Algoritmos de Ordenação

Relatório do projeto | AEDS I – Prática

Aluno: João Antonio Lassister Melo **RA:** 2024.1.08.027

Introdução

O seguinte relatório tem como base a análise do desempenho de três algoritmos de ordenação, o BubbleSort, o InsertionSort e o SelectionSort, operando em vetores de ordenação aleatória, de ordenação crescente e de ordenação decrescente, cujo tamanho variava em intervalos regulares.

Referencial Teórico

Wikipedia e material disponibilizado no Google Classroom.

Material Utilizado

NetBeans IDE, para criação e implementação dos códigos.

Google Sheets, para criação das tabelas e dos gráficos.

Microsoft Word, para criação deste relatório.

Métodos Implementados

```
/*
 * Implementação do algoritmo BubbleSort.
 * Parâmetros:
 *   v[], é o endereço do vetor.
 *   tam, é o tamanho do vetor.
 * Retorno:
 *   n, é o número de operações realizadas.
 */
unsigned long int bubblesort(int v[], int tam) {
    int aux;
    unsigned long int n = 0;
    bool ordenado = false;
    for (int i = tam - 1; i > 0 && !ordenado; i--) {
        ordenado = true;
        for (int j = 0; j < i; j++) {
            if (v[j] > v[j + 1]) {
                ordenado = false;
                aux = v[j];
                v[j] = v[j + 1];
                v[j + 1] = aux;
                n = n + 4;
            }
            n = n + 2;
        }

    }

    return n;
}
```

```
/*
 * Implementação do algoritmo InsertionSort.
 * Parâmetros:
 *   v[], é o endereço do vetor.
 *   tam, é o tamanho do vetor.
 * Retorno:
 *   n, é o número de operações realizadas.
 */
unsigned long int insertionsort(int v[], int tam) {
    int aux, j;
    unsigned long int n = 0;
    for (int i = 1; i < tam; i++) {
        aux = v[i];
        j = i - 1;
        n = n + 1;
        while (j >= 0 && v[j] > aux) {
            v[j + 1] = v[j];
            j = j - 1;
            n = n + 2;
        }
        v[j + 1] = aux;
        n = n + 2;
    }

    return n;
}
```

```
/*
 * Implementação do algoritmo SelectionSort.
 * Parâmetros:
 *   v[], é o endereço do vetor.
 *   tam, é o tamanho do vetor.
 * Retorno:
 *   n, é o número de operações realizadas.
 */
unsigned long int selectionsort(int v[], int tam) {
    int aux, min;
    unsigned long int n = 0;
    for (int i = 0; i < tam - 1; i++) {
        min = i;
        aux = v[i];
        n = n + 1;
        for (int j = i + 1; j < tam; j++) {
            if (v[j] < aux) {
                min = j;
                aux = v[j];
                n = n + 1;
            }
            n = n + 1;
        }
        aux = v[i];
        v[i] = v[min];
        v[min] = aux;
        n = n + 4;
    }

    return n;
}
```

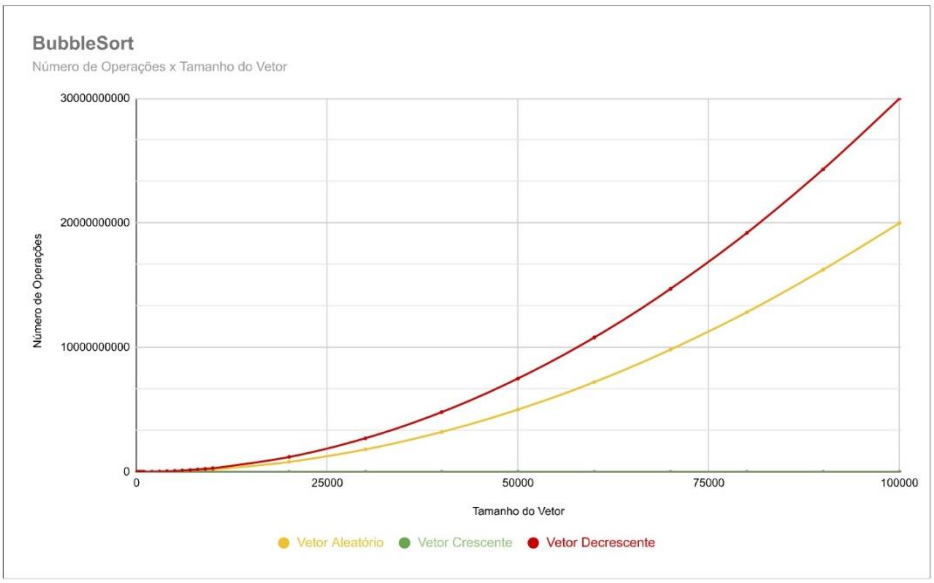
```
/*
 * Função para gerar um vetor desordenado e escreve-lo em um arquivo.
 * Parâmetros:
 *   vetor[], é o endereço do vetor.
 *   TAM, é o tamanho do vetor.
 * Retorno:
 */
void desordenado(int vetor[], int TAM) {
    srand(time(NULL));

    for (int i = 0; i < TAM; i++) {
        vetor[i] = -1;
    }

    int j = 0;
    for (int i = 0; i < TAM; i++) {
        int k = rand() % TAM + 1;
        if (j + k >= TAM) {
            j = (j + k) % TAM;
        } else
            j = j + k;
        if (vetor[j] == -1) {
            vetor[j] = i;
        } else {
            while (vetor[j] != -1) {
                j++;
                if (j >= TAM) {
                    j = (j + j) % TAM;
                }
            }
            vetor[j] = i;
    }

    ofstream arquivo("desordenado.txt");

    for (int i = 0; i < TAM; i++) {
        arquivo << vetor[i] << endl;
    }

    arquivo.close();
}
```

```
/*
 * Função para ler o vetor a partir do arquivo.
 * Parâmetros:
 *   vetor[], é o endereço do vetor.
 *   TAM, é o tamanho do vetor.
 * Retorno:
 *   Retorna um valor inteiro para verificar se a execução foi bem sucs
 */
int leituraArquivo(int vetor[], int TAM) {
    ifstream arquivo("desordenado.txt");

    if (!arquivo.is_open()) {
        cout << "Erro. Arquivo inexistente." << endl;
        return 1;
    }

    for (int i = 0; i < TAM; i++) {
        arquivo >> vetor[i];
    }

    arquivo.close();

    return 0;
}
```

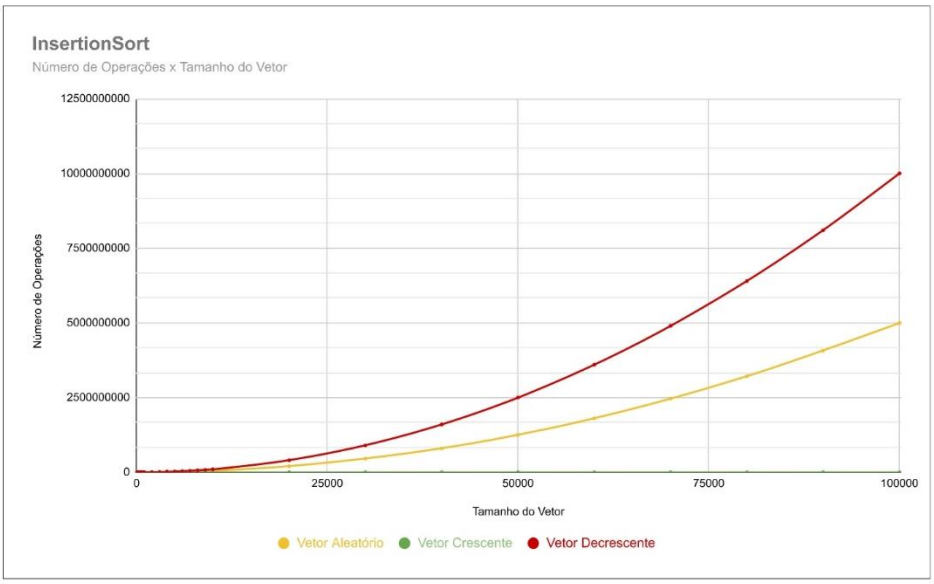
```
/*
 * Função para inverter a ordem dos valores do vetor.
 * Parâmetros:
 *   vetor[], é o endereço do vetor.
 *   TAM, é o tamanho do vetor.
 * Retorno:
 */
void inverter(int vetor[], int TAM) {
    int aux;
    for (int i = 0; i < TAM / 2; i++){
        aux = vetor[i];
        vetor[i] = vetor[TAM - 1 -i];
        vetor[TAM -1 -i] = aux;
    }
}
```

Resultados Obtidos

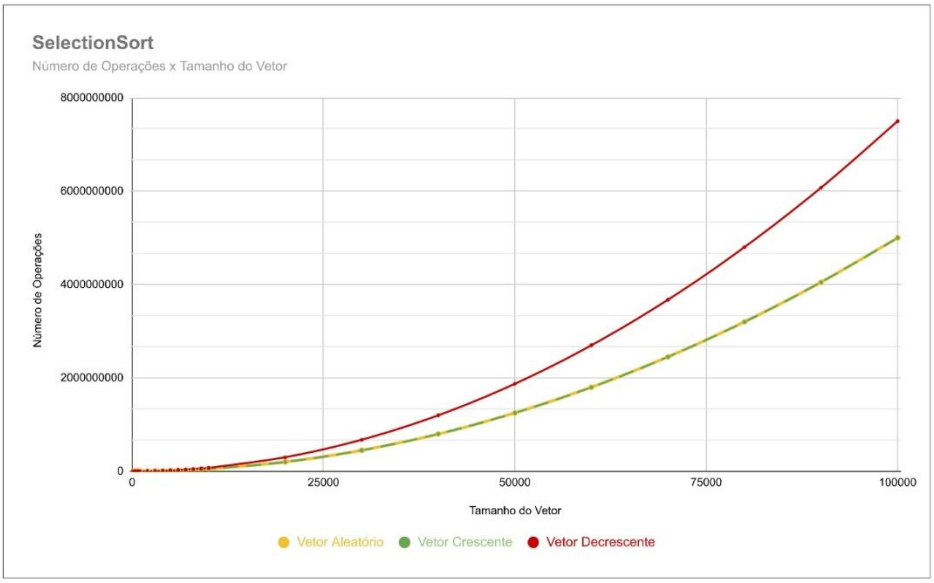
Tamanho Do Vetor	Op. Vetor Aleatório	Op. Vetor Crescente	Op. Vetor Decrescente
100	18578	198	29700
200	80996	398	119400
300	170112	598	269100
400	313224	798	478800
500	512442	998	748500
600	718070	1198	1078200
700	987774	1398	1467900
800	1281878	1598	1917600
900	1567050	1798	2427300
1000	2087736	1998	2997000
2000	8032940	3998	11994000
3000	17827388	5998	26991000
4000	32385628	7998	47988000
5000	49790258	9998	74985000
6000	71555556	11998	107982000
7000	97775162	13998	146979000
8000	128224084	15998	191976000
9000	161414112	17998	242973000
10000	197453176	19998	299970000
20000	804781746	39998	1199940000
30000	1815495054	59998	2699910000
40000	3200230078	79998	4799880000
50000	5002596330	99998	7499850000
60000	7208476854	119998	10799820000
70000	9829359488	139998	14699790000
80000	12827825702	159998	19199760000
90000	16248314260	179998	24299730000
100000	19985882636	199998	29999700000



Tamanho Do Vetor	Op. Vetor Aleatório	Op. Vetor Crescente	Op. Vetor Decrescente
100	4637	297	10197
200	21601	597	40397
300	41181	897	90597
400	78639	1197	160797
500	132989	1497	250997
600	181177	1797	361197
700	251487	2097	491397
800	324087	2397	641597
900	385413	2697	811797
1000	548185	2997	1001997
2000	2023603	5997	4003997
3000	4426021	8997	9005997
4000	8210297	11997	16007997
5000	12415401	14997	25009997
6000	17802261	17997	36011997
7000	24412403	20997	49013997
8000	32140339	23997	64015997
9000	40258653	26997	81017997
10000	48766241	29997	100019997
20000	202475235	59997	400039997
30000	457861569	89997	900059997
40000	800264627	119997	1600079997
50000	1251487527	149997	2500099997
60000	1804505715	179997	3600119997
70000	2464976101	209997	4900139997
80000	3214217379	239997	6400159997
90000	4074489147	269997	8100179997
100000	4993308893	299997	10000199997



Tamanho Do Vetor	Op. Vetor Aleatório	Op. Vetor Crescente	Op. Vetor Decrescente
100	5759	5445	7945
200	21615	20895	30895
300	47579	46345	68845
400	83474	81795	121795
500	129655	127245	189745
600	185549	182695	272695
700	251656	248145	370645
800	327759	323595	483595
900	413555	409045	611545
1000	509886	504495	754495
2000	2021246	2008995	3008995
3000	4533583	4513495	6763495
4000	8044201	8017995	12017995
5000	12557359	12522495	18772495
6000	18069044	18026995	27026995
7000	24583217	24531495	36781495
8000	32094165	32094165	48035995
9000	40608481	40540495	60790495
10000	50120310	50044995	75044995
20000	200256792	200089995	300089995
30000	450396932	450134995	675134995
40000	800538768	800179995	1200179995
50000	1250683329	1250224995	1875224995
60000	1800840075	1800269995	2700269995
70000	2450981168	2450314995	3675314995
80000	3201145855	3200359995	4800359995
90000	4051284856	4050404995	6075404995
100000	5001445665	5000449995	7500449995



Conclusão

BubbleSort:

• Pior algoritmo para vetores de ordem aleatória, apresentando um desempenho bastante inferior em relação aos demais.

• Melhor algoritmo para vetores de ordenação crescente ou próximos a isso, apresentando uma linha no gráfico quase que paralela ao eixo horizontal, com um número de operações igual a

2n – 2 (n sendo o tamanho do vetor).

• Pior algoritmo para vetores de ordenação decrescente ou próximos a isso, apresentando um desempenho bastante inferior em relação aos demais, com um número de operações igual a **$3n^2$ - 3n** (n sendo o tamanho do vetor).

InsertionSort:

• Melhor algoritmo para vetores de ordem aleatória junto com o SelectionSort, ambos apresentando desempenhos similares.

• Segundo melhor algoritmo para vetores de ordenação crescente, apresentando um desempenho próximo ao BubbleSort ainda que inferior, com um número de operações igual a

3n – 3 (n sendo o tamanho do vetor).

• Segundo melhor algoritmo para vetores de ordem decrescente, apresentando um desempenho próximo ao SelectionSort ainda que inferior, com um número de operações igual a

n^2 + 2n – 3 (n sendo o tamanho do vetor).

SelectionSort:

• Melhor algoritmo para vetores de ordem aleatória junto com o InsertionSort, ambos apresentando desempenhos similares.

• Pior algoritmo para vetores de ordenação crescente, apresentando um desempenho semelhante ao com um vetor de ordem aleatória, com as curvas do gráfico se sobrepondo, sendo bastante inferior aos demais, com um número de operações igual a **n^2 / 2 + 9n / 2 – 5** (n sendo o tamanho do vetor)

• Melhor algoritmo para vetores de ordem decrescente, com um número de operações igual a **$3n^2$ / 4 + 9n / 2 – 5** (n sendo o tamanho do vetor).

Conclusões finais:

• Vetor de ordem aleatória: Algoritmo mais eficiente **InsertionSort** (**SelectionSort**).

• Vetor de ordenação crescente: Algoritmo mais eficiente **BubbleSort**.

• Vetor de ordenação decrescente: Algoritmo mais eficiente **SelectionSort**.