Nome: Jonas Oliveira Silva Campos | Matrícula: 202304322741

Disciplina: Estrutura de dados | Turma: 3001 | Profª: Maria Bernadete

Estácio Nova América

Trabalho: Desafios na Escolha de Algoritmos de Ordenação

Para otimizar a ordenação dos dados, levando em conta o tipo de dado e a capacidade da máquina, João pode optar por usar Insertion Sort para sequências quase ordenadas, porque ele faz poucas comparações e rapidamente coloca tudo no lugar, rodando de forma eficiente com uma complexidade de tempo O(n).

Agora, se os dados estiverem completamente desordenados, o Merge Sort é mais adequado, pois ele tem uma performance garantida de O(n log n), o que o torna uma escolha segura para lidar com grandes volumes de dados, mesmo nos piores cenários.

A solução é combinar os dois: usar o Insertion Sort quando os dados estiverem quase ordenados e o Merge Sort quando estiverem desordenados. Isso deixa a ordenação rápida e eficiente, ajudando João a resolver os problemas de lentidão e entregar o projeto no prazo.

Código:

```
#include <stdio.h>
#include <stdlib.h>
using namespace std;

void mostrarArray(int arr[], int tam);
void insertionSort(int arr[], int tam);
void merge(int arr[], int esq, int meio, int dir);
void mergeSort(int arr[], int esq, int dir);
int contarInversoes(int arr[], int tam);
void verificarArray(int arr[], int tam);

int main() {
    // Arrays
    int arr1[] = {1, 2, 3, 4, 5};
    int arr2[] = {1, 3, 2, 4, 5};
```

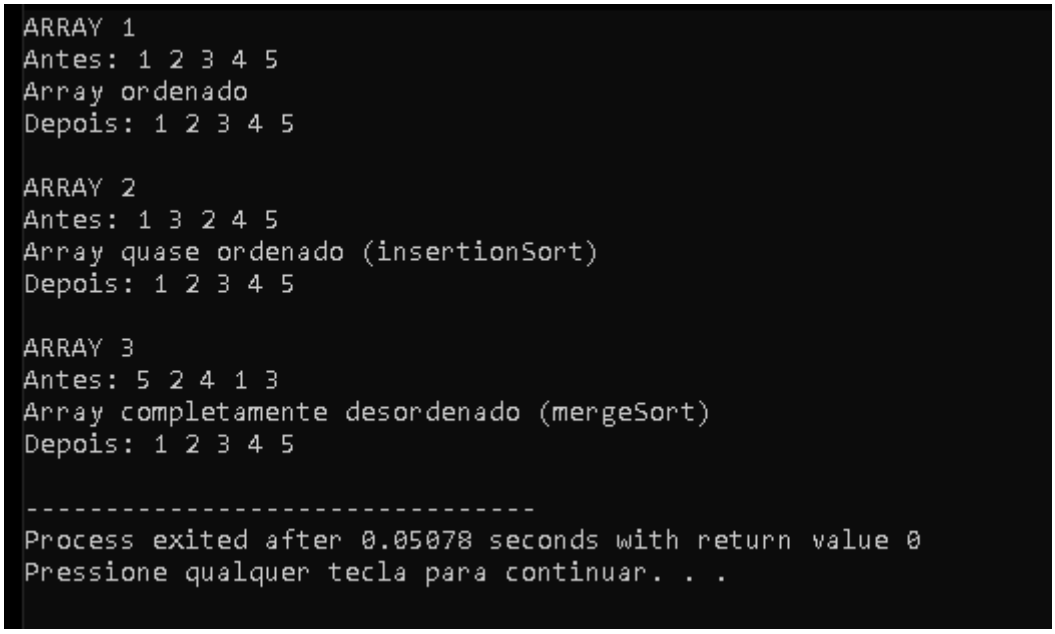
```
    int arr3[] = {5, 2, 4, 1, 3};

    // Tamanhos dos arrays
    int tam1 = sizeof(arr1) / sizeof(arr1[0]);
    int tam2 = sizeof(arr2) / sizeof(arr2[0]);
    int tam3 = sizeof(arr3) / sizeof(arr3[0]);

    // Array 1
    printf("ARRAY 1\nAntes: ");
    mostrarArray(arr1, tam1);
    verificarArray(arr1, tam1);
    printf("Depois: ");
    mostrarArray(arr1, tam1);

    // Array 2
    printf("\nARRAY 2\nAntes: ");
    mostrarArray(arr2, tam2);
    verificarArray(arr2, tam2);
    printf("Depois: ");
    mostrarArray(arr2, tam2);

    // Array 3
    printf("\nARRAY 3\nAntes: ");
    mostrarArray(arr3, tam3);
    verificarArray(arr3, tam3);
    printf("Depois: ");
    mostrarArray(arr3, tam3);

    return 0;
}

void mostrarArray(int arr[], int tam) {
```

```
    // Exibe o array
    for (int i = 0; i < tam; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

void insertionSort(int arr[], int tam) {
    // Ordena o array com insertionSort
    for (int i = 1; i < tam; i++) {
        int chave = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > chave){
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = chave;
    }
}

void merge(int arr[], int esq, int meio, int dir) {
    // Merge para combinar subarrays com alocação dinâmica
    int tamEsq = meio - esq + 1;
    int tamDir = dir - meio;
    int *esqArr = new int[tamEsq];
    int *dirArr = new int[tamDir];

    // Copia dados para arrays temporários
    for (int i = 0; i < tamEsq; i++)
        esqArr[i] = arr[esq + i];
    for (int i = 0; i < tamDir; i++)
        dirArr[i] = arr[meio + 1 + i];
```

```
    // Combina os arrays temporários de volta no array original
    int i = 0, j = 0, k = esq;
    while (i < tamEsq && j < tamDir) {
        if (esqArr[i] <= dirArr[j]) {
            arr[k] = esqArr[i];
            i++;
        } else {
            arr[k] = dirArr[j];
            j++;
        }
        k++;
    }

    // Copia os elementos restantes
    while (i < tamEsq)
        arr[k++] = esqArr[i++];
    while (j < tamDir)
        arr[k++] = dirArr[j++];

    delete[] esqArr;
    delete[] dirArr;
}

void mergeSort(int arr[], int esq, int dir) {
    // Divide o array e aplica o mergeSort
    if (esq < dir) {
        int meio  = esq + (dir - esq) / 2;
        mergeSort(arr, esq, meio);
        mergeSort(arr, meio + 1, dir);
        merge(arr, esq, meio, dir);
    }
}
```

```
int contarInversoes(int arr[], int tam) {
    // Conta o número de inversões no array
    int inversoes = 0;
    for (int i = 0; i < tam - 1; i++){
        for (int j = i + 1; j < tam; j++){
            if (arr[i] > arr[j])
                inversoes++;
        }
    }
    return inversoes;
}

void verificarArray(int arr[], int tam) {
    // Verifica o estado do array e decide qual algoritmo de ordenação usar
    int inversoes = contarInversoes(arr, tam);
    int max_inv = (tam * (tam - 1)) / 2;

    if (inversoes == 0){
        printf("Array ordenado\n");
    } else if (inversoes >  max_inv * 0.1) {
        printf("Array completamente desordenado (mergeSort)\n");
        mergeSort(arr, 0, tam - 1);
    } else {
        printf("Array quase ordenado (insertionSort)\n");
        insertionSort(arr, tam);
    }
}
```

Saída:


ARRAY 1
Antes: 1 2 3 4 5
Array ordenado
Depois: 1 2 3 4 5
ARRAY 2
Antes: 1 3 2 4 5
Array quase ordenado (insertionSort)
Depois: 1 2 3 4 5
ARRAY 3
Antes: 5 2 4 1 3
Array completamente desordenado (mergeSort)
Depois: 1 2 3 4 5

Process exited after 0.05078 seconds with return value 0
Pressione qualquer tecla para continuar. . .