UNIANCHIETA – UNIVERSIDADE PADRE ANCHIETA
CIÊNCIAS DA COMPUTAÇÃO

VICTOR HUGO MARTINS DE OLIVEIRA
RA: 2004526

ESTUDO DE COMPARAÇÃO DE TEMPOS DE EXECUÇÃO DE ESTRUTURAS DE DADOS PARA ORDENAR VETORES EM C

JUNDIAÍ

2020

1. INTRODUÇÃO

Em aula foram apresentadas 04 estruturas de dados, estas: Bubble Sort, Quick Sort, Selection Sort e Insertion Sort, todas estruturadas com suas características e complexidade. Então foi proposto este presente trabalho para que testamos a performance de ordenação dessas estruturas e mais uma não listada, e foi indicado a estrutura Merge Sort para comparação.

2. OBJETIVO

Obter dados de processamento das estruturas de dados: Bubble Sort, Quick Sort, Selection Sort, Insertion Sort e Marge Sort e compara-las entre si.

3. METODOLOGIA

Foram criados 5 arquivos do tipo texto. O primeiro “arquivo300.txt” contendo 300 valores gerados aleatoriamente entre 0 e 999. O segundo “arquivo999.txt” contendo 999 valores gerados aleatoriamente entre 0 e 999. E o terceiro “arquivo 9990.txt” contendo 9990 valores gerados aleatoriamente entre 0 e 9990, “arquivo20.txt” contendo 20 valores gerados aleatoriamente entre 0 e 999 e um último “arquivo99900.txt” contendo 99900 valores gerados aleatoriamente entre 0 e 9990 (este foram realizados apenas 3 laços, pois eram muitos números).

Foi criado uma função chamada carregarNumero() que carrega estes arquivos e devolve o vetor.

```c
#define QTLOOP 20

int * carregaNumero(int * tam, int test){
    int i = 0, n, *vetor, qt;

    FILE *arquivo;

    switch(test)
    {
        case 1:
            arquivo = fopen("arquivo300.txt", "r");
        break;
        case 2:
            arquivo = fopen("arquivo999.txt", "r");
        break;
        case 3:
            arquivo = fopen("arquivo9990.txt", "r");
        break;
        case 4:
            arquivo = fopen("arquivo20.txt", "r");
        break;
        case 5:
            arquivo = fopen("arquivo99900.txt", "r");
        break;
    }

    fscanf(arquivo, "%d\n", &qt);

    vetor = malloc (sizeof(int)*qt);

    while(!feof(arquivo)){
        fscanf(arquivo, "%d\n", &n);
        vetor[i] = n;
        i++;
    }
```

Foi criada também uma função chamada criaFile() que é chamada após o arquivo ser ordenado, esta função é responsável por criar os arquivos com o resultado da performance.

```
40
41  void criaFile(char * fileName, int * vector){
42      int i;
43
44      FILE *arquivo;
45      arquivo = fopen(fileName, "w");
46      for(i = 0; i < QTLOOP; i++){
47          fprintf(arquivo, "%i\n", vector[i]);
48
49      }
50      fclose(arquivo);
51
52      printf("\nArquivo %s criado!\n", fileName);
53
54  }
55
```

Para cada tipo de estrutura e arquivo txt foi criada a seguinte linha de código:

```
99      //bubble
100     printf("BUBBLE SORT\n");
101     printf("Carregando resultados\n");
102     for(i = 0; i < QTLOOP; i++){
103         vetorBubble = carregaNumero(&tam, 1);
104         QueryPerformanceCounter(&tempoInicial);
105         //gerando sort
106         bubble(vetorBubble, tam);
107
108         QueryPerformanceCounter(&tempoFinal);
109         tempo[i] = tempoFinal.QuadPart - tempoInicial.QuadPart;
110
111         printf(".");
112     }
113     criaFile("bubblesort.txt", tempo);
```

Este código se repete com algumas variações, entre o tipo de estruturas a ordenar e o arquivo .txt que será usado para consulta, este definido pelo segundo parâmetro da função carregaNumero(). Nele a variável QTLOOP está setada para 20, isto é, o programa testará em forma de laço por 20 vezes (menos o arquivo com 99900 dados, que foram feitos 3 laços), carregando novamente o vetor com os valores dos arquivos .txt, então é iniciado o contador de performance, armazenando-

o em uma variável chamada tempoInicial, em seguida, é executado a função de uma estrutura de dados que ordenará o vetor de acordo, estes como já citados são: Bubble Sort, Quick Sort, Selection Sort e Insertion Sort e Merge Sort. Após executado, o contador de performance finaliza armazenando o valor em uma outra variável chamada tempoFinal. Um vetor chamado tempo alocado com 20 unidades de memória inteiros é montado, dando a cada índice os valores da subtração da variável tempoFinal com a variável tempoInicial. E então executado a função criaFile() dando como um dos parâmetros o vetor tempo.

Para que não houvessem grandes interferências externas, foram fechados os navegadores, arquivos-texto, arquivos excel, pastas e alguns programas que estavam rodando no processo que não seriam utilizados no processo, e, por fim, compilado.

4. DADOS OBTIDOS

Tabela 1 – Dados gerados pelas estruturas com 300 dados carregados no vetor.

Arquivo com 300 dados					
	Bubble Sort	Insertion Sort	Quick Sort	Selection Sort	Merge Sort
Teste 1	1508	539	207	1009	3267
Teste 2	1500	540	181	1006	3140
Teste 3	1490	538	154	1001	3321
Teste 4	1481	538	134	997	3120
Teste 5	1479	534	124	996	3058
Teste 6	1483	604	118	996	3049
Teste 7	1489	533	115	994	3059
Teste 8	1490	533	112	992	3083
Teste 9	1494	532	110	993	3147
Teste 10	1477	532	110	993	3108
Teste 11	1482	535	109	991	3223
Teste 12	1474	533	107	991	2949
Teste 13	1474	535	107	991	3208
Teste 14	1477	529	104	991	3083
Teste 15	1476	534	104	996	3196
Teste 16	1495	536	104	995	2961
Teste 17	1483	534	105	993	3237
Teste 18	1482	534	117	993	3036
Teste 19	1485	531	111	990	3059
Teste 20	1489	533	107	994	2970
Média	1483	534	110,5	993,5	3095,5

Tabela 2 – Dados gerados pelas estruturas com 999 dados carregados no vetor.

Arquivo com 999 dados					
	Bubble Sort	Insertion Sort	Quick Sort	Selection Sort	Merge Sort
Teste 1	14624	5557	751	10208	11168
Teste 2	14583	5464	742	10203	11621
Teste 3	14516	5453	718	10242	10997
Teste 4	14606	5566	716	10197	10973
Teste 5	14579	5451	710	10781	10855
Teste 6	14543	5457	702	10265	10322
Teste 7	14753	5452	702	10239	10954
Teste 8	14567	5447	701	10249	10515
Teste 9	14580	5452	697	10696	10500
Teste 10	16201	5455	699	10210	10873
Teste 11	14608	5453	704	10200	9778
Teste 12	14580	5451	701	10200	10062
Teste 13	14525	5458	702	10200	9705
Teste 14	14560	5457	711	10204	9875
Teste 15	14710	5463	700	10188	9654
Teste 16	14544	6123	705	10200	10012
Teste 17	14571	5460	704	10261	9990
Teste 18	14556	5458	752	10227	9788
Teste 19	16404	5468	803	10186	9833
Teste 20	14601	5456	733	10254	9454
Média	14580	5457	704,5	10209	10192

Tabela 3 – Dados gerados pelas estruturas com 9990 dados carregados no vetor.

Arquivo com 9990 dados					
	Bubble Sort	Insertion Sort	Quick Sort	Selection Sort	Merge Sort
Teste 1	1826512	538998	7908	957977	104943
Teste 2	1806029	531855	7605	953589	105093
Teste 3	2196051	542184	7610	950519	111862
Teste 4	1956773	533116	7628	952409	112070
Teste 5	1757329	534922	7784	953450	109318
Teste 6	1771591	534117	8629	954425	109152
Teste 7	1780056	522091	7616	947130	111254
Teste 8	1761493	525336	7612	953480	109500
Teste 9	1773255	522911	7639	956292	109088
Teste 10	1778774	531008	7636	954843	108261
Teste 11	1782238	533608	7791	954808	108650
Teste 12	1772034	534144	7801	954596	111199
Teste 13	1756349	533307	7816	948850	109640
Teste 14	1766791	523072	7798	936951	110428
Teste 15	1772010	522437	7795	947955	109716
Teste 16	1771785	530137	10838	943180	108275
Teste 17	1770344	524448	7823	938336	109478
Teste 18	1775137	534646	7814	948252	110059
Teste 19	1754804	532694	7818	937013	110519
Teste 20	1759942	533067	7809	936610	108594
Média	1772022	532880,5	7796,5	951464	109489

Tabela 4 – Dados gerados pelas estruturas com 20 dados carregados no vetor.

Arquivo com 20 dados					
	Bubble Sort	Insertion Sort	Quick Sort	Selection Sort	Merge Sort
Teste 1	11	5	11	10	35
Teste 2	9	4	8	8	25
Teste 3	7	4	7	6	23
Teste 4	6	2	5	5	30
Teste 5	5	3	5	5	26
Teste 6	5	3	4	5	23
Teste 7	5	3	4	4	23
Teste 8	5	3	4	4	30
Teste 9	5	3	4	4	25
Teste 10	5	2	3	4	28
Teste 11	5	3	3	5	22
Teste 12	5	3	3	4	23
Teste 13	4	3	4	4	21
Teste 14	4	3	3	5	21
Teste 15	5	2	3	10	20
Teste 16	5	2	4	8	21
Teste 17	5	2	4	7	20
Teste 18	4	3	3	6	20
Teste 19	4	3	4	5	20
Teste 20	5	3	4	5	20
Média	5	3	4	5	23

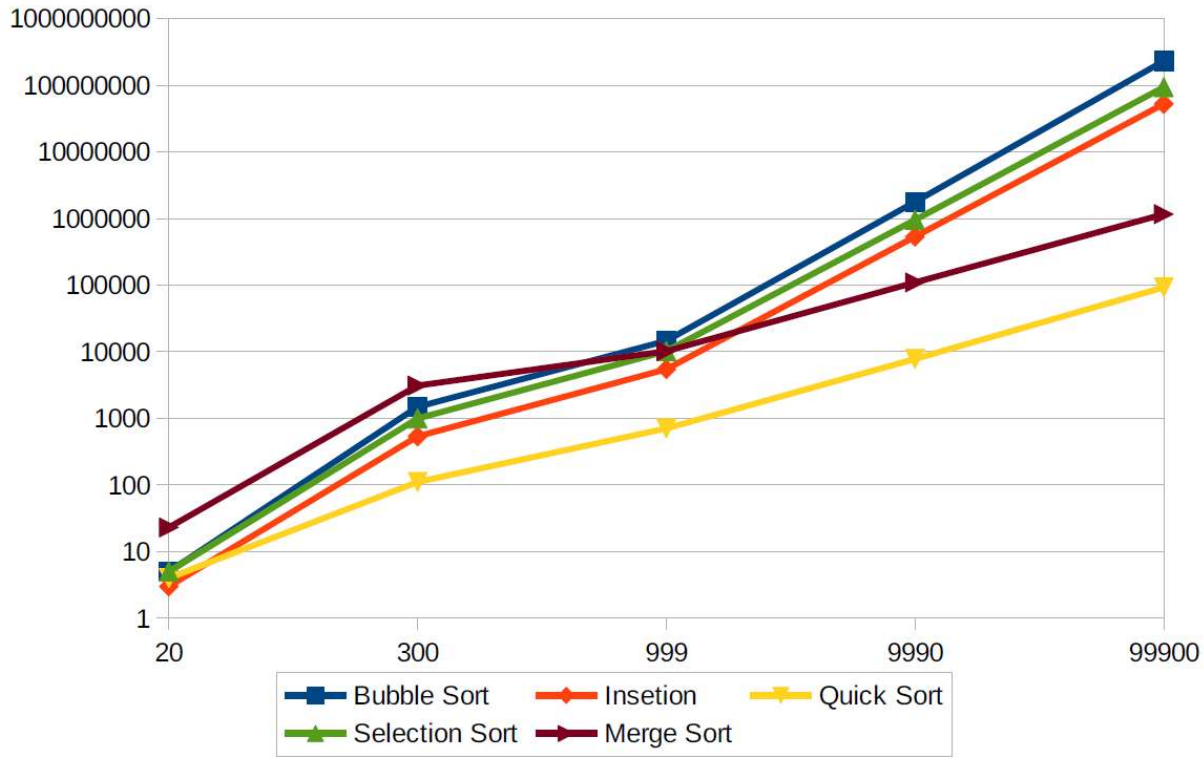
Tabela 5 – Dados gerados pela estrutura com 99900 dados carregados no vetor.

Arquivo com 99900 dados					
	Bubble Sort	Insertion Sort	Quick Sort	Selection Sort	Merge Sort
Teste 1	230726125	52349714	92137	93400538	1176420
Teste 2	229995611	52291259	92546	93439158	1160331
Teste 3	229956815	52320740	93715	93353532	1159103
Média	229995611	52320740	92546	93400538	1160331

Tabela 6 – Comparação de dados gerados em relação a estruturas e valores médios obtidos.

COMPARAÇÃO MÉDIA DE QUANTIDADE DE DADOS E PROCESSAMENTO					
	20	300	999	9990	99900
Bubble Sort	5	1483	14580	1772022	229995611
Insetion	3	534	5457	532880,5	52320740
Quick Sort	4	110,5	704,5	7796,5	92546
Selection Sort	5	993,5	10209	951464	93400538
Merge Sort	23	3095,5	10192	109489	1160331

Gráfico 1 – Comparação de dados gerados em relação a estruturas e valores médios obtidos.



4. CONCLUSÃO

Ao concretizar esta bateria de testes com diferentes estruturas de organização e tamanho de vetores, obtive resultados variados, nos 3 primeiros testes, de 300, 999 e 9990 dados as estruturas Bubble Sort, Selection Sort, Insertion Sort e Quick Sort se manteve em uma posição subsequente da outra, com uma pequena diferença da Merge Sort, que quanto mais dados, mais se aproximava da Quick Sort.

(Em ordem **Tabela 1**, **Tabela 2 e Tabela 3**).

- 300 dados

  Merge Sort, Bubble Sort, Selection Sort, Insertion Sort, Quick Sort

- 999 dados

  Bubble Sort, Selection Sort, Merge Sort, Insertion Sort, Quick Sort

- 9990 dados

  Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort

Foi possível perceber também que os dados de testes gerados a partir do arquivo de 20 dados foi quase indiferente entre as estruturas Bubble Sort, Selection Sort, Insertion Sort e Quick Sort, com destaque da Merge Sort, que apresentou uma média de 500% a mais de processamento vide **Tabela 4**.

Outro detalhe da estrutura de dados Merge Sort, é que em uma determinada quantidade de dados, ainda assim não passa o Quick Sort e velocidade de processamento, é possível perceber na **Tabela 5** este fenômeno.

Contudo, é conclusivo que a estrutura de Merge Sort é um tanto quanto interessante em relação a diminuição de processamento exponencial com a quantidade de dados, e que chega a um limite em uma determinada quantidade de dados em relação ao Quick Sort, que impressiona pela sua velocidade de processamento, tendo apenas em torno de 10% do processamento do Merge Sort, segundo colocado no caso com mais dados.