Universidade Evangélica de Goiás – UniEVANGÉLICA
CURSO DE ENGENHARIA DE SOFTWARE

Aprendendo a resolver problemas (ciclo 02)

Antônio Claudio Ferreira Filho

Matrícula: 2110854

Anápolis - GO

2023

Antônio Claudio Ferreira Filho

Aprendendo a resolver problemas (ciclo 02)

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

Anápolis – GO

2023

a. Métodos de ordenação implementados utilizando a linguagem Python:

a. BubbleSort:

```python
def bubbleSort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    print(arr)
```

b. SelectionSort:

```python
def selectionSort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[min_idx] > arr[j]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    print(arr)
```

c. InsertionSort:

```python
def insertionSort(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    print(arr)
```

d. QuickSort:

```python
def partition(arr, low, high):
    i = low - 1
    pivot = arr[high]
    for j in range(low, high):
        if arr[j] < pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

def quickSort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quickSort(arr, low, pi - 1)
        quickSort(arr, pi + 1, high)
    print(arr)
```

e. MergeSort:

```
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    L = [0] * n1
    R = [0] * n2
    for i in range(n1):
        L[i] = arr[l + i]
    for j in range(n2):
        R[j] = arr[m + 1 + j]
    i = 0
    j = 0
    k = l
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

def mergeSort(arr, l, r):
    if l < r:
        m = (l + r) // 2
        mergeSort(arr, l, m)
        mergeSort(arr, m + 1, r)
        merge(arr, l, m, r)
    print(arr)
```

f. HeapSort:

```
def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < n and arr[i] < arr[l]:
        largest = l

    if r < n and arr[largest] < arr[r]:
        largest = r

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)


def heapSort(arr):
    n = len(arr)

    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

    print(arr)
```

b. Métodos de busca:

a. Busca sequencial:

```python
def busca_sequencial(array, elemento):
    for i in range(len(array)):
        if array[i] == elemento:
            return i
    return -1
```

b. Busca binária:

```python
def busca_binaria(array, elemento):
    inicio = 0
    fim = len(array) - 1
    while inicio <= fim:
        meio = (inicio + fim) // 2
        if array[meio] == elemento:
            return meio
        elif array[meio] < elemento:
            inicio = meio + 1
        else:
            fim = meio - 1
    return -1
```