Relatório de Análise de Algoritmos-Aula 02(06/03/25)

Felipe Fazio da Costa; RA: 23.00055-4

Este relatório apresenta a análise dos algoritmos implementados para os exercícios propostos, seguindo os Modelos de Knuth, Detalhado e Simplificado. São fornecidos os códigos-fonte, tabelas de medições e gráficos das médias.

Código do Exercício 1

```
public class ex_1 {
   public static void main(String[] args) {
      int[] array_test = new int[10];
      for (int j = 0; j < 10; j++) {
         array_test[j] = 0;
      }
   }
}
```

Modelos de Knuth:

int j = 0: Sigma rec + Sigma arm

Simplificado: 2

j < 10: 2* Sigma rec + Sigma <

Simplificado: 3

j++: 2 * Sigma rec + Sigma"+" + Sigma arm

Simplificado: 4

*array_test[j] = 0: 3 * Sigma rec + Sigma . + Sigma arm*

Simplificado: *5*

*Total: 14 * n.*

Após deve-se multiplicar pelo tamanho do vetor, pois é o número de vezes que essas mesmas iterações vão acontecer.

Código do Exercício 2

```
public class ex_2 {
   public static void main(String[] args) {
      int[] array_test = new int[10];
      int[] array_copy = new int[10];
      for (int j = 0; j < 10; j++) {
         array_copy[j] = array_test[j];
      }
   }
}
```

Modelos de Knuth:

int j = 0: Sigma rec + Sigma arm

Simplificado: *2*

j < 10: 2 Sigma rec + Sigma <*

Simplificado: *3*

*j++: 2 * Sigma rec + Sigma"+" + Sigma arm*

Simplificado: *4*

*array_test[j] lado direito: 3 * Sigma rec + Sigma .*

Simplificado: *4*

*array_copy[j] lado esquerdo: 2 * Sigma rec + Simga . + Sigma arm*

Simplificado: *4*

*Total: 17 * n.*

Após deve-se multiplicar pelo tamanho do vetor, pois é o número de vezes que essas mesmas iterações vão acontecer.

Código do Exercício 3

```
public class ex_3 {
   public static void main(String[] args) {
      double matriz[][] = new double[10][10];
      for (int i = 0; i < 10; i++) {
```

```
        for (int j = 0; j < matriz[i].length; j++) {
          matriz[i][j] = -1;
        }
      }
    }
}
```

Modelos de Knuth:

int j = 0: Sigma rec + Sigma arm

Simplificado: *2*

j < 10: 2 Sigma rec + Sigma <*

Simplificado: *3*

*j++: 2 * Sigma rec + Sigma"+" + Sigma arm*

Simplificado: *4*

int i = 0: Sigma rec + Sigma arm

Simplificado: *2*

i < 10: 2 Sigma rec + Sigma <*

Simplificado: *3*

*i++: 2 * Sigma rec + Sigma"+" + Sigma arm*

Simplificado: *4*

*matriz[i][j] = -1: 4 * Sigma rec + Sigma . + Sigma arm*

Simplificado: *6*

*Total: 24 * n * m.*

Após deve-se multiplicar pelo tamanho do vetor, pois é o número de vezes que essas mesmas iterações vão acontecer.

Código do Exercício 4

```
public class ex_4{
   public static void main(String[] args) {
      int[][] matrix = new int[10][10];
      boolean isSymmetric = checkSymmetry(matrix);
```

```
  }
  private static boolean checkSymmetry(int[][] matrix) {
    int n = 10;
    for (int i = 0; i < n; i++) {
      for (int j = i + 1; j < n; j++) {
        if (matrix[i][j] != matrix[j][i]) {
          return false;
        }
      }
    }
    return true;
  }
}
```

Modelos de Knuth:

boolean isSymmetric = checkSymmetry(matrix, counter): Sigma chamada + Sigma retorno + 2 * Sigma arm + Sigma rec

Simplificado: 4

int n = 10: Sigma rec + Sigma arm

Simplificado: *2*

int j = 0: Sigma rec + Sigma arm

Simplificado: *2*

j < 10: 2 Sigma rec + Sigma <*

Simplificado: *3*

*j++: 2 * Sigma rec + Sigma"+" + Sigma arm*

Simplificado: *4*

int i = 0: Sigma rec + Sigma arm

Simplificado: *2*

i < 10: 2 Sigma rec + Sigma <*

Simplificado: *3*

*i++: 2 * Sigma rec + Sigma"+" + Sigma arm*

Simplificado: *4*

*matrix[i][j] != matrix[j][i]: 5 * Sigma rec + 2 * Sigma . + Sigma < + Sigma arm*

Simplificado: *9*

*Total: 33 * n*

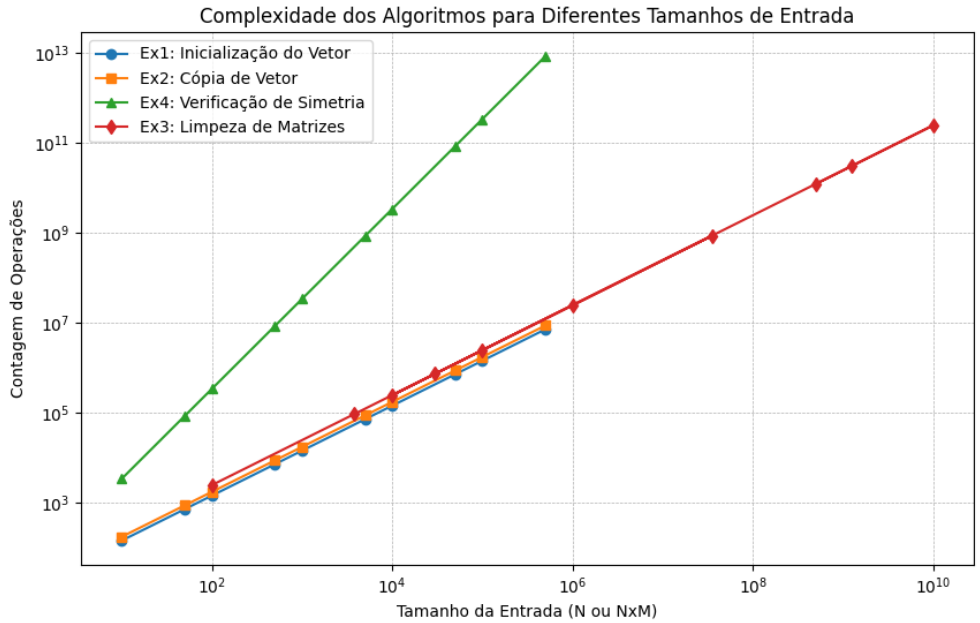
Após deve-se multiplicar pelo tamanho do vetor, pois é o número de vezes que essas mesmas iterações vão acontecer.

Tabelas de Medições

N	Ex1	Ex2	Ex4
10	140	170	3300
50	700	850	82500
100	1400	1700	330000
500	7000	8500	8250000
1000	14000	17000	33000000
5000	70000	85000	825000000
10000	140000	170000	3,3E+09
50000	*700000*	*850000*	8,25E+10
100000	1,4E+07	17000000	3,3E+11
500000	7000000	8500000	8,25E+12

NxM	Ex3
10x10	2400
50x75	90000
100x300	720000
500x200	2400000
1000x1000	2,4E+07
5000x7000	8,4E+08
10000x1	240000
50000x25000	3E+10
100000x100000	2,4E+11
500000x1000	1,2E+10

Gráfico de comparação



Assim Podemos afirmar que:

A vermelha, laranja e azul: O (n*log(n)) – Bad

A verde: O (n^c) - Worst

Conclusão:

Em termos de desempenho, os algoritmos com crescimento linear (Ex1 e Ex2) são mais eficientes para grandes valores de NNN, enquanto Ex3 e Ex4 apresentam maior custo computacional conforme o tamanho da entrada aumenta. Esse tipo de análise é essencial para otimizar implementações e escolher a melhor abordagem para problemas envolvendo grandes volumes de dados.