Atividade

Valor da Atividade = **5,0**

Como será avaliada a Atividade?

1. A codificação será avaliada por:
    1. Apresentação do código de forma limpa, ou seja, identação do código;
    2. Nome de variáveis e métodos com nomes significativos;
    3. Comentários no código;
    4. Uso do português correto;
    5. Apresentação do resultado do problema.
2. Teste de mesa e passo a passo do algoritmo serão avaliados por:
    1. Clareza do teste de mesa;
    2. Apresentação do passo a passo;
    3. Uso do português correto;
    4. Apresentação do resultado do problema.
3. Análise de complexidade será avaliado por:
    1. Apresentação do valor de cada linha do algoritmo;
    2. Uso do português correto;
    3. Resultado final da análise de complexidade.
4. Ao ser identificado uma cópia os alunos envolvidos receberão a nota 0,0 automaticamente.

Como será a entrega da Atividade?

1. Faça um arquivo com extensão “.zip” com todos os arquivos da atividade;
    1. Identifique o arquivo com: POTA_Atividade_<SeuNome>_<SeuRA>.zip
2. Entrega via Blackboard;
3. A atividade vai ficar aberta até o dia 16/09/2021 até 23h59.

Exercícios da Atividade

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)**

$$Comb(n,k) = \begin{cases} n & \text{se } k = 1 \\ 1 & \text{se } k = n \\ Comb(n-1,k-1) + Comb(n-1,k) & \text{se } 1 < k < n \end{cases}$$

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);

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

```
int func(int n) {
     int i, r, j;
     r = 1;
     i = 1;
     j = 1;
     while (i <= n) {
          while (j <= n){
               r = r*n; i++;
          }
     }
     return r;
}
```

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

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)**