FCT/Unesp – Presidente Prudente
Projeto e Análise de Algoritmos
Prof. Danilo Medeiros Eler

Exercícios Aula 03 – Parte I
Folha de apoio para o vídeo de resolução
https://daniloeler.github.io/teaching/PAA2020/index.html

1) Verifique se cada questão abaixo é verdadeira ou falsa.

(a) $10^{56} \cdot n^2 \in O(n^2)$?
(b) $10^{56} \cdot n^2 \in O(n^3)$?
(c) $10^{56} \cdot n^2 \in O(n)$?
(d) $2^{n+1} \in O(2^n)$?
(e) $2^{2n} \in O(2^n)$?
(f) $n \in O(n^3)$?

(a) $10^{56}n^2 = O(n^2)$ – Verdadeiro

$10^{56}n^2 <= cn^2$; $c = 10^{56}$
$10^{56}n^2 <= 10^{56}n^2$; $c = 10^{56}$

(b) $10^{56}n^2 = O(n^3)$ – Verdadeiro
$10^{56}n^2 = O(n^3)$
$10^{56}n^2 <= cn^3$; $c = 1$
$10^{56}n^2 <= 1n^3$
$10^{56}n^2 / n^3 <= 1$
$10^{56}/ n <= 1$

(c) $10^{56}n^2 = O(n)$ – Falso
$10^{56}n^2 <= cn$
$10^{56}n^2 / n <= c$
$10^{56}n <= c$

(d) $2^{n+1} = O(2^n)$ – Verdadeiro
$2^{n+1} <= c2^n$
$2^n2^1 <= c2^n$; $c = 2$
$2^n2^1 <= 22^n$; $c = 2$

(e) $2^{2n} = O(2^n)$ – Falso
$2^{2n} <= c2^n$
$2^n2^n <= c2^n$
$2^n2^n / 2^n <= c$
$2^n <= c$

(f) $n = O(n^3)$ – Verdadeiro
$n <= cn^3$
$n/ n^3 <= c$
$1/n^2 <= c$

$n <= cn^3$; $c = 1$
$n <= n^3$; $c = 1$; $a = 1$

2) Coloque em ordem crescente de complexidade as principais classes de problemas listadas a seguir.

$$O(n!),\ O(n\ log\ n),\ O(n),\ O(log\ n),\ O(1),\ O(2^n),\ O(n^3), O(n^2)$$

O(1)
O(logn)
O(n)
O(nlogn)
$O(n^2)$
$O(n^3)$
$O(2^n)$
O(n!)

3) Expresse a função abaixo em termos da notação assintótica "O".

$$n^3/1000 - 100n^2 - 100n + 3$$

$O(n^3)$

4) Analise o algoritmo abaixo e identifique o seu pior caso usando a notação assintótica.

```
exibe_matriz_3D(M)
    for i ← 1 to comprimento_x[M]
        for j ← 1 to comprimento_y[M]
            for k ← 1 to comprimento_z[M]
                do escreva(M[i][j][k]))
```

Complexidade: ~~Θ~~(xyz)

5) Apresente a análise detalhada de complexidade do método **main1**. Neste caso, é necessário analisar a complexidade dos métodos **subAlgoritmo01** e **subAlgoritmo02**. Utilize a notação assintótica para indicar a complexidade do algoritmo. Lembre que **size()** é um método que retorna o número de elementos de uma lista.

Complexidade: $\Theta(n^{10})$

```
public void main1(ArrayList args){….. Θ(1) + 4004+4n+ Θ(n^10) = Θ(n^10)
        subAlgoritmo01(); ..... Θ(1)
        double x, y, z;..... 3
        for(int i = (1000+args.size()); i >= 0; i--){ ..... 4004+4n
        x=(double)i/2; y=(double)x/2; z=(double)x+y; ..... 3
        System.out.print(z); ..... 1
        }
        subAlgoritmo02(args);..... Θ(n^10)
}
public void subAlgoritmo01(){ ..... 4007 = Θ(1)
        double x, y, z;..... 3
        for(int i = 1000; i >= 0; i--){ ..... 4004
        x=i/2; y=x/2; z=x+y;..... 3
        System.out.print(z); ..... 1
        }
}
public void subAlgoritmo02(ArrayList args){….n + n^10+4007+4n^5 = Θ(n^10)
        for(int i = 0; i < args.size(); i++){..... n
          System.out.print(args.get(i));..... 1
        }
        for(int i = 0; i < Math.pow(args.size(),10); i++){..... n^10
          System.out.println("aloAlo");..... 1
        }
        double x, y, z;..... 3
        for(int i = 1000+Math.pow(args.size(),5); i >= 0; i--){ ..... 4004+4n^5
         x=(double)i/2; y=(double)x/2; z=(double)x+y;  ..... 3
         System.out.print(z); ..... 1
        }
}
```

6) Conforme o exercício anterior, apresente a análise detalhada de complexidade do algoritmo **main2**. Utilize a notação assintótica para indicar a complexidade. Lembre que **size()** é um método que retorna o número de elementos de uma lista.

Complexidade: $\Theta(n^3)$

```
public void main2(ArrayList args){…… Θ(1)+ Θ(n)+ Θ(n^3)+Θ(n^2) = Θ(n^3)
        subAlgoritmo01(); ..... Θ(1)
        subAlgoritmo02(args);..... Θ(n)
        subAlgoritmo03(args); ..... Θ(n^3)
        subAlgoritmo04(args); ..... Θ(n^2)
}

public void subAlgoritmo01(){ ..... 4007= Θ(1)
        double x, y, z;..... 3
        for(int i = 1000; i >= 0; i--){ ..... 4004
        x=i/2; y=x/2; z=x+y;..... 3
        System.out.print(z); ..... 1
        }
}
public void subAlgoritmo02(ArrayList args){..n+n+4007 = 2n+4007 = Θ(n)
        for(int i = 0; i < args.size(); i++){..... n
        System.out.print(args.get(i));..... 1
        }
       for(int i = 0; i < args.size(); i++){..... n
        System.out.print(args.get(i));..... 1
        }
        double x, y, z;..... 3
        for(int i = 1000; i >= 0; i--){ ..... 4004
        x=(double)i/2; y=(double)x/2; z=(double)x+y; ..... 3
        System.out.print(z); ..... 1
        }
}
public void subAlgoritmo03(ArrayList args){ ..... Θ(n^3)
        for(int i = 0; i < args.size(); i++){……n*n*n
          for(int j = 0; j < args.size(); j++){…….n*n
            for(int ki = 0; ki < args.size(); k++){…….n
                System.out.print("Alo mundo " + i*j*k);…1
            }
          }
        }
}

public void subAlgoritmo04(ArrayList args){  ..... Θ(n^2)
        for(int i = 0; i < Math.pow(args.size(),2); i++){..... n^2
          System.out.print("Alo mundo " + i); ..... 1
        }
}
```

7) Apresente a análise detalhada de complexidade dos subprogramas abaixo. Lembre que **size()** é um método que retorna o número de elementos de uma lista. Utilize a notação assintótica.

```
Pessoa busca(String nome){
  for (int i = 0; i< pessoas.size(); i++){ ..... .....n+1
       if (pessoas.get(i).getNome().equals(nome)) ..... .....1
          return pessoas.get(i);..... 1
  }
  return null;
}
```
Complexidade: Ω(1) O(n)

a)
```
void exibir(String nome){
   Pessoa p = busca(nome); ..... n+1
   if (p != null){ ..... 1
      p.exibirDados();..... 1
   }
  else{  System.out.println(“Pessoa não encontrada”);
  }
}
```
Complexidade: Ω(1) O(n)

b)
```
void exibir(String nome){
   if (busca(nome) != null){..... n+1
      busca(nome).exibirDados(); ..... n+1
   }
  else{
       System.out.println(“Pessoa não encontrada”);
  }
}
```
Complexidade: Ω(1) O(n)

c)

```
void atualizar(String nome, int idade, float salario){
     Pessoa p = busca(nome); ..... n+1
    if (p != null){ ..... 1
        p.setIdade(idade); ..... 1
        p.setSalario(salario); ..... 1
    }
   else{
         System.out.println(“Pessoa não encontrada”);
   }
}
```

Complexidade: Ω(1) O(n)

d)

```
void atualizar(String nome, int idade, float salario){
    if (busca(nome) != null){ ..... n+1
        busca(nome).setIdade(idade); ..... n+1
        busca(nome).setSalario(salario); ..... n+1
    }
   else{
         System.out.println(“Pessoa não encontrada”);
   }
}
```

Complexidade: Ω(1) O(n)

8) Apresente a análise detalhada de complexidade dos subprogramas abaixo. Utilize a notação assintótica para apresenta a análise.

Algoritmo UnicoElemento (A, n)
//Determina se todos os elementos de um dado vetor são distintos
//**Entrada:** um vetor **A** e o número de elementos do vetor **n**
//Saída: retorna "verdadeiro" se todos os elementos em **A** são distintos e "falso" caso contrário

```
Para i = 1 até n – 1 faça
   Para j = i+1 até n faça
      Se A[i] = A[j] retorne falso
retorne verdadeiro
```

i = 1	j = 2	n-1
i = 2	j = 3	n-2
i = 3	j = 4	n-3
.....		
i = n-3	j=n-2	3
i = n-2	j=n-1	2
i = n-1	j=n	1

1 +2 + 3 + ... + n-3 + n-2 + n-1

$$1+2+3+4+\ldots+(n-2)+(n-1)$$

$$\sum_{i=1}^{n-1} i = \left(\sum_{i=1}^{n} i\right) - n = \left(\frac{n(n+1)}{2}\right) - n$$

$$\frac{n^2+n}{2} - n = \frac{n^2+n-2n}{2}$$

$$\frac{n^2-n}{2} \qquad = O(n^2)$$

Complexidade: $\Omega(1)$ $O(n^2)$