T198 – Construção e Análise de Algoritmos
Prof. Napoleão Nepomuceno
AV1 - Lab01

Exercício 1

- Passo 1: Implementar o seguinte código em Java ou equivalente em outra linguagem de programação. (0%)

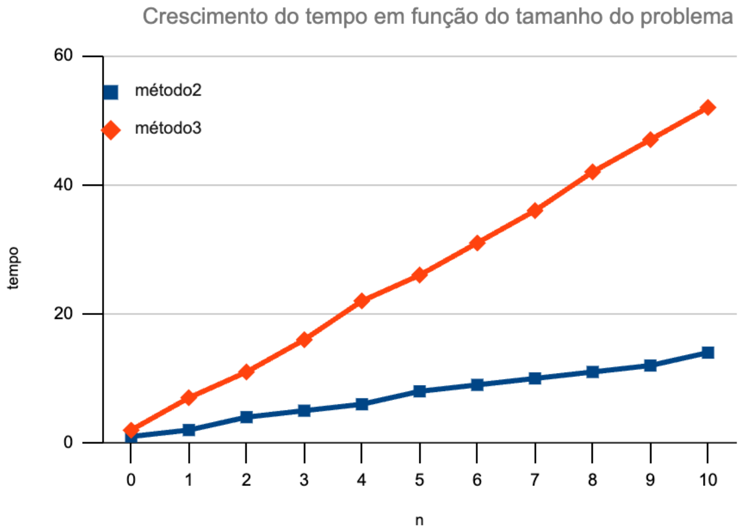
```
import java.util.concurrent.TimeUnit;
public class Exercicio1 {
	public static void main(String[] args) {
		System.out.printf("metodo1\n");
		System.out.printf("%10s%20s%10s\n","n", "solucao", "tempo");
		for (int n = 0; n <= 10; n+=1) {
			metodo1(n);
		}
		System.out.printf("metodo2\n");
		System.out.printf("%10s%20s%10s\n","n", "solucao", "tempo");
		for (int n = 0; n <= 10; n+=1) {
			metodo2(n);
		}
		System.out.printf("metodo3\n");
		System.out.printf("%10s%20s%10s\n","n", "solucao", "tempo");
		for (int n = 0; n <= 10; n+=1) {
			metodo3(n);
		}
		System.out.printf("metodo4\n");
		System.out.printf("%10s%20s%10s\n","n", "solucao", "tempo");
		for (int n = 0; n <= 10; n+=1) {
			metodo4(n);
		}
		System.out.printf("metodo5\n");
		System.out.printf("%10s%20s%10s\n","n", "solucao", "tempo");
		for (int n = 0; n <= 10; n+=1) {
			metodo5(n);
		}

	}

	static void metodo1 (long n) {
		double inicio = System.currentTimeMillis();
		long valor = 0;
		long termo = n * n * n;
		try {
			TimeUnit.MILLISECONDS.sleep(1);
		} catch (InterruptedException e) {
```

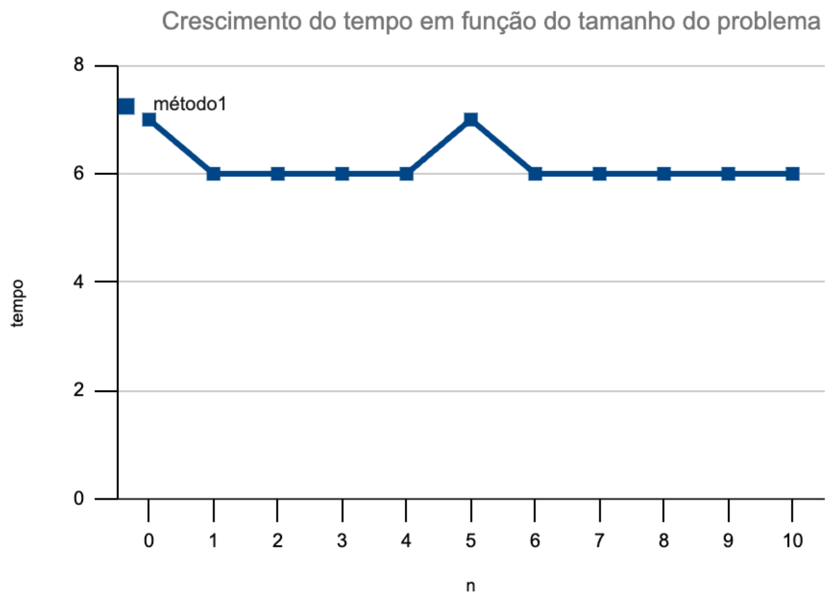
```
				e.printStackTrace();
			}
			for (long i = 1; i <= 4; i++) {
				valor += termo;
				try {
					TimeUnit.MILLISECONDS.sleep(1);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
			double fim = System.currentTimeMillis();
			double tempo = fim - inicio;
			System.out.printf("%10d%20d%10.0f\n", n, valor, tempo);
		}

		static void metodo2 (long n) {
			double inicio = System.currentTimeMillis();
			long valor = 0;
			long termo = 4 * n * n * n;
			try {
				TimeUnit.MILLISECONDS.sleep(1);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
			for (long i = 1; i <= n; i++) {
				valor += termo;
				try {
					TimeUnit.MILLISECONDS.sleep(1);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
			double fim = System.currentTimeMillis();
			double tempo = fim - inicio;
			System.out.printf("%10d%20d%10.0f\n", n, valor, tempo);
		}

		static void metodo3 (long n) {
			double inicio = System.currentTimeMillis();
			long valor = 0;
			long termo = n * n * n;
			try {
				TimeUnit.MILLISECONDS.sleep(1);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
			for (long i = 1; i <= 4; i++) {
				for (long j = 1; j <= n; j++) {
```

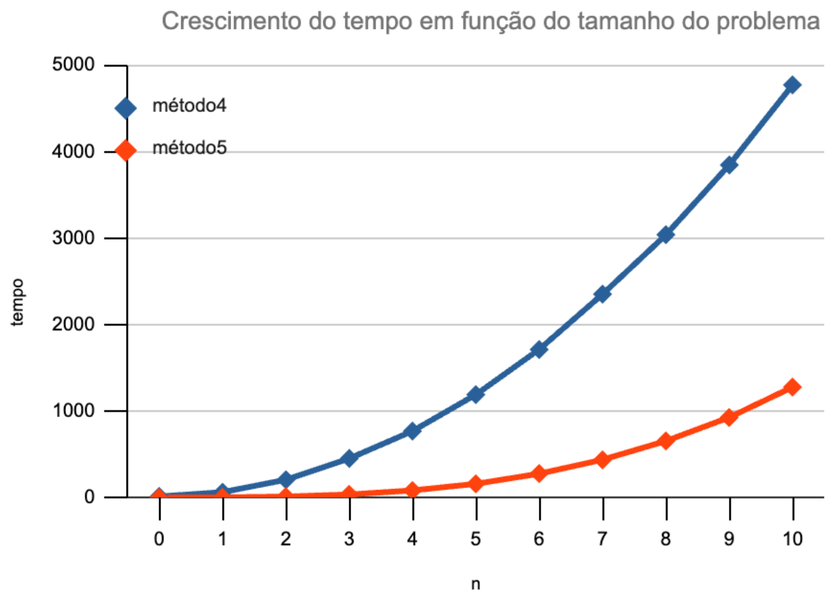
```
                valor += termo;
                try {
                    TimeUnit.MILLISECONDS.sleep(1);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        double fim = System.currentTimeMillis();
        double tempo = fim - inicio;
        System.out.printf("%10d%20d%10.0f\n", n, valor, tempo);
    }

    static void metodo4 (long n) {
        double inicio = System.currentTimeMillis();
        long valor = 0;
        long termo = n * n;
        try {
            TimeUnit.MILLISECONDS.sleep(1);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        for (long i = 1; i <= 2 * n; i++) {
            for (long j = 1; j <= 2 * n; j++) {
                valor += termo;
                try {
                    TimeUnit.MILLISECONDS.sleep(1);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        double fim = System.currentTimeMillis();
        double tempo = fim - inicio;
        System.out.printf("%10d%20d%10.0f\n", n, valor, tempo);
    }

    static void metodo5 (long n) {
        double inicio = System.currentTimeMillis();
        long valor = 0;
        long termo = 4 * n;
        try {
            TimeUnit.MILLISECONDS.sleep(1);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        for (long i = 1; i <= n; i++) {
            for (long j = 1; j <= n; j++) {
```

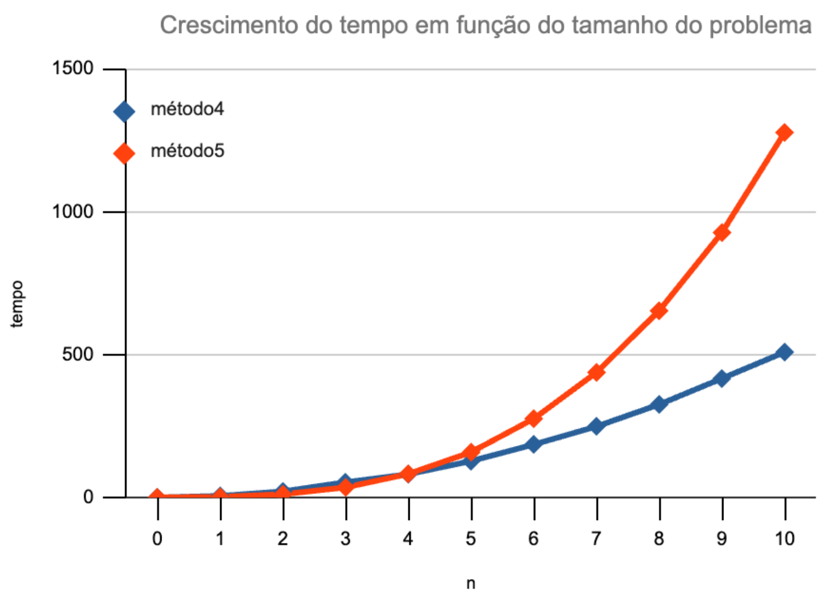
```
                    for (long k = 1; k <= n; k++) {
                        valor += termo;
                        try {
                            TimeUnit.MILLISECONDS.sleep(1);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
            double fim = System.currentTimeMillis();
            double tempo = fim - inicio;
            System.out.printf("%10d%20d%10.0f\n", n, valor, tempo);
        }

}
```

- Passo 2: Os diferentes métodos computam o valor de 4n^4. Executar o código e preencher o resultado na planilha disponibilizada (aba Exercicio1). Copiar os gráficos neste documento. (0%)


Crescimento do tempo em função do tamanho do problema
60
40
20
0
método2
método3
tempo
0
1
2
3
4
5
6
7
8
9
10
n


Crescimento do tempo em função do tamanho do problema
8
6
4
2
0
método1
tempo
0
1
2
3
4
5
6
7
8
9
10
n





- Passo 3: Realizar a análise de complexidade para cada um dos métodos. Desconsiderar na análise as instruções do try catch (utilizadas apenas para simular uma máquina mais lenta) e as de rastreamento do tempo de execução. (10%)

metodo 1: Θ(1)

c1	1
c2	1
c3	5
c4	4

c1 * 1 + c2 * 1 + c3 * 5 + c4 * 4 = T(N) = Θ(1) = Constante metodo 2: Θ(n)

c1	1
c2	1
c3	n+1
c4	n

c1 * 1 + c2 * 1 + c3 * n+1 + c4 * n = T(N) = Θ(N) = Linear

metodo 3: Θ(n)

c1	1
c2	1
c3	5
c4	4(n+1)
c5	4(n)

c1 * 1 + c2 * 1 + c3 * 5 + c4 * 4(n+1) = T(N) = Θ(n) = Linear

metodo 4:Θ(n^2)

c1	1
c2	1
c3	2n + 1
c4	2n
c5	2n (2n + 1)
c6	4n^2

c1 * 1 + c2 * 1 + c3 * (2n + 1) + c4 * 2n + c5 * (2n (2n + 1)) + c6 * (4n^2) = T(N) = Θ(n^2) = Quadrática

metodo 5:Θ(n^3)

c1	1
c2	1
c3	n+1
c4	n(n+1)
c5	n(n(n+1))
c6	n^3

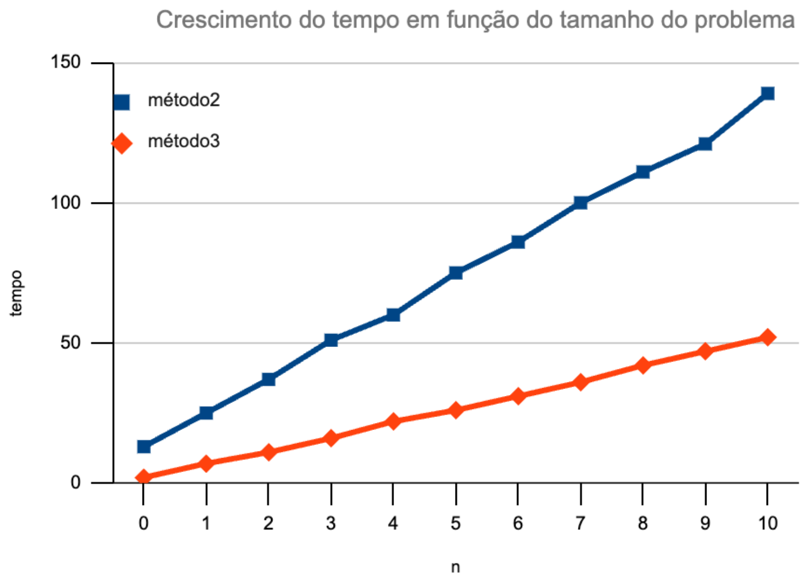
c1 * 1 + c2 * 1 + c3 * (n + 1) + c4 * (n(n+1)) + c5 * (n(n(n+1))) + c6 * n^3 = T(N) = Θ(n^3)

- Passo 4: Explicar a que se deve a variação de tempo de execução para o método metodo1? (2%)

A complexibilidade do método 1 é constante. Levando isso em consideração, a variação apresentada durante a execução se deve a influência de diversos fatores do sistema, como a carga da CPU, alocação de recursos entre outros.

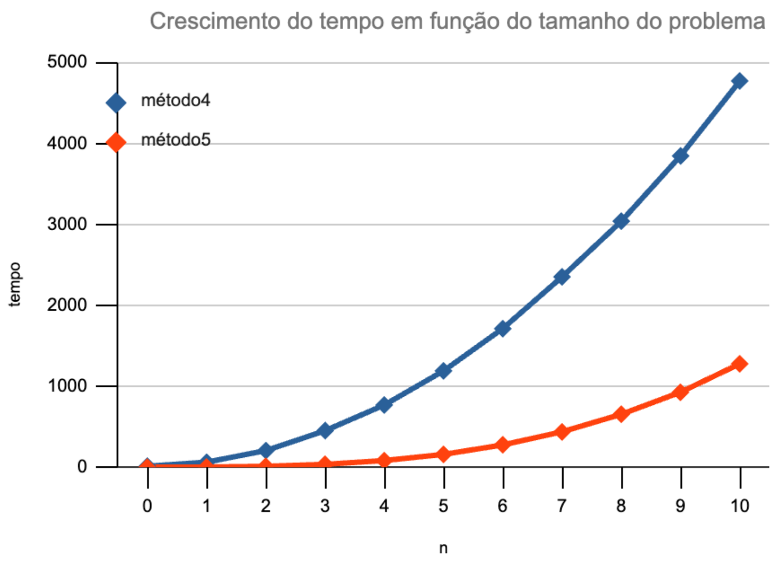
- Passo 5: Em seu experimento, qual método tem melhor tempo de execução: metodo2 ou metodo3? Para simular a execução do metodo2 em uma máquina 10 vezes mais lenta, modificar a instrução de sleep para TimeUnit.MILLISECONDS.sleep(10) apenas para este método, executar novamente o programa, alterar a planilha e copiar o gráfico respectivo neste documento. Neste novo experimento, qual método tem o melhor tempo de execução para n suficientemente grande: metodo2 ou metodo3? Explicar a que se deve este comportamento. (4%)

O primeiro experimento o metodo 2 possui melhor tempo de execução. Após modificar a instrução sleep de 1 para 10, o metodo3 passou a ter o melhor tempo de execução. O método 2 quando aumentamos o sleep, ele fica proporcional a N * 10, enquanto o método 3 permanece 4 * N.
Para valores suficientemente grandes de N, o método 2 terá maior tempo de execução por conta do seu tempo de espera. Já método 3 para valores suficientemente grandes se torna mais rápido, devido á sua estrutura de loop mais eficiente.



- Passo 6: Em seu experimento, qual método tem melhor tempo de execução: método 4 ou método 5? Para simular a execução do método 4 em uma máquina 10 vezes mais lenta, modificar a instrução de sleep para TimeUnit.MILLISECONDS.sleep(10) apenas para este método, executar novamente o programa, alterar a planilha e copiar o gráfico respectivo neste documento. Neste novo experimento, qual método tem o melhor tempo de execução para n suficientemente grande: metodo4 ou metodo5? Explicar este comportamento. (4%)

O primeiro experimento o método 4 possui melhor tempo de execução. Após modificar a instrução sleep de 1 para 10, o método 5 passou a ter o melhor tempo de execução para N pequenos, porém para N suficientemente grande o método 4 continua sendo mais rápido. O método 4 quando aumentamos o sleep, ele fica proporcional a 2 * N * 10, enquanto o método 5 permanece 4 * N * N * N * 1.
Para valores suficientemente grandes de N, o método 5 terá maior tempo de execução por conta do tempo de espera no loop mais interno.



- Passo 7: Indicar o nome dos integrantes da equipe que participaram efetivamente na resolução deste exercício. (0%)

Angelo Barcelos Rodrigues
Alberto
Alessandro
Vithor Vilas Boas
Ana Clara de Sá

Exercício 2

- Passo 1: Implementar o seguinte código em Java ou equivalente em outra linguagem de programação. (0%)

```
import java.util.concurrent.TimeUnit;

public class Exercicio1 {
	public static void main(String[] args) {
		double inicio1, fim1, tempo1;
		double inicio2, fim2, tempo2;
		double inicio3, fim3, tempo3;
		double inicio4, fim4, tempo4;
```

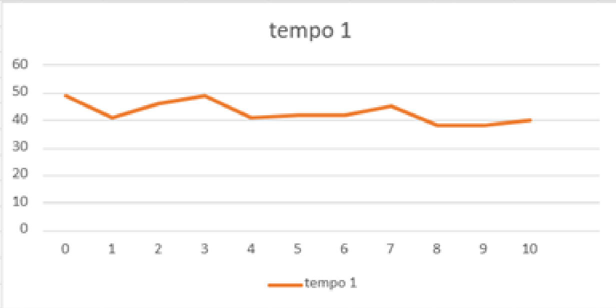
```
		System.out.printf("%5s%10s%10s%10s%10s\n","n", "tempo1", "tempo2",
"tempo3", "tempo4");
		System.out.println("-------------------------------------------");
		for (int n = 0; n <= 10; n+=1) {
			inicio1 = System.currentTimeMillis();
			metodo1(n);
			fim1 = System.currentTimeMillis();
			tempo1 = fim1 - inicio1;
			inicio2 = System.currentTimeMillis();
			metodo2(n);
			fim2 = System.currentTimeMillis();
			tempo2 = fim2 - inicio2;
			inicio3 = System.currentTimeMillis();
			metodo3(n);
			fim3 = System.currentTimeMillis();
			tempo3 = fim3 - inicio3;
			inicio4 = System.currentTimeMillis();
			metodo4(n);
			fim4 = System.currentTimeMillis();
			tempo4 = fim4 - inicio4;
			System.out.printf("%5d%10.0f%10.0f%10.0f%10.0f\n", n, tempo1,
tempo2, tempo3, tempo4);
		}

	}

	static void metodo1 (long n) {
		long valor = 0;
		try {
			TimeUnit.MILLISECONDS.sleep(1);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		for (long i = 10; i <= 12; i++) {
			for (long j = 4; j <= 10; j++) {
				valor += 1;
				try {
					TimeUnit.MILLISECONDS.sleep(1);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		}
	}

	static void metodo2 (long n) {
		long valor = 0;
```

```
		try {
			TimeUnit.MILLISECONDS.sleep(1);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		for (long i = 1; i <= n; i++) {
			for (long j = 1; j <= 3; j++) {
				valor += 1;
				try {
					TimeUnit.MILLISECONDS.sleep(1);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		}
	}

	static void metodo3 (long n) {
		long valor = 0;
		try {
			TimeUnit.MILLISECONDS.sleep(1);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		for (long i = 0; i < n; i++) {
			for (long j = 1; j <= n - i; j++) {
				valor += 1;
				try {
					TimeUnit.MILLISECONDS.sleep(1);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		}
	}

	static void metodo4 (long n) {
		long valor = 0;
		try {
			TimeUnit.MILLISECONDS.sleep(1);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		for (long i = 1; i <= n * n; i++) {
			for (long j = 1; j <= i; j++) {
				valor += 1;
				try {
					TimeUnit.MILLISECONDS.sleep(1);
```

```
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }

}
```

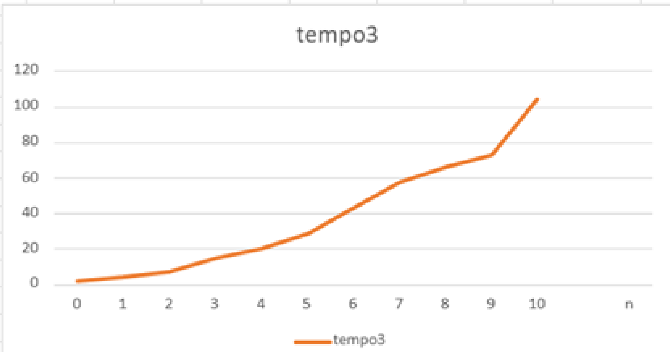
- Passo 2: Executar o código e preencher o resultado na planilha disponibilizada (aba Exercicio2). Copiar os gráficos neste documento. (0%)

n	tempo 1
0	49
1	41
2	46
3	49
4	41
5	42
6	42
7	45
8	38
9	38
10	40

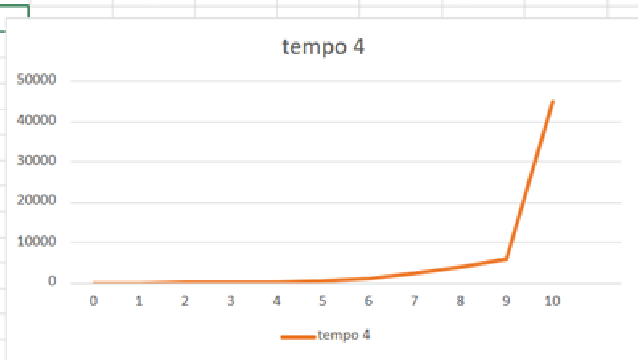


n	tempo 2
0	3
1	9
2	15
3	22
4	30
5	29
6	38
7	45
8	46
9	46
10	53

n	tempo3
0	2
1	4
2	7
3	15
4	20
5	29
6	43
7	58
8	66
9	73
10	104



n	tempo 4
0	2
1	5
2	25
3	90
4	258
5	635
6	1246
7	2555
8	3884
9	5925
10	44937



- Passo 3: Realizar a análise de complexidade para cada um dos métodos. Desconsiderar na análise as instruções do try catch (utilizadas apenas para simular uma máquina mais lenta) e as de rastreamento do tempo de execução. (10%)

metodo 1: Θ(1)

C1 1
C2 2 + 1
C3 6 + 1
C4 2 * 6

C1 * 1 + C2 * 3 + C3 * 7 + C4 * 12 = T(N) = Θ(1) = Constante

metodo 2: Θ(n)

C1 1
C2 N + 1
C3 2 + 1
C4 N*(2)

C1 * 1 + C2 * (N + 1) + C3 * 3 + C4 * 2N=T(N) = Θ(N) = Linear

Metodo3 : Θ(N^3)

C1: 1

C2: N+1

C3: (N*(N+1))/2

C4:N*N*N/2

C1 * 1 + C2 * N+1 + C3 * (N*(N+1))/2+ c4 * N*N*N = T(N) = Θ(N^3)

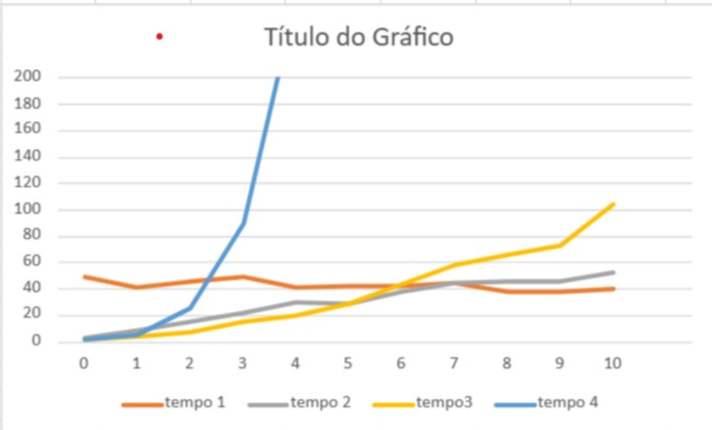
metodo 4:Θ(n^4)

C1 1
C2 Nˆ2 + 1
C3 Nˆ4+1
C4 Nˆ4

C1 * 1 + C2 * (Nˆ2 + 1) + C3 * (Nˆ4 + 1) + C4 * Nˆ4= T(N)=Θ(Nˆ4)

- Passo 4: Observando os gráficos obtidos e considerando as análises de complexidade assintótica: (1) discutir sobre as curvas de crescimento do tempo de execução de cada método; (2) indicar qual é o método assintoticamente mais eficiente; (3) indicar qual é o método assintoticamente menos eficiente; e (4) indicar a partir de que ponto o método mais eficiente passou a ser efetivamente mais rápido que os demais no experimento realizado. (5%)

empo 1	tempo 2	tempo3	tempo 4
49	3	2	2
41	9	4	5
46	15	7	25
49	22	15	90
41	30	20	258
42	29	29	635
42	38	43	1246
45	45	58	2555
38	46	66	3884
38	46	73	5925
40	53	104	44937



(1) - método 1: Θ(1) Crescimento constante, método 2: Θ(n)Crescimento Linear, Método 3 : Θ(N^3) Crescimento cúbico, método 4:Θ(n^4) Crescimento Biquadrático
(2) - método 1 é a mais eficiente
(3) - método menos eficiente é o 4
(4) - A partir do ponto 7 do eixo x, observamos que o método 1 se torna o mais eficiente

- Passo 5: Indicar o nome dos integrantes da equipe que participaram efetivamente na resolução deste exercício. (0%)

  Alberto Perdigão Lopes,
  Vithor Vilas Boas
  Iury Azevedo
  Ana Clara de Sá
  Alessandro

Exercício 3

- Passo 1: Implementar o seguinte código em Java ou equivalente em outra linguagem de programação. (0%)

```
import java.util.Random;
import java.util.concurrent.TimeUnit;

public class Exercicio2{
	public static void main(String[] args) {
		int n = 1000;
		int[] A;
		A = criaVetorAleatorio(n);
		double inicio, fim, tempo;
		inicio = System.currentTimeMillis();
		metodo(A, n);
		fim = System.currentTimeMillis();
		tempo = fim - inicio;
		System.out.printf("Tempo: %1.0f", tempo);
	}

	static double metodo (int[] vetor, int n) {
		double v = 1;
		for (int i = 0; i < n; i++) {
			try {
				TimeUnit.MILLISECONDS.sleep(1);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
			v = v * vetor[i];
			if (v == 0) {
				return 0;
			}
		}
		return v;
	}

	static int[] criaVetorAleatorio (int n) {
		Random randomGenerator = new Random();
		int[] A = new int[n];
		for (int i = 0; i < n; i++) {
			A[i] = randomGenerator.nextInt(100);
		}
		return A;
	}
}
```

- Passo 2: Dado um vetor, o que exatamente a função metodo está computando matematicamente? (2%)

Ela computa o produto de todos os valores contidos no vetor e retorna esse valor.

- Passo 3: Executar o código 10 vezes e copiar a saída de cada execução do programa aqui abaixo. Visto que o tamanho do problema não se modifica, o que justifica a grande variação do tempo de uma execução para outra? (3%)

Tempo: 42	Tempo: 16
Tempo: 74	Tempo: 37
Tempo: 232	Tempo: 106
Tempo: 80	Tempo: 33
Tempo: 20	Tempo: 148

Essa grande variância no tempo de execução é dada pela aleatoriedade dos termos do vetor e o fato de existir um melhor e pior caso dentro da função "metodo".

- Passo 4: Realizar a análise de complexidade de melhor e pior casos para o método. Obs.: Desconsiderar na análise as instruções do try catch (utilizadas apenas para simular uma máquina mais lenta). (3%)

Melhor caso:

$c1$ 1
$c2$ 1
$c3$ 1
$c4$ 1
$c5$ 1
$c6$ 0

$T(n) = c1 + c2 + c3 + c4 + c5 = \Theta(1)$

pior caso:

$c1$ 1
$c2$ $n+1$
$c3$ n
$c4$ n
$c5$ 0
$c6$ 1

$T(n) = c1 + (n+1)c2 + n*c3 + n*c4 + c6 = \Theta(n)$

- Passo 5: Se o vetor A, em vez de 1000 elementos, tivesse 1.000.000 elementos, a complexidade do algoritmo aumentaria? Justificar. (2%)

Não, o que aumentaria seria o tempo de execução e não a complexidade, pois a complexidade independe do tamanho da instância.

- Passo 6: Indicar o nome dos integrantes da equipe que participaram efetivamente na resolução deste exercício. (0%)

  Ana Clara de Sá
  Vithor Vilas
  Angelo Rodrigues
  Iury azevedo
  Alessandro

Exercício 4

- Passo 1: Considerar o seguinte código em Java ou equivalente em outra linguagem de programação. (0%)

```
import java.util.Random;
import java.util.concurrent.TimeUnit;

public class Exercicio1 {
	public static void main(String[] args) {

		int[] A;
		double inicio1, fim1, tempo1;
		double inicio2, fim2, tempo2;

		System.out.printf("%5s%10s%10s%10s%10s\n","n", "soma1", "tempo1",
"soma2", "tempo2");
		System.out.println("-------------------------------------------");
		for (int n = 1; n <= 50; n++) {
			A = criaVetorAleatorio(n);
			inicio1 = System.currentTimeMillis();
			int soma1 = soma1(A, n);
			fim1 = System.currentTimeMillis();
			tempo1 = fim1 - inicio1;
			inicio2 = System.currentTimeMillis();
			int soma2 = soma2(A, 0, n-1);
			fim2 = System.currentTimeMillis();
			tempo2 = fim2 - inicio2;
```

```
                    System.out.printf("%5d%10d%10.0f%10d%10.0f\n", n, soma1,
tempo1, soma2, tempo2);
            }
      }

      static int soma1 (int[] vetor, int n) {
            int total = 0;
            for (int i = 0; i < n; i++) {
                  try {
                        TimeUnit.MILLISECONDS.sleep(1);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }
                  total = total + vetor[i];
            }
            return total;
      }

      static int soma2 (int[] vetor, int i, int f) {
            if (i == f) {
                  return vetor[i];
            } else {
                  int m = (i+f) / 2;
                  try {
                        TimeUnit.MILLISECONDS.sleep(2);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }
                  return soma2(vetor, i, m) + soma2(vetor, m+1, f);
            }
            2* (t/2) + Θ(1)

      }
      static int[] criaVetorAleatorio (int n) {
            Random randomGenerator = new Random();
            int[] A = new int[n];
            for (int i = 0; i < n; i++) {
                  A[i] = randomGenerator.nextInt(100*n);
            }
            return A;
      }

}
```

- Passo 2: Realizar a análise de complexidade da função soma1. (1%)
  c1   1
  c2   n + 1

c3 n
c4 1

T(n) = Θ(1) = constante

- Passo 3: Montar a equação de recorrência para a função soma2. (2%)
  2* (t/2) + Θ(1)

- Passo 4: Resolver a equação de recorrência pelo teorema mestre. (2%)
  N^(log2 base2)
  n^1
  Θ(N)
  T(n) = Θ(N)

- Passo 5: Considerar o seguinte código em Java ou equivalente em outra linguagem de programação. (0%)

```
import java.util.concurrent.TimeUnit;

public class Exercicio2 {
	public static void main(String[] args) {
		double inicio1, fim1, tempo1;
		double inicio2, fim2, tempo2;
		System.out.printf("%5s%20s%10s%20s%10s\n","n", "pot1", "tempo1", "pot2",
"tempo2");
		System.out.println("-----------------------------------------------------------------");
		for (int n = 1; n <= 30; n++) {
			inicio1 = System.currentTimeMillis();
			int pot1 = potencia1(2, n);
			fim1 = System.currentTimeMillis();
			tempo1 = fim1 - inicio1;
			inicio2 = System.currentTimeMillis();
			int pot2 = potencia3(2, n);
			fim2 = System.currentTimeMillis();
			tempo2 = fim2 - inicio2;
			System.out.printf("%5d%20d%10.0f%20d%10.0f\n", n, pot1, tempo1,
pot2, tempo2);
		}
	}

	static int potencia1 (int a, int n) {
		int total = 1;
		for (int i = 1; i <= n; i++) {
			try {
				TimeUnit.MILLISECONDS.sleep(1);
			} catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
            total = total * a;
        }
        return total;
    }

    static int potencia2 (int a, int n) {
        if (n == 0) {
            return 1;
        } else {
            try {
                TimeUnit.MILLISECONDS.sleep(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            int aux = potencia2 (a, n/2);
            if (n % 2 == 0) {
                return aux * aux;
            } else {
                return aux * aux * a;
            }
        }
    }
}
```

- Passo 6: Realizar a análise de complexidade da função potencia1. (1%)
  C1: 1
  C2: N+1
  C3: N*1
  C4: 1
  T(n) = Θ(1) = constante

- Passo 7: Montar a equação de recorrência para a função potencia2. (2%)

  T(a, n) = T(a, n/2) + O(1)

- Passo 8: Resolver a equação de recorrência pelo teorema mestre. (2%)

- Passo 9: Indicar o nome dos integrantes da equipe que participaram efetivamente na resolução deste exercício. (0%)
  Ana Clara de Sá
  Angelo Rodrigues
  Iury azevedo
  Alessandro

Exercício 5

Q1. Suponha que dois algoritmos, A e B, resolvem um mesmo problema. Assuma ainda que o tamanho das instâncias do problema é dado por um parâmetro n. Para cada item abaixo, assumindo-se n suficientemente grande, indique se A é mais rápido que B para toda e qualquer instância, se B é mais rápido que A para toda e qualquer instância, ou se não podemos inferir qual dos dois algoritmos é mais rápido. Só serão pontuados os itens devidamente justificados. (10%)

- O algoritmo A consome tempo O(nˆ2) e o B consome tempo Ω(nˆ3).
  O "A" é mais eficiente, pois não existe uma intersecção entre os intervalos de complexibilidade de O(Nˆ2) e Ω(n3), sendo O(n2) sempre menor que Ω(n3).
- O algoritmo A consome tempo Ω(n2) e o B consome tempo O(n3).
  Impossível de inferir, pois existe uma intersecção entre os intervalos de complexibilidade de Ω(nˆ2) e O(nˆ3).
- O algoritmo A consome tempo Θ(n2) e o B consome tempo Θ(n3).
  O "A" é mais eficiente, complexibilidade de Θ(nˆ2) e Θ(nˆ3), sendo Θ(nˆ2) sempre menor que Θ(nˆ3).
- O algoritmo A consome tempo O(n2) e o B consome tempo O(n3).
  Impossível de inferir, pois existe uma intersecção entre os intervalos de complexibilidade de O(nˆ2) e O(nˆ3). Podendo A ser Θ(nˆ2) e B Θ(n), podendo ser tanto quanto B igual a Θ(nˆ2) e A Θ(n).
- O algoritmo A consome tempo O(n2) no pior caso e o B consome tempo Ω(n3) no melhor caso.
  No pior caso de A, ele ainda terá um consumo de tempo inferior ao de B no melhor caso.
- O algoritmo A consome tempo O(n3) no pior caso e o B consome tempo O(n2) no pior caso.
  Impossível de inferir, pois só é dado o limite superior.
- O algoritmo A consome tempo O(n2) no pior caso e o B consome tempo O(n3) no pior caso.
  Impossível de inferir, pois só é dado o limite superior.

- O algoritmo A consome tempo Ω(n2) no melhor caso e o B consome tempo Ω(n3) no melhor caso.

  Impossível de inferir, pois só é dado o limite inferior.

Q2. Aplique o método mestre para resolver as seguintes recorrências. (10%)

- T(n) = 4T(n/3) + n2
- T(n) = T(n/8) + 1
- T(n) = 8T(n/2) + n2
- T(n) = 16T(n/4) + n2

a - T(n) = Θ(n^2)

b - T(n) = Θ(logn)

c - T(n) = Θ(n^3)

d - T(n) = Θ(n^2log)

Q3. Dada o método abaixo, encontre um limite assintótico, utilizando notação Θ, para determinar sua complexidade. E qual o valor de retorno do método em função do valor da entrada? (7%)

```
int funcao(n)
   sum = 0;                               c1   1
   for (i = 1; i <= n; i++)               c2   n + 1
      for (j = 1; j <= i; j++)            c3  2 + (n + 1)* n / 2
         for (k = 1; k <= n*n; k++)       c4  (1+n)  * n / 2 (n^2 + 1)
            sum = sum + k;                c5   (1+n)  * n * (n^2) /2
   return sum
```

Θ(n^3)

Q4. Dada o método abaixo, encontre um limite assintótico, utilizando notação Θ, para determinar sua complexidade. E qual o valor de retorno do método em função do valor da entrada? (8%)

```
int funcao(n)
    sum = 0                                     c1        1
    for (i = 1; i <= n; i++)                    c2        n+1
                                                            i=1      i=2       i=3      ...   i=n
        for (j = i; j <= n*n; j++)              c3        (n^2+1)  (n^2)   (n^2-1)   ... (n^2-n+2) = [(n^2+1)+(n^2-n+2)]*(n)/2
            for (k = 1; k <= 5; k++)            c4         (5+1)   (5+1)    (5+1)    ...  (5+1) = (6+6)*{[(n^2)+(n^2-n+1)]*(n)/2}/2
                sum = sum + k                   c5           5       5        5      ...    5  = (5+5)*{[(n^2)+(n^2-n+1)]*(n)/2}/2
    return sum                                  c6        1
limite assintotico = θ(n^3)
```

Q5. Seja um vetor A de n elementos inteiros. É possível determinar o produto dos elementos do vetor em Θ(n) percorrendo-se os elementos do vetor de forma iterativa. Alternativamente, pode-se utilizar um método de divisão-e-conquista. Faça uma função recursiva para determinar o produto dos elementos do vetor. O algoritmo deve recursivamente dividir o vetor ao meio até se chegar a um caso trivial. Determine e resolva a equação de recorrência para o seu algoritmo. O algoritmo recursivo é mais eficiente do que o algoritmo iterativo? (10%

Passo 6: Indicar o nome dos integrantes da equipe que participaram efetivamente na resolução deste exercício. (0%)
Vithor Vilas Boas
Angelo Barcelos
Ana Clara de Sá
Iury Azevedo
Alberto
Alexandro