1. Prove ou conteste as seguintes relações:

i) $\log_2 n \sim \log_3 n + 1/n$;

Solução:

De acordo com a definição de "$\sim$", a relação i) é falsa pois:

$$\lim_{n\to\infty} \frac{\log_2 n}{\log_3 n + 1/n} = \lim_{n\to\infty} \frac{\log_2 n}{\log_3 n} = \log_3 / \log 2 \neq 1$$

ii) $3^n = O(e^n)$;

Solução:

Para duas funções estritamente positivas $f(n)$ e $g(n)$, a definição de $f(n) = O(g(n))$ estabelece que para algum valor constante c e todo valor de n maior que uma constante N será verdade que $f(n) \leq cg(n)$.

De acordo com essa definição, a relação ii) é falsa pois:

Para constante positiva c teríamos $3^n \leq ce^n$, o que leva a $\left(\frac{3}{e}\right)^n \leq c$

porém $\frac{3}{e} > 1$ o que implicaria em $n \leq \log_{3/e} c$ (o que não respeita a definição de O)

Por outro lado, isso também implica que para $n > \log_{3/e} c$ temos que $3^n = \Omega(e^n)$ o que é o oposto de ii)

iii) $\sum_{i=1}^{n} i = \omega(n^2)$;

Solução:

A definição da notação $\omega(\cdot)$ é $f(n) = \omega(g(n))$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Temos que a soma de $n/2$ termos valendo $n+1$ é $\sum_{i=1}^{n} = n(n+1)/2$

Ao aplicar a definição fazemos: $\lim_{n\to\infty} \frac{n(n+1)/2}{n^2} = \lim_{n\to\infty} \frac{n+1}{2} = \infty$

Portanto, a relação iii) não atende à definição da notação e é falsa.

iv) $1 = o(1/n)$;

Solução:

Uma função $f(n)$ é $o(g(n))$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

Portanto, a relação iv) é falsa pois:

$$\lim_{n\to\infty} \frac{1}{1/n} = \infty$$

v) $\sqrt{n} = \Omega(\log_{10} n)$;

Solução:

Para duas funções estritamente positivas $f(n)$ e $g(n)$, a definição de $f(n) = \Omega(g(n))$ estabelece que para algum valor constante c e todo valor de $n > N$, sendo N uma constante arbitrária, será verdade que $f(n) \geq cg(n)$.

Sejam $f(n) = \sqrt{n}$ e $g(n) = \log_{10} n$.

Soluçõ 1) (usando a definição de $\Omega(\cdot)$) a relação v) é verdade pois para $n > 0$ podemos achar o menor valor constante $c > 1$ tal que seja verdade:

$$f(n)/g(n) \geq c$$

Isso é equivalente a encontrar um valor para n que faz a inclinação (isto é, derivada) de $f(n)/g(n)$ mudar. A inclinação muda quando a derivada é igual a zero. Assim, devemos achar o valor n que satisfaz a igualdade:

$$\left(\frac{f(N)}{g(N)}\right)' = 0$$

$$
\begin{aligned}
& & \left(\frac{f(N)}{g(N)}\right)' = \frac{f'(N)}{g'(N)} = \\
&\text{(derivada de produto de funções)} & = \frac{1}{2}\frac{1}{n^{1/2}}\frac{1}{\log_{10} n} + \sqrt{n}\,(\log_{10} n)' = \\
&\text{(regra da cadeia)} & = \frac{1}{2}\frac{1}{n^{1/2}}\frac{1}{\log_{10} n} + \sqrt{n}\frac{1}{n\log 10}\cdot -1\,(\log_{10} n)^{-2} = \\
& & = \frac{1}{2}\frac{1}{n^{1/2}}\frac{1}{\log_{10} n} + \frac{-1}{\sqrt{n}\ln 10\log_{10}^2 n} = \\
&\text{(usando } \log_{10} n = \frac{\ln n}{\ln 10}\text{)} & = \frac{1}{2}\frac{1}{n^{1/2}}\frac{\ln 10}{\ln n} + \frac{-\ln 10}{\sqrt{n}\ln^2 n} = \\
&\text{(reorganizando)} & = \frac{\ln 10}{2\sqrt{n}\ln n} - \frac{\ln 10}{\sqrt{n}\ln^2 n} = \\
&\text{(obtendo mesmo denominador)} & = \frac{\ln 10\ln n}{2\sqrt{n}\ln^2 n} - \frac{2\ln 10}{2\sqrt{n}\ln^2 n} = \\
&\text{(juntando frações)} & = \frac{\ln 10\ln n - 2\ln 10}{2\sqrt{n}\ln^2 n} = \\
&\text{(numerador em evidência)} & = \frac{\ln 10(\ln n - 2)}{2\sqrt{n}\ln^2 n} = 0 \\
&\text{Portanto, a fração é igual a zero quando:} & \ln n = 2
\end{aligned}
$$

Portanto, a partir de $n = e^2$ a derivada muda e torna-se positiva. Concluindo, para $n > e^2$ e $c = \ln e^2/\sqrt{e^2} \approx 0.735$, relação v) é verdade.

Solução 2) se provarmos que $\sqrt{n} = \omega(\log_{10} n)$ isso implicaria em $\sqrt{n} = \Omega(\log_{10} n)$.

Dessa forma, usando a definição de $\omega(\cdot)$:

$$
\lim_{n\to\infty}\frac{\sqrt{n}}{\log_{10} n} = \text{ (regra de L'Hopital) } \lim_{n\to\infty}\frac{1/2\frac{1}{n^{1/2}}}{\frac{1}{n}\log_e 10} = \lim_{n\to\infty}\frac{1}{2\log_e 10}n^{\frac{1}{2}} = \infty
$$

o que comprova, pela definição da notação $\omega(\cdot)$ que v) é verdade.

vi) $\lg(n!) = \Theta(n^2)$

Para essa relação ser verdadeira é necessário que $\lg(n!) = \Omega(n^2)$ e $\lg(n!) = O(n^2)$.

Como $\lg(n!) = \sum_{k=1}^{n} \lg k \le n\lg n$, é necessário provar que $n\lg n \le cn^2$, sendo $c > 0$ uma constante:

$$
\begin{aligned}
n\lg n &\le cn^2 \\
\lg n &\le n
\end{aligned}
$$

Pela definição de log isso é verdade (log de um número é sempre menor ou igual a ele mesmo) para $n > 1$.

Contudo não é verdade que $n\lg n \ge c_2 n^2$ para ser verdade que $\lg(n!) = \Omega(n^2)$. Isso pode ser provado da seguinte forma para $c_2 = 1$:

$$
\begin{aligned}
n\lg n &\ge c_2 n^2 \\
\lg n &\ge n
\end{aligned}
$$

O que não é verdade para $n > 1$. Portanto, a relação **vi)** não é verdadeira.

Solução alternativa: provar que $n\lg n = o(n^2)$ por limite:

$$
\begin{aligned}
&\lim_{n\to\infty}\frac{n\lg n}{n^2} = \lim_{n\to\infty}\frac{\lg n}{n} = \\
= &\text{ (regra de L'Hopital) } \lim_{n\to\infty}\frac{1/(n\ln 2)}{1} = 0
\end{aligned}
$$

2. Considere um array quase ordenado em que qualquer elemento pode estar em no máximo $c \geq 0$ posições (um número constante) à direita ou esquerda da posição em que estaria se o array estivesse ordenado. Escreva um algoritmo de ordenação para esse cenário baseado em divisão e conquista para resolver esse problema. Obtenha a forma fechada para o número de comparações em termos de n e c.

Custo total: mergesortEmAteC: $C_N = (N-1) + \sum_{c<k<N} \lfloor \lg k \rfloor + 1$ + custo do insertion sort $n \times c$.

```
void algoritmoDeOrdenacao(int v[], int c) {
  mergesortEmAteC(v,0,v.length-1,c);// ordena pedaços com mais que c elementos:
  insertionSort(v); // compara cada número no máximo $c$ posições: custo $n\times c$
}

void mergesortEmAteC(int v[], int lo, int hi, int c) {
    if (hi <= lo +c) return;

    int mid = lo + (hi-lo)/2;
    mergesortEmAteC(a, lo, mid);  mergesortEmAteC(a, mid+1, hi);

    for (int k = lo; k <= mid; k++)    b[k-lo]     = a[k];
    for (int k = mid+1; k <= hi; k++) c[k-mid-1] = a[k];
    b[mid-lo+1] = c[hi-mid] = Integer.MAX_VALUE;

    int i =0, j = 0;
    for (int k = lo; k <= hi; k++)
      if (c[j] < b[i]) a[k] = c[j++];
      else             a[k] = b[i++];
}

void insertionSort(int v[]) {
  for (int i = lo; i < v.length ; i++)
    for (int j = i; j>= 1 && v[j-1] > v[j]; j--) {
      int aux = v[j];
      v[j] = v[j-1];
      v[j-1] = aux;
    }
}
```

3. Analise as recorrências: **i)** $a_n = a_{n-1} - a_{n-2}/4$, $n > 1$, $a_0 = 0$,$a_1 = 4$;

O polinômio característico da recorrência é $x^2 - x + 1/4$. Esse polinômio tem raiz dupla igual a 0.5. Portanto a solução para a_n é da forma $cn(.5)^n$ para uma constante c que depende das condições iniciais. Quando $n = 1$, obtemos $4 = c1(.5)^1$ e $c = 8$. Assim, $a_n = 8n(.5)^n$.

ii) $a_{n+1} = n + \sum_{k=1}^{n}(a_k + a_{n-k+1})$, $a_1 = 1$,$n > 1$;

Primeiro, reescrever na forma $a_n = n + 2\sum_{k=1}^{n-1} a_k$, $a_0 = 1$, $a_1 = 3$.

Obter $a_{n+1} = (n+1) + 2\sum_{k=1}^{n} a_k$ e subtrair $a_{n+1} - a_n = 1 + 2a_n$.

Dessa forma, $a_{n+1} = 1 + 3a_n$, $a_1 = 3$. Usar 3^{n+1} como fator para dividir a recorrência. Definir $u_{n+1} = a_{n+1}/3^{n+1}$ e $u_1 = 1$ com $n > 1$ para obter $u_{n+1} = u_n + 1/3^{n+1}$. A solução de u_n é $7/6 - 3^{-n}/2$ e portanto $a_{n+1} = (7/6 \times 3^n - 1/2)$.

iii) $a_n = 3a_{n/2} + 3$, $a_1 = 0$, $n > 1$;

Seja $\alpha = 3$ e $\beta = 2$ e $f(n) = 3 = O(n^{\log_\beta \alpha - \epsilon})$, com $\epsilon = \log_\beta \alpha$ então usando o Caso 1 do Teorema mestre, $a(n) = \Theta(n^{\log_\beta \alpha})$.

iv) $a_n = 3a_{n/2} + n$, $a_1 = 0$, $n > 1$;

Seja $\alpha = 3$, $\beta = 2$, então usando o segundo caso do teorema para recorrências da forma $a(x) = \alpha a(x/\beta) + n$, temos que

$$a_n = \frac{3}{3-2}\left(\frac{2}{3}\right)^{\{\log_2 3\}} n^{\log_2 3}$$

v) $a_n = 2a_{n/3} + n \log n$, $a_1 = 1$,$n > 1$

Seja $a = 2$, $b = 3$, $f(n) = n \log n = \Omega(n^{\log_3 2 + \epsilon})$, então usando o terceiro caso do Teorema mestre, $a_n = \Theta(f(n))$.

4. O número de inversões em um array é igual o número de vezes que pares de números a_i e a_j nas posições i e j estão invertidos, ou seja, $a_i > a_j$. Por exemplo no array "[2,1,3,0]" há 4 inversões. O número de cópias que o insertion sort faz para ordenar um array é igual ao total de inversões. Escreva um algoritmo para contar a quantidade total de inversões cujo número de comparações seja $o(n^2)$ no pior caso.

```
int contarInversoes(int v[], int lo, int hi) {
    if (hi <= lo) return;
    int mid = lo + (hi-lo)/2;
```

```
    int inversoes = contarInversoes(a, lo, mid);
    inversoes +=    contarInversoes(a, mid+1, hi);

    for (int k = lo; k <= mid; k++)   b[k-lo]     = a[k];
    for (int k = mid+1; k <= hi; k++) c[k-mid-1] = a[k];

    b[mid-lo+1] = c[hi-mid] = Integer.MAX_VALUE;

    int i =0, j = 0;
    for (int k = lo; k <= hi; k++)
      if (c[j] < b[i])  {
        inversoes += mid -i; // todos os elementos de i até mid estão invertidos
        a[k] = c[j++];
      }
      else              a[k] = b[i++];

    return inversoes;
}
```