ETEC PARQUE BELÉM

Ensino Médio com Habilitação Profissional Técnico em Desenvolvimento de Sistema

Matheus Lourenço Soares

Exercícios Vetor

São Paulo

2024

Matheus Lourenço Soares

Exercícios Vetor

Trabalho a ser entregue para obtenção de nota total ou parcial na disciplina de Técnicas de Programação em Algoritmo para a avaliação do professor Juliano Ratusnei sobre o tema vetores em C realizado pelos alunos

Matheus Lourenço Soares

Prof: Juliano Ratuznei.

São Paulo

2024

RESUMO

Este projeto tem como objetivo explorar a aplicação dos algoritmos "QuickSort" e "BubbleSort" em um vetor contendo 1000 elementos gerados aleatoriamente. Utilizando os conhecimentos adquiridos nas aulas de Técnicas de Programação em Algoritmos e aplicando criatividade e raciocínio para resolver o problema, conseguimos desenvolver uma solução que atende às exigências propostas. Através de pesquisa e utilizando o material complementar fornecido pelo professor Juliano Ratusznei, foi possível compreender a melhor abordagem para alcançar uma solução lógica e eficaz.

ABSTRACT

This project aims to explore the application of the "QuickSort" and "BubbleSort" algorithms on a vector containing 1000 randomly generated elements. Leveraging the knowledge acquired during the Algorithm Programming Techniques classes and applying creativity and problem-solving skills, we developed a solution that meets the project's requirements. Through research and utilizing supplementary materials provided by Professor Juliano Ratusznei, we gained insights into the best approach to achieve a logical and effective solution.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Quicksort: O método Quicksort recebe uma lista de elementos, seleciona um dos elementos como o 'pivô', e rearranja os outros elementos para que os valores menores estejam à esquerda do elemento pivô e os valores maiores estejam à direita dele.

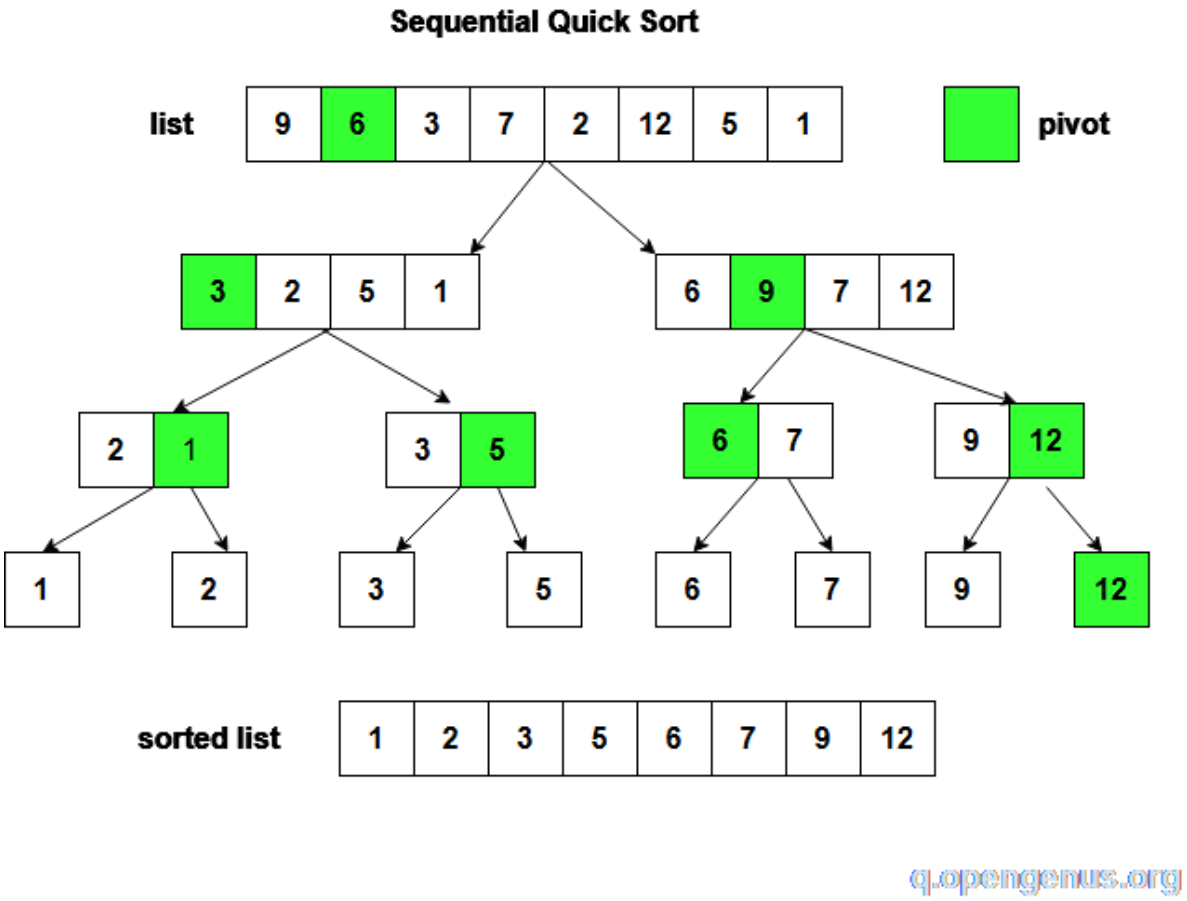

Figura 1: QuickSort

BubbleSort: É um método básico de classificação que recebe uma sequência de elementos como entrada e gera uma sequência organizada conforme um critério. É um método conhecido, porém menos eficiente comparado a outros métodos de classificação.

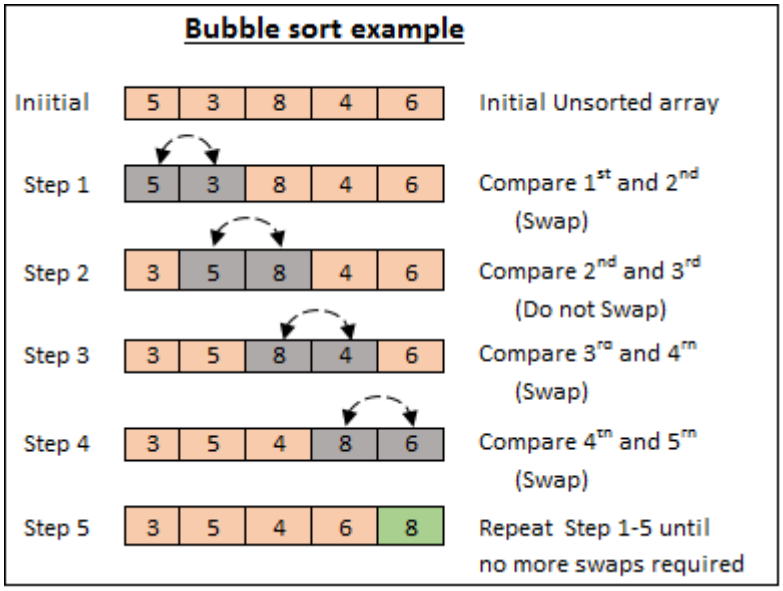

Figura 2: BubbleSort

MergeSort: Este algoritmo de ordenação por mistura divide recursivamente a lista em sublistas até que cada sublista contenha um único elemento e, em seguida, mescla as sublistas de maneira ordenada para produzir a lista final ordenada. Ele é estável e eficiente, especialmente em grandes conjuntos de dados.

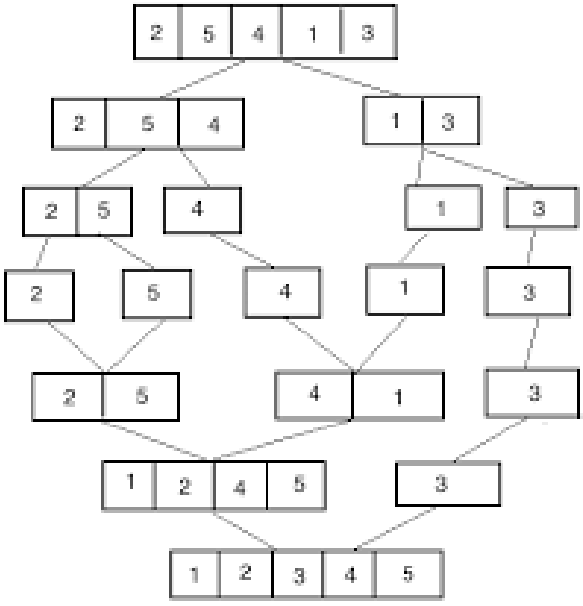

Figura 3: MergeSort

1.1 Tema

O tema do trabalho se baseia na utilização da linguagem C para implementarmos três algoritmos clássicos de ordenação: “QuickSort”, “MergeSort” e “BubbleSort”.

- QuickSort: Este algoritmo escolhe um elemento como pivô e particiona os elementos em torno dele, de modo que os menores fiquem à esquerda e os maiores à direita. Ele é eficiente em média, mas pode ter pior desempenho em alguns casos.

- MergeSort: Este algoritmo divide a lista em sublistas até que cada uma contenha um único elemento e depois as mescla de maneira ordenada. Ele é estável e eficiente para grandes conjuntos de dados.

- BubbleSort: Este algoritmo compara repetidamente pares adjacentes de elementos e os troca se estiverem na ordem errada. Ele é fácil de implementar, mas menos eficiente em listas grandes.

O trabalho incluirá a implementação dos algoritmos e testes comparativos para avaliar o desempenho de cada um, discutindo suas vantagens e desvantagens.

1.2 Problema

Nosso desafio principal está na execução completa do projeto, especialmente na implementação eficiente da ordenação utilizando os algoritmos “QuickSort”, “MergeSort” e “BubbleSort”.

1.3 Objetivos

Nosso objetivo principal é ampliar nosso conhecimento na linguagem C através do uso do aplicativo DevC++ e VS Code, o que nos permitirá aprimorar nossas habilidades e facilitar a implementação dos algoritmos de ordenação mencionados.

2 METODOLOGIA

1. Crie um vetor de 8 valores monetários de produtos o qual o usuário insere os valores

```c
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>

float valor[8];

int main(void) {
    setlocale(LC_ALL,"Portuguese");

    int i;

    for(i = 0; i < 8; i++) {
        printf("Insira o %d valor: ", i+1);
        scanf("%f", &valor[i]);
    }

    printf("Os valores inseridos s o: ");
    for(i = 0; i < 8; i++) {
        printf("%.2f\t", valor[i]);
    }

    getchar();
}
```

---

2. Crie um vetor que armazene 15 números inteiros inseridos pelo usuário

```c
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>

int valor[15];

int main(void) {
    setlocale(LC_ALL,"Portuguese");

    int i;

    for(i = 0; i < 15; i++) {
        printf("Insira o %d° valor: ", i+1);
        scanf("%d", &valor[i]);
    }

    printf("Os valores inseridos são: ");
    for(i = 0; i < 15; i++) {
        printf("%d\t", valor[i]);
    }

    getchar();
```

```
}
```

3. Crie um vetor que armazene os 60 números naturais maiores que 0.

```c
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>

int valor[60];

int main(void) {
    setlocale(LC_ALL,"Portuguese");

    int i;

    for(i = 0; i < 60; i++){
        valor[i] = i + 1;
        printf("%d\t", valor[i]);
    }

    getchar();
}
```

4. Crie um vetor de 5 nomes e outro vetor de 5 alturas mostre ao usuário os nomes e as respectivas alturas.

```c
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

char nome[5][50] = {"Ana","Julia","Anderson","Carlos","Gusmão"};
float altura[5];

int main(void) {
    setlocale(LC_ALL, "Portuguese");

    int i;

    altura[0] = 1.67;
    altura[1] = 1.80;
    altura[2] = 1.77;
    altura[3] = 1.45;
    altura[4] = 1.32;

    printf("Nomes e Alturas:\n");
    for (i = 0; i < 5; i++){
        printf("%s %.2f\n", nome[i], altura[i]);
    }

    getchar();
}
```

5. Crie um vetor de 5 nomes inseridos pelo usuário e outro vetor de 5 alturas inseridos pelo usuário. O programa deve exibir o nome e a altura do usuario distinto na mesma linha.

```c
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>

char nome[5][50];
float altura[5];

int main(void){
    setlocale(LC_ALL, "Portuguese");

    int i;

    for (i = 0; i < 5; i++){
        printf("Insira o %d nome:\n", i + 1);
        scanf("%s", nome[i]);
    }

    for (i = 0; i < 5; i++){
        printf("Insira a %d altura:\n", i + 1);
        scanf("%f", &altura[i]);
    }

    printf("Nomes e alturas:\n");
    for (i = 0; i < 5; i++){
        printf("%s %.2f\n", nome[i], altura[i]);
    }

    getchar();
}
```

---

6. Crie um vetor com 30 números pares maiores que um número inserido pelo usuário.

```c
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int x[30];

int main(void) {
    setlocale(LC_ALL, "portuguese");

    int i, num, num2;

    printf("Insira seu n mero: ");
    scanf("%d", &num);
```

```
    if(num % 2 == 0){
        for(i = 0; i < 30; i++){
            x[i] = num += 2;
            printf("%d\t", x[i]);
        }
    } else {
        (num2 = num + 1);
        for(i = 0; i < 30; i++){
            x[i] = num2 += 2;
            printf("Os %d\t", x[i]);
        }
    }

getchar();
}
```

---

7. Crie um vetor com 30 números ímpares maiores que um número inserido pelo usuário quando o número for divisível por 3 escreva "PIM" no lugar do número.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int x[30];

int main(void) {
    setlocale(LC_ALL, "Portuguese");

    int i, num, num2;

    printf("Insira seu n◆mero: ");
    scanf("%d", &num);

    if (num % 2 == 1) {
        for (i = 0; i < 30; i++) {
            x[i] = num + 2 * i;
            if (x[i] % 3 == 0) {
                printf("PIM\t");
```

```
            } else {
                printf("%d\t", x[i]);
            }
        }
    } else {
        num2 = num + 1;
        for (i = 0; i < 30; i++) {
            x[i] = num2 + 2 * i;
            if (x[i] % 3 == 0) {
                printf("PIM\t");
            } else {
                printf("%d\t", x[i]);
            }
        }
    }

    getchar();
}
```

---

8. Crie um vetor com 200 números gerados de 0 até 300 e mostre ao usuário.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int numA[200];

int main(void) {
    setlocale(LC_ALL, "Portuguese");

    srand(time(NULL));
    int i;

    for (i = 0; i < 200; i++) {
        numA[i] = rand() % 300;
        printf("%d\n", numA[i]);
    }

    getchar();
}
```

9. Crie um vetor com 100 números distintos gerados de 0 até 200 e mostre ao usuário.

```c
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int numA[100];

int main(void) {
    setlocale(LC_ALL, "Portuguese");

    srand(time(NULL));

    int i = 0, j, ver;

    do{
        numA[i] = rand() % 200;
        ver = 0;
        for(j = 0; j < i; j++){
            if(numA[j] == numA[i])
                ver = 1;
        }
        if(ver == 0)
            i++;
    }while(i < 100);

    for(i = 0; i < 100; i++){
        printf("%d\t", numA[i]);
    }

    getchar();
}
```

10. Crie um vetor com 200 números gerados de 0 até 300 e mostre ao usuário ordenado em ordem crescente.

```c
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int numA[200];

int main(void) {
    setlocale(LC_ALL, "Portuguese");

    srand(time(NULL));
    int i, j, numT;

    for (i = 0; i < 200; i++) {
        numA[i] = rand() % 300;
```

```
    }

    for (i = 0; i < 200 - 1; i++) {
        for (j = 0; j < 200; j++) {
            if (numA[j] > numA[j + 1]) {
                numT = numA[j];
                numA[j] = numA[j + 1];
                numA[j + 1] = numT;
            }
        }
    }

    printf("\nVetor ordenado em ordem crescente:\n");
    for (i = 0; i < 200; i++) {
        printf("%d ", numA[i]);
    }
    printf("\n");

    getchar();
}
```

---

11. Crie um vetor com 100 números distintos gerados de 0 até 200 e mostre ao usuário em ordem decrescente.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int numA[100];

int main(void) {
    setlocale(LC_ALL, "Portuguese");

    srand(time(NULL));

    int i = 0, j, ver, numT;

    do{
        numA[i] = rand() % 200;
        ver = 0;
        for(j = 0; j < i; j++){
            if(numA[j] == numA[i])
                ver = 1;
        }
        if(ver == 0)
            i++;
    }while(i < 100);

    for (i = 0; i < 100 - 1; i++) {
        for (j = 0; j < 100; j++) {
            if (numA[j] < numA[j + 1]) {
                numT = numA[j];
                numA[j] = numA[j + 1];
                numA[j + 1] = numT;
```

```
            }
        }
    }

    printf("\nVetor ordenado em ordem decrecente:\n");
    for (i = 0; i < 100; i++) {
        printf("%d ", numA[i]);
    }

    getchar();
}
```

---

12. Busque e aplique em um vetor com 1000 elementos aleatórios os algoritmos "Quick-Sort", "MergeSort" e "BubbleSort".

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int particionar(int arr[], int inicio, int fim);

void quickSort(int arr[], int inicio, int fim);

void mergeSort(int arr[], int inicio, int fim);

void merge(int arr[], int inicio, int meio, int fim);

void bubbleSort(int arr[], int n);

void preencherVetor(int arr[], int tamanho);

void exibirVetor(int arr[], int tamanho);

int main() {
    int vetor_quick[1000], vetor_merge[1000], vetor_bubble[1000];
    srand(time(NULL));

    preencherVetor(vetor_quick, 1000);
    preencherVetor(vetor_merge, 1000);
    preencherVetor(vetor_bubble, 1000);

    quickSort(vetor_quick, 0, 999);
    printf("Vetor ordenado com Quick Sort:\n");
    exibirVetor(vetor_quick, 1000);
    printf("\n");

    mergeSort(vetor_merge, 0, 999);
    printf("Vetor ordenado com Merge Sort:\n");
    exibirVetor(vetor_merge, 1000);
    printf("\n");

    bubbleSort(vetor_bubble, 1000);
    printf("Vetor ordenado com Bubble Sort:\n");
    exibirVetor(vetor_bubble, 1000);
```

```
    printf("\n");

    return 0;
}

int particionar(int arr[], int inicio, int fim) {
    int pivo = arr[fim];
    int i = inicio - 1;
    int j;

    for (j = inicio; j < fim; j++) {
        if (arr[j] <= pivo) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[fim];
    arr[fim] = temp;

    return (i + 1);
}

void quickSort(int arr[], int inicio, int fim) {
    if (inicio < fim) {
        int pi = particionar(arr, inicio, fim);

        quickSort(arr, inicio, pi - 1);
        quickSort(arr, pi + 1, fim);
    }
}

void mergeSort(int arr[], int inicio, int fim) {
    if (inicio < fim) {
        int meio = inicio + (fim - inicio) / 2;


        mergeSort(arr, inicio, meio);
        mergeSort(arr, meio + 1, fim);

        merge(arr, inicio, meio, fim);
    }
}

void merge(int arr[], int inicio, int meio, int fim) {
    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int L[n1], R[n2];

    int i, j, k;
    for (i = 0; i < n1; i++)
        L[i] = arr[inicio + i];
    for (j = 0; j < n2; j++)
```

```
        R[j] = arr[meio + 1 + j];

    i = 0;
    j = 0;
    k = inicio;
    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++;
    }
}

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

void preencherVetor(int arr[], int tamanho) {
    int i;
    for (i = 0; i < tamanho; i++) {
        arr[i] = rand() % 1000;
    }
}

void exibirVetor(int arr[], int tamanho) {
    int i;
    for (i = 0; i < tamanho; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

3 RECURSOS

Os recursos fundamentais utilizados para a realização deste trabalho foram as plataformas DevC++ e VS Code, que desempenharam um papel crucial na programação e configuração do ambiente para a execução eficaz dos algoritmos de ordenação.

4 CONCLUSÃO

Com esse trabalho, adquirimos um entendimento significativo sobre a importância dos algoritmos de ordenação como o "BubbleSort", "QuickSort" e "MergeSort". Aprendemos como esses algoritmos facilitam e otimizam o processo de ordenação de dados, contribuindo para a eficiência e organização em diversas aplicações computacionais.

REFERÊNCIAS

ElemarJr.; **O que é e como funciona o BubbleSort?**. Disponível em: https://elemarjr.com/clube-de-estudos/artigos/o-que-e-e-como-funciona-o-bubblesort/

Acessado em 20/05/2024.

W3schools.; **DSA Quicksort** Disponível em: https://www.w3schools.com/dsa/dsa_algo_quicksort.php

Acessado em 21/05/2024.

W3schools.; **DSA Mergesort** Disponível em: https://www.w3schools.com/dsa/dsa_algo_mergesort.php#:~:text=The%20Merge%20Sort%20algorithm%20is,so%20that%20it%20is%20sorted.

Acessado em 23/05/2024.