 0));
Collections.fill(list, 1);  // list: {1, 1, 1, 1, 1}
```

48. `Math.random()`: 0.0과 1.0 사이의 랜덤한 double 값을 반환합니다.

```
double random = Math.random();  // 0.0 <= random < 1.0
```

49. `Math.round()`: 가장 가까운 정수로 반올림합니다.

```
long rounded = Math.round(1.5);  // rounded: 2
```

50. `Math.ceil()`, `Math.floor()`: 숫자를 올림 또는 내림합니다.

```
double ceil = Math.ceil(1.1);
```