ANÁLISE DE ALGORITMOS

ALISSON LÜDTKE
FILIPE TIETBÖHL
GABRIEL SCHRAMM
WILLIAN CAVALHEIRO

ALGORITMOS DE ORDENAMENTO

ANÁLISE EFICIÊNCIA

UNIRITTER
2023

Bubble_Sort

$$f(n) = C1+C2(N+1)+C3(N+1)+C4(N)+C5(N)$$

$$f(n) = n^2$$

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

Bubble_Otimizado

$$f(n) = C1+C2+C3(N+1)+C4(N)+C5(N+1)+C6(N)+C7(N)$$

$$f(n) = n^2$$

```python
def bubble_sort(arr):
    n = len(arr)
    swapped = True
    while swapped:
        swapped = False
        for i in range(1, n):
            if arr[i - 1] > arr[i]:
                arr[i - 1], arr[i] = arr[i], arr[i - 1]
                swapped = True
        n -= 1
```

Selection_sort

$f(n) = C1(N)+C2(N)+C3(N+1)+C4(N)+C5+C6$

$f(n) = n^2$

```python
def selection_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
```

Selection_Otimizado

$$f(n) = C1(N)+C2(N)+C3(N+1)+C4(N)+C5(N+1)+C6(N)+C7$$

$$f(n) = n^2$$

```python
def selection_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        if min_idx != i:
            arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
```

Insertion_Sort

$$f(n) = C1(N)+C2+C3+C4(N)+C5(N)+C6+C7+C8(N+1)+C9+C10+C11$$

$$f(n) = n^2$$

```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1

        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1

        arr[j + 1] = key

    return arr
```

Insertion_Otimizado

$f(n) = C1+C2(N)+C3+C4$

$f(n) = n^2$

```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]

        for j in range(i - 1, -1, -1):
            if arr[j] >= key:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
            else:
                break

    return arr
```

Merge_Sort

$$f(n) = C1 + C2 + C3(n) + C4 + C5(n) + C6(n+1)+C7+C8(n)+C9(n)+C10+C11+C12+C13(n+1)+C14(n+1)+C15(n+1)$$

$$f(n) = n\ log(n)$$

```python
def mergeSort(array, inicio=0, fim=None):
    if fim is None:
        fim = len(array)
    if fim - inicio > 1:
        meio = (fim + inicio) // 2
        mergeSort(array, inicio, meio)
        mergeSort(array, meio, fim)
        merge(array, inicio, meio, fim)

def merge(array, inicio, meio, fim):
    left = array[inicio:meio]
    right = array[meio:fim]
    top_left, top_right = 0, 0
    for k in range(inicio, fim):
        if top_left >= len(left):
            array[k] = right[top_right]
            top_right += 1
        elif top_right >= len(right):
            array[k] = left[top_left]
            top_left += 1
        elif left[top_left] < right[top_right]:
            array[k] = left[top_left]
            top_left += 1
        else:
            array[k] = right[top_right]
            top_right += 1
```

Merge_Hibrido

$$f(n) = C1(N+1)+C2(N)+C3(N)+C4+C5+C6(N)+C7(N)+C8(N+1)+C9+C10+C11+C12+C13$$

$$f(n) = n\ (\log n)$$

```python
def merge_sort_hibrido(arr):
    if len(arr) <= 16:
        insertion_sort(arr)
    else:
        meio = len(arr) // 2
        esquerda = arr[:meio]
        direita = arr[meio:]
        merge_sort_hibrido(esquerda)
        merge_sort_hibrido(direita)
        merge(arr, esquerda, direita)
def merge(arr, esquerda, direita):
    i = j = k = 0
    while i < len(esquerda) and j < len(direita):
        if esquerda[i] < direita[j]:
            arr[k] = esquerda[i]
            i += 1
        else:
            arr[k] = direita[j]
            j += 1
        k += 1
    while i < len(esquerda):
        arr[k] = esquerda[i]
        i += 1
        k += 1
    while j < len(direita):
        arr[k] = direita[j]
        j += 1
        k += 1
def insertion_sort(arr):
    for i in range(1, len(arr)):
        chave = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > chave:
            arr[j + 1] = arr[j]
```

Heap_sort

$$f(n) = C1+C2+C3+C4+C5+C6(N+1)+C7(N+1)+C8+C9(N+1)+C10(N+1)+C11+C12(N)+C13+C14$$

$$f(n) = n\ (\log n)$$

```python
def heapify(arr, n, i):
    maior = i
    esquerda = 2 * i + 1
    direita = 2 * i + 2

    if esquerda < n and arr[i] < arr[esquerda]:
        maior = esquerda

    if direita < n and arr[maior] < arr[direita]:
        maior = direita

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


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

    # Construindo o heap máximo
    for i in range(tamanho // 2 - 1, -1, -1):
        heapify(arr, tamanho, i)

    # Extrai os elementos um por um
    for i in range(tamanho - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]  # Troca o primeiro e último elemento
        heapify(arr, i, 0)
```

Heap_Iterativo

$$f(n) = C1+C2+C3+C4+C5+C6+C7(N)+C8(N)+C9+C10(N)$$

$$f(n) = n\ (\log n)$$

```python
def heap_sort_iterativo(arr):
    n = len(arr)

    # Construir o heap máximo
    for i in range(n // 2 - 1, -1, -1):
        heapfy(arr, n, i)

    # Extrair elementos do heap um por um
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]  # Troca
        heapfy(arr, i, 0)

def heapfy(arr, n, i):
    maior = i
    esquerda = 2 * i + 1
    direita = 2 * i + 2

    if esquerda < n and arr[esquerda] > arr[maior]:
        maior = esquerda

    if direita < n and arr[direita] > arr[maior]:
        maior = direita

    if maior != i:
        arr[i], arr[maior] = arr[maior], arr[i]  # Troca
        heapfy(arr, n, maior)
```

Quick Sort

$$f(n) = C1(N)+C2(N)+C3(N)+C4+C5+C6+C7+C8(N)+C9(N)$$

$$f(n) = n\ (\log n\ )$$

```python
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivo = arr[0]
        menores = [x for x in arr[1:] if x <= pivo]
        maiores = [x for x in arr[1:] if x > pivo]
        return quicksort(menores) + [pivo] + quicksort(maiores)
```

Quick_Randomizado

$$f(n) = C1(N+1)+C2+C3(N)+C4(N)+C5+C6(N)+C7+C8(N)+C9+C10+C11$$

$$f(n) = n\ (\log n)$$

```python
def quick_sort_randomizado(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = random.choice(arr)
        menores = [x for x in arr if x < pivot]
        iguais = [x for x in arr if x == pivot]
        maiores = [x for x in arr if x > pivot]
        return quick_sort_randomizado(menores) + iguais + quick_sort_randomizado(maiores)
```

GRÁFICO

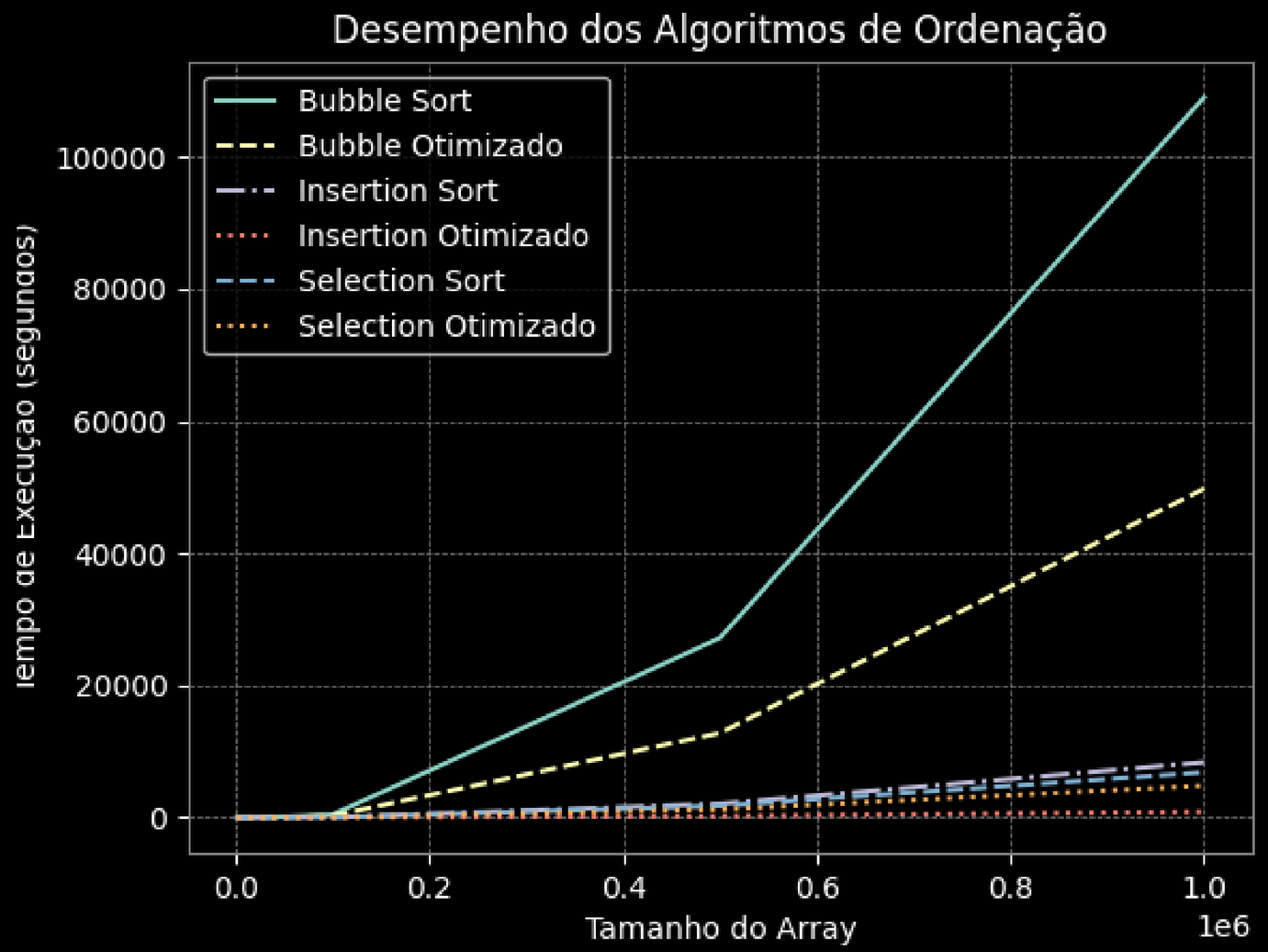
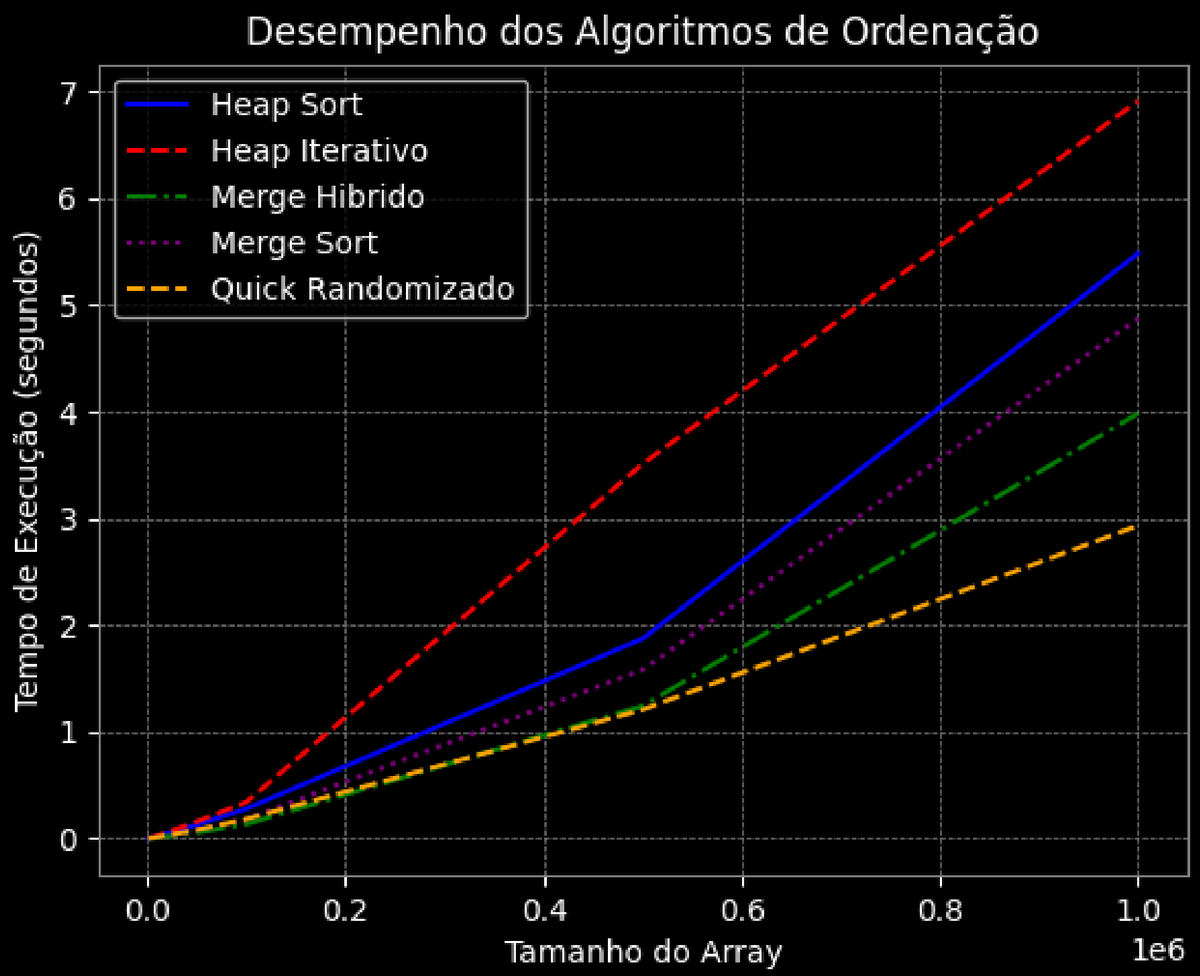

Desempenho dos Algoritmos de Ordenação
Bubble Sort
Bubble Otimizado
Insertion Sort
Insertion Otimizado
Selection Sort
Selection Otimizado
100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Tamanho do Array

GRÁFICO ALGORITMOS ESPECIAIS



COMPARAÇÃO FUNÇÃO VS EMPÍRICO