1. (Factorial 계산: 반복) 다음은 n!을 반복적으로 계산하는 코드이다. 아래 코드를 입력하고 실행하면서 반복에 기반한 팩토리얼 구현을 익히시오.

```
public class Test {
	public static void main(String[] args) {
		int	n=10;
		System.out.println(fact(n));
	}
	private static double fact(int n) {
		double	v=1;
		for (int i = 2; i <= n; i++) v*=i;
		return v;
	}
}
```

2. (Factorial 계산: 재귀) 다음은 n!을 재귀적으로 계산하는 코드이다. 아래 코드를 입력하고 실행하면서 재귀(recursion)에 기반한 팩토리얼 구현을 익히시오. 점화식은 다음과 같다.

- 점화식

n=0 : fact(n)=1
n>=1: fact(n)=n*fact(n-1)

```
public class Test {
	public static void main(String[] args) {
		int	n=10;
		System.out.println(fact(n));
	}
	private static double fact(int n) {
		if(n==0) return 1;
		return n*fact(n-1);
	}
}
```

3. (Power 함수: 반복) 다음은 m의 n승을 계산하는 코드로 m을 n회 반복적으로 곱하여 m의 n승을 계산한다. 아래 코드를 입력하고 실행하면서 반복 기반 power 함수 구현을 익히시오.

```
public class Test {
	public static void main(String[] args) {
		int	m=2, n=10;
		System.out.println(power(m,n));
	}
	private static long power(int m, int n) {
		long v=1;
		for (int i = 0; i < n; i++) v*=m;
		return v;
	}
}
```

4. (Power 함수: 재귀) 다음은 m의 n승을 재귀적으로 계산하는 코드이다. 아래 코드를 입력하고 실행하면서 재귀에 기반한 power 함수 구현을 익히시오. 점화식은 다음과 같다.

- 점화식

n=0: power(m,n)=1
n>0: power(m,n)=m*power(m,n-1)

```
public class Test {
	public static void main(String[] args) {
		int	m=2, n=60;
		System.out.println(power(m,n));
	}
	private static long power(long m, long n) {
		if(n==0) return 1;
		return m*power(m,n-1);
	}
}
```

5. (Power 함수: 재귀) 다음은 m의 n승을 재귀적으로 계산하는 보다 효율적인 코드이다. 아래 코드를 입력하고 실행하면서 재귀에 기반한 power 함수 구현을 익히시오. 점화식은 다음과 같다. (참고: C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.)

- 점화식

n=0: power(m,n)=1
n>0 AND n이 짝수: power(m,n)=power(m*m,n/2)
n>0 AND n이 홀수: power(m,n)=power(m*m,n/2)*m

```
public class Test {
	public static void main(String[] args) {
		System.out.println(power(2,60));
	}
	private static long power(long m, long n) {
		if(n==0) return 1;
		return power(m*m,n/2)*(n%2==0? 1:m);
	}
}
```

6. (**실습**: 재귀연습, 배열 총합) 다음은 정수 배열 n의 총합을 재귀적으로 계산하는 점화식과 코드를 보인 것이다. 아래 점화식을 참고하여 재귀함수 sum()을 완성하시오. 아래 코드에서 재귀함수 호출은 sum(n, n.length-1)로부터 시작하고 있다.

- 점화식

i=0: sum(n,i)=n[0]
i>0: sum(n,i)=n[i]+sum(n,i-1)

```
public class Test {
	public static void main(String[] args) {
		int	n[]={7,2,8,4,1};
		System.out.println(sum(n, n.length-1));
	}
	private static int sum(int[] n, int i) {
	}
}
```

7. (**실습**: 재귀연습, 배열 총합) 다음은 정수 배열 n의 총합을 sum(n,0)의 호출로부터 시작하여 재귀적으로 계산하는 코드이다. 점화식을 작성한 후 점화식에 기반하여 아래 코드의 재귀함수 sum()을 완성하시오.

```
public class Test {
	public static void main(String[] args) {
		int	n[]={1,2,3,4,5};
		System.out.println(sum(n, 0));
	}
	private static int sum(int[] n, int i) {
	}
}
```

8. (**실습**: 재귀연습, 배열 최대값) 다음은 정수 배열의 최대값을 재귀적으로 계산하는 점화식과 코드의 일부분이다. 아래 코드를 완성하시오.

- 점화식

i=0: max(n,i)=n[i]

i>0: max(n,i)=n[i]와 max(n,i-1) 중 큰 값

```
public class Test {
	public static void main(String[] args) {
		int		n[]={5,1,8,3,2};
		System.out.println(max(n, n.length-1));
	}
	private static int max(int[] n, int i) {
	}
}
```

9. (**실습**: 재귀연습, 배열 최대값) 다음 재귀적 접근에 따라 배열의 최대값을 재귀적으로 계산하는 아래 코드를 완성하시오..

- 재귀적 접근: 배열의 최대값은 왼쪽 반의 최대값과 오른쪽 반의 최대값 중 큰 값

```
public class Test {
	public static void main(String[] args) {
		int		n[]={5,1,8,3,2};
		System.out.println(sum(n, 0, n.length-1));
	}
	private static long sum(int[] n, int low, int high) {
	}
}
```

10. (**실습**: 대문자변환) 다음 코드는 문자 배열 내 영어 소문자를 대문자로 변환하기 위해 재귀함수 toUpper()를 호출하고 있다. 아래 코드를 완성하시오. (아스키 코드표 참고: https://en.wikipedia.org/wiki/ASCII).

```
public class Test {
	public static void main(String[] args) {
			String	s="South, Korea 안녕";
			char	v[]=s.toCharArray();
			toUpper(v, 0);
			System.out.println(v);
	}
	private static void toUpper(char[] v, int i) {
	}
}
```

11. (**실습**: 대문자출현횟수) 다음 코드는 문자 배열 내 영어 대문자 출현 횟수를 계산하기 위해 countUpper()를 호출하고 있다. 아래 코드를 완성하시오. (아스키 코드표 참고: https://en.wikipedia.org/wiki/ASCII)

```
public class Test {
	public static void main(String[] args) {
			String	s="South, Korea";
			char	v[]=s.toCharArray();
			System.out.println(countUpper(v, 0));
	}
	private static int countUpper(char[] v, int i) {
	}
}
```

12. (**실습**: 재귀 기반 palindrome 검사) 다음은 문자열의 내용이 회문(palindrome)인지 여부를 검사하기 위해 재귀함수 palindrome(s, 0)을 호출하고 있다. 아래 재귀함수 palindrome을 완성하시오.

```
public class Test {
	public static void main(String[] args) {
		String	s="ABCDEDCBA";
		System.out.println(palindrome(s, 0));
	}
	private static boolean palindrome(String s, int i) {
	}
}
```

13. (**실습**: 재귀 기반 배열 reverse) 다음은 배열 n의 내용을 역순으로 변경하기 위해 재귀함수 reverse(n, 0)을 호출하고 있다. 아래 재귀함수 reverse를 완성하시오.

- 실행결과: [8, 7, 6, 4, 5, 3, 2, 1]

```
public class Test {
	public static void main(String[] args) {
		int		n[]={1,2,3,4,5,6,7,8};
		reverse(n, 0);
		System.out.println(Arrays.toString(n));
	}
	private static void reverse(int[] n, int i) {
	}
}
```

14. (**실습**: 최대공약수) 다음은 두 수의 최대공약수(GCD, Greatest Common Divisor)를 계산하는 유클리드 호제법을 보인 것이다. 아래 코드의 함수 gcd를 재귀 및 비재귀 방식으로 각각 구현하시오.

- 유클리드 호제법:

b=0: gcd(a,b)=a
b>0: gcd(a,b)=gcd(b,a%b)

```
public class Test {
	public static void main(String[] args) {
		System.out.println(gcd(64,48));
	}
	private static int gcd(int a, int b) {
	}
}
```

15. (**실습**: DFS, Depth First Search) 아래 예시와 같이 표현된 2차원 평면 내 물웅덩이 개수를 구하는 아래 코드를 완성하시오. 아래 예시에서 문자 '0'은 마른 위치이고 문자 '1'은 젖은 위치를 표현한다. 하나의 물웅덩이는 상하, 좌우, 대각선 위치의 연속된 1로 이어진 공간으로 정의한다. 아래 예시에서 물웅덩이는 3개이다. (Reference: "프로그래밍 콘테스트 챌린징", Akiba 등 저, 박건태, 김승엽 역, 로드북, 2011. pp.47-49)

01111
00101
00001
10000
10111

➔ 물웅덩이 개수: 3개

- **실습** #1: '0', '1'로 표현된 R x C 크기 평면 내 물웅덩이 개수를 구하는 아래 코드를 완성하시오. 해결 방법은 다음과 같다. 평면 상의 각 위치를 검사하면서 젖은 위치인 경우 해당 위치가 포함된 하나의 물웅덩이를 탐색하는 함수 dfs를 호출한 후 물웅덩이 개수를 1 증가시킨다. dfs 함수는 젖은 위치 좌표를 파라미터로 전달받아 해당 위치를 마른 위치로 변경한 후 그 위치의 상하좌우대각선 방향 각 인접 위치를 검사하여 젖은 위치가 있으면 dfs 함수를 재귀호출한다.
- **실습** #2: 물웅덩이 각각의 크기를 출력하는 코드를 추가하시오. 특정 물웅덩이의 크기는 해당 물웅덩이를 구성하는 젖은 위치의 개수로 정의한다. 위 예의 경우 3개 물웅덩이 각각의 크기에 해당하는 7, 2, 3이 출력되어야 한다.

```
public class Test {
	public static void main(String[] args) {
		int		R=5, C=5, count=0;
		char		ground[][]=new char[R][C];
		Random	random=new Random();
		for (int i = 0; i < ground.length; i++){ // 임의 개수의 물웅덩이 포함 평면 생성
			for (int j = 0; j < ground[i].length; j++) ground[i][j]=(random.nextInt(3)==0)? '1' : '0';
		}
		for (int i = 0; i < ground.length; i++){ // 평면 출력
			for (int j = 0; j < ground[i].length; j++) System.out.print(ground[i][j]);
			System.out.println();
		}

		for (int i = 0; i < ground.length; i++) {
			for (int j = 0; j < ground[i].length; j++){
				if(ground[i][j]=='1'){ dfs(ground, R, C, i, j); count++; }
			}
		}
		System.out.println(count);
	}

	private static void dfs(char[][] ground, int R, int C, int i, int j) {
	}
}
```

16. (피보나치수열: 비효율적 재귀호출 구현) 다음은 피보나치수열의 n(n>=0)번째 수를 반환하는 함수 fibo를 재귀함수로 구현한 코드이다. 그러나 이 코드에서는 fibo(n)을 얻기 위해 fibo(n-1)과 fibo(n-2)를 호출하게 되고, fibo(n-1)을 계산하기 위해 fibo(n-3)와 fibo(n-2)를 호출하는 방식을 사용하므로, fibo(n) 계산을 위해 호출하는 fibo(n-2)를 fibo(n-1)의 계산을 위해 중복 호출하는 비효율이 발생한다. 아래 코드의 30을 50으로 수정 실행하여 소요 시간 지연을 확인하시오.

- 점화식:
n<=1: fibo(n)=n
n>=2: fibo(n)=fibo(n-2)+fibo(n-1)

```
public class Test {
	public static void main(String[] args) {
		System.out.println(fibo(30));
	}
	private static long fibo(int n) {
		if(n<2) return n;
		return fibo(n-2)+fibo(n-1);
	}
}
```

17. (피보나치수열: 메모이제이션 기반 재귀호출) 다음은 피보나치수열의 n(n>=0)번째 수를 반환하는 함수 fibo를 재귀함수로 구현한 코드이다. 그러나 이전의 비효율적 재귀 호출 구현과 달리 계산 시간을 줄이기 위해 이전에 계산되지 않은 피보나치 수는 계산 후 반환 전에 저장해 두고, 이전에 계산된 피보나치 수의 경우 저장된 값을 단순히 반환함으로써 동일 피보나치 수에 대한 중복 계산을 피하고 있다.

- 실행결과: 12586269025

```
public class Test {
	public static void main(String[] args) {
		System.out.println(fibo(50));
	}
	private static long fibo(int n) {
		long memo[]=new long[n+1];
		return fiboMemo(n, memo);
	}
	private static long fiboMemo(int n, long[] memo) {
		if(memo[n]>0) return memo[n];
		if(n<2) memo[n]=n;
		else memo[n]=fiboMemo(n-2, memo)+fiboMemo(n-1, memo);
		return memo[n];
	}
}
```

18. (**실습**: 피보나치수열) 피보나치수열의 n(n>=0)번째 수를 반환하는 함수 fibo를 반복적으로 구현하는 아래 코드를 완성하시오.

```
public class Test {
	public static void main(String[] args) {
		System.out.println(fibo(50));
	}
	private static long fibo(int n) {
	}
}
```

19. (**실습**: 자바 swap) 다음은 swap 함수를 통해 배열 내 두 정수를 교환하는 예시 코드이다. 교환이 정상적으로 수행되도록 아래 코드의 swap 함수 정의문 및 호출문을 수정하시오.

```
public class Test {
	public static void main(String[] args) {
		int	n[]={3,4};
		System.out.println(Arrays.toString(n));
		swap(n[0],n[1]);
		System.out.println(Arrays.toString(n));
	}
	private static void swap(int i, int j) {
		int	temp=i;
		i=j;
		j=temp;
	}
}
```

20. (순열) 집합 {0,1,2}에 대한 순열의 모든 결과는 012, 021, 102, 120, 210, 201이다. 다음은 순열 생성 코드이다. 아래 코드를 입력하고 실행하면서 순열 생성 코드 구현법을 익히시오.

```
public class Test {
	public static void main(String[] args) {
		int	n[]=new int[3];
		for (int i = 0; i < n.length; i++) n[i]=i;
		perm(n, 0);
	}
	private static void perm(int[] n, int i) {
		if(i==n.length){
			System.out.println(Arrays.toString(n));
			return;
		}
		for (int j = i; j < n.length; j++) {
			int temp=n[i]; n[i]=n[j]; n[j]=temp; // swap
			perm(n, i+1);
			temp=n[i]; n[i]=n[j]; n[j]=temp; // swap
		}
	}
}
```

21. (**실습**) {서울, 대구, 부산}의 모든 순열을 출력하는 아래 코드를 완성하시오.

- 실행결과:

```
서울 대구 부산
서울 부산 대구
대구 서울 부산
대구 부산 서울
부산 대구 서울
부산 서울 대구
```

```
public class Test {
	public static void main(String[] args) {
		String	s[]={"서울", "대구", "부산"};
	}
}
```

22. (**실습**) 20개 도시가 있으며 각 도시 쌍은 서로 연결된 도로가 있을 때, 20개 도시를 방문하는 각 경로에 대해 어떤 처리를 한다고 가정하자. 20개 도시를 순차 방문하는 모든 가능한 경로는 총 몇 개인가?

- n!의 크기를 계산하는 함수 long fact(int n)를 작성하고, fact 함수를 이용하여 20!을 계산하여 출력해 보시오.
- 20!개의 경로를 모두 처리하는 시간을 초,분,시간,일,년 단위로 출력하시오. (1초에 1G개의 경로를 처리한다고 가정하시오.)

23. (**실습**) 다음은 3비트 이진수의 모든 가능한 숫자열을 생성하는 코드이다. 아래 코드를 입력하고 실행하면서 비트열 생성 코드 구현법을 익히시오. (참고: 다양한 예제로 학습하는 데이터 구조와 알고리즘 문제 해결법부터 개선법까지. 나라심하 카루만치 저. 전계도 역. 인사이트(insight). 2014.)

- 실행결과: 000,001,010,011,100,101,110,111

- **실습** #1: 다음과 같이 4비트의 모든 가능한 비트열을 생성하도록 아래 코드를 수정하시오.
  0000,0001,0010,0011,0100,0101,0110,0111,1000,1001,1010,1011,1100,1101,1110,1111
- **실습** #2: 다음과 같은 비트열 순으로 결과가 출력되도록 아래 코드를 수정하시오. (비트열이 생성되는 순서가 다름)
  111,110,101,100,011,010,001,000
- **실습** #3: 다음과 같이 문자집합 {A,B}로부터 생성 가능한 길이 3의 모든 가능한 문자열을 출력하는 코드를 작성하시오.
  AAA,AAB,ABA,ABB,BAA,BAB,BBA,BBB
- **실습** #4: 2자리 3진수(0,1,2)의 모든 가능한 표현을 출력하는 코드를 작성하시오.
  00,01,02,10,11,12,20,21,22
- **실습** #5: 2자리 8진수(0,1,2,3,4,5,6,7)의 모든 가능한 표현을 출력하는 코드를 작성하시오.
  00,01,02,03,04,05,06,07,10,11,12,13,14,15,16,17,20,21,22,23,24,25,26,27,30,31,32,33,34,35,36,37,40,41,42,43,44,45,46,47,50,51,52,53,54,55,56,57,60,61,62,63,64,65,66,67,70,71,72,73,74,75,76,77
- **실습** #6: {0,1,2}의 모든 부분집합 {},{1},{2},{0,1},{0,2},{1,2},{0,1,2}을 출력하는 코드를 작성하시오.

```
public class Test {
	public static void main(String[] args) {
		int	v[]=new int[3];
		bitString(v, 0);
	}
	private static void bitString(int[] v, int i) {
		if(i==v.length){
			System.out.println(Arrays.toString(v));
			return;
		}
		v[i]=0;
		bitString(v, i+1);
		v[i]=1;
		bitString(v, i+1);
	}
}
```

24. (**실습**: subset sum problem - 부분집합의 합 모든 해 탐색) 다음은 입력으로 주어진 n개 정수 집합의 부분집합에 속한 정수들의 합이 특정한 값이 될 수 있는 경우 그러한 모든 부분집합들을 출력하는 코드이다. 이 코드를 완성하시오. (참조: https://en.wikipedia.org/wiki/Subset_sum_problem)

- 입력(sum=21): 1,3,4,5,8,9
- 출력:

4,8,9
3,4,5,9
1,3,8,9
1,3,4,5,8

- 구현 방법: 입력 정수 배열과 같은 크기 N의 정수 배열 subset을 만든 후, 배열 subset이 모든 가능한 N-자리 이진비트열 표현을 순차 생성하는 코드를 작성한 다음, 배열 subset이 표현하는 이진비트열의 각 비트값 0,1을 입력 배열 n 내 대응하는 각 원소를 부분집합에 포함할 것인지 포함하지 않을 것인지로 해석하여 이후 부분집합의 합 검사를 진행한다. 예를 들어 subset 배열이 {0,0,1,0,1,1}인 경우 이를 입력 배열 n에서 n[2],n[4],n[5]에 해당하는 정수들의 부분집합 {4,8,9}을 표현하는 것으로 해석하여 4+8+9=21인지 여부를 검사한다.

```
public class Test {
	static	int	sum=21; // 부분집합 내 정수들의 합이 되어야 할 값
	static	int	n[]={1,3,4,5,8,9}; // 입력: n개 정수
	public static void main(String[] args) {
		int	subset[]=new int[n.length];
		subsetSum(subset, 0);
	}
	private static void subsetSum(int[] subset, int i) {
		if(i==n.length){
			// 부분집합 내 정수의 합이 sum이 되는지 검사 후 부분집합 출력
			return;
		}
		subset[i]=0;
		subsetSum(subset, i+1);
		subset[i]=1;
		subsetSum(subset, i+1);
	}
}
```

25. (subset sum problem - 부분집합의 합 모든 해 탐색: 누계 기반 총합 검사) 다음은 입력으로 주어진 n개 정수 집합의 부분집합에 속한 정수들의 합이 특정한 값이 될 수 있는 경우 그러한 모든 부분집합들을 출력하는 코드이다. 이전의 구현에서는 하나의 부분집합이 결정된 시점에 부분집합에 속하는 정수들의 합 조건 검사를 수행하였다. 아래 구현에서는 최초 0의 누계 값(partialSum=0)에서 출발하여 하나의 부분집합에 속하는 각 정수가 선택되는 시점에 해당 정수를 partialSum에 누계해 나가면서 그 부분집합이 결정된 시점에는 최종 누계 값(partialSum)이 도달되어야 할 총합과 일치하는지 여부를 검사하는 방법이 사용되었다.

- 입력(sum=21): 1,3,4,5,8,9
- 출력:

```
4,8,9
3,4,5,9
1,3,8,9
1,3,4,5,8
```

```
public class Test {
	static	int	sum=21;
	static	int	n[]={1,3,4,5,8,9};
	public static void main(String[] args) {
		int	subset[]=new int[n.length];
		subsetSum(subset, 0, 0); // 최초 누계 값 0에서 출발
	}
	private static void subsetSum(int[] subset, int i, int partialSum) {
		if(i==n.length){
			if(partialSum==sum){
				for (int j = 0; j < subset.length; j++) if(subset[j]==1) System.out.print(n[j]+" ");
				System.out.println();
			}
			return;
		}
		subset[i]=0;
		subsetSum(subset, i+1, partialSum);
		subset[i]=1;
		subsetSum(subset, i+1, partialSum+n[i]);
	}
}
```

26. (**실습**: subset sum problem - 부분집합의 합 모든 해 탐색: 차감 기반 총합 검사) 이전 문제에서는 부분집합의 총합을 계산하기 위해 누계 기반 방법을 사용하였다. 누계 기반 방법 대신, 도달되어야 할 총합(partialSum=K)에서 출발하여 하나의 부분집합에 속하는 정수들이 선택될 때마다 해당 정수를 partialSum에서 차감하는 방식으로 진행하여 하나의 부분집합이 결정되는 시점에는 partialSum이 0이 되는지를 검사함으로써 부분집합의 총합 문제를 해결할 수 있다. 이 방식으로 아래 코드를 완성하시오.

```
public class Test {
	static	int	sum=21;
	static	int	n[]={1,3,4,5,8,9};
	public static void main(String[] args) {
		int	subset[]=new int[n.length];
		subsetSum(subset, 0, sum); // 최초 도달되어야 할 총합 sum에서 출발
	}
	private static void subsetSum(int[] subset, int i, int partialSum) {

	}
}
```

27. (**실습**: subset sum problem - 부분집합의 합 해 존재 여부 결정: 누계 기반) 다음은 입력으로 주어진 n개 정수 집합의 모든 부분집합 중 (부분집합에 속한) 정수의 합이 특정한 값이 되는 부분집합이 존재하는 경우 true를 그렇지 않은 경우 false를 출력하는 코드이다. 아래 코드는 최초 누계 값 0에서 출발하여 부분집합에 속하는 정수가 결정될 때마다 해당 정수를 누계 값에 합산하는 식으로 진행하여 하나의 부분집합이 결정되는 시점에 누계 값이 도달되어야 할 총합과 일치하는지 여부를 검사하는 방법을 사용하였다. 아래 코드 중 두 번째 미완성코드를 이전 문제에서 설명한 차감 기반 방법으로 구현해 보시오.

- 입력(sum=21): 1,3,4,5,8,9
- 출력: true

```
public class Test {
	static	int	sum=21;
	public static void main(String[] args) {
		int	n[]={1,3,4,5,8,9};
		System.out.println(subsetSum(n, 0, 0));
	}
	private static boolean subsetSum(int[] n, int i, int partialSum) {
		if(i==n.length) return partialSum==sum;
		return subsetSum(n, i+1, partialSum) || subsetSum(n, i+1, partialSum+n[i]);
	}
}
```

```
public class Test { // 미완성 코드
	static	int	sum=21;
	public static void main(String[] args) {
		int	n[]={1,3,4,5,8,9};
		System.out.println(subsetSum(n, 0, sum));
	}
	private static boolean subsetSum(int[] n, int i, int partialSum) {
	}
}
```

28. (**실습**: 조합) 다음은 크기 N의 집합에서 K개 원소를 선택하는 모든 조합을 생성하는 코드이다. 아래 코드를 입력하고 실행하면서 조합 생성 코드 구현법을 익히시오.

- 실행결과: 10100은 5개 원소 중 첫번째, 세번째 원소를 선택한 것을 의미한다.

```
11000
10100
10010
10001
01100
01010
01001
00110
00101
00011
```

- **실습**: 아래 코드는 중복을 허용하지 않는 조합을 다룬다. 다음과 같이 중복을 허용하는 조합을 생성하도록 아래 코드를 수정하시오. 20000은 첫 번째 원소를 중복하여 2번 선택한 것을 의미한다.

```
20000
11000
10100
10010
10001
02000
01100
01010
01001
00200
00110
00101
00020
00011
00002
```

```
public class Test {
	public static void main(String[] args) {
		int		N=5, K=2;
		int		v[]=new int[N];
		genCombinations(N, K, 0, v, 0);
	}
	private static void genCombinations(int N, int K, int startIndex, int v[], int len) {
		if(len==K) {
			String	s="";
			for (int i = 0; i < v.length; i++) s+=v[i];
			System.out.println(s);
			return;
		}
		for (int i = startIndex; i < N; i++) {
			v[i]=1; genCombinations(N, K, i+1, v, len+1); v[i]=0;
		}
	}
}
```

References

- C로 쓴 자료구조론 (Fundamentals of Data Structures in C, Horowitz et al.). 이석호 역. 사이텍 미디어. 1993.
- 쉽게 배우는 알고리즘: 관계 중심의 사고법. 문병로. 한빛아카데미. 2013.
- C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.
- 김윤명. (2008). 뇌를 자극하는 Java 프로그래밍. 한빛미디어.
- 남궁성. 자바의 정석. 도우출판.
- 김윤명. (2010). 뇌를 자극하는 JSP & Servlet. 한빛미디어.