

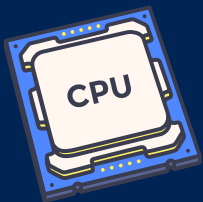
# TRABALHO ORDENAÇÃO – AEDS

João Victor Dutra Martins Silva

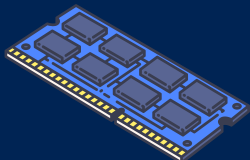


- Equipamentos utilizado para o trabalho :

- Computador Pessoal :



Processador : I5-4440



RAM : 16GB , DDR3, 1333MHZ



SO : Windows 10 – Não possuo SSD

# CÓDIGO UTILIZADO PARA TESTES :

- Main :

```
1 package trabalho.aedsl;
2
3 import java.util.Random;
4
5 public class TesteAlgoritmosOrdenacao {
6
7     public static void main(String[] args) {
8         int[] tamanhos = {100, 1000, 100000, 500000};
9
10        for (int tamanho : tamanhos) {
11            int[] arrAleatorio = gerarArrayAleatorio(tamanho);
12            int[] arrCrescente = gerarArrayCrescente(tamanho);
13            int[] arrDecrescente = gerarArrayDecrescente(tamanho);
14
15            System.out.println("Testando com tamanho do vetor: " + tamanho);
16
17            testarOrdenacao(algoritmo:"BubbleSort (Aleatório)", arr: arrAleatorio.clone());
18            testarOrdenacao(algoritmo:"SelectionSort (Aleatório)", arr: arrAleatorio.clone());
19            testarOrdenacao(algoritmo:"InsertionSort (Aleatório)", arr: arrAleatorio.clone());
20            testarOrdenacao(algoritmo:"MergeSort (Aleatório)", arr: arrAleatorio.clone());
21            testarOrdenacao(algoritmo:"QuickSort (Aleatório)", arr: arrAleatorio.clone());
22
23            testarOrdenacao(algoritmo:"BubbleSort (Crescente)", arr: arrCrescente.clone());
24            testarOrdenacao(algoritmo:"SelectionSort (Crescente)", arr: arrCrescente.clone());
25            testarOrdenacao(algoritmo:"InsertionSort (Crescente)", arr: arrCrescente.clone());
26            testarOrdenacao(algoritmo:"MergeSort (Crescente)", arr: arrCrescente.clone());
27            testarOrdenacao(algoritmo:"QuickSort (Crescente)", arr: arrCrescente.clone());
28
29            testarOrdenacao(algoritmo:"BubbleSort (Decrescente)", arr: arrDecrescente.clone());
30            testarOrdenacao(algoritmo:"SelectionSort (Decrescente)", arr: arrDecrescente.clone());
31            testarOrdenacao(algoritmo:"InsertionSort (Decrescente)", arr: arrDecrescente.clone());
32            testarOrdenacao(algoritmo:"MergeSort (Decrescente)", arr: arrDecrescente.clone());
33            testarOrdenacao(algoritmo:"QuickSort (Decrescente)", arr: arrDecrescente.clone());
34
35            System.out.println();
36        }
37    }
```

# • Método Testar Ordenação :

```
39 public static void testarOrdenacao(String algoritmo, int[] arr) {
40     long tempoInicial = System.nanoTime();
41     switch (algoritmo) {
42         case "BubbleSort (Aleatório)":
43             bubbleSort(arr, size: arr.length);
44             break;
45         case "SelectionSort (Aleatório)":
46             selectionSort(arr, size: arr.length);
47             break;
48         case "InsertionSort (Aleatório)":
49             insertionSort(arr, size: arr.length);
50             break;
51         case "MergeSort (Aleatório)":
52             mergeSort(arr, l: 0, arr.length - 1);
53             break;
54         case "QuickSort (Aleatório)":
55             quickSort(arr, low: 0, arr.length - 1);
56             break;
57         case "BubbleSort (Crescente)":
58             bubbleSort(arr, size: arr.length);
59             break;
60         case "SelectionSort (Crescente)":
61             selectionSort(arr, size: arr.length);
62             break;
63         case "InsertionSort (Crescente)":
64             insertionSort(arr, size: arr.length);
65             break;
66         case "MergeSort (Crescente)":
67             mergeSort(arr, l: 0, arr.length - 1);
68             break;
69         case "QuickSort (Crescente)":
70             quickSort(arr, low: 0, arr.length - 1);
71             break;
72         case "BubbleSort (Decrescente)":
73             bubbleSort(arr, size: arr.length);
74             break;
75         case "SelectionSort (Decrescente)":
76             selectionSort(arr, size: arr.length);
77             break;
78         case "InsertionSort (Decrescente)":
79             insertionSort(arr, size: arr.length);
80             break;
81         case "MergeSort (Decrescente)":
82             mergeSort(arr, l: 0, arr.length - 1);
83             break;
84         case "QuickSort (Decrescente)":
85             quickSort(arr, low: 0, arr.length - 1);
86             break;
87     }
88     long tempoFinal = System.nanoTime();
89     System.out.println(algoritmo + " levou " + (tempoFinal - tempoInicial) + " nanossegundos");
90 }
```

## • Métodos Gerar Array :

```
92  public static int[] gerarArrayAleatorio(int tamanho) {
93      int[] arr = new int[tamanho];
94      Random random = new Random();
95      for (int i = 0; i < tamanho; i++) {
96          arr[i] = random.nextInt(bound: 1000);
97      }
98      return arr;
99  }

100
101  public static int[] gerarArrayCrescente(int tamanho) {
102      int[] arr = new int[tamanho];
103      for (int i = 0; i < tamanho; i++) {
104          arr[i] = i;
105      }
106      return arr;
107  }

108
109  public static int[] gerarArrayDecrescente(int tamanho) {
110      int[] arr = new int[tamanho];
111      for (int i = 0; i < tamanho; i++) {
112          arr[i] = tamanho - i;
113      }
114      return arr;
115  }

116
```

# • Métodos de Ordenação :

```
153 public static void merge(int arr[], int l, int m, int r) {
154     int i, j, k;
155     int n1 = m - l + 1;
156     int n2 = r - m;
157
158     int[] L = new int[n1];
159     int[] R = new int[n2];
160
161     for (i = 0; i < n1; i++) {
162         L[i] = arr[l + i];
163     }
164     for (j = 0; j < n2; j++) {
165         R[j] = arr[m + 1 + j];
166     }
167
168     i = 0;
169     j = 0;
170     k = l;
171     while (i < n1 && j < n2) {
172         if (L[i] <= R[j]) {
173             arr[k] = L[i];
174             i++;
175         } else {
176             arr[k] = R[j];
177             j++;
178         }
179         k++;
180     }
181
182     while (i < n1) {
183         arr[k] = L[i];
184         i++;
185         k++;
186     }
187
188     while (j < n2) {
189         arr[k] = R[j];
190         j++;
191         k++;
192     }
193 }
```

```
195 public static void quickSort(int arr[], int low, int high) {
196     while (low < high) {
197         int middle = low + (high - low) / 2;
198         int pivot = arr[middle];
199         swap(arr, middle, high);
200         int pivotIndex = partition(arr, low, high);
201         if (pivotIndex - low < high - pivotIndex) {
202             quickSort(arr, low, pivotIndex - 1);
203             low = pivotIndex + 1;
204         } else {
205             quickSort(arr, pivotIndex + 1, high);
206             high = pivotIndex - 1;
207         }
208     }
209 }
```

```
117 public static void selectionSort(int arr[], int size) {
118     for (int i = 0; i < size - 1; i++) {
119         int menorElemento = i;
120         for (int j = i + 1; j < size; j++) {
121             if (arr[j] < arr[menorElemento]) {
122                 menorElemento = j;
123             }
124         }
125         int swap = arr[i];
126         arr[i] = arr[menorElemento];
127         arr[menorElemento] = swap;
128     }
129 }
130 }
```

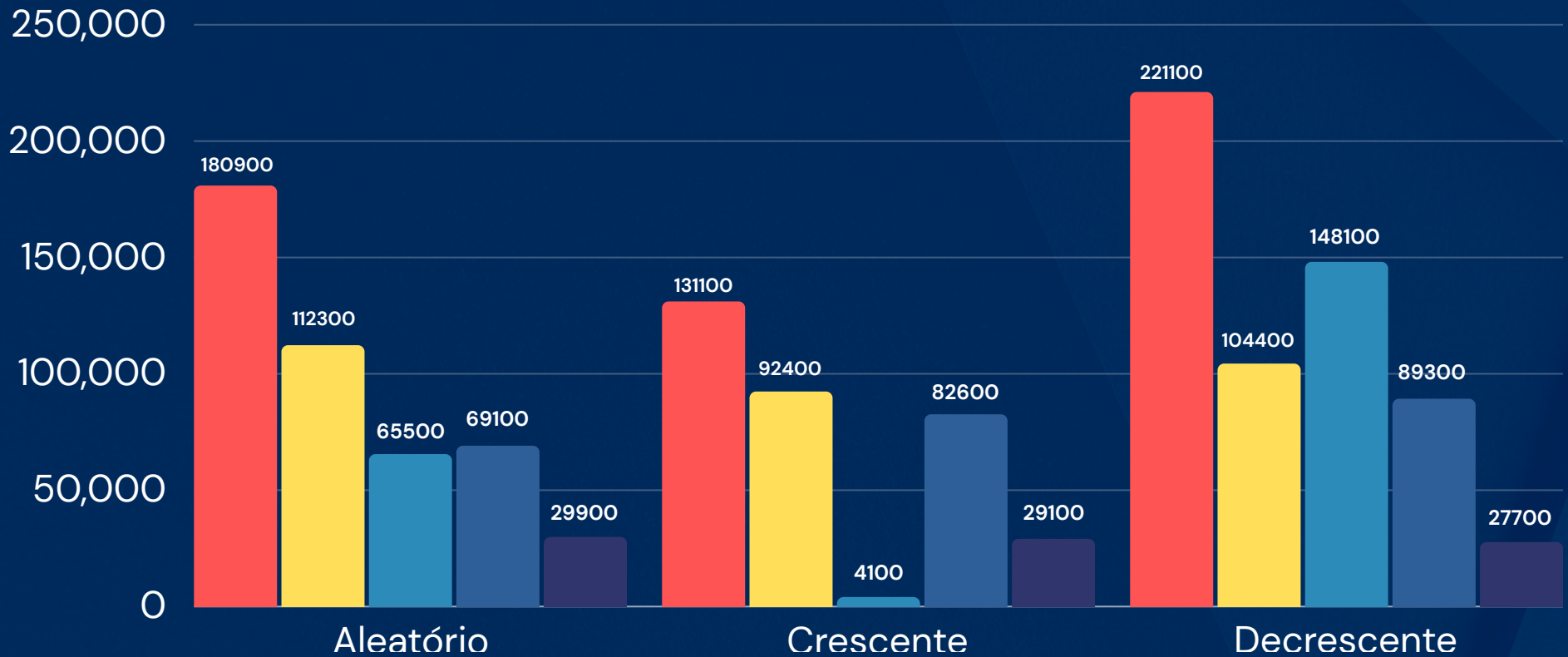
```
131 public static void insertionSort(int arr[], int size) {
132     int temp, j;
133     for (int i = 1; i < size; i++) {
134         temp = arr[i];
135         j = i - 1;
136         while (j >= 0 && arr[j] > temp) {
137             arr[j + 1] = arr[j];
138             j = j - 1;
139         }
140         arr[j + 1] = temp;
141     }
142 }
143 }
```

```
144 public static void mergeSort(int arr[], int l, int r) {
145     if (l < r) {
146         int m = l + (r - l) / 2;
147         mergeSort(arr, l, m);
148         mergeSort(arr, m + 1, r);
149         merge(arr, l, m, r);
150     }
151 }
```

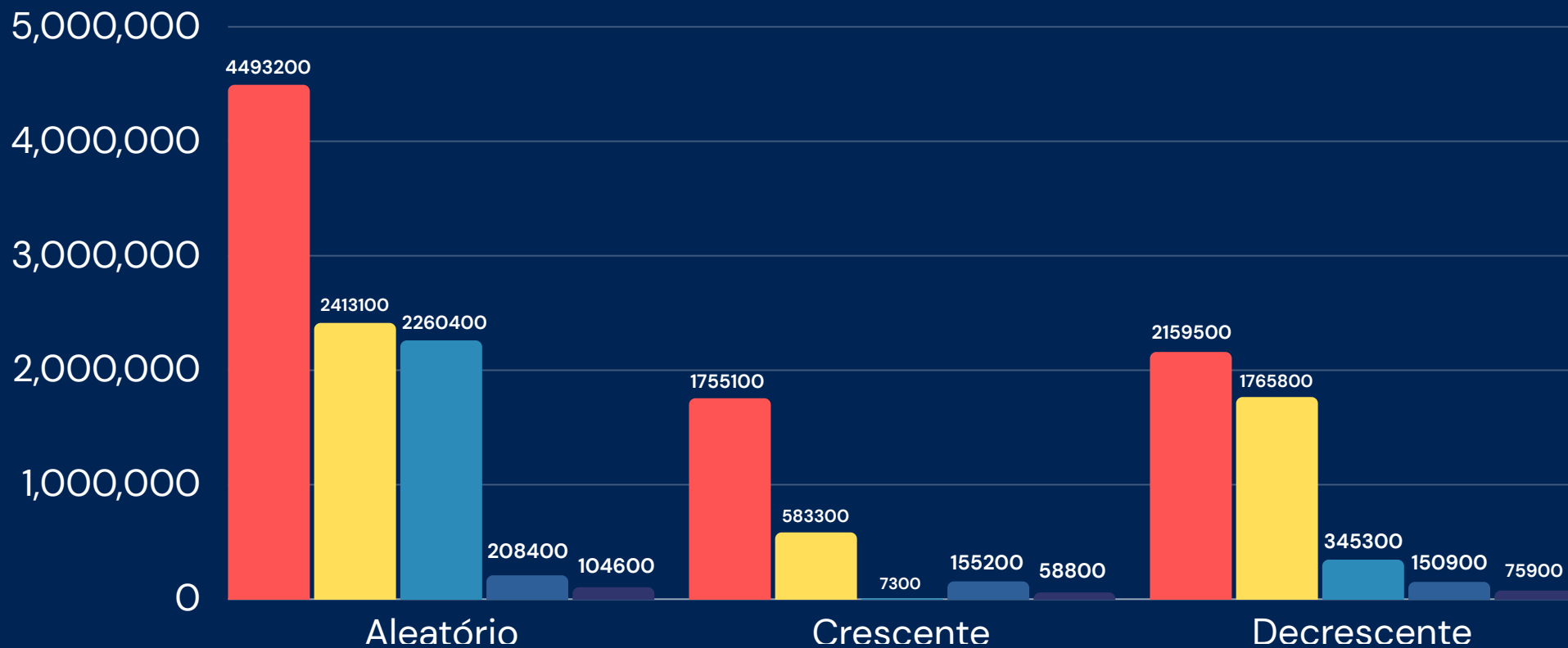
## • Métodos de Ordenação (2) :

```
211 public static int partition(int arr[], int low, int high) {
212     int pivot = arr[high];
213     int i = low - 1;
214     for (int j = low; j <= high - 1; j++) {
215         if (arr[j] <= pivot) {
216             i++;
217             int temp = arr[i];
218             arr[i] = arr[j];
219             arr[j] = temp;
220         }
221     }
222     int temp = arr[i + 1];
223     arr[i + 1] = arr[high];
224     arr[high] = temp;
225     return i + 1;
226 }
227
228 public static void bubbleSort(int arr[], int size) {
229     for (int i = 0; i < size - 1; i++) {
230         for (int j = 0; j < size - i - 1; j++) {
231             if (arr[j] > arr[j + 1]) {
232                 int temp = arr[j];
233                 arr[j] = arr[j + 1];
234                 arr[j + 1] = temp;
235             }
236         }
237     }
238 }
239
240 public static void swap(int arr[], int i, int j) {
241     int temp = arr[i];
242     arr[i] = arr[j];
243     arr[j] = temp;
244 }
245 }
246
```

# VETORES TAMANHO [100]

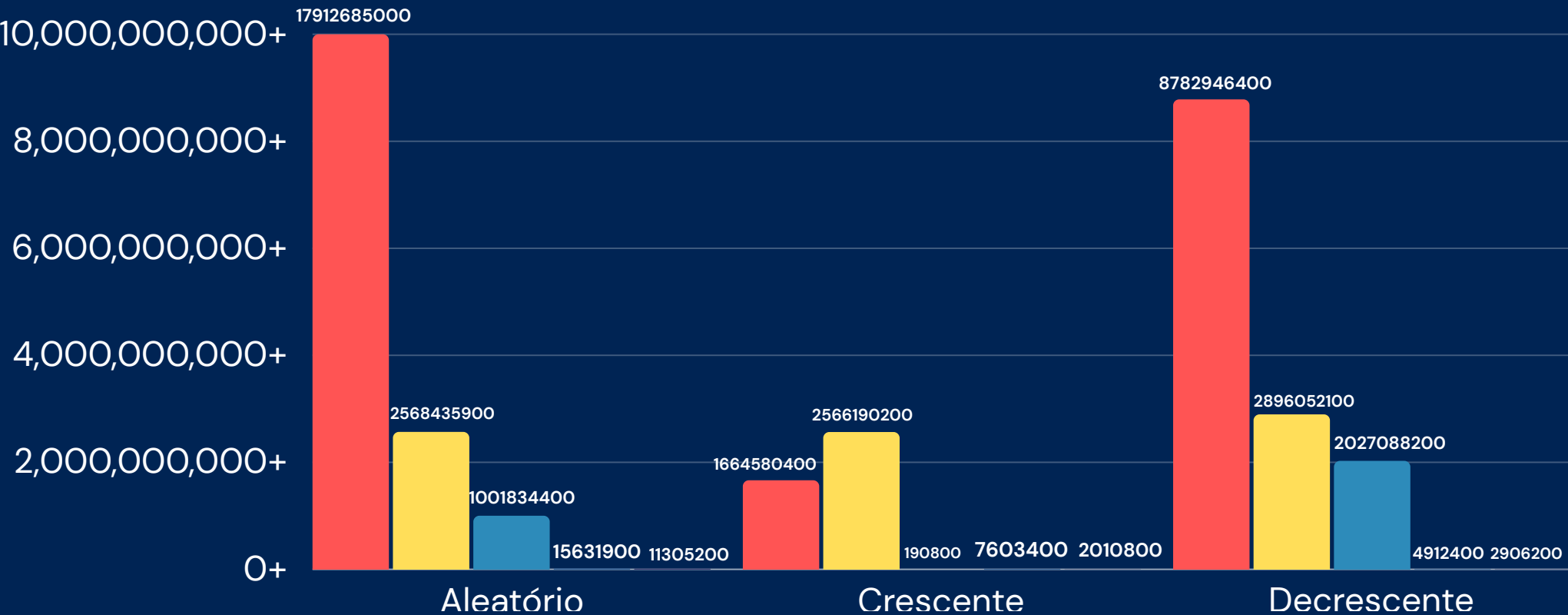


# VETORES TAMANHO [1000]

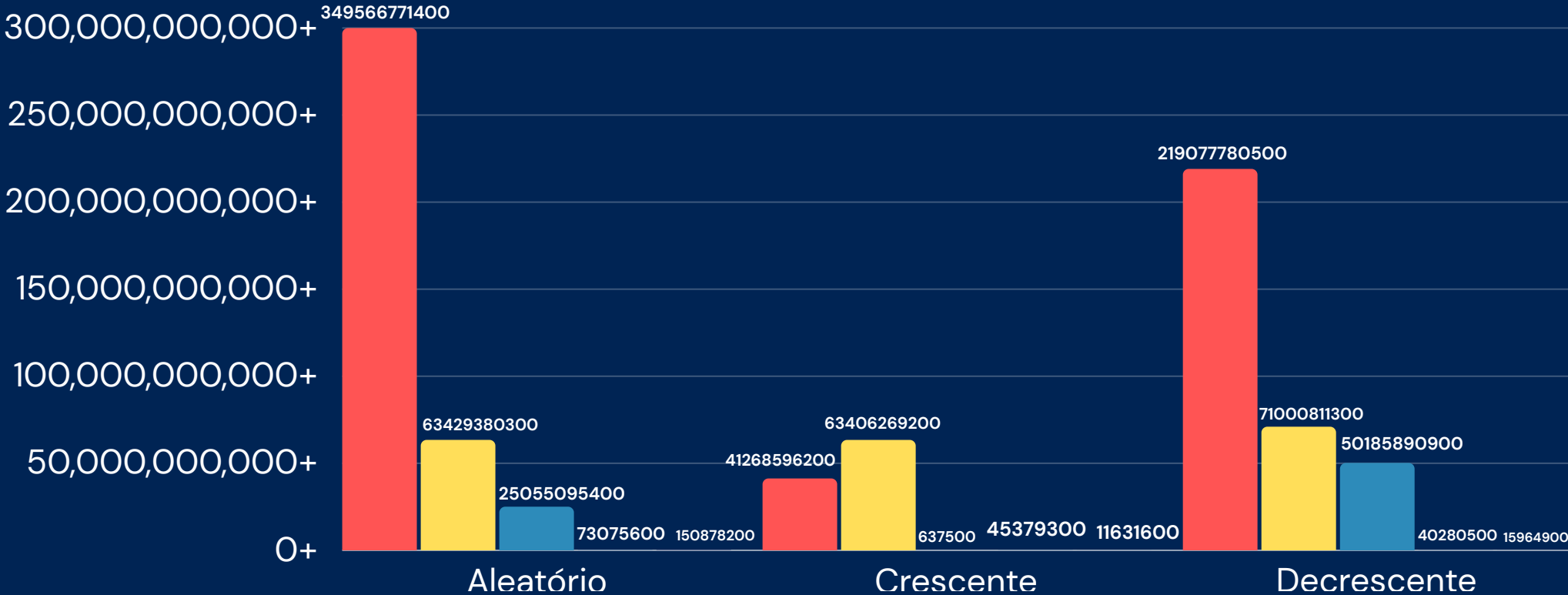




# VETORES TAMANHO [100K]



# VETORES TAMANHO [500K]



# COMPLEXIDADE DOS METODOS DE ORD. :

- **Bubble Sort:**

Pior caso:  $O(n^2)$

Caso médio:  $O(n^2)$

Melhor caso:  $O(n)$

- **Selection Sort:**

Pior caso:  $O(n^2)$

Caso médio:  $O(n^2)$

Melhor caso:  $O(n^2)$

- **Insertion Sort:**

Pior caso:  $O(n^2)$

Caso médio:  $O(n^2)$

Melhor caso:  $O(n)$

- **Merge Sort:**

Pior caso:  $O(n \log n)$

Caso médio:  $O(n \log n)$

Melhor caso:  $O(n \log n)$

- **Quick Sort:**

Pior caso:  $O(n^2)$  [raro, mas possível]

Caso médio:  $O(n \log n)$

Melhor caso:  $O(n \log n)$

# CONCLUI-SE ENTÃO :

## 1. Bubble Sort, Selection Sort e Insertion Sort:

- Esses métodos de ordenação são menos eficientes em termos de tempo, especialmente em grandes conjuntos de dados.
- Eles são mais adequados para listas pequenas ou já quase ordenadas.
- O Bubble Sort é útil apenas para fins educacionais ou quando a simplicidade do algoritmo é mais importante do que o desempenho.

## 1. Merge Sort:

- O Merge Sort é um algoritmo de ordenação eficiente com desempenho  $O(n \log n)$  em todos os casos.
- É uma escolha sólida quando a eficiência é fundamental, independentemente do tamanho da lista ou do grau de desordem.

## 1. Quick Sort:

- O Quick Sort é altamente eficiente na média, com desempenho  $O(n \log n)$ .
- É uma boa escolha quando a eficiência é importante, mas deve-se estar ciente de que o pior caso pode ser  $O(n^2)$  em situações raras.
- Pode ser otimizado para mitigar o risco do pior caso.

# BIBLIOGRAFIAS E REFERÊNCIAS :

## Referências :

<https://medium.com/@macedo.g/tempo-de-execu%C3%A7%C3%A3o-dos-principais-algoritmos-de-ordena%C3%A7%C3%A3o-feb5d06fb674>

[https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261?source=post\\_page-----feb5d06fb674-----](https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261?source=post_page-----feb5d06fb674-----)

-----

[https://www.freecodecamp.org/portuguese/news/algoritmos-de-ordenacao-explicados-com-exemplos-em-python-java-e-c/?source=post\\_page-----](https://www.freecodecamp.org/portuguese/news/algoritmos-de-ordenacao-explicados-com-exemplos-em-python-java-e-c/?source=post_page-----)

[feb5d06fb674-----](https://www.freecodecamp.org/portuguese/news/algoritmos-de-ordenacao-explicados-com-exemplos-em-python-java-e-c/?source=post_page-----feb5d06fb674-----)

<https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao>