Trabalho Teórico 6 - Bruno Rodrigues Faria

• Resolva as equações abaixo:

a) 2^{10} = 1024

b) lg(1024) = 10

c) lg(17) = 4,087

d) $\lceil lg(17) \rceil$ = 5

e) $\lfloor lg(17) \rfloor$ = 4

• Plote um gráfico com todas as funções abaixo:

a) f(n) = n^3

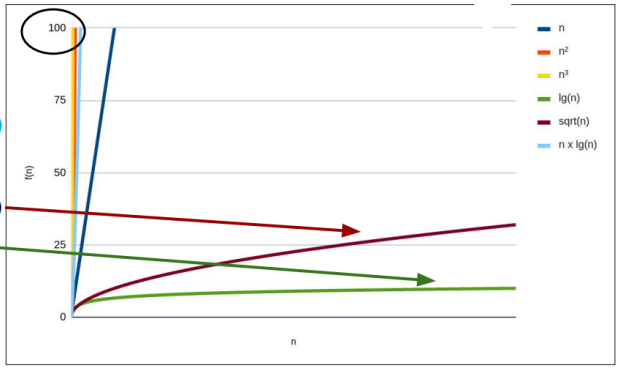
b) f(n) = n^2

c) f(n) = n x lg(n)

d) f(n) = n

e) f(n) = sqrt(n)

f) f(n) = lg(n)



• Calcule o número de subtrações que o código abaixo realiza:

```
...
for (int i = 0; i < n; i++){
    if (i % 2 == 0){
        a--;
        b--;
    } else {
        c--;
    }
}
```

Resposta: Melhor caso ele irá realizar n subtrações e no pior caso o código irá realizar 2n subtrações

• Calcule o número de subtrações que o código abaixo realiza:

(n-3) x 1 = (n-3)

```
...
for (int i = 3; i < n; i++){
    a--;
}
```

- Calcule o número de multiplicações que o código abaixo realiza:

```
for (int i = n; i > 0; i /= 2)
        a *= 2;
```

Resposta: Quando n é uma potência de 2 temos, que o código faz lg(n) + 1 multiplicações. Mas quando n é um valor qualquer temos, que o código faz [PISO]lg(n) + 1 multiplicações.

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];
for (int i = 1; i < n; i++){
    if (min > array[i]){
        min = array[i];
    }
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

R: Se tivermos n elementos: T(n) = n - 1

3º) O nosso T(n) = n – 1 é para qual dos três casos?

Todos os casos

```
int min = array[0];

for (int i = 1; i < n; i++){
    if (min > array[i]){
        min = array[i];
    }
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

R: Se tivermos n elementos: T(n) = n - 1

3º) O nosso T(n) = n – 1 é para qual dos três casos?

R: Para os três casos

4º) O nosso algoritmo é ótimo? Por que?

Testa tudo, pois assim irá garantir

```
boolean resp = false;
for (int i = 0; i < n; i++){
    if (array[i] == x){
        resp = true;
        i = n;
    }
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do array

2º) Quantas vezes ela será executada?

R: Melhor caso: f(n) = 1 → 1ª pos = X
Pior caso: f(n) = n percorre tudo
Caso médio: f(n) = (n + 1) / 2

3º) O nosso algoritmo é ótimo? Por que?

Sim, pois testa tudo para garantir a resposta

- Encontre o maior e menor valores em um *array* de inteiros e, em seguida, encontre a função de complexidade de tempo para sua solução

```
maior = array[0];
menor = array[0];
for(int i=1;i<n;i++){
    if(array[i] > maior){
        maior = array[i];
    }
    if(array[i] < menor) {
        menor = array[i];
    }
}

f(n) = 2(n-1);
O(n), θ(n), Ω(n);
```

- Um aluno deve procurar um valor em um *array* de números reais. Ele tem duas alternativas. Primeiro, executar uma pesquisa sequencial. Segundo, ordenar o *array* e, em seguida, aplicar uma pesquisa binária. O que fazer?

O aluno deve escolher a primeira opção pois tem custo n. A segunda tem custo (n * lg n). A terceira tem custo (lg n).

- Responda se as afirmações são verdadeiras ou falsas:

a) $3n^2 + 5n + 1$ é $O(n)$: Falsa
b) $3n^2 + 5n + 1$ é $O(n^2)$: Verdadeira
c) $3n^2 + 5n + 1$ é $O(n^3)$: Verdadeira
d) $3n^2 + 5n + 1$ é $\Omega(n)$: Verdadeira
e) $3n^2 + 5n + 1$ é $\Omega(n^2)$: Verdadeira
f) $3n^2 + 5n + 1$ é $\Omega(n^3)$: Falsa
g) $3n^2 + 5n + 1$ é $\Theta(n)$: Falsa
h) $3n^2 + 5n + 1$ é $\Theta(n^2)$: Verdadeira
i) $3n^2 + 5n + 1$ é $\Theta(n^3)$: Falsa

- Preencha verdadeiro ou falso na tabela abaixo:

	O(1)	O(lg n)	O(n)	O(n.lg(n))	$O(n^2)$	$O(n^3)$	$O(n^5)$	$O(n^{20})$
f(n) = lg(n)		X						
f(n) = n . lg(n)				X				
f(n) = 5n + 1			X		X	X	X	X
$f(n) = 7n^5 - 3n^2$							X	X
$f(n) = 99n^3 - 1000n^2$						X	X	X
$f(n) = n^5 - 99999n^4$							X	X

O → pior caso $g(n)$
→ melhor caso $f(n)$
→ exatamente

- Preencha verdadeiro ou falso na tabela abaixo:

	Ω(1)	Ω(lg n)	Ω(n)	Ω(n.lg(n))	$\Omega(n^2)$	$\Omega(n^3)$	$\Omega(n^5)$	$\Omega(n^{20})$
f(n) = lg(n)	X	X						
f(n) = n . lg(n)	X	X	X	X				
f(n) = 5n + 1	X	X	X					
$f(n) = 7n^5 - 3n^2$	X	X	X		X	X	X	
$f(n) = 99n^3 - 1000n^2$	X	X	X		X	X		
$f(n) = n^5 - 99999n^4$	X	X	X		X	X	X	

- Preencha verdadeiro ou falso na tabela abaixo:

	Θ(1)	Θ(lg n)	Θ(n)	Θ(n.lg(n))	$\Theta(n^2)$	$\Theta(n^3)$	$\Theta(n^5)$	$\Theta(n^{20})$
f(n) = lg(n)		X						
f(n) = n . lg(n)				X				
f(n) = 5n + 1			X					
$f(n) = 7n^5 - 3n^2$							X	
$f(n) = 99n^3 - 1000n^2$						X		
$f(n) = n^5 - 99999n^4$							X	

- Dado $f(n) = 3n^2-5n-9$, $g(n) = n*lg(n)$, $l(n) = n.lg^2(n)$ e $h(n) = 99n^8$, qual é a ordem de complexidade das operações:

a) $f(n) + g(n) - h(n)$
b) $O(f(n) + O(g(n)) - O(h(n))$
c) $f(n)$ *x* $g(n)$
d) $g(n)$ *x* $l(n) + h(n)$
e) $f(n)$ *x* $g(n)$ *x* $l(n)$
f) $O(O(O(O(f(n)))))$

a) f(n) + g(n) - h(n)

O(f(n)) + O(g(n)) - O(h(n))

O(maximo(f(n),g(n))) - O(h(n))

b) O(f(n) + O(g(n)) - O(h(n))

O(f(n) + g(n) - h(n))

c) f(n) x g(n)

O(f(n) x g(n))

d) g(n) x l(n) + h(n)

O(g(n) x l(n)) + O(h(n))

O(maximo(g(n)xl(n) + h(n)))

e) f(n) x g(n) x l(n)

O(f(n) x g(n)) x l(n)

O(f(n) x g(n) x l(n))

f) O(O(O(O(f(n)))))

O(O(O(f(n))))

O(O(f(n)))

O(f(n))

Dada a definição da notação O:

Mostre um valor c e outro m tal que, para n ≥ m, |3n2 + 5n +1| ≤ c x |n2|, provando que 3n^2 + 5n +1 é O(n2)

3n^2 + 5n +1 é Ω(n^2) pois para a análise de complexidade,

o que vale é a maior potência.

Mostre um valor c e outro m tal que, para n ≥ m, |3n2 + 5n +1| ≤ c x |n2|, provando que 3n^2 + 5n +1 é O(n^3)

O indica o limite superior, ou seja, O(n^2) também será O(n^3)

Prove que 3n^2 + 5n +1 não é O(n).

O indica o limite superior, e como 3n^2 + 5n +1 é O(n^2),

ele nao pode ser O(n)

Dada a definição da notação Ω:

Mostre um valor c e outro m tal que, para n ≥ m, |3n2 + 5n +1| ≤ c x |n2|,

provando que 3n^2 + 5n +1 é Ω(n2)

3n^2 + 5n +1 é Ω(n^2) pois para a análise de complexidade,

o que vale é a maior potência.

Mostre um valor c e outro m tal que, para n ≥ m, |3n2 + 5n +1| ≤ c x |n2|,

provando que 3n^2 + 5n +1 é Ω(n)

Ω indica o limite inferior, logo, Ω(n^2) também será Ω(n)

Prove que 3n^2 + 5n +1 não é O(n^3)

Como Ω indica o limite inferior, e já que 3n^2 + 5n +1 é Ω(n^2),

ele não pode ser Ω(n^3)

Dada a definição da notação Θ:

Mostre um valor c e outro m tal que, para n ≥ m, |3n2 + 5n +1| ≤ c x |n2|,

provando que 3n^2 + 5n +1 é Ω(n2)

3n^2 + 5n +1 é Θ(n^2) pois para a análise de complexidade,

o que vale é a maior potência.

Prove que 3n2 + 5n +1 não é Θ(n)

Θ é o limite justo, e já que 3n^2 + 5n +1 é Θ(n^2),

ele não pode ser Θ(n).

Prove que 3n2 + 5n +1 não é Θ(n^3)

Θ é o limite justo, e já que 3n^2 + 5n +1 é Θ(n^2),

ele não pode ser Θ(n^3).

RESUMO DE COMPLEXIDADE

1- Introdução

Dada a definição de um algoritmo como uma sequência de passos para chegar a uma solução, sabe-se que podem existir diversas, entretanto, precisamos medir aquela que será a melhor de todas em uma determinada situação. Para isso, utilizamos a Complexidade de Algoritmos para projetar algoritmos eficientes e prever a quantidade de recursos que ele irá demandar a medida que o tamanho do problema cresce, principalmente o tempo de execução do algoritmo

2- Analise de Algoritmos

Conta-se o número de operações relevantes realizadas por um algoritmo e expressa-se esse número como uma função de n. Essas operações podem ser comparações, operações aritméticas, movimento de dados, etc. Sobre a quantidade de recursos utilizados no algoritmo, podemos destacar três casos:

a) **Melhor caso:**
E o menor custo possível na execução de um algoritmo, normalmente as funções de melhor caso podem ser delimitadas inferiormente usando a notação assintótica Ω.

b) **Pior caso:**
Indica o maior tempo de execução de um algoritmo qualquer. A ordem de crescimento da complexidade de pior caso normalmente ´e usada para compara a eficiência de dois algoritmos. O pior caso ´e comumente mais utilizado pois o tempo de execução dele estabelece um limite superior para o tempo de execução para qualquer entrada, conhecê-lo nos garante que o algoritmo nunca demorará mais do que esse tempo esperado.

c) **Caso médio:**
E a quantidade de algum recurso computacional utilizado pelo algoritmo, numa média sobre todas as entradas possíveis.

3- Notações O, Ω e Θ

As propriedades do somatório facilitam o desenvolvimento das expressões algébricas, com o objetivo de chegar `as somas simples ou somas de quadrados

a) **Notação O:**
Usamos a notação O para dar um limite superior a uma função, dentro de um fator constante. Para todos os valores n em n0 ou `a direita de n0, o valor da função f(n) está abaixo de cg(n). Com a notação O, podemos descrever frequentemente o tempo de execução de um algoritmo apenas inspecionando a estrutura global do algoritmo.

b) **Notação Ω:**
Da mesma maneira que a notação O fornece um limite assintótico superior para uma função, a notação Ω nos d´a um limite assintótico inferior.

c) **Notação Θ:**
A notação Θ fornece uma simbologia simplificada para representar um limite justo de desempenho para um algoritmo. Um limite exato de tempo que um algoritmo leva para ser executado. Ou seja, a notação Θ representa o ponto de encontro entre as notações Ω (limite inferior) e Big O (limite superior).

Apresente a função e a complexidade para os números de comparações e

movimentações de registros para o pior e melhor caso:

```
void imprimirMaxMin( int [] array, int n){
    int maximo, minimo;

    if (array[0] > array[1]){
        maximo = array[0];    minimo = array[1];
    } else {
        maximo = array[1];    minimo = array[0];
    }

    for (int i = 2; i < n; i++){
        if (array[i] > maximo){
            maximo = array[i];
        } else if (array[i] < minimo){
            minimo = array[i];
        }
    }
}
```

Função de complexidade

	MOV	CMP
PIOR	f(n) = 2 + (n – 2)	f(n) = 1 + 2(n – 2)
MELHOR	f(n) = 2 + (n – 2) x 0	f(n) = 1 + (n – 2)

Complexidade

	MOV	CMP
PIOR	O(n), Ω(n) e Θ(n)	O(n), Ω(n) e Θ(n)
MELHOR	O(1), Ω(1) e Θ(1)	O(n), Ω(n) e Θ(n)

- Apresente a função e a complexidade para o número de subtrações para o pior e melhor caso

```
i = 0;

while (i < n) {
    i++;
    a--;
}

if (b > c) {
    i--;
} else {
    i--;
    a--;
}
```

	Função	**Complexidade**
PIOR	f(n) = n + 2	O(n), Ω(n) e Θ(n)
MELHOR	f(n) = n + 1	O(n), Ω(n) e Θ(n)

- Apresente a função e a complexidade para o número de subtrações para o pior e melhor caso

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a--;
        b--;
    }
    c--;
}
```

	Função	**Complexidade**
TODOS	f(n) = (2n + 1)n	O(n2), Ω(n2) e Θ(n2)

- Apresente a função e a complexidade para o número de subtrações para o pior e melhor caso

```
for (i = 0; i < n; i++) {
    for (j = 1; j <= n; j *= 2) {
        b--;
    }
}
```

Função	Complexidade
TODOS $f(n) = (\lg(n) + 1) * n = n * \lg(n) + n$	$O(n \times \lg(n))$, $\Omega(n \times \lg(n))$ e $\Theta(n \times \lg(n))$

- Apresente um código, defina duas operações relevantes e apresente a função e a complexidade para as operações escolhidas no pior e melhor caso

```
public static void sort(int[] array) {
        int n = array.length;
        for (int i = 1; i < n; i++) {
                int tmp = array[i];
                int j = i - 1;

                while ((j >= 0) && (array[j] > tmp)) {
                        array[j + 1] = array[j];
                        j--;
                }
                //pesqBin(vet, x);
                array[j + 1] = tmp;
        }
}

public static boolean pesqBin(int[] vet, int x) {
        boolean resp = false;
        int dir = (vet.length - 1), esq = 0, meio;

        while (esq <= dir) {
                meio = (esq + dir) / 2;
                if (x == vet[meio]) {
                        resp = true;
                        esq = dir + 1;
                } else if (x > vet[meio]) {
                        esq = meio + 1;
                } else {
                        dir = meio - 1;
                }
        }
```

Insertion Sort
PIOR CASO: $f(n) = ((n-1)n)/2$
MELHOR CASO: $f(n) = n-1$

Pesquisa Binaria:
PIOR CASO: $f(n) = \lg(n)$
MELHOR CASO: $f(n) = 1$

- Apresente o tipo de crescimento que melhor caracteriza as funções abaixo (Khan Academy, adaptado)

	Constante	Linear	Polinomial	Exponencial
$3n$		✓		
1	✓			
$(3/2)n$		✓		
$2n^3$			✓	
2^n				✓
$3n^2$			✓	
1000	✓			
$(3/2)^n$				✓

- Classifique as funções $f_1(n) = n^2$, $f_2(n) = n$, $f_3(n) = 2^n$, $f_4(n) = (3/2)^n$, $f_5(n) = n^3$ e $f_6(n) = 1$ de acordo com o crescimento, do mais lento para o mais rápido (Khan Academy, adaptado)

$$f_6(n) = 1$$

$$f_2(n) = n$$

$$f_1(n) = n^2$$

$$f_5(n) = n^3$$

$$f_4(n) = (3/2)^n$$

$$f_3(n) = 2^n$$

- Classifique as funções $f_1(n) = n.\log_6(n)$, $f_2(n) = \lg(n)$, $f_3(n) = \log_8(n)$, $f_4(n) = 8n^2$, $f_5(n) = n.\lg(n)$, $f_6(n) = 64$, $f_7(n) = 6n^3$, $f_8(n) = 8^{2n}$ e $f_9(n) = 4n$ de acordo com o crescimento, do mais lento para o mais rápido (Khan Academy, adaptado)

$$f_6(n) = 64$$
$$f_3(n) = \log_8(n)$$
$$f_2(n) = \lg(n)$$
$$f_9(n) = 4n$$
$$f_1(n) = n.\log_6(n)$$
$$f_5(n) = n.\lg(n)$$
$$f_4(n) = 8n^2$$
$$f_7(n) = 6n^3$$
$$f_8(n) = 8^{2n}$$

- Faça a correspondência entre cada função f(n) com sua g(n) equivalente, em termos de Θ. Essa correspondência acontece quando $f(n) = \Theta(g(n))$ (Khan Academy, adaptado)

f(n)	g(n)
$n + 30$	$3n - 1$
$n^2 + 2n - 10$	$n^2 + 3n$
$n^3 . 3n$	n^4
$\lg(n)$	$\lg(2n)$

- No Exercício Resolvido (10), verificamos que quando desejamos pesquisar a existência de ***um*** elemento em um *array* de números reais é adequado executar uma pesquisa sequencial cujo custo é $\Theta(n)$. Nesse caso, o custo de ordenar o *array* e, em seguida, aplicar uma pesquisa binária é mais elevado, $\Theta(n * \lg(n)) + \Theta(\lg(n)) = \Theta(n * \lg(n))$. Agora, supondo que desejamos efetuar ***n*** pesquisas, responda qual das duas soluções é mais eficiente

Nesse caso, a solução mais eficiente é ordenar e fazer pesquisa binária, pois com n pesquisas, a sequencial fica O(n^2), enquanto ordenar e pesquisar continua Θ(n * lg n) + Θ(lg n).