UNIVERSIDADE FEDERAL DO TRIÂNGULO MINEIRO – CÂMPUS UBERABA – ICTE 2

Aluno: Arthur Lopes Morais Arantes

R.A: 2018.1055.4

ALGORITMOS DE SORT - ORDENAÇÃO

Docente: Elder Vicente de Paulo Sobrinho

Uberaba – MG

Novembro – 2019

1. INTRODUÇÃO

No mundo da computação é essencial entregar um código limpo, bem estruturado e com um processo de compilação em um menor e armazenamento possível (mas é praticamente impossível obter todos esses resultados ao mesmo tempo). Portanto, para obter um código eficiente e otimizado (afim de alcançar bons resultado citados anteriormente, deve-se analisar os algoritmos específicos de cada área e comparar cada um, sendo definido por busca, ordenação e outros.

É importante salientar que para se ter um resultado plausível, em relação ao processo de compilação e armazenamento, analisa-se a quantidade de termos presentes no algoritmo, pois, um algoritmo de tamanho 5 seria sempre menor que 1000. Outro fato seria a junção da quantidade de termos com a complexidade, notação BigOnotation, do algoritmo, seja para um caso médio ou bom.

Nesse caso, afunilando mais o assunto, tem-se os algoritmos de ordenação: Quick Sort, Merge Sort, Selection Sort, Bubble Sort e Insertion Sort, sendo os 2 primeiros sando recursividade e os 3 últimos usando apenas vetores.

2. MÉTODO

Antes de inicializar a implementação do algoritmo, foi explicada a matéria de Algoritmos de Ordenação pelo professor Elder. Após isso, foi passado aos discentes um trabalho sobre os algoritmos de ordenação, pedindo para comparar com 4 vetores de diferentes tamanhos.

Logo depois, foi implementado o código com o auxílio do CodeBlocks e DevC++ -pois em alguns casos o CodeBlocks dava erros – e com o auxílio da biblioteca Time.H foi medido o tempo de execução de cada algoritmo. Foi realizado uma quantidade de 5 interações, mas nesse caso, o usuário estaria disposto a fazer o tanto que quisesse. Também, foi utilizado 2 vetores, um original e outro cópia. A cópia sempre seria parte da ordenação, já a original apenas serviria para fazer a cópia. Nesse caso, se não colocasse um vetor cópia, o vetor sempre estaria ordenado, afetando outros algoritmos, pois suas ações sempre teriam tempo 0.

Por meio disso, foi possível a plotagem de gráfico e término de código, demonstrado a seguir:

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <conio.h>
#include <time.h>
#include <math.h>



void copia ( int vetormagico [], int vetormalabarista [], int tamanho)
{
   for (int conttransferencia = 0; conttransferencia<tamanho; conttransferencia++)
   {
```

```
        vetormalabarista [conttransferencia] = vetormagico [conttransferencia];
    }
}



void tempomedio(double tempos[], double med, int iteraor, char tipo[])
{
    int con;

    printf("\nTempo médio gasto para o ordenador %s: ", tipo);

    for(con=0; con<iteraor; con++)
    {
        med = med + tempos[con];
    }
    printf("%lf\n", med/iteraor);
}




void BubbleSort(int vetormagico[], int tamanho)
{
    int i, j, temporario;
    for (i=0; i<tamanho-1; i++)
    {
        for (j=0; j<tamanho-1-i ; j++)
        {
            if (vetormagico[j+1]<vetormagico[j])
            {

                temporario = vetormagico[j+1];
                vetormagico[j+1] = vetormagico [j];
                vetormagico[j] = temporario;
            }
        }
    }
}




void InsertionSort(int vetormagico[], int tamanho)
{
    int i, j, temporario;
    for (i=1; i<tamanho; i++)
    {
        temporario = vetormagico[i];
        j = i-1;
        while ((temporario <  vetormagico[j]) && (j>=0))
        {
            vetormagico[j+1] = vetormagico[j];
            j--;
        }
        vetormagico[j+1] = temporario;
    }
}



int SMALLEST (int vetormagico[], int tamanho, int movimento)
{
    int elemento = vetormagico[movimento], i;
    for (i=(movimento+1); i<tamanho; i++)
    {
        if (vetormagico[i]<elemento)
        {
            elemento = vetormagico[i];
            movimento = i;
        }
    }
```

```
    return movimento;
}

void SelectionSort(int vetormagico[], int tamanho)
{
    int movimento, posicao, temporario;
    for (movimento=0; movimento<tamanho; movimento++)
    {
        posicao = SMALLEST(vetormagico, tamanho, movimento);
        temporario = vetormagico[movimento];
        vetormagico[movimento] = vetormagico [posicao];
        vetormagico [posicao] = temporario;
    }
}



void MergeFazTudo (int vetormagico [], int beg, int meio, int end, int tamanho)
{
    int i = beg, j = meio+1, index = beg, temp[tamanho], k;
    while ((i<=meio)&& (j<=end))
    {
        if (vetormagico[i]< vetormagico[j])
            temp [index++] = vetormagico [i++];
        else
            temp [index++] = vetormagico [j++];
    }
    if (i>meio)
    {
        while (j<=end)
            temp [index++] = vetormagico [j++];
    }
    else
    {
        while (i<=meio)
            temp [index++] = vetormagico [i++];
    }
    for (k=beg; k<index; k++)
        vetormagico[k] = temp[k];
}


void MergeSort(int vetormagico [], int beg, int end, int tamanho)
{
    int meio;
    if (beg<end)
    {
        meio = floor ((beg+end)/2);
        MergeSort(vetormagico, beg, meio, tamanho);
        MergeSort(vetormagico, meio+1, end, tamanho);
        MergeFazTudo (vetormagico, beg, meio, end, tamanho);
    }
}




int PARTITION (int vetormagico [], int beg, int end)
{
    int esquerda = beg;
    int direita = end;
    int posicaorepartevetor = beg;
    int elemento = vetormagico[posicaorepartevetor];
    int temporario;
    while (esquerda < direita)
    {
        while (vetormagico[esquerda]<=elemento)
            esquerda ++;
        while (vetormagico[direita]>elemento)
            direita --;
        if (esquerda <direita)
        {
            temporario = vetormagico[esquerda];
```

```
            vetormagico[esquerda] = vetormagico[direita];
            vetormagico[direita] = temporario;
        }
    }
    vetormagico [beg] = vetormagico[direita];
    vetormagico [direita] = elemento;
    return direita;
}


void QuickSort(int vetormagico[], int beg, int end)
{
    int posicaorepartevetor;
    if (beg<end)
    {
        posicaorepartevetor = PARTITION (vetormagico, beg, end);
        QuickSort (vetormagico, beg,posicaorepartevetor-1);
        QuickSort (vetormagico,posicaorepartevetor+1, end);
    }
}




void main()
{
    srand(time(NULL));
    clock_t comeco;
    clock_t termino;
    double tempogasto, tempogasto2, tempogasto3, tempogasto4, tempogasto5;
    double med = 0;
    setlocale(LC_ALL, "");
    int tamanho;
    int iteraor;
    int elementomax;
    int k, contadoritera;



    printf ("Qual o tamanho do seu vetor ? ");
TAMANHO:
    scanf ("%d", &tamanho);
    if(tamanho<1)
    {
        printf("COMO SE QUER ORDENAR VETOR VAZIO OUU NEGATIVO ? TÁ DOIDO ??:\n");
        goto TAMANHO;
    }
    puts ("\n\n");



    printf ("Quantas interações cada algoritmo de ordenação tem que ter ? ");
ITERACAO:
    scanf ("%d", &iteraor);
    if(iteraor<1)
    {
        printf("COMO SE QUER INTERAGIR COM NÚMERO NEGATIVO? TÁ DOIDO ??:\n");
        goto ITERACAO;
    }
    puts ("\n\n");



    printf ("Qual o tamanho máximo de cada elemento do vetor?:");
ELEMENTOMAX:
    scanf ("%d", &elementomax);
    if(elementomax<1)
    {
        printf("SÓ POSITIVO FERA !!\n");
        goto ELEMENTOMAX;
    }



    //TEMPOS DOS ALGORITMOS
```

```
double tempos[iteraor];
double tempos2[iteraor];
double tempos3[iteraor];
double tempos4[iteraor];
double tempos5[iteraor];


//VETORES CRIADOS
int vetormagico[tamanho];
int vetormalabarista [tamanho];


for (k=0; k<tamanho; k++)
{
   vetormagico [k] = (rand()%elementomax);
}


for (contadoritera=0; contadoritera< iteraor; contadoritera++)
{
   puts ("\n");



   //BUBBLE SORT
   comeco = 0, termino = 0;
   copia(vetormagico, vetormalabarista, tamanho);
   comeco = clock();
   BubbleSort(vetormalabarista, tamanho);
   termino = clock();
   copia(vetormagico, vetormalabarista, tamanho);
   tempogasto = (double)(termino-comeco)/CLOCKS_PER_SEC;
   tempos[contadoritera] = tempogasto;
   puts ("\n");
   printf("Tempo gasto para iteração a %dº Iteração com BubbleSort: ", contadoritera+1);
   printf("%lf", tempogasto);
   tempogasto = 0;



   //INSERTION SORT
   comeco = 0, termino = 0;
   comeco = clock();
   InsertionSort(vetormalabarista, tamanho);
   termino = clock();
   copia(vetormagico, vetormalabarista, tamanho);
   tempogasto2 = (double)(termino-comeco)/CLOCKS_PER_SEC;
   tempos2[contadoritera] = tempogasto2;
   puts ("\n");
   printf("Tempo gasto para iteração a %dº Iteração com InsertionSort: ", contadoritera+1);
   printf("%lf", tempogasto2);
   tempogasto = 0;



   //SELECTION SORT
   comeco = 0, termino= 0;
   comeco = clock();
   SelectionSort(vetormalabarista, tamanho);
   termino = clock();
   copia(vetormagico, vetormalabarista, tamanho);
   tempogasto3 = (double)(termino-comeco)/CLOCKS_PER_SEC;
   tempos3[contadoritera] = tempogasto3;
   puts ("\n");
   printf("Tempo gasto para iteração a %dº Iteração com Selection Sort:  ", contadoritera+1);
   printf("%lf", tempogasto3);
   tempogasto = 0;



   //QUICK SORT
   int beg = 0, end = tamanho-1;
   comeco = clock();
   QuickSort(vetormalabarista, beg, end);
   termino = clock();
   copia(vetormagico, vetormalabarista, tamanho);
```

```
        tempogasto5 = (double)(termino-comeco)/CLOCKS_PER_SEC;
        tempos5[contadoritera] = tempogasto5;
        puts ("\n");
        printf("Tempo gasto para iteração a %dº Iteração com Quick Sort:  ", contadoritera+1);
        printf("%lf", tempogasto5);
        tempogasto = 0;



        //MERGE SORT
        beg = 0, end = tamanho-1;
        comeco = clock();
        MergeSort(vetormalabarista, beg, end, tamanho);
        termino = clock();
        copia(vetormagico, vetormalabarista, tamanho);
        tempogasto4 = (double)(termino-comeco)/CLOCKS_PER_SEC;
        tempos4[contadoritera] = tempogasto4;
        puts ("\n");
        printf("Tempo gasto para iteração a %dº Iteração com Merge Sort:  ", contadoritera+1);
        printf("%lf", tempogasto4);
        tempogasto = 0;
    }


    puts ("\n");

    tempomedio(tempos, med, iteraor, "BubbleSort");
    med = 0;
    tempomedio(tempos2, med, iteraor, "InsertionSort");
    med = 0;
    tempomedio(tempos3, med, iteraor, "SelectionSort");
    med = 0;
    tempomedio(tempos4, med, iteraor, "MergeSort");
    med = 0;
    tempomedio(tempos5, med, iteraor, "QuickSort");
    med = 0;

     getchar ();

}
```

3. RESULTADOS E DISCUSSÕES

Portanto, por meio do código acima, após inúmeras iterações (5, nesse caso) e com os tamanhos dos vetores de 10000, 30000, 60000, 90000, como também, uma quantidade máxima de elementos até 100000, obtém-se o seguinte resultado das compilações referentes aos algoritmos de ordenação.

Tabela 1: Algoritmos Ordenação para 10000 elementos

10000	BUBBLESORT	INSER.SORT	SELECT.SORT	MERGESORT	QUICKSORT
1ºIteração	0.317	0.093	0.162	0.003	0.001
2ºIteração	0.317	0.089	0.171	0.002	0.002
3ºIteração	0.323	0.092	0.168	0.002	0.001
4ºIteração	0.318	0.09	0.163	0.002	0.002
5ºIteração	0.332	0.089	0.171	0.003	0.002
Média	0.3214	0.0906	0.167	0.0024	0.0016

Tabela 2: Algoritmos Ordenação para 30000 elementos					
30000	BUBBLESORT	INSER.SORT	SELECT.SORT	MERGESORT	QUICKSORT
1ºIteração	3.25	0.788	1.45	0.01	0.005
2ºIteração	3.24	0.802	1.459	0.011	0.005
3ºIteração	3.24	0.797	1.447	0.013	0.005
4ºIteração	3.365	0.777	1.491	0.013	0.005
5ºIteração	3.2	0.789	1.488	0.012	0.005
Média	3.2832	0.7906	1.4576	0.0124	0.005

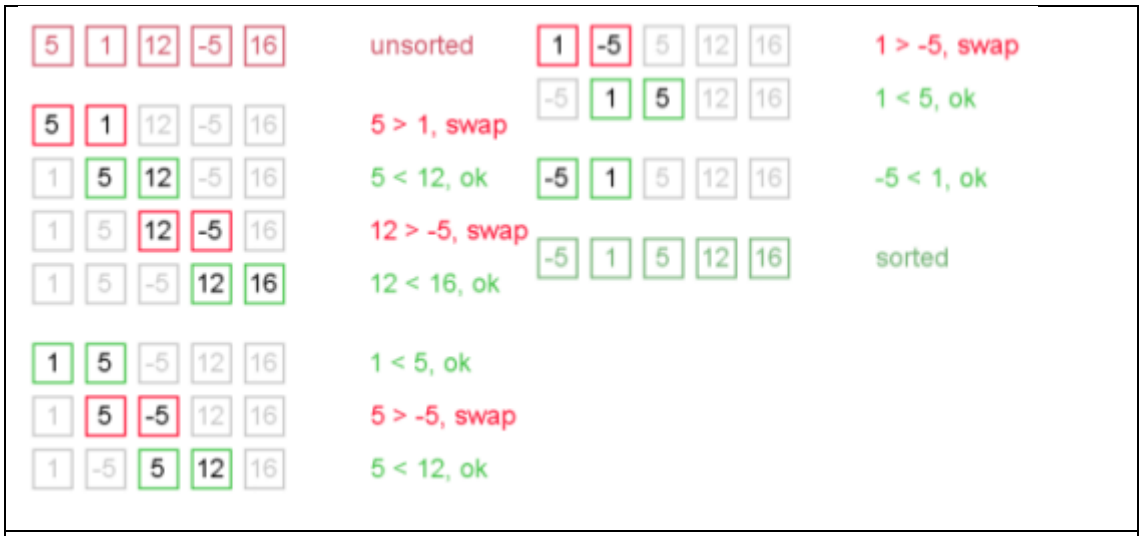
Tabela 3 : Algoritmos Ordenação para 60000 elementos					
60000	BUBBLESORT	INSER.SORT	SELECT.SORT	MERGESORT	QUICKSORT
1ºIteração	13.543	3.509	5.843	0.038	0.011
2ºIteração	14.015	3.442	6.08	0.038	0.01
3ºIteração	14.479	3. 384	6.089	0.044	0.012
4ºIteração	14.151	3. 363	6.218	0.042	0.011
5ºIteração	14.465	3.266	6.146	0.039	0.011
Média	14.1306	3. 3928	6.0752	0.0402	0.011

Tabela 4 : Algoritmos Ordenação para 90000 elementos					
90000	BUBBLESORT	INSER.SORT	SELECT.SORT	MERGESORT	QUICKSORT
1ºIteração	30.891	7.169	13.148	0.099	0.019
2ºIteração	31.076	7. 161	13.117	0.096	0.017
3ºIteração	33.458	8.032	14.254	0.098	0.017
4ºIteração	31.857	7.1	13.013	0.099	0.018
5ºIteração	35.963	13.721	16.496	0.097	0.018
Média	32.649	8.6366	14.0056	0.0978	0.0178

Explicando esses algoritmos de ordenação, temos :

- Bubble Sort: Algoritmo de ordenação em que temos dois for, variando de 0 até tamanho indicado -1, como também outro variando de tamanho indicado -1-i. Nesse caso, compara-se cada elemento do vetor com o seguinte. Representa-se na figura abaixo:

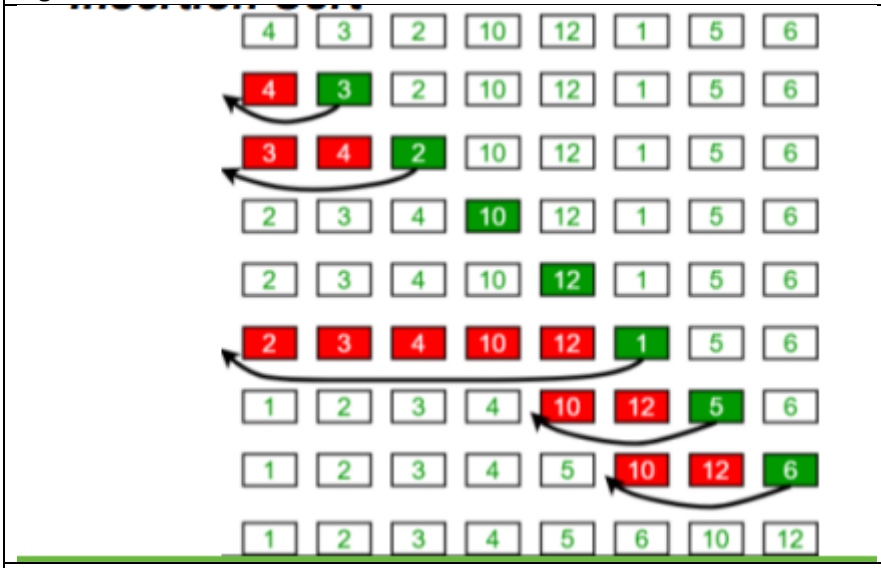
Figura 1: BubbleSort



Autor: Elder Vicente

- Insertion Sort: Nesse caso, compara-se o vetor na posição i e compara com um anterior, logo depois, acrescentamos o vetor na posição i, e compara-se com o anterior.

Figura 2: InsertionSort



Autor: Elder Vicente

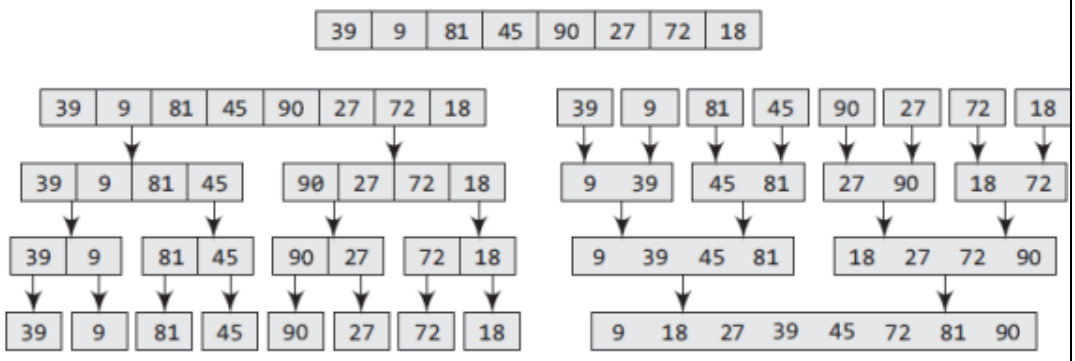
- Selection Sort: Nesse caso, é comparado o menor valor possível e adicionado no primeiro termo da lista, sendo assim:

- Figura 3: SelectionSort

Autor: Elder Vicente

- Merge Sort: Nesse caso, divide-se o vetor por meio recursivamente até ficar com apenas um elemento e ordena-se cada separação. Nesse caso, divide até ter um elemento, ordena e combina até voltar ao vetor normal.

Figura 4: MergeSort



Autor: Elder Vicente

- QuickSort: Nesse caso, tem-se a estratégia de divisão e conquista. Esse divide em duas sublistas, em chaves maiores ou menores até que a lista esteja recursivamente ordenada. Nesse caso, por meio da escolha do pivô, e observando o lado esquerdo e direito da lista, ordena-se o vetor.

4. CONCLUSÃO

Observando o pior e médio caso para os algoritmos supracitados tem-se:

Figura 5: Complexidade BigO

Algorithm	Average Case	Worst Case
Bubble sort	$O(n^2)$	$O(n^2)$
Bucket sort	O(n.k)	$O(n^2.k)$
Selection sort	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n^2)$	$O(n^2)$
Shell sort	–	$O(n \log^2 n)$
Merge sort	O(n log n)	O(n log n)
Heap sort	O(n log n)	O(n log n)
Quick sort	O(n log n)	$O(n^2)$

Autor: Elder Vicente

Portanto, por meio da complexidade BigO dos algoritmos e os gráficos, pode-se observar algumas semelhanças com o BubbleSort, SelectionSort e InsertionSort, pois, ambos são comparações com uso de for's e cada comparação há uma troca, por isso, tem-se o O (N^2). Já para QuickSort e MergeSort, tem-se uma complexidade O (N^2) e nlog(n) no pior caso.