Estruturas de Dados 2

Prof. Silvia Brandão

2024.1

Momento N2 – 25 pts

- **Atividade avaliativa** – 15pts
  - Implementação e avaliação comparativa dos métodos de ordenação (10pts):
    - Quick Sort; Merge Sort; Shell Sort e Heap Sort
  - Apresentação (5pts): códigos e gráfico comparativo das curvas de desempenho dos n dados x tempo de ordenação
  - **Data de entrega e apresentação**: 22/05, postagem dos arquivos e gráfico no diário de bordo (basta um aluno postar, com o nome dos integrantes da equipe. **Só receberei o trabalho pelo diário de bordo**.)
- **Avaliação Subjetiva** sobre métodos de ordenação – 10pts **(Google Forms)**

Momento N3 – 25 pts

- Projeto Prático
  - Ver arquivo com descrição do projeto (tema, avaliação e apresentação)

Algoritmo Merge Sort

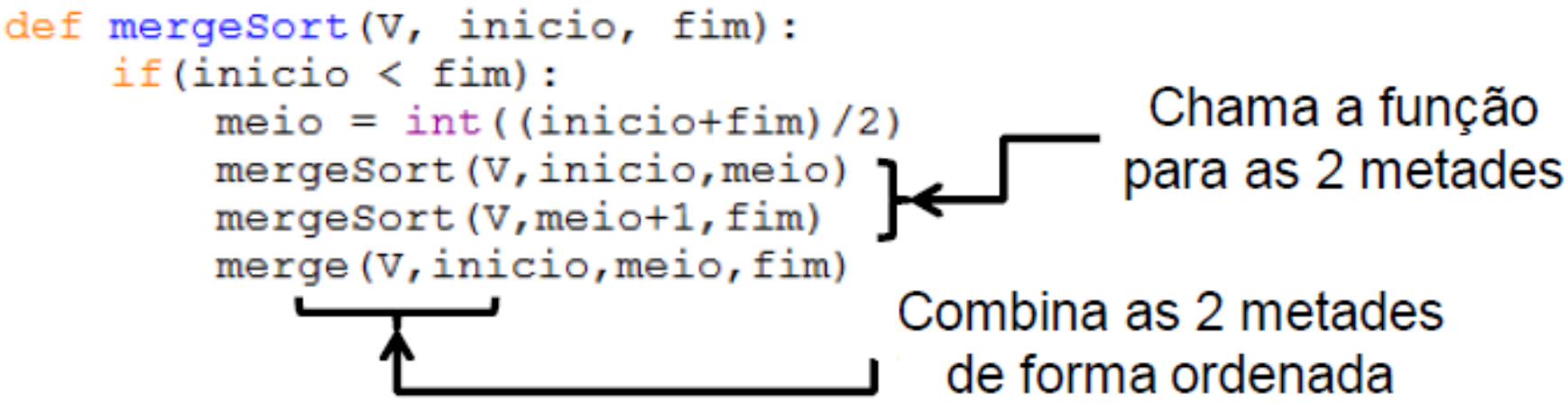
- Também conhecido como ordenação por intercalação
  - Algoritmo recursivo que usa a ideia de *dividir para conquistar* para ordenar os dados
    - Parte do princípio de que é mais fácil ordenar um conjunto com poucos dados do que um com muitos
- O algoritmo divide os dados em conjuntos cada vez menores para depois ordená-los e combina-los por meio de intercalação (merge)

Algoritmo Merge Sort

- **Funcionamento:**
  - Divide, recursivamente, o array em duas partes
    - Continua até cada parte ter apenas um elemento
  - Em seguida, combina dois array de forma a obter um array maior e ordenado
    - A combinação é feita intercalando os elementos de acordo com o sentido da ordenação (crescente ou decrescente)
  - Este processo se repete até que exista apenas um array

Algoritmo Merge Sort

- O algoritmo usa 2 funções
  - mergeSort: divide os dados em arrays cada vez menores
  - merge: intercala os dados de forma ordenada em um array maior



Função mergeSort()

Algoritmo Merge Sort

Função merge()

```
def merge(V, inicio, meio, fim):
    tamanho = fim-inicio+1
    p1 = inicio
    p2 = meio+1
    fim1 = False
    fim2 = False

    temp = [0 for i in range(tamanho)]

    for i in range(tamanho):
        if(not fim1 and not fim2):
            if(V[p1] < V[p2]):
                temp[i] = V[p1]
                p1 = p1 + 1
            else:
                temp[i] = V[p2]
                p2 = p2 + 1

            if(p1 > meio):
                fim1 = True

            if(p2 > fim):
                fim2 = True
        else:
            if(not fim1):
                temp[i] = V[p1]
                p1 = p1 + 1
            else:
                temp[i] = V[p2]
                p2 = p2 + 1

    k = inicio
    for j in range(tamanho):
        V[k] = temp[j]
        k = k + 1
```

Combinar ordenando

Vetor acabou?

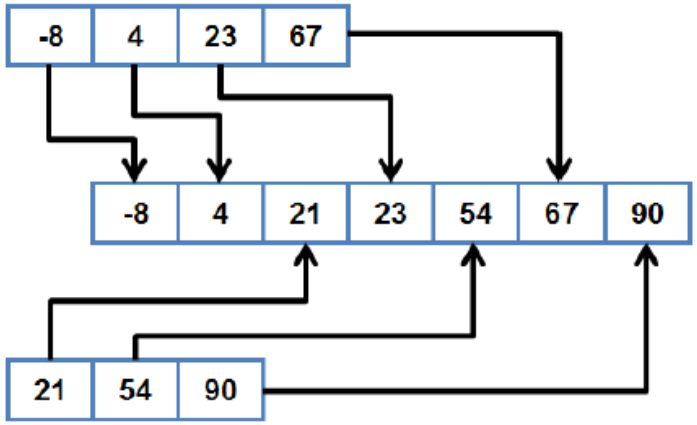
Copia o que sobrar

Copiar do auxiliar para o original

Algoritmo Merge Sort

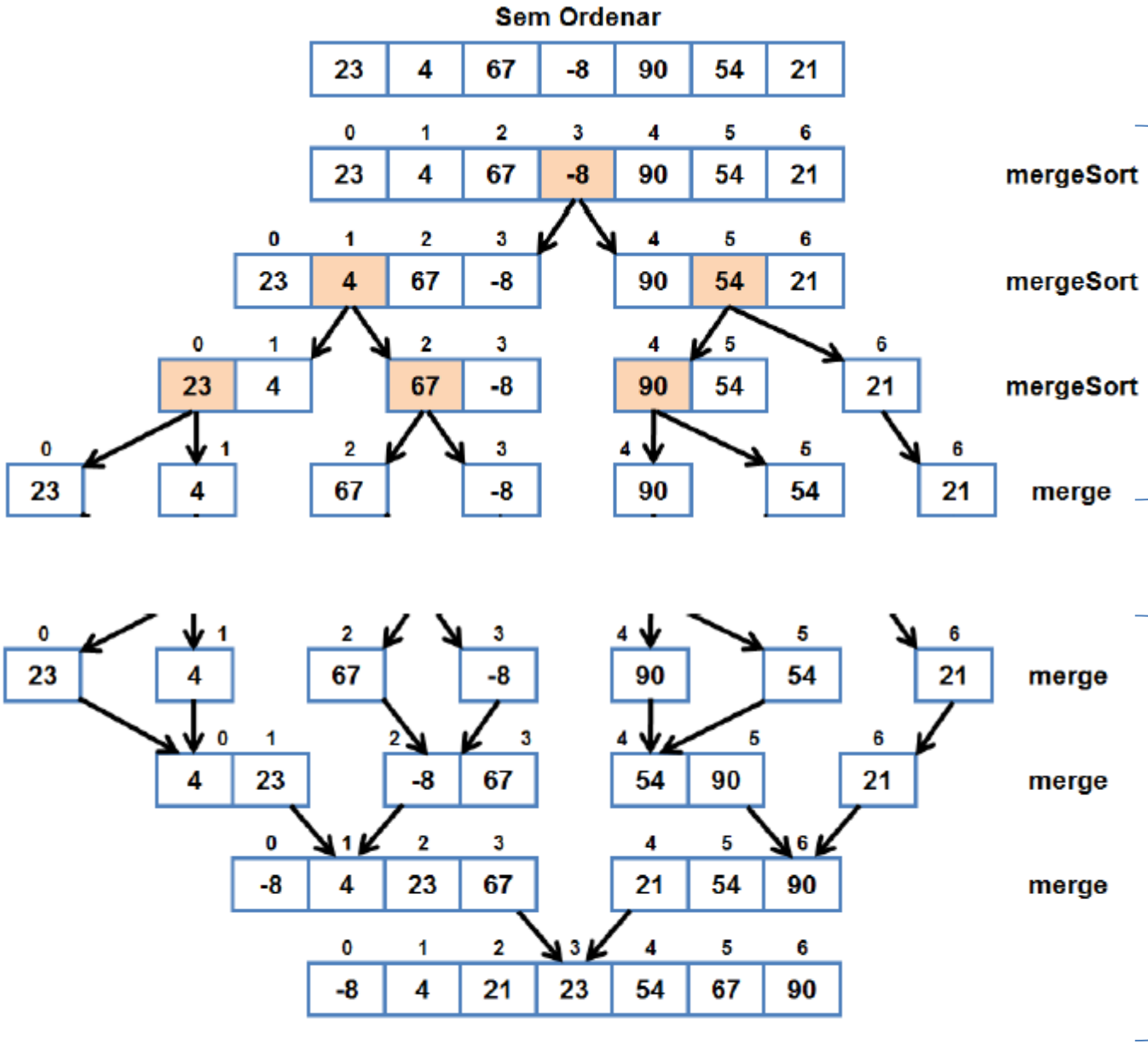
Passo a passo: função merge

- Intercala os dados de forma ordenada em um array maior
- Utiliza um array auxiliar



Simulando:
https://www.youtube.com/watch?v=XaqR3G_NVoo

Passo a passo



Divide o array até ter **N** arrays de 1 elemento cada.

Intercala os arrays até obter um único array de **N** elementos

Algoritmo Merge Sort

- **Complexidade:**
  - Considerando um array com N elementos, o tempo de execução é **O(n log n),** em todos os casos.
    - Sua eficiência não depende da ordem inicial dos elementos
  - **No pior caso**, realiza cerca de 39% menos comparações do que o quick sort no seu caso médio
  - Já **no seu melhor caso**, o merge sort realiza cerca de metade do número de iterações do seu pior caso

Algoritmo Merge Sort

- **Vantagem:**
  - Estável: não altera a ordem dos dados iguais

- **Desvantagens**
  - Possui um gasto extra de espaço de memória em relação aos demais métodos de ordenação
  - Ele cria uma cópia do array para cada chamada recursiva
  - Em outra abordagem, é possível utilizar um único array auxiliar ao longo de toda a sua execução

TAREFA: implementação do método merge sort

Algoritmo Shell Sort

Algoritmo Shell Sort

- Proposto por Shell em 1959.
- É uma extensão do algoritmo de ordenação por inserção.
- Problema com o algoritmo de ordenação por inserção:
  - Troca itens adjacentes para determinar o ponto de inserção.
  - São efetuadas $n - 1$ comparações e movimentações quando o menor item está na posição mais à direita no vetor.
- O método de Shell contorna este problema permitindo trocas de registros distantes um do outro.

Algoritmo Shell Sort

- **Funcionamento:**
  - Os itens separados de h posições são rearranjados.
  - Todo h-ésimo item leva a uma sequência ordenada.
  - Tal sequência é dita estar h-ordenada.
  - **Exemplo de utilização:**

	1	2	3	4	5	6
Chaves iniciais:	*O*	*R*	*D*	*E*	*N*	*A*
h = 4	*N*	*A*	*D*	*E*	*O*	*R*
h = 2	*D*	*A*	*N*	*E*	*O*	*R*
h = 1	*A*	*D*	*E*	*N*	*O*	*R*

  - Quando h = 1 shell sort corresponde ao algoritmo de inserção.

Algoritmo Shell Sort

- Funcionamento
  - Como escolher o valor de h:
    - Sequência para h:
      - $h(s) = 3h(s - 1) + 1$, para $s > 1$
      - $h(s) = 1$, para $s = 1$.
    - Knuth (1973, p. 95) mostrou experimentalmente que esta sequência é difícil de ser batida por mais de 20% em eficiência.
    - A sequência para h corresponde a 1, 4, 13, 40, 121, 364, 1.093, 3.280, . . .

Simulação:
https://www.youtube.com/watch?v=CmPA7zE8mx0

	1	2	3	4	5	6
Chaves iniciais:	*O*	*R*	*D*	*E*	*N*	*A*
h = 4	*N*	*A*	*D*	*E*	*O*	*R*
h = 2	*D*	*A*	*N*	*E*	*O*	*R*
h = 1	*A*	*D*	*E*	*N*	*O*	*R*

Algoritmo Shell Sort

```
def shellSort(nums):
    h = 1
    n = len(nums)
    while h > 0:
            for i in range(h, n):
                c = nums[i]
                j = i
                while j >= h and c < nums[j - h]:
                    nums[j] = nums[j - h]
                    j = j - h
                    nums[j] = c
            h = int(h/2.2)
    return nums
```

Algoritmo Shell Sort

- **Complexidade**
  - A razão da eficiência do algoritmo ainda não é conhecida.
  - Ninguém ainda foi capaz de analisar o algoritmo.
  - A sua análise contém alguns problemas matemáticos muito difíceis. A começar pela própria sequência de incrementos.
  - O que se sabe é que cada incremento não deve ser múltiplo do anterior.
  - Conjecturas referente ao número de comparações para a sequência de Knuth:
    - Conjectura 1 : C(n) = O($n^{1,25}$)
    - Conjectura 2 : C(n) = O(n $(\ln n)^2$)

Algoritmo Shell Sort

- **Vantagens:**
  - Shell sort é uma ótima opção para arquivos de tamanho moderado.
  - Sua implementação é simples e requer uma quantidade de código pequena.
  - Devido a sua complexidade possui excelentes desempenhos em N muito grandes, inclusive sendo melhor que o Merge Sort.
- **Desvantagens:**
  - O tempo de execução do algoritmo é sensível à ordem inicial do arquivo.
  - O método não é estável.

TAREFA: implementação do método shell sort

Referência

- Viana, Daniel. **Conheça os principais algoritmos de ordenação.** https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao
- Sorting. https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

PRÓXIMA AULA:

Algoritmo Heap Sort