Aula 04 – Exercícios

1. Que tipos de problemas podem ser resolvidos por um algoritmo recursivo?

R: Muitos problemas têm a seguinte propriedade: cada instância do problema contém uma instância menor do mesmo problema. Para resolver uma instância de um problema desse tipo, podemos aplicar o seguinte método:

```
se a instância em questão for pequena,
    resolva-a diretamente (use força bruta se necessário);
senão
    reduza-a a uma instância menor do mesmo problema,
    aplique o método à instância menor,
    volte à instância original.
```

2. Qual o valor de X(4) se X é dada pelo seguinte código? Faça um teste de mesa para comprovar seu resultado.

```
int X (int n) {
 if (n == 1 || n == 2) return n;
   else return X (n-1) + n * X (n-2); }
```

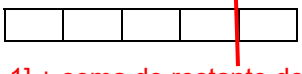
R: 13

Entrada X(n)	n	Condição (n=1 ou n=2)	Saída (X(n-1) + n*X(n-2))
X(4)	4	F	X(3) + 4*X(2) 5 + 4*2 = ***13***
X(3)	3	F	X(2) + 3*X(1) 2 + 3*1 = 5
X(2)	2	V	2
X(1)	1	V	1

3. Escreva uma função recursiva que some todos os valores de um array de n posições.

R: Caso base? Tamanho do array = 0. Soma é 0.

- Passo da recursão:

v 

v[n-1] + soma do restante do array.

```
int soma_a(int v[], int n)
{   if ( n == 0 ) return 0;
    return v[n-1] + soma_a(v, n-1); }
```

4. Dado um array de inteiros e o seu número de elementos, crie uma função recursiva que inverta a posição dos seus elementos.

R: Caso base? Tamanho do array menor ou igual a 1 ;

- Passo da recursão:

- Troca primeiro e último elementos, e inverte o resto do array.

```
void inverte(int v[], int esq, int dir)
{ int t;
   if (esq >= dir) return;
    t = v[esq];
    v[esq] = v[dir];
    v[dir] = t;
    inverte(v, esq+1, dir-1); }
```

5. O que esse programa vai imprimir ao final ?

```
void func1 (int x, int y) {
      int aux = x; x = y; y = aux; }

int main() {
  int a = 3, b = 5;
  func1(a,b);
  cout << a << " " << b << "\n";
  return 0 ; }
```

R: c) 3 e 5

6. Qual a complexidade do algoritmo abaixo?

```
int Prime1 (const int n){
    int t;
    for (int i = 1; i<=t; i++){
            int m = pow(n, 0.5);
            int j = rand() % m + 2;
            if (!(n%j))
               return 0 ;  }
    return 1; }
```

R: d) O(n)

⇨ 1
⇨ n * (
  ⇨ 1
  ⇨ 1
  ⇨ 1
  ⇨ 1
⇨ 1
= 1 + n * (1 + 1 + 1 + 1) + 1
= 2 + 4n

7. Seja a função :

```
int F (int n) {
      for (int i = 0; i <= 4; i++)
          for (int j = 0; j < n-1 ; j++)
             cout << i << " - "  << j << endl; }
```

Conte o número de passos e dê a complexidade (notação O).

R:
```
i = 0;          ---------------------------------  1
for (  ; i <= 4;  )  ---------------------------  6 vezes
{    ------------------------------------------
     j = 0;  ---------------------------------- 1
     for ( ; j < n-1 ; )  ----------------------  n testes
     {
        cout <<  i  << " - "  <<  j ;   ----  1  (n-1) vezes      Entrou 5 vezes  no 1º for
         j++;                           ----  1  (n-1) vezes
     }
     i++; }  ---------------------------------- 1
```

Número. de passos : 1 + 6 + 5.(1 + n + 2.(n-1) + 1) = 15n + 7
Complexidade : O(n)

8. Seja a função :

```
int S (int n) {
     int  sum = 0, i, j;
     for(i =0; i < n; i++ )
         for(j =0; j < n*n; j++ )
              sum++;
     return sum; }
```

Conte o número de passos e dê a complexidade (notação O).

R: Nº de passos :
$1 + 1 + (n+1) + n\,(1 + n^2 + 1 + 2n^2 + 1) =$
$3 + n + n\,(3\,n^2 + 3)$
$3 + n + 3\,n^3 + 3\,n$
$3\,n^3 + 4n + 3$
Complexidade : $O(n^3)$