*ACH2002 – Introdução à Ciência da Computação 2 – Tarefa 1/2011*
*Prof. Fábio Nakano.*
*Introdução:*

*Um colega seu pegou esta prova do ano passado, resolveu (com tempo cronometrado) e pediu para você corrigir.*

*Objetivos:*

*corrigir,*
*comentar os erros,*
*atribuir nota e*
*propor uma solução correta.*

*Observações:*

*Tarefa individual;*
*Seja rigoroso;*
*Faça todos os comentários que julgar necessários.*

*Entrega: 14.10.2011 no início da aula.*

*Formatos permitidos:*

*Papel (se manuscrito);*
*CoL (se totalmente eletrônico)*

---

1-) **(1,5pt)** Escreva o código recursivo que calcula a raiz quadrada de n. Ele deve implementar a sequência

$x_{i+1}=\frac{1}{2}\left(x_i+\frac{n}{x_i}\right)$ e calcular novos termos enquanto $\varepsilon=|x_{i+1}-x_i|>EPS$ .

Sugestão: o método recebe $x_i$ como parâmetro, calcula $x_{i+1}$ e $\varepsilon$ e age de acordo com o necessário. (Apenas números positivos serão passados ao método).

```
import java.lang.Math.*;

class Raiz {
	public static final double EPS=1e-10;
	public static double raizrec (double x0, double n) {
		// este é o método:
		double proxx= 0.5*(x+(n/x));
		double raiz;
		while (Math.abs(proxx-x)<EPS) {
			raiz= raizrec (proxx, n);
		}
		return raiz;
	}
	public static double raiz (double n) {
		return raizrec (1, n);
	}
	public static void main (String[] args) {
		System.out.println (raiz (1.44));
	}
}
```

2-) Dado o código abaixo, obtenha a recorrência que descreve sua complexidade de tempo **(1pt)** e explique a que corresponde cada um de seus termos **(1pt)**. Note que o algoritmo é de ordenação por inserção!!

```
class InsertionSort {
	int[] Array;
	void ordena (int IElemento) {  // o fim é o final do array mm.
		System.out.println (IElemento);
		if (IElemento<(Array.length-1)) {
			ordena (IElemento+1);
			insere (IElemento);
		}
	}

	/** Insere o elemento de índice IElemento na posicao "certa"
	  * do array ordenado. */
	void insere (int IElemento) {
		int Elemento=Array[IElemento];
		int i=IElemento;
		while ((i<(Array.length-1))&&(Elemento>Array[i+1])) {
			i++;
			Array[i-1]=Array[i];
		}
		Array[i]=Elemento;
	}
	/** Métodos auxiliares fornecidos por completude. */
	InsertionSort (int[] Arr) {
		Array=Arr;
	}

	void imprime () {
		for (int i=0;i<Array.length;i++)
			System.out.print (Array[i] + " ");
		System.out.println ();
	}
	public static void main (String[] args) {
		int[] Arr={98, 56, 33, -4, -98, 100, 7};
		InsertionSort IS=new InsertionSort (Arr);
		IS.ordena(0);
		IS.imprime();
	}
}
```

$T(n)=(n-1)*(n)$ pois insere tem um loop que percorre o array

inteiro e ordena se chama n vezes.

3-) Dada a recorrência $T(n)=\begin{cases} k\ para\ n=1 \\ 2*T(\frac{n}{3})+k\ para\ n>1 \end{cases}$ , **prove por indução** que a fórmula fechada que a descreve é $T(n)=(2^{\log_3(n)+1}-1)k$ . **(base 0,3pt, passo 1,2pt)**

Obs.: a=2, b=3, f(n)=k

base n=1

$T(1)=(2^{\log(1)+1}-1)*k$

$T(1)=(2^1-1)*k=1$

paso n+1

$T(n+1)=(2^{\log(n+1)+1}-1)*k=(2^{\log(n)+1+1}-1)*k$

$(2*2^{\log(n+1)}-1)*k=(2^{\log(2*n)+1}-1)*k$ cqd.

4-) Dadas f(n) e g(n) abaixo, demonstre que $f(n)\in O(g(n))$ . No ítem b, use o conceito de dominância – não use a definição usando limites.

a-) **(1,0pt)** $f(n)=n*(\log(n)+sen(n));g(n)=n^2$

$\lim_{x\to\infty}(\frac{n*(\log(n)+sen(n))}{n^2})=\lim_{x\to\infty}(\frac{n*(k*1/n+\cos(n))+(\log(n)+sen(n))}{2*n})=($

b-) **(1,0pt)** $f(n)=500*n^2+10000;g(n)=2^n$ **use dominância!!**

$500*n^2+10000\leqslant c*2^n$ para algum valor de c e n0.

Sabemos que uma função polinomial sempre cresce mais do que uma quadrática e portanto a desigualdade acima é obviamente verdadeira.

5-) **(1,5pt)** A recorrência $T(n)=\begin{cases} k\ para\ n=1 \\ 2*T(\frac{n}{3})+20*n+5\ para\ n>1 \end{cases}$ descreve a complexidade de tempo de um algoritmo. Use o teorema mestre para demonstrar a que classe de complexidade o algoritmo pertence.

$a=2,b=3,f(n)=20*n+5$

Obs: $0{,}5<\log_3 2<1$

Caso 2: ? $20*n+5\in\Theta(n^{\log_3 2})$ ? Não

Caso 1:? $20*n+5\in O(n^{\log_3 2-\epsilon})$ ? Não.

Caso 3:? $20*n+5\in\Omega(n^{\log_3 2+\epsilon})$ ? Não pois para $\epsilon=1$

$20*n+5=n$ e $n<n^{\log_3 2+1}$ , ou seja, f(n) não é ômega-grande.

Portanto a complexidade de tempo deve ser exponencial pois não se enquadra em nenhum caso do Teorema Mestre.

6-) **(1,5pt)** A sequência de operações nas questões 1 a 5 apresentam uma forma de criar e analisar algoritmos. Explique como cada passo se conecta ao outro a fim de cumprir seu objetivo **(1pt)**. Você deseja resolver um problema e tem dois algoritmos para isso. Um com complexidade O(log(n)) e outro O(n^2). **Demonstre** qual dos dois é melhor (**0,5pt**).

As recorrências são extraídas a partir de algoritmos de divisão e conquista a fim de obter a complexidade de tempo, que comparamos a fim de saber qual algoritmo é melhor. Em ordem crescente de classe de complexidade temos:

$O(c)\in O(\log(n))\in O(n)\in O(n*\log(n))\in O(n^c)\in O(2^n)$

considerando a relação acima, o algoritmo com complexidade de tempo O(log(n)) é melhor que o com complexidade O(n^2).