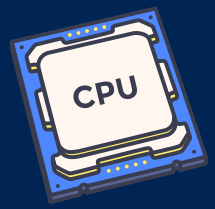
TRABALHO ORDENAÇÃO – AEDS

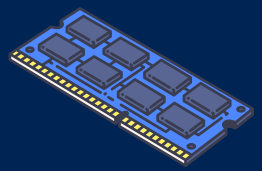
João Victor Dutra Martins Silva

- **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 :**

```java
package trabalho.aedsl;

import java.util.Random;

public class TesteAlgoritmosOrdenacao {

    public static void main(String[] args) {
        int[] tamanhos = {100, 1000, 100000, 500000};

        for (int tamanho : tamanhos) {
            int[] arrAleatorio = gerarArrayAleatorio(tamanho);
            int[] arrCrescente = gerarArrayCrescente(tamanho);
            int[] arrDecrescente = gerarArrayDecrescente(tamanho);

            System.out.println("Testando com tamanho do vetor: " + tamanho);

            testarOrdenacao(algoritmo: "BubbleSort (Aleatório)", arr: arrAleatorio.clone());
            testarOrdenacao(algoritmo: "SelectionSort (Aleatório)", arr: arrAleatorio.clone());
            testarOrdenacao(algoritmo: "InsertionSort (Aleatório)", arr: arrAleatorio.clone());
            testarOrdenacao(algoritmo: "MergeSort (Aleatório)", arr: arrAleatorio.clone());
            testarOrdenacao(algoritmo: "QuickSort (Aleatório)", arr: arrAleatorio.clone());

            testarOrdenacao(algoritmo: "BubbleSort (Crescente)", arr: arrCrescente.clone());
            testarOrdenacao(algoritmo: "SelectionSort (Crescente)", arr: arrCrescente.clone());
            testarOrdenacao(algoritmo: "InsertionSort (Crescente)", arr: arrCrescente.clone());
            testarOrdenacao(algoritmo: "MergeSort (Crescente)", arr: arrCrescente.clone());
            testarOrdenacao(algoritmo: "QuickSort (Crescente)", arr: arrCrescente.clone());

            testarOrdenacao(algoritmo: "BubbleSort (Decrescente)", arr: arrDecrescente.clone());
            testarOrdenacao(algoritmo: "SelectionSort (Decrescente)", arr: arrDecrescente.clone());
            testarOrdenacao(algoritmo: "InsertionSort (Decrescente)", arr: arrDecrescente.clone());
            testarOrdenacao(algoritmo: "MergeSort (Decrescente)", arr: arrDecrescente.clone());
            testarOrdenacao(algoritmo: "QuickSort (Decrescente)", arr: arrDecrescente.clone());

            System.out.println();
        }
    }
```

• Método Testar Ordenação :

```java
public static void testarOrdenacao(String algoritmo, int[] arr) {
    long tempoInicial = System.nanoTime();
    switch (algoritmo) {
        case "BubbleSort (Aleatório)":
            bubbleSort(arr, size: arr.length);
            break;
        case "SelectionSort (Aleatório)":
            selectionSort(arr, size: arr.length);
            break;
        case "InsertionSort (Aleatório)":
            insertionSort(arr, size: arr.length);
            break;
        case "MergeSort (Aleatório)":
            mergeSort(arr, l: 0, arr.length - 1);
            break;
        case "QuickSort (Aleatório)":
            quickSort(arr, low:0, arr.length - 1);
            break;
        case "BubbleSort (Crescente)":
            bubbleSort(arr, size: arr.length);
            break;
        case "SelectionSort (Crescente)":
            selectionSort(arr, size: arr.length);
            break;
        case "InsertionSort (Crescente)":
            insertionSort(arr, size: arr.length);
            break;
        case "MergeSort (Crescente)":
            mergeSort(arr, l: 0, arr.length - 1);
            break;
        case "QuickSort (Crescente)":
            quickSort(arr, low:0, arr.length - 1);
            break;
        case "BubbleSort (Decrescente)":
            bubbleSort(arr, size: arr.length);
            break;
        case "SelectionSort (Decrescente)":
            selectionSort(arr, size: arr.length);
            break;
        case "InsertionSort (Decrescente)":
            insertionSort(arr, size: arr.length);
            break;
        case "MergeSort (Decrescente)":
            mergeSort(arr, l: 0, arr.length - 1);
            break;
        case "QuickSort (Decrescente)":
            quickSort(arr, low:0, arr.length - 1);
            break;
    }
    long tempoFinal = System.nanoTime();
    System.out.println(algoritmo + " levou " + (tempoFinal - tempoInicial) + " nanossegundos");
}
```

- **Métodos Gerar Array :**

```java
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 :

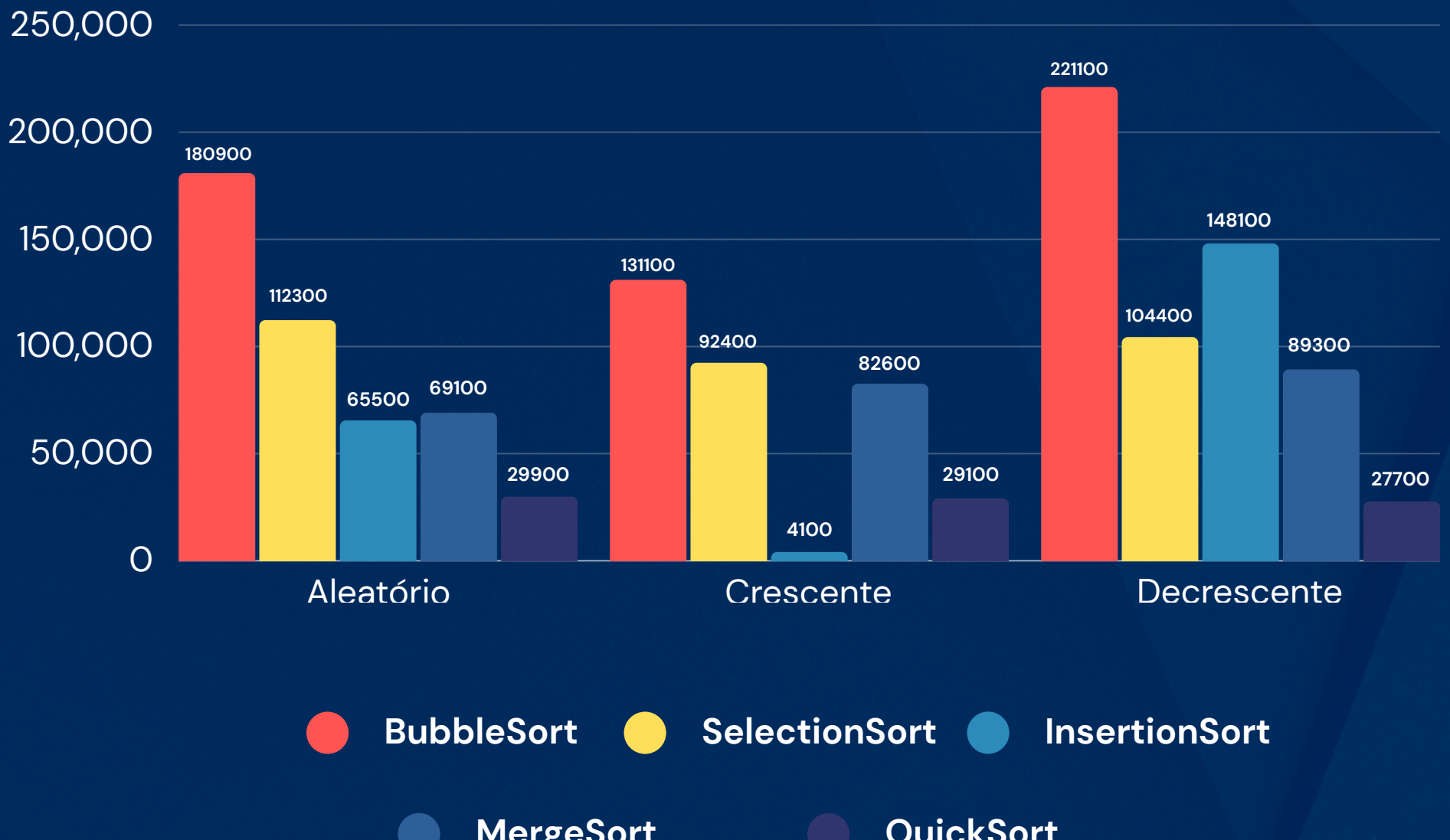
```java
public static void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int[] L = new int[n1];
    int[] R = new int[n2];

    for (i = 0; i < n1; i++) {
        L[i] = arr[l + i];
    }
    for (j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
    }

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

public static void quickSort(int arr[], int low, int high) {
    while (low < high) {
        int middle = low + (high - low) / 2;
        int pivot = arr[middle];
        swap(arr, i: middle, j: high);
        int pivotIndex = partition(arr, low, high);
        if (pivotIndex - low < high - pivotIndex) {
            quickSort(arr, low, pivotIndex - 1);
            low = pivotIndex + 1;
        } else {
            quickSort(arr, pivotIndex + 1, high);
            high = pivotIndex - 1;
        }
    }
```

```java
public static void selectionSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int menorElemento = i;
        for (int j = i + 1; j < size; j++) {
            if (arr[j] < arr[menorElemento]) {
                menorElemento = j;
            }
        }
        int swap = arr[i];
        arr[i] = arr[menorElemento];
        arr[menorElemento] = swap;
    }
}

public static void insertionSort(int arr[], int size) {
    int temp, j;
    for (int i = 1; i < size; i++) {
        temp = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > temp) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = temp;
    }
}

public static void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, r: m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
```

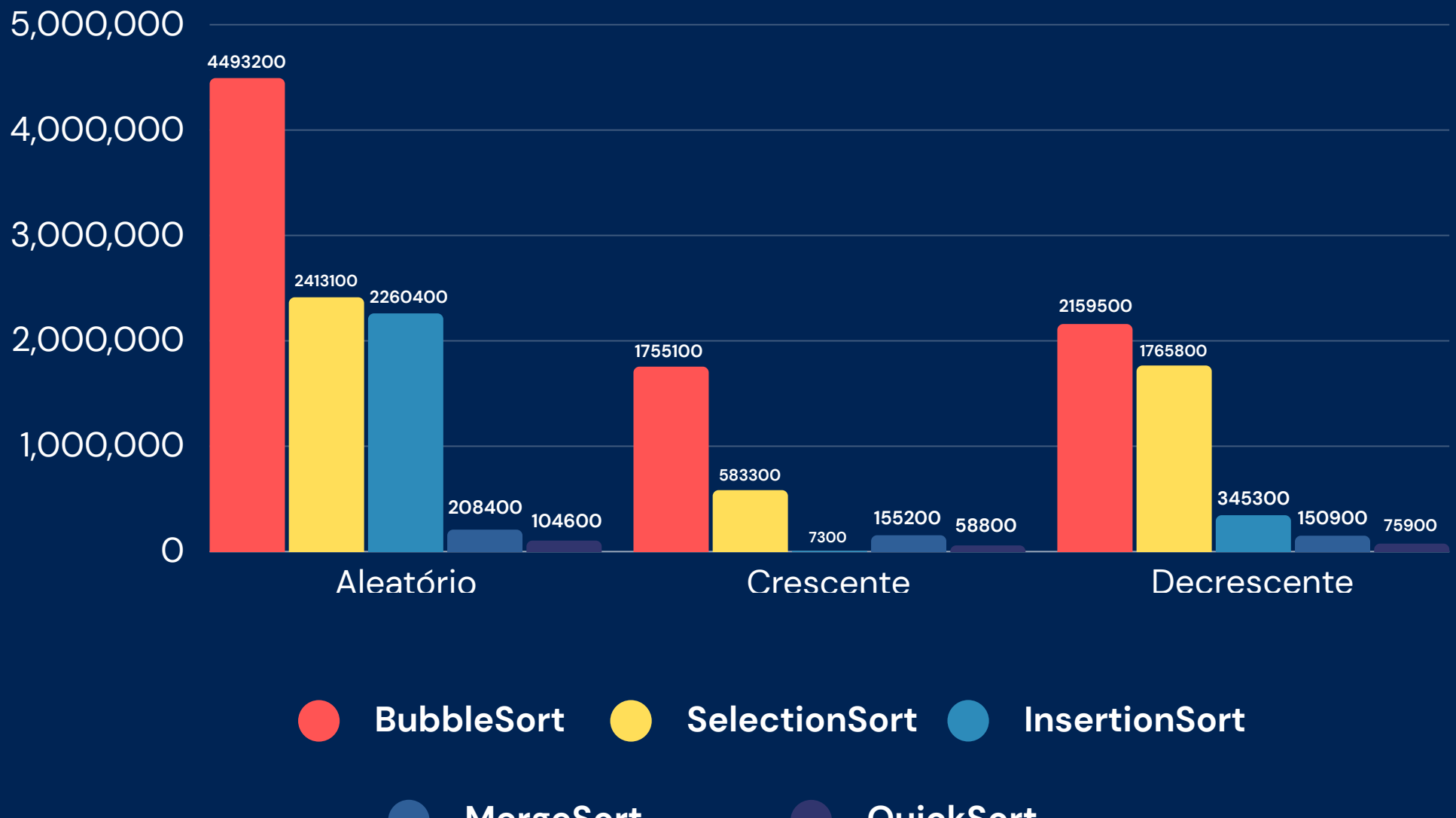
- **Métodos de Ordenação (2) :**

```java
    public static int partition(int arr[], int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j <= high - 1; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    public static void bubbleSort(int arr[], int size) {
        for (int i = 0; i < size - 1; i++) {
            for (int j = 0; j < size - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    public static void swap(int arr[], int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

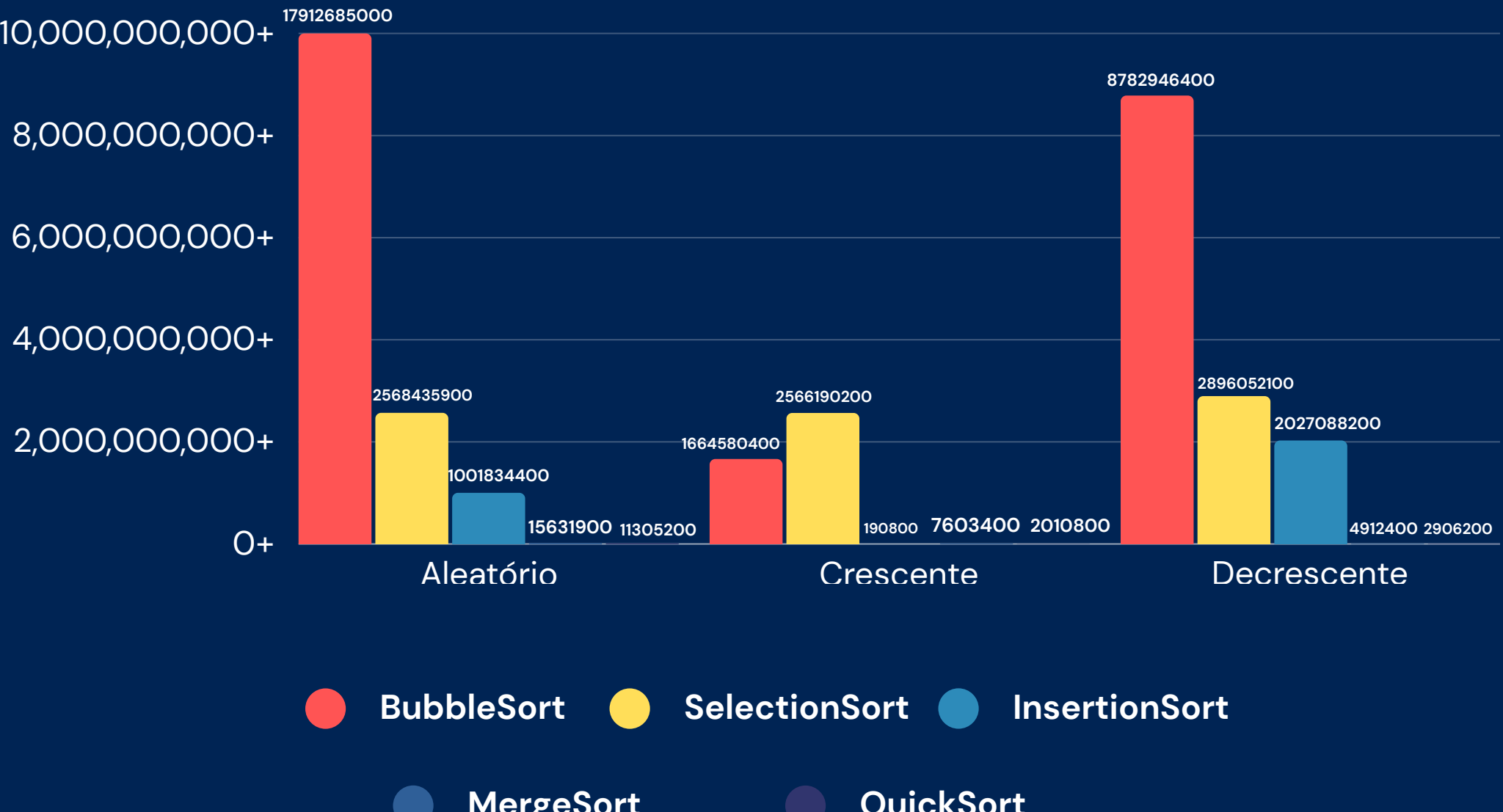
VETORES TAMANHO [100]



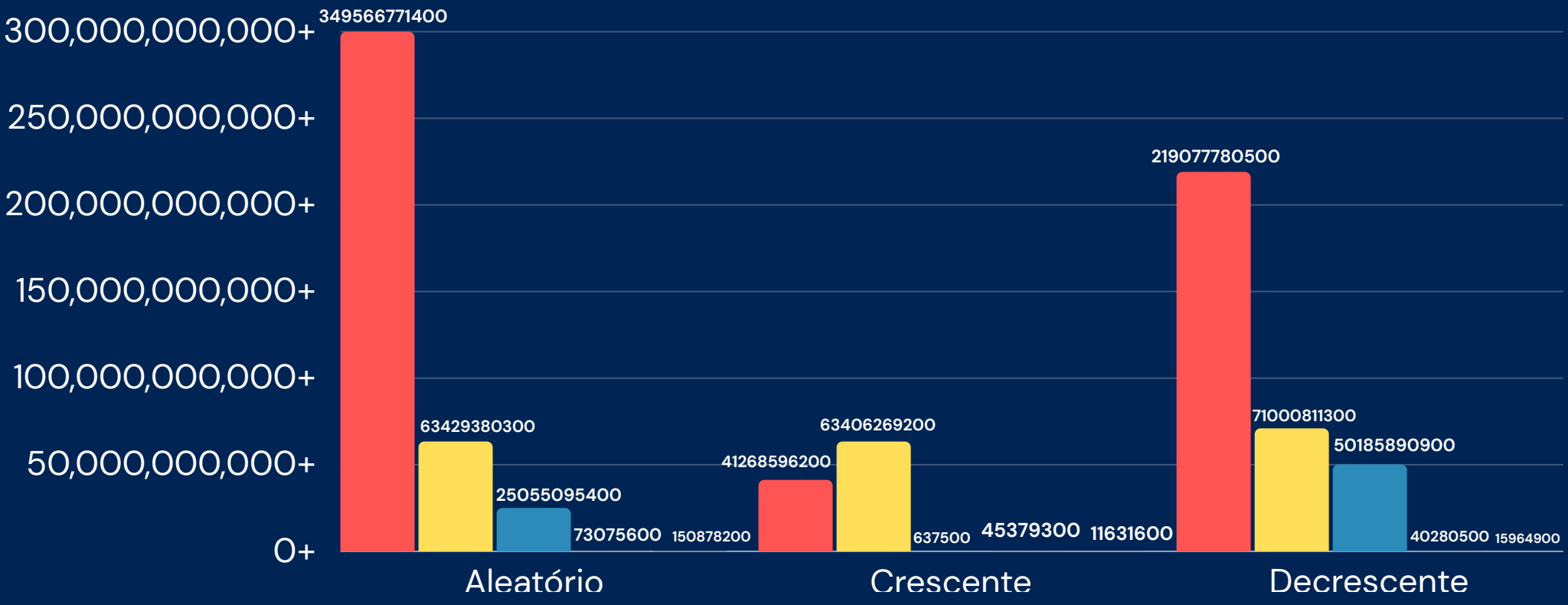
VETORES TAMANHO [1000]



VETORES TAMANHO [100K]



VETORES TAMANHO [500K]



BubbleSort SelectionSort InsertionSort

MergeSort QuickSort

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(1/2 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.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