Funções de Complexidade baseadas em Equações de Recorrência

1 – Implementação recursiva Polinômio

Considera a forma recursiva do polinômio: pn(x) = (((anx + an-1)x + ... + a2).x + a1)x + a0

Algoritmo Horner;

Entrada: vetor de coeficientes inteiros: [a0,a1,..., na];n: inteiro; x: inteiro

Saida: p: inteiro (valor do polinômio *)*

```
Função termo(i: inteiro): inteiro
Inicio
 Se ( i < n) Então
       Retorne <-  ai + x.termo(i+1)
Senão
       Retorne an
Fim_Se;
Fim_termo;
Inicio
     Se n=0 Então
         p <- a0
     Senão
         p <- termo(0);
Fim.
```

Simulação recursiva:

Suponha n=4:

termo(0);

retorne a0 + x.(termo(1))

retorne a0 + x.(a1 + x.(termo(2)))

retorne a0 + x.(a1 + x.(a2 + x.(termo(3))))

retorne a0 + x.(a1 + x.(a2 + x.(a3 + x.(termo(4)))))

retorne a0 + x.(a1 + x.(a2 + x.(a3 + x.(a4))))

Função de complexidade:

$T(n) = 1$

$T(n) = 4 + T(1)$

Segundo nível:

$T(n) = 4 + 4 + T(2)$

Késimo nível:

$T(n) = 4.k + T(k)$

Condição de retorno:

$k = n$

Derivando:

$T(n) = 4.n +1$

Portanto $T(n)$ é $O(n)$.

2 – Progressão Aritmética

(1,3,5,7,...)

Enésimo termo = (2n -1)

Função Impar(n:inteiro): inteiro;

Inicio

Se n=1 Então

Retorne 1

Senão

Retorne 2 + Impar(n-1);

Fim.

Função recursiva:

F(1) = 1

F(n) = F(n-1) + 2 p/ n>1

Função de complexidade:

T(1) = 1

T(n) = 3 + T(n-1) p/ n>1

Segundo nível:

T(n) = 3 + (3+ T(n-2))

Késimo nível:

T(n) = 3.k + T(n-k)

Condição de retorno:

n-k=1 => k=n-1

Portanto:

T(n) = 3.(n-1) + 1 => T(n) é O(n).

3 – Progressão Geométrica

(1,2,4,8,...)

Enésimo termo = (2^{n-1})

Função Quadrado (n:inteiro): inteiro;

Inicio

Se n=1 Então

Retorne 1

Senão

Retorne 2.Quadrado(n-1);

Fim.

Função recursiva:

F(1) = 1

F(n) = 2F(n-1) p n>1

Função de complexidade:

T(1) = 1

T(n) = 3 + T(n-1) p/ n>1

Segundo nível:

T(n) = 3 + (3+ T(n-2))

Késimo nível:

T(n) = 3.k + T(n-k)

Condição de retorno:

n-k=1 => k=n-1

Portanto:

T(n) = 3.(n-1) + 1 => T(n) é O(n).