Universidade Evangélica de Goiás – UniEVANGÉLICA

CURSO DE ENGENHARIA DE SOFTWARE

Atividade pré-aula (semana 04)

Antônio Claudio Ferreira Filho

Matrícula: 2110854

Anápolis - GO

2023

Antônio Claudio Ferreira Filho

Atividade pré-aula (semana 04)

Trabalho apresentado à disciplina de Árvores e grafos como requisito parcial para aprovação.

Anápolis – GO

2023

Faça uma breve pesquisa sobre o Método de Ordenação QuickSort e faça o que se pede:

1 - Vantagens e Desvantagens sobre os outros métodos de ordenação

2 - Mostre o código de implementação em C ou em Python.

O Método de Ordenação QuickSort é um dos algoritmos mais populares para ordenação de arrays. Ele utiliza a técnica de divisão e conquista para dividir o array em subarrays menores e ordená-los recursivamente. O QuickSort tem uma complexidade média de O(n log n), o que o torna eficiente para grandes conjuntos de dados.

Algumas vantagens do QuickSort em relação a outros métodos de ordenação são:

1. Desempenho rápido em média e melhor caso: O QuickSort tem uma complexidade média de O(n log n) e é muito rápido em conjuntos de dados maiores.
2. Uso eficiente de memória: O QuickSort usa memória de forma eficiente, o que o torna adequado para dispositivos com recursos limitados.
3. Possibilidade de paralelização: O QuickSort pode ser facilmente paralelizado, o que o torna ideal para hardware moderno com vários núcleos de processamento.

Algumas desvantagens do QuickSort em relação a outros métodos de ordenação são:

1. Desempenho ruim no pior caso: O QuickSort pode ter um desempenho muito ruim no pior caso, quando o conjunto de dados está quase ordenado ou totalmente reverso. Nesses casos, a complexidade do algoritmo pode chegar a O(n^2).
2. Não é estável: O QuickSort não preserva a ordem relativa dos elementos com chaves iguais.
3. Requer funções de comparação: O QuickSort requer uma função de comparação para determinar a ordem dos elementos, o que pode ser um problema em alguns casos.

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

```c
void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```

```c
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++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

```c
void quicksort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quicksort(arr, low, pi - 1);
        quicksort(arr, pi + 1, high);
    }
}
```

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

```c
int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    quicksort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```