```java
package exercicio1;
public class Exercicio1 {

   /*
   1) Escreva uma função recursiva que calcule o número de grupos distintos com k
   pessoas que podem ser formados a partir de um conjunto de n pessoas.
   A definição abaixo da função Comb(n, k) define as regras: (Valor 1,0)


   Comb(n,k)={n                                    se K =1
             {1                                   se K = n
             {Comb(n-1, k-1) + Comb(n-1,k)          se 1 < k < n

    */


   public static void main(String[] args) {

      System.out.println(comb(4,3)); // Imprimir resultado

   }

   public static int comb(int n, int k){

      if(k == 1 ){  // Caso o parâmetro K seja igual a 1
         return n;
// o retorno será n, ou seja, um grupo com 1 pessoa pode ser formado N vezes
      }else { // Caso contrário
         if(k == n){
 // Com K igual a N, seria como um grupo com valor X de pessoas sendo formado X
// vezes, ou seja, valores iguais
            return 1; // Então deve ser retornado 1,
         }else{ // Caso contrário
            if( 1 < k && k < n){ // Se os valores de K nao forem igual a 1 ou a N
            return (comb((n-1),(k-1)) + comb((n-1), k));
// Retorna então a soma entre a mesma funçao, sendo
// funcao(os dois mesmos valores -1 ) + funcao(n-1, k)
// assim com essa recursividade é possivel calcular o numero de grupos distintos com k
//pessoas que podem ser formados a
// partir de um conjunto n de pessoas
            }
         }
      }
      return 0; // Caso os parametros nao entrarem na lógica acima, retorne 0
   }
```

```
}
```

2) Mostre, através de teste de mesa, o resultado das seguintes funções: (Valor 1,0)

```
public int funcao(int n) {
   if (n == 0) {
         System.out.println(“Zero”);
  return 0;
 }        else {
         System.out.println(n);
         System.out.println(n);
         return funcao(n-1);
          }
}
```

a) Considere as entradas:
1) funcao(0);
2) funcao(1);
3) funcao(5);

==========

1) funcao(0):

se 0 = 0 --> true
    imprimir("Zero")
    retornar 0

resultado = 0

2) funcao(1)

se 1 = 0 --> false
então
    imprimir(1)
    imprimir(1)
retornar funcao(1-1)

se 0 = 0 --> true
    imprimir("Zero")
    retornar 0

resultado = 0

3) funcao(5)

se 5 = 0 --> false
entao
    imprimir(5)
    imprimir(5)
    retornar funcao(5-1)

se 4 = 0 --> false
entao
    imprimir(4)
    imprimir(4)
    retornar funcao(4-1)

se 3 = 0 --> false
entao
    imprimir(3)
    imprimir(3)
    retornar funcao(3-1)

se 2 = 0 --> false
entao
    imprimir(2)
    imprimir(2)
    retornar funcao(2-1)

se 1 = 0 --> false
então
    imprimir(1)
    imprimir(1)
retornar funcao(1-1)

se 0 = 0 --> true
    imprimir("Zero")
    retornar 0

resultado = 0

3) Analise o pior caso do método quanto a complexidade de tempo. Ache o T(n) e o O(n). (Valor 1,5)

```
Linha 1:        int func(int n) {
Linha 2:                int i, r, j;
Linha 3:                r = 1;
Linha 4:                i = 1;
Linha 5:                j = 1;
Linha 6:                        while (i <= n) {
Linha 7:                                while (j <= n){
Linha 8:                                        r = r*n; i++;
Linha 9:                                }
Linha 10:                       }
Linha 11:               return r;
Linha 12:       }
```

==========

Ache T(n):

Linha 2 Declaração de 3 variáveis, a linha será executada uma vez
sendo assim, T(n) = 1

Linha 3,4,5 há 3 Atribuições de valores, cada um será executado uma vez
Logo, T(n) = (1) + 1 + 1 + 1

Linha 6, será executada apenas 1 vez devido ao looping na linha sequente.
Ficando assim: T(n) = (1) + (1 + 1 + 1) + (1)

Linha 7, será executada para cada valor de j, de 1 a n + 1
Logo, será executada n + 1 vezes, porém não há incremento para J, sendo
Executada indeterminada vezes.
Ficando assim: T(n) = (1)+ (1 + 1 + 1) + (1) + (n + 1) -> (infinito)

Linha 8, há **duas** atribuições de valores, a linha será executada até que o valor de j seja <= n e i <= n, ou seja n*n, porém não há incremento para j, o que deixa a linha sendo executada indeterminadamente,.
Ficando assim: T(n) = (1)+ (1 + 1 + 1) + (1) + (n + 1) + (n*n) + (n*n) -> (infinito)

Linha 11, não será executada pois não há como sair do looping
Logo, T(n) = (1)+ (1 + 1 + 1) + (1) + (n + 1) + (n*n) + (n*n)+ (0)

No fim, teremos o resultado:

Resposta -> T(n)=infinito

Caso fosse uma função com incremento no J:

$T(n) = 2n^2 + n + 6$

Ache O(n):

Resposta -> O(n) = 0 (infinito)

Caso fosse uma função com incremento no J:

$O(n) = n^2$

4)Faça o passo a passo da insertion sort no vetor abaixo. Mostre os ponteiros e descreva rapidamente o passo que está realizando. (Valor 0,5)

[26 | 32 | 2 | 45 | 15 | 68 | 34]

Varredura 01:

[26 | 32 | 2 | 45 | 15 | 68 | 34]

Vetor[0] > Vetor[1]
26 é maior que 32 ? -> False -> Mantém posição

Varredura 02:

[26 | 32 | 2 | 45 | 15 | 68 | 34]

Vetor[1] > Vetor[2]
32 é maior que 2 -> True -> Troca posição

[26 | 2 | 32 | 45 | 15 | 68 | 34]

Varredura 03:

[26 | 2 | 32 | 45 | 15 | 68 | 34]

Vetor[0] > Vetor[1]
26 é maior que 2 -> True -> Troca posição

[2 | 26 | 32 | 45 | 15 | 68 | 34]

Varredura 04:

[2 | 26 | 32 | 45 | 15 | 68 | 34]

Vetor[2] > Vetor[3]
32 é maior que é 45-> False -> Mantém posição

Varredura 05:

[2 | 26 | 32 | 45 | 15 | 68 | 34]

Vetor[3] > Vetor[4]
45 é maior que 15 -> True -> Troca posição

[2 | 26 | 32 | 15 | 45 | 68 | 34]

Varredura 06:

[2 | 26 | 32 | 15 | 45 | 68 | 34]

Vetor[2] > Vetor[3]
32 é maior que 15 -> True -> Troca posição

[2 | 26 | 15 | 32 | 45 | 68 | 34]

Varredura 07:

[2 | 26 | 15 | 32 | 45 | 68 | 34]

Vetor[1] > Vetor[2]
26 é maior que 15 -> True -> Troca posição

[2 | 15 | 26 | 32 | 45 | 68 | 34]

Varredura 08:

[2 | 15 | 26 | 32 | 45 | 68 | 34]

Vetor[0] > Vetor[1]
2 é maior que 15 -> False -> Mantém

Varredura 09:

[2 | 15 | 26 | 32 | 45 | 68 | 34]

Vetor[4] > Vetor[5]
45 é maior 68 -> False -> Mantém posição

Varredura 10:

[2 | 15 | 26 | 32 | 45 | 68 | 34]

Vetor[5] > Vetor[6]
68 é maior 34 -> True -> Troca posição

[2 | 15 | 26 | 32 | 45 | 34 | 68]

Varredura 11:

[2 | 15 | 26 | 32 | 45 | 34 | 68]

Vetor[4] > Vetor[5]
45 é maior que 34 -> True -> Troca posição

[2 | 15 | 26 | 32 | 34 | 45 | 68]

Varredura 12:

[2 | 15 | 26 | 32 | 34 | 45 | 68]

Vetor[3] > Vetor[4]
32 é maior que 34 -> False -> Mantém

Array Final -> [2 | 15 | 26 | 32 | 34 | 45 | 68]

```java
package exercicio5;

/**
 5) Implemente o método de busca binária em java ou em C e teste com um vetor de tamanho
10 com qualquer número no vetor. (Valor 1,0)
 * @author munizera
 */
public class Exercicio5 {

   public static void main(String[] args) {

      int vetor[] = {0,12,24,37,48,56,62,79,85,93}; // Vetor de tamanho 10

      int num = 56; // Numero na qual estou procurando

      int resultado = buscaBinaria(num, vetor); // Armazenando o resultado para melhor visualização

      System.out.println("Encontrando o numero " + num + " no vetor");

      if(resultado > 0){ // Se caso for encontrado, imprimir a posição na qual foi encontrada
         System.out.println("O numero "+ num + " foi encontrado na posição: " + resultado);
      }else{ // Se caso nao encontrado avisa ao usuário
         System.out.println("Não foi encontrado o numero " + num);
      }

   }

   public static int buscaBinaria( int num, int vetor[]){
// A função recebe um num, na qual é o que o usuário deseja buscar
// E recebe um vetor, na qual é o lugar onde iremos procurar o num

      int inicio = 0;  // Posição Inicial Padrão do vetor
      int fim = vetor.length - 1; // Posição final do vetor, no caso 10 - 1, pois o inicio é igual a 0

      while( inicio <= fim ){
         int centro = (inicio + fim) / 2; // Lógica básica para determinar o centro do vetor
         if(num == vetor[centro]){
            return centro; // Quando o num que buscarmos estiver no centro, ele será retornado
         }
         if( num < vetor[centro]){
```

```
            fim = centro - 1;  // Se o numero estiver na primeira parte, ou seja do começo até o centro
            //  O valor de centro é setado como o fim, ou seja, estamos vendo apenas  a primeira metade do vetor
            // pois na segunda metade não há necessidade de busca pois  NUM < VETOR[centro]
            // Tendo em vista que o vetor esteja ordenado é claro.
          }
          if( num > vetor[centro]){
            inicio = centro + 1; // Segue a mesma ideologia acima, porém trabalha com a segunda parte do vetor
            // O que leva a entender o que o numero que buscamos é maior que numero que está no centro.
          }
      }
      return -1;

   }

}
```