USP-ICMC-BMACC/BCDados

SCC0223 - Estrutura de Dados I

2023

Resolução da Lista de Exercícios 2

Parte 1

1. ★ Seja a seguinte definição: "Dadas duas funções, $f(n)$ e $g(n)$, diz-se que $f(n)$ é da ordem de $g(n)$ ou que $f(n)$ é $\mathcal{O}(g(n))$, se existirem inteiros positivos a e b tais que $f(n) \leq a * g(n)$ para todo $n \geq b$." Verifique se as seguintes proposições estão corretas:

   i. $7 \in \mathcal{O}(n)$: $\surd$
   ii. $n \in \mathcal{O}(1)$: $\times$
   iii. $n + 7 \in \mathcal{O}(n)$: $\surd$
   iv. $n + 7 \in \mathcal{O}(1)$: $\times$
   v. $n^2 + 2 \in \mathcal{O}(n)$: $\times$
   vi. $n + 2 \in \mathcal{O}(n^2)$: $\surd$
   vii. $3n^3 + n \in \mathcal{O}(n^3)$: $\surd$
   viii. $2n^4 \in \mathcal{O}(n^4)$: $\surd$
   ix. $n^4 \in \mathcal{O}(2n^4)$: $\surd$
   x. $3n^4 + 2n^3 \in \mathcal{O}(2n^4)$: $\surd$
   xi. $2n^4 \in \mathcal{O}(3n^4 + 2n^3)$: $\surd$
   xii. $log\ n \in \mathcal{O}(1)$: $\times$
   xiii. $log\ n + 1 \in \mathcal{O}(log\ n)$: $\surd$
   xiv. $log\ n + 1 \in \mathcal{O}(n)$: $\surd$
   xv. $log\ n + 1 \in \mathcal{O}(n^2)$: $\surd$
   xvi. $log\ n + 1 \in \mathcal{O}(n^3)$: $\surd$
   xvii. $n * log\ n \in \mathcal{O}(1)$: $\times$
   xviii. $n * log\ n + 1 \in \mathcal{O}(log\ n)$: $\times$
   xix. $n * log\ n + 1 \in \mathcal{O}(n)$: $\surd$
   xx. $n * log\ n + 1 \in \mathcal{O}(n^2)$: $\surd$
   xxi. $n * log\ n + 1 \in \mathcal{O}(n^3)$: $\surd$
   xxii. $2log\ n \in \mathcal{O}(n * log\ n)$: $\surd$
   xxiii. $3n * log\ n \in \mathcal{O}(log\ n)$: $\times$
   xxiv. $2n + n \in \mathcal{O}(2^3)$: $\times$
   xxv. $n^2 \in \mathcal{O}(2^n)$: $\surd$
   xxvi. $100n^4 \in \mathcal{O}(2^n)$: $\surd$

xxvii. $100n^4 \in \mathcal{O}(n^n)$: $\surd$

xxviii. $2^n \in \mathcal{O}(100n^4)$: $\times$

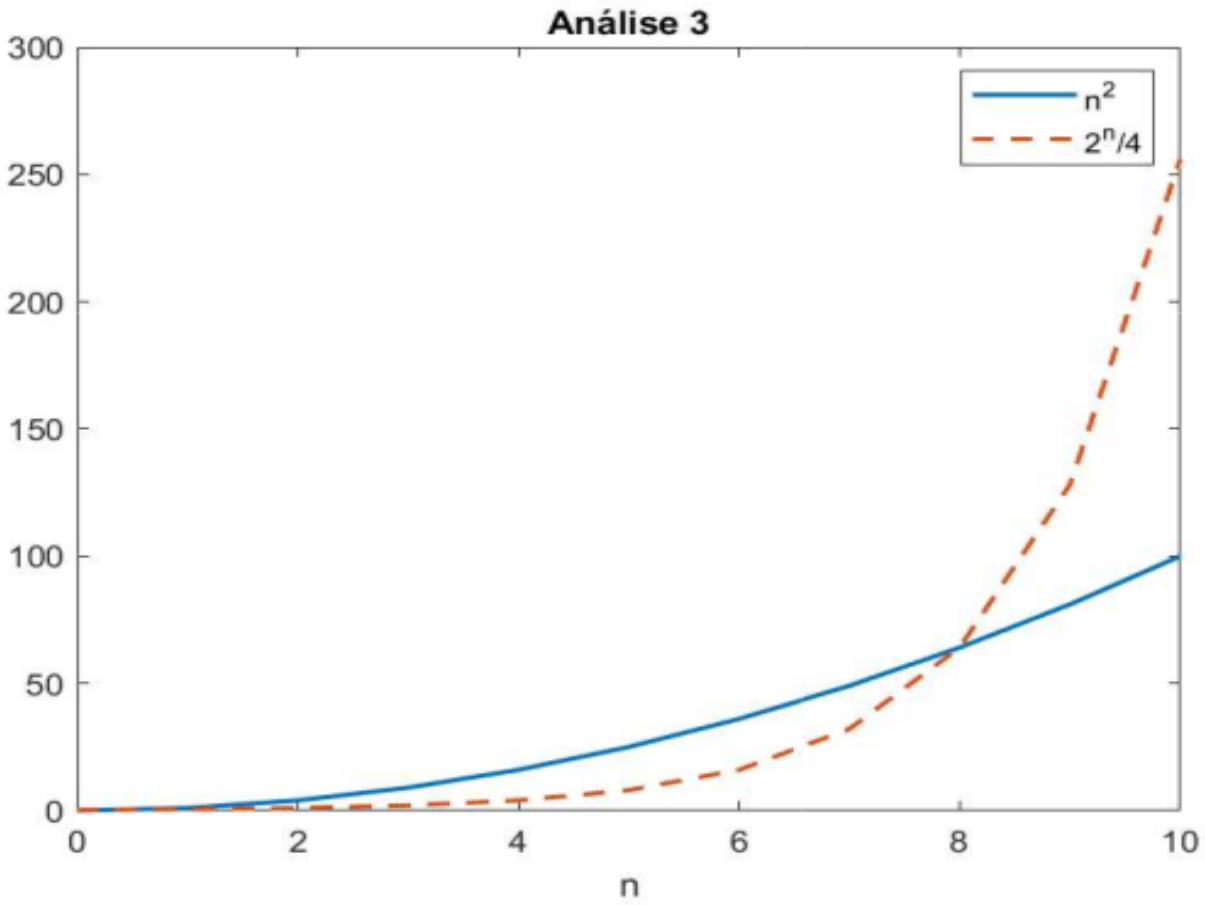
xxix. $2^n \in \mathcal{O}(n^n)$: $\surd$

xxx. $n^n \in \mathcal{O}(2^n)$: $\times$

xxxi. $n^{100} \in \mathcal{O}(n^n)$: $\surd$

2. ★ Compare as duas funções n^2 e $\frac{2^n}{4}$ para vários valores de n. Determine quando a segunda se torna maior que a primeira.

Resolução:

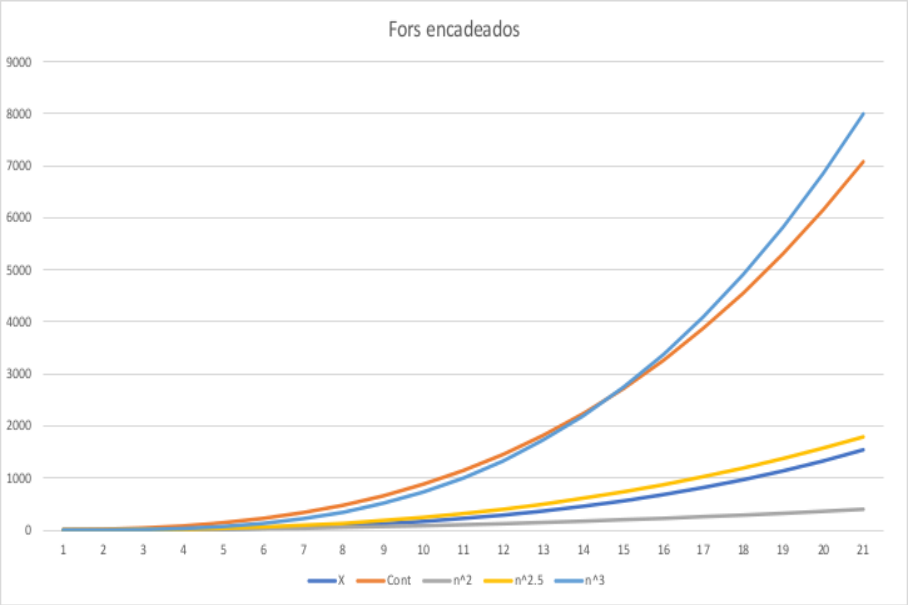


Resolução da Lista 2 (continuação)

3. ★ Determine os contadores de frequência para todos os comandos nos seguintes dois segmentos de algoritmos:

   i.
   ```
   (1) para i ← 1 até n faça
   (2)   para j ← 1 até i faça
   (3)     para k ← 1 até j faça
   (4)       x ← x + 1;
   ```

   Resolução:

   Pelo gráfico, o algoritmo é $\mathcal{O}(n^3)$.

   ii.
   ```
   (1) i ← 1;
   (2) enquanto (i ≤ n) faça
   (3) {
   (4)   x ← x + 1;
   (5)   i ← i + 1;
   (6) }
   ```

   Resolução:

   O algoritmo é $\mathcal{O}(n)$.

Resolução da Lista 2 (continuação)

4. ★ Considere um computador com *clock* de 2GHz, que realiza cada operação relevante em 1 ciclo. Estime, apenas com esses dados, o tempo necessário para que ele execute um algoritmo que realiza $(n^2 - n)/2$ operações relevantes, considerando que há 4M dados de entrada.

   Resolução:

   $2\text{GHz} = 2 * 10^9 s^{-1}$

   $\text{ciclo} = 0,5 * 10^{-9} s$

   $n = 4M = 4 * 10^6$:

   $(n^2 - n)/2 \ * \ 0,5 * 10^{-9} s =$

   $((4 * 10^6)^2 - 4 * 10^6)/2 \ * \ 0,5 * 10^{-9} s =$

   $(16 * 10^{12} - 4 * 10^6)/2 \ * \ 0,5 * 10^{-9} s \approx$

   $8 * 10^{12} \ * \ 0,5 * 10^{-9} s = 4 * 10^3 s = 4000 s \approx 1h06$

   Para $n = 4K$:

   $((4 * 10^3)^2 - 4 * 10^3)/2 \ * \ 0,5 * 10^{-9} s \approx$

   $8 * 10^6 * 0,5 * 10^{-9} s = 4ms.$

5. ★ Idem, usando um algoritmo que realiza n^3 operações relevantes.

   Resolução:

   $n^3 \ * \ 0,5 * 10^{-9} s =$

   $(4 * 10^6)^3 \ * \ 0,5 * 10^{-9} s =$

   $(64 * 10^{18} \ * \ 0,5 * 10^{-9} s =$

   $32 * 10^9 s \approx 1015$ anos.

   Para $n = 4K$:

   $(4 * 10^3)^3 \ * \ 0,5 * 10^{-9} s =$

   $64 * 10^9 * 0,5 * 10^{-9} s = 32s.$

Resolução da Lista 2 (continuação)

6. ★ Idem, usando um algoritmo que realiza 2^n operações relevantes.

   Resolução:

   Para $n = 4K$:

   $2^{4K} \ * \ 0,5 * 10^{-9}s =$

   $1,32 * 10^{1204} * 0,5 * 10^{-9}s = 6,6 * 10^{1194}s.$

7. ★ Idem, usando um algoritmo que realiza n^n operações relevantes.

   Resolução:

   Para $n = 4K$:

   $4K^{4K} \ * \ 0,5 * 10^{-9}s =$

   $1,74 * 10^{14408} * 0,5 * 10^{-9}s = 8,7 * 10^{14400}s.$

8. ★ Idem, para um computador com *clock* de 100MHz, ordenando a mesma seqüência, usando um algoritmo que realiza $4/3 * n \ log \ n$ operações relevantes. Analise os resultados.

   Resolução:

   $100\text{MHz} = 100 * 10^6 s^{-1} = 1 * 10^8 s^{-1}$

   $\text{ciclo} = 1 * 10^{-8}s$

   Lembre-se de que $log_2(n) = ln(n)/ln(2) = log(n)/log(2)$

   $n = 4M = 4 * 10^6$:

   $4/3 * 4 * 10^6 \ log \ (4 * 10^6) \ * \ 1 * 10^{-8}s =$

   $16/3 * 10^6 * 21,93 * 10^{-8} = 116,97 * 10^{-2} = 1,17s$

   Para $n = 4K$:

   $4/3 * 4 * 10^3 \ log \ (4 * 10^3) \ * \ 1 * 10^{-8}s =$

   $16/3 * 10^3 * 11,97 * 10^{-8} = 63,82 * 10^{-5} = 0,64 * 10^{-3}s = 0,64ms$

Resolução da Lista 2 (continuação)

Parte 2

9. Resolva as seguintes equações de recorrência:

(a)

$$\begin{cases} T(n) = T(n-1) + c & c \text{ constante}, n > 1 \\ T(1) = 0 \end{cases}$$

(b)

$$\begin{cases} T(n) = cT(n-1) & c, k \text{ constantes}, n > 0 \\ T(0) = k \end{cases}$$

(c) ★

$$\begin{cases} T(n) = 3T(n/2) + n & n > 1 \\ T(1) = 1 \end{cases}$$

cuja solução satisfaz $T(n) = \mathcal{O}(n\ log^2\ n)$. Prove usando indução matemática em n.

Solução:

Primeira forma: Um tipo especial de equação de recorrência que é frequentemente encontrada em análise de algoritmos:

$$T(n) = aT(n/b) + cn^i,$$

para alguma constante inteira i e coeficientes a, b e c. Três casos:

i. $a = b^i$, a solução é $T(n) = \mathcal{O}(n^i\ log_b\ n)$;

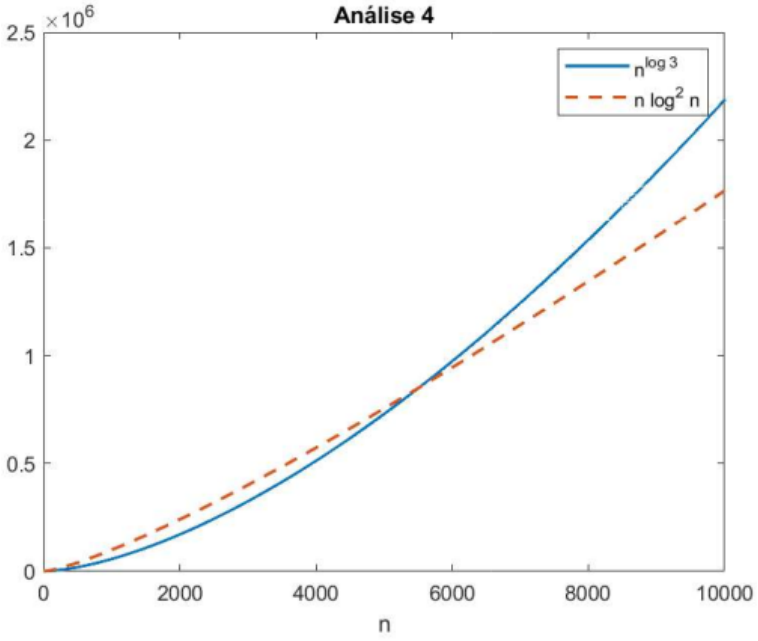
ii. $a > b^i$, a solução é $T(n) = \mathcal{O}(n^{log_b\ a})$;

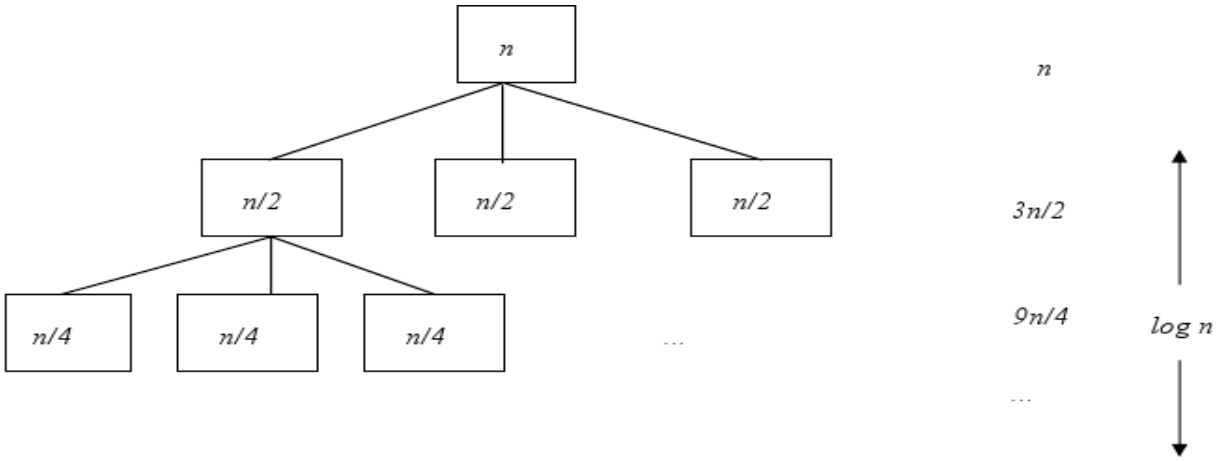
iii. $a < b^i$, a solução é $T(n) = \mathcal{O}(n^i)$;

Fonte: `cs.fit.edu/~dmitra/Algorithms/lectures/RECEQUN.doc`.

Neste caso, $a = 3$; $b = 2$; $c = 1$ e $i = 1$:

Caso 2. $a > b^i$, pois $3 > 2^1$: Solução $T(n) = \mathcal{O}(n^{log\ 3})$ e não $\mathcal{O}(n\ log^2\ n)$ como está no enunciado.





Segunda forma: Árvore de recursão:

$T(n) = 3T(n/2) + n$

$= n\ (1 + 3(1/2) + 3^2(1/4) + ... + 3^{log\ n}(1/2^{log\ n}))$
$= n\ (1 + 3/2 + (3/2)^2 + ... + (3/2)^{log\ n})$
$= n\ (\sum_{i=0}^{log\ n}(3/2)^i)$
$= n\ (\frac{(3/2)^{log\ n+1}-1}{3/2-1})$
$= n\ (\frac{(3/2)^{log\ n}\ (3/2)-1}{1/2})$
$= (\frac{n\ (3/2)^{log\ n}\ (3/2)}{1/2}) - 2n$

$= 3n\ (3/2)^{log\ n} - 2n$
$= 3n\ n^{log\ 3/2} - 2n$
$= 3n^{1+log\ 3/2} - 2n$
$= 3n^{log\ 2+log\ 3/2} - 2n$
$= 3n^{log\ 3} - 2n$
$= \mathcal{O}(n^{log\ 3})$

pois

$a^{log\ b} = b^{log\ a}$
$\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$
$1 = log\ 2$
$log\ a + log\ b = log\ ab$

Obs.: $log\ 3 = 1.5849625$

Solução:

Terceira forma: Iteração:

$T(n) = 3T(n/2) + n$
$= 3(3T(n/4) + n/2) + n$
$= 9T(n/4) + 3n/2 + n$
$= 9(3T(n/8) + n/4) + 3n/2 + n$
$= 27T(n/8) + 9n/4 + 3n/2 + n$
$= 3^iT(n/2^i) + 3^{i-1}n/2^{i-1} + 3^{i-2}n/2^{i-2} + 3^{i-3}n/2^{i-3}$
$= 3^iT(n/2^i) + \sum_{k=0}^{i-1}(3/2)^{i-k-1}n$

$\sum_{k=0}^{i-1}(3/2)^{i-k-1}n = \frac{(3/2)^i-1}{(3/2)-1}n = 2n(3/2)^i - 1$

Para $n = 2^i \Rightarrow i = \log n$

$= 3^{\log n}T(1) + 2n(3/2)^{\log n} - 1$
$= n^{\log 3} + 2n(3/2)^{\log n} - 1$
$= n^{\log 3} + 2n(n^{\log 3/2}) - 1$
$= n^{\log 3} + 2n^{1+log\ 3/2} - 1$
$= n^{\log 3} + 2n^{log\ 2+log\ 3/2} - 1$
$= n^{\log 3} + 2n^{log\ 3} - 1$
$= \mathcal{O}(n^{\log 3})$

pois

$a^{log\ b} = b^{log\ a}$
$\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$
$1 = log\ 2$
$log\ a + log\ b = log\ ab$

10. Considere o algoritmo a seguir. Suponha que a operação crucial é o fato de inspecionar um elemento. O algoritmo inspeciona os n elementos de um conjunto e, de alguma forma, isso permite descartar $\frac{2}{5}$ dos elementos e então fazer uma chamada recursiva sobre os $\frac{3n}{5}$ elementos restantes.

```
void Pesquisa (int n)
{
 if (n <= 1)
    'inspecione elemento' e termine
 else {
    para cada um dos n elementos 'inspecione o elemento';
    Pesquisa (3n/5);
    }
}
```

    (a) Escreva uma **equação de recorrência** que descreva este comportamento.

    (b) Converta esta equação para um somatório.

    (c) Dê a fórmula fechada para este somatório.

11. Escreva um algoritmo recursivo para o problema das Torres de Hanói. Mostre a equação de recorrência e calcule a complexidade do algoritmo. Suponha três pinos e, num deles, alguns discos dispostos um sobre o outro em ordem de tamanho do diâmetro (o menor no topo). O problema consiste em passar todos os discos de um pino para outro qualquer, usando um dos pinos como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor em nenhuma situação.

12. Seja o algoritmo recursivo de Busca Binária mostrado abaixo. Encontre a equação de recorrência do algoritmo. Em seguida, defina a sua complexidade para o pior caso.

```
int busca_b(int a[], int x, int baixo, int alto)
{
 int meio;
```

```
 if (baixo > alto)
   return(-1);
 meio = (baixo + alto)/2;
 return(x == a[meio]) ? meio : x < a[meio] ?
      busca_b(a, x, baixo, meio-1) :
      busca_b(a, x, meio+1, alto);
}
```

13. Um elemento majoritário em um arranjo A de tamanho n, é um elemento que aparece mais que $\frac{n}{2}$ vezes. Por exemplo, o arranjo 3, 3, 4, 3, 4, 4, 2, 4, 4 tem o 4 como elemento majoritário. Se não existir um elemento majoritário seu algoritmo deve indicar esse fato. Temos a seguir, um esboço de um algoritmo para solucionar o problema acima:

    > Primeiro, um candidato a elemento majoritário é encontrado (esta é a pior parte). Este candidato é o único elemento que poderá ser um possível majoritário. O segundo passo determina se o candidato selecionado é realmente majoritário. Isto é feito com uma pesquisa seqüencial no arranjo. Para encontrar um candidato a majoritário no arranjo A, forme um segundo arranjo B. Compare $A[1]$ e $A[2]$. Se forem iguais, coloque um deles em B. Caso contrário não faça nada. Compare $A[3]$ e $A[4]$. Novamente, se forem iguais coloque um deles em B. Caso contrário não faça nada. Continue dessa forma até ter processado todo o arranjo A. Então, recursivamente, encontre um candidato em B.

    (a) Como a recursividade termina?

    (b) Qual é o tempo de execução desse algoritmo?

    (c) Como podemos evitar o uso do arranjo extra B?

    (d) Escreva o programa que encontre o elemento majoritário.

Resolução da Lista 2 (continuação)

Parte 3

Exercícios Propostos no slide 89 - SCC0223Cap2

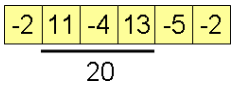
1 ★ Implemente o algoritmo da busca binária em um arranjo ordenado, teste e analise o algoritmo,

Resolução:

```
int busca_bin(int x, int vet[], int ini, int fim)
{
    int pos = (ini + fim)/2;
    if (ini > fim)
        return -1;
    else
    if (vet[pos] == x)
        return pos;
    else
    if (x < vet[pos])
        return busca_bin(x, vet, ini, pos - 1);
    else
        return busca_bin(x, vet, pos + 1, fim);
}
```

Análise: $T(n) = 5$, $n < 2$
$T(n) = 7 + T(n/2)$, $n \geq 2$
$\Theta(\log n)$: árvore de recursão: c em cada nível; altura $= \log n$

2 ★ Faça um algoritmo para resolver o problema da maior soma de subsequência em um arranjo e analise-o:



Resolução:

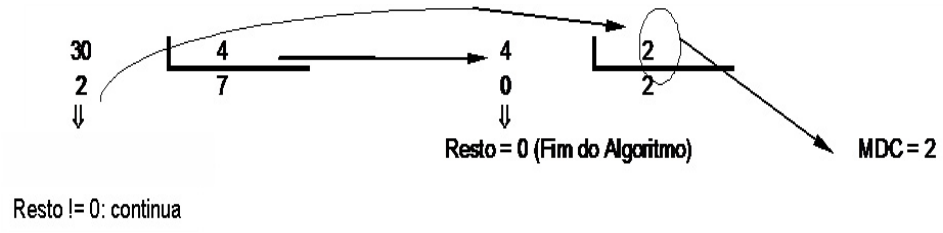
```
int max (int a, int b, int c)
{
    if ((a >= b) && (a >= c)) return a;
    if ((b >= a) && (b >= c)) return b;
    if ((c >= a) && (c >= b)) return c;
}
```

```
int mssub(int vet[], int ini, int fim)
{
    // divisão e conquista
    int meio, x1, x2, y1, y2, x, i, j, s;
    if (ini == fim)
        return vet[ini];
    else
    {
        meio = (ini+fim)/2;
        x1 = mssub(vet,ini,meio);
        x2 = mssub(vet,meio+1,fim);
        y1 = s = vet[meio];
        for (i=meio-1;i>=ini;i--)
        {
            s = vet[i]+s;
            if (s > y1) y1 = s;
        }
        y2 = s = vet[meio+1];
        for (j=meio+2;j<=fim;j++)
        {
            s = s + vet[j];
            if (s > y2) y2 = s;
        }
        x = max(x1,y1+y2,x2);
        return x;
    }
}
```

Análise: Divisão e conquista: $T(n) = T([n/2]) + T([n/2]) + n$
Complexidade: $\mathcal{O}(n \log n)$

3 ★ Implemente o algoritmo de Euclides para calcular o máximo divisor comum para 2 números, e faça a análise de recorrência do mesmo,

Resolução:




```
int MDC (int i, int j)
{
    if (i % j != 0)
        return MDC(j, i % j);
    else
        return j;
}
```

Análise: $\log(min(i, j))$

4 ★ Escreva e analise um algoritmo recursivo para calcular x^n.

Resolução:

```
int pot_r (int base, int expoente)
{
    /* pot_r: Calcula a potência de um número de forma recursiva
     * entrada: base e expoente
     * saida: resultado da potência
     * suposicao: expoente é um inteiro positivo
     */

    if (expoente == 0)
        return 1;
    return base * pot_r(base, expoente-1);
}

int pot_i (int base, int expoente)
{
    /* pot_i: Calcula a potência de um número de forma iterativa
     * entrada: base e expoente
```

```
     * saida: resultado da potência
     * suposicao: expoente é um inteiro positivo
     */

    int i, aux=1;
    if (expoente == 0)
        return 1;
    else
    {
        for (i = 1; i <= expoente; i++)
        aux *= base;
    }
    return aux;
}
```

Análise: disfarce da repetição: expoente vezes $\Rightarrow \mathcal{O}(n)$