**INSTITUTO FEDERAL DE ALAGOAS – IFAL / CAMPUS MACEIÓ**
**Curso Bacharelado em Sistemas de Informação**
**Disciplina de Estruturas de Dados**

*Prof. MSc. Ricardo Nunes*
*Ricardo (arroba) ifal.edu.br*

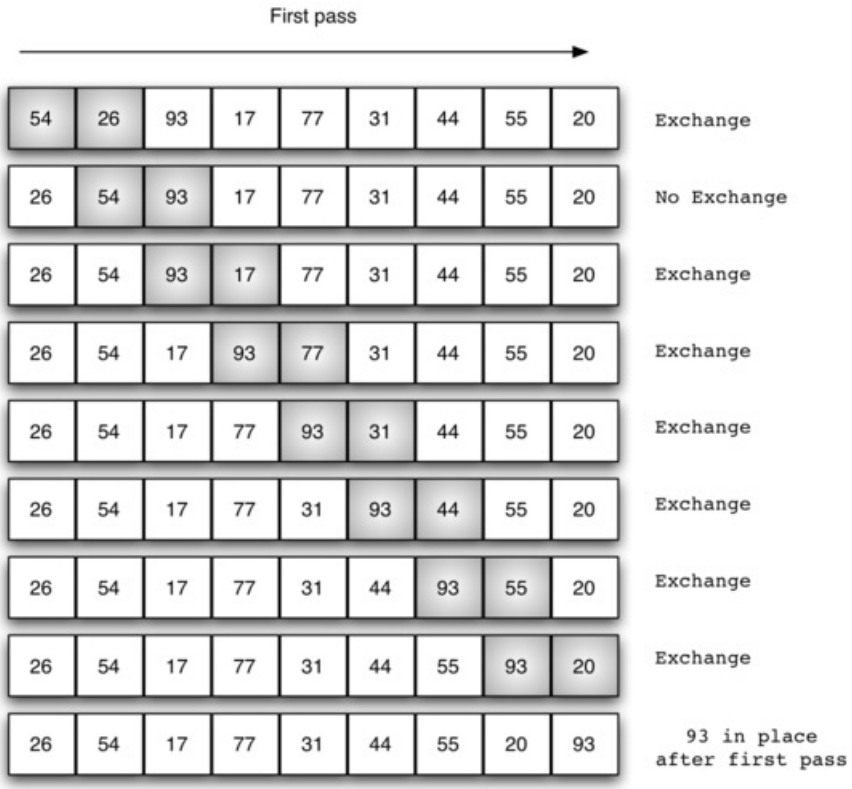
# Ordenação

## Objetivos e Conteúdo

1. Ser capaz de explicar e implementar os algoritmos de ordenação: - bubble sort - insertion sort - selection sort - shell sort - merge sort - quick sort

## BubbleSort

- Faz várias varreduras na lista, comparando os itens das posições adjacentes e trocando de lugar quando necessário




## BubbleSort (2)

- Após a primeira varredura, o 93 está ordenado
- Considere
- N elementos
- Então há N-1 pares
- Após a primeira varredura ainda será necessário mais N-1 varreduras

Visualize em https://visualgo.net/en/sorting

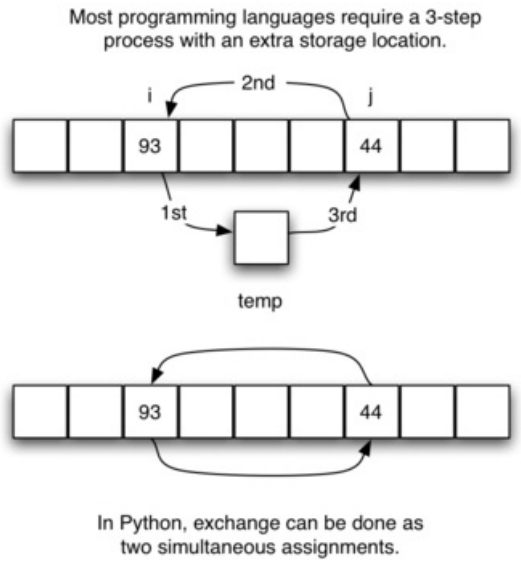
# BubbleSort (Troca de posições)

```
temp = uma_lista[i]
uma_lista[i] = uma_lista[j]
uma_lista[j] = temp
```

Em python também é possível trocar os elementos com o seguinte código

```
a,b=b,a
```

–




# BubbleSort (Algoritmo)

```
def bubbleSort(uma_lista):
    for varredura in range(len(uma_lista)-1,0,-1):
        for i in range(varredura):
            if uma_lista[i]>uma_lista[i+1]:
                temp = uma_lista[i]
                uma_lista[i] = uma_lista[i+1]
                uma_lista[i+1] = temp
```

Visualize em https://visualgo.net/en/sorting

# Análise do bubbleSort

| Varredura | Comparações |
|---|---|
| 1 | n□1 |
| 2 | n□2 |
| 3 | n□3 |
| … | … |
| n□1 | 1 |

Soma das comparações: (n^2 * 1/2)□(n * 1/2)

> O(N^2)

## bubbleSort otimizado

- Se não houver mais trocas, a lista está ordenada, então pode parar as varreduras

```
def shortBubbleSort(uma_lista):
    trocas = True
    varreduras = len(uma_lista)-1
    while varreduras > 0 and trocas:
      trocas = False
      for i in range(varreduras):
          if uma_lista[i]>uma_lista[i+1]:
              trocas = True
              temp = uma_lista[i]
              uma_lista[i] = uma_lista[i+1]
              uma_lista[i+1] = temp
      varreduras = varreduras-1
```

---

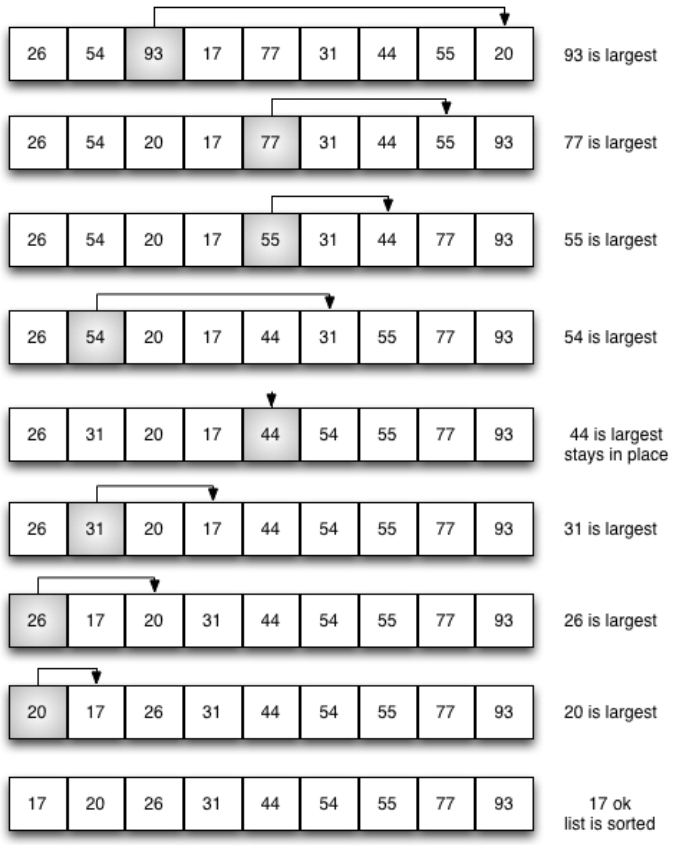
## Questão

Suponha que você tem a seguinte lista de números para ordenar:
[19, 1, 9, 7, 3, 10, 13, 15, 8, 12]
qual das resposta abaixo representa a lista parcialmente ordenada após completar três varreduras do bubble sort?

(A) [1, 9, 19, 7, 3, 10, 13, 15, 8, 12]
(B) [1, 3, 7, 9, 10, 8, 12, 13, 15, 19]
(C) [1, 7, 3, 9, 10, 13, 8, 12, 15, 19]
(D) [1, 9, 19, 7, 3, 10, 13, 15, 8, 12]

## Selection Sort

- Melhora o bubble sort fazendo uma troca para cada varredura da lista




## Selection Sort (algoritmo)

```
def selectionSort1(uma_lista):
  for posicao_verificada in range(len(uma_lista)-1,0,-1):
    posicao_maior = 0
    for posicao in range(1,posicao_verificada+1):
      if uma_lista[posicao]>uma_lista[posicao_maior]:
        posicao_maior = posicao

    temp = uma_lista[posicao_verificada]
    uma_lista[posicao_verificada] = uma_lista[posicao_maior]
    uma_lista[posicao_maior] = temp
```

## Análise do Selection Sort

- Faz o mesmo número de comparações do Bubble Sort, logo O(N^2)

visualize em: https://visualgo.net/en/sorting

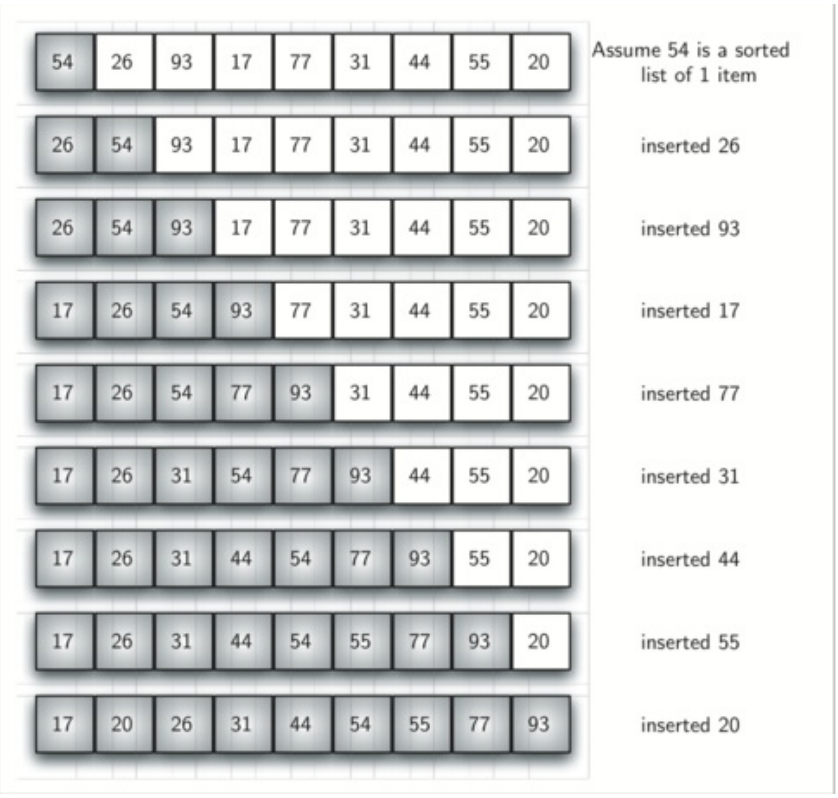
## Questão - Selection Sort

Suponha que você tem a seguinte lista de números para ordenar:
[11, 7, 12, 14, 19, 1, 6, 18, 8, 20]
qual das resposta abaixo representa a lista parcialmente ordenada após completar três varreduras completas do selection sort?

(A) [7, 11, 12, 1, 6, 14, 8, 18, 19, 20]
(B) [7, 11, 12, 14, 19, 1, 6, 18, 8, 20]
(C) [11, 7, 12, 14, 1, 6, 8, 18, 19, 20]
(D) [11, 7, 12, 14, 8, 1, 6, 18, 19, 20]

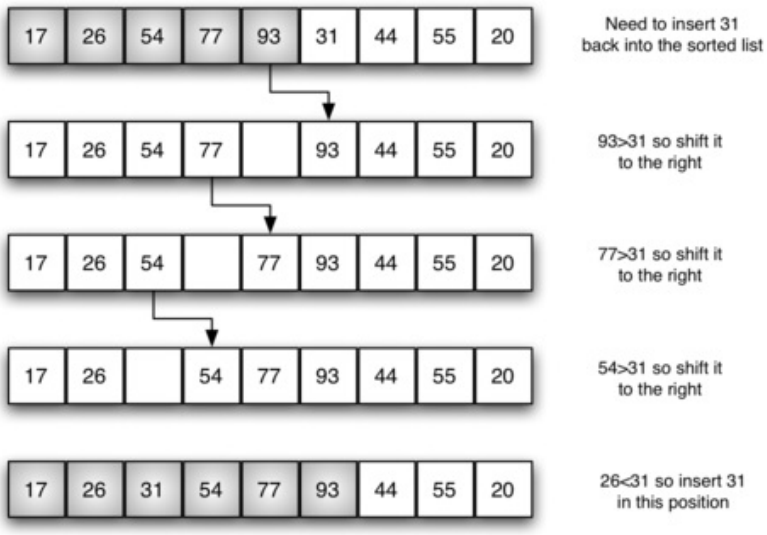
## Insertion Sort

- Outro algoritmo O(N^2)




visualize em: https://visualgo.net/en/sorting

## Insertion Sort (Varredura)




- 5ª varredura em detalhes

visualize em: https://visualgo.net/en/sorting

## Insertion Sort (Algoritmo)

```
def insertionSort(uma_lista):
  for index in range(1,len(uma_lista)):

    valor_atual = uma_lista[index]
    posicao = index

    while posicao>0 and uma_lista[posicao-1]>valor_atual:
      uma_lista[posicao]=uma_lista[posicao-1]
      posicao = posicao-1

    uma_lista[posicao]=valor_atual
```

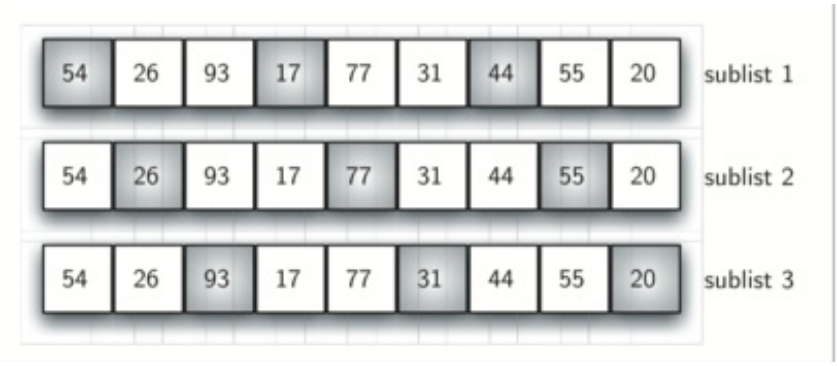
# Comparando os algoritmos estudados
# https://www.toptal.com/developers/sorting-algorithms

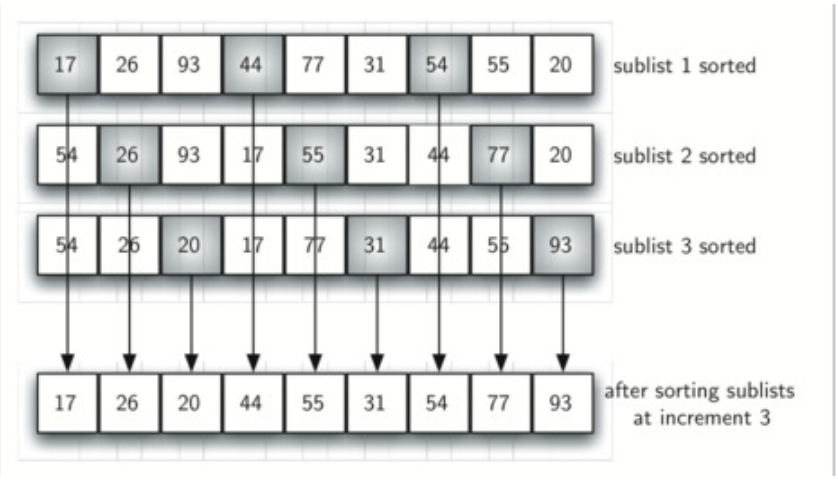
## Shell Sort (1)

- Melhora o Insertion Sort quebrando a lista original de números ordenados em várias sublistas
- As sublistas não são de elementos contíguos, são intercalados em i posições (Incremento I)

## Shell Sort (2)

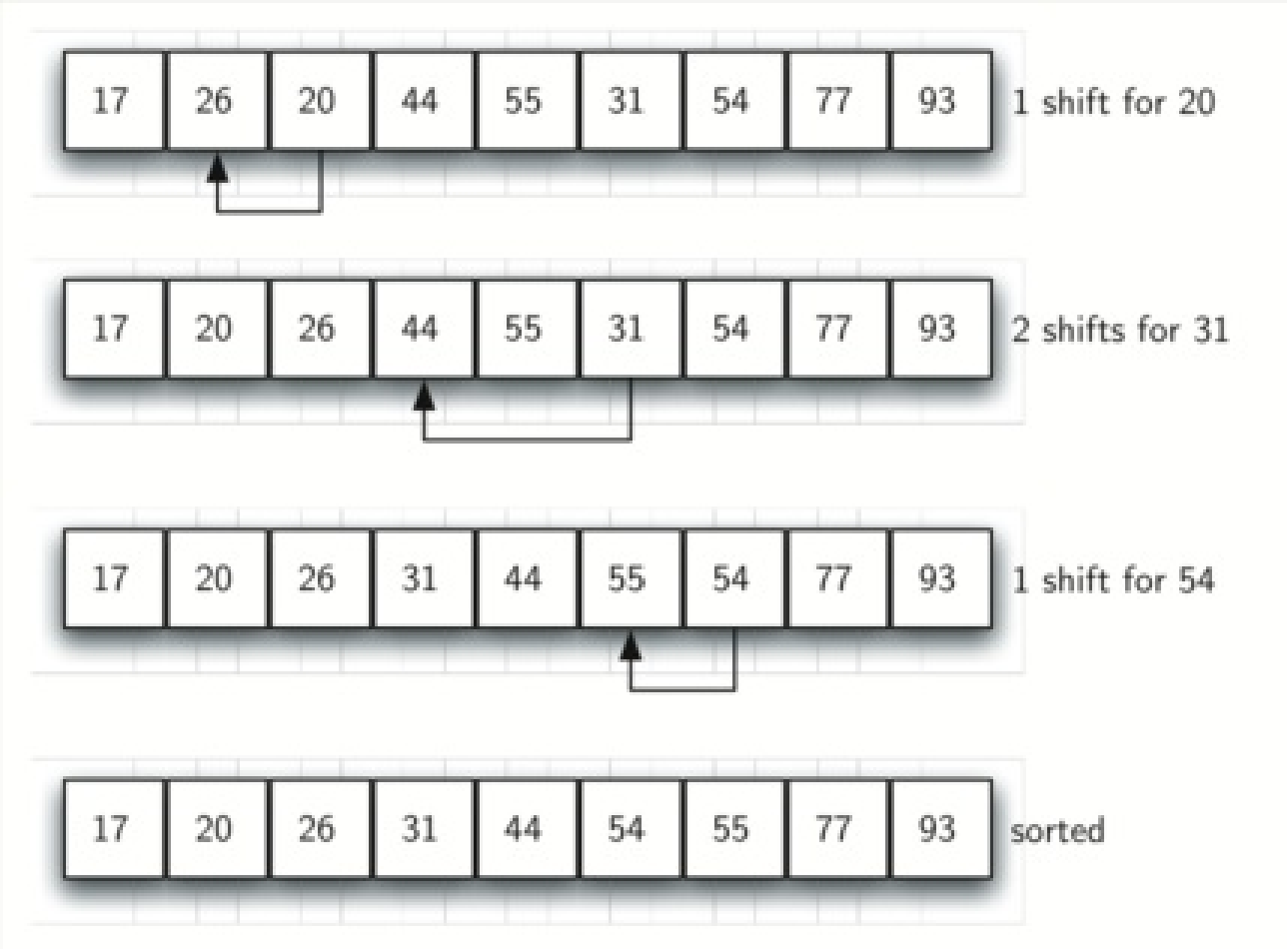
Incremento de 3



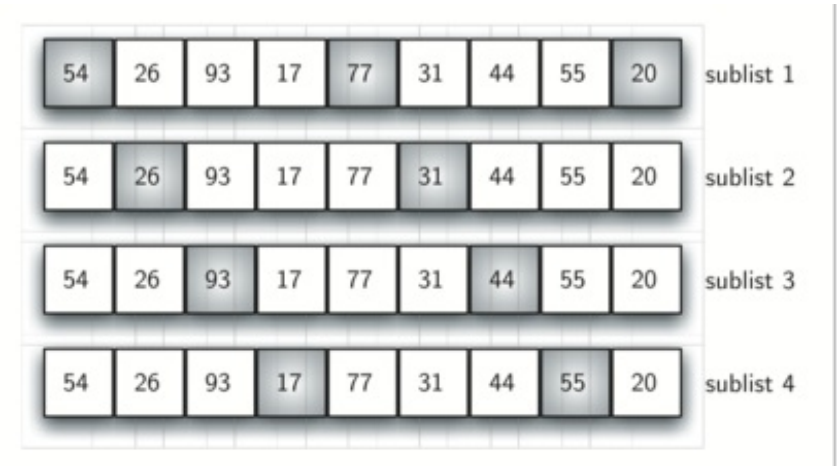

## Shell Sort (3)

Incremento de 1 (insertion sort tradicional)




## Shell Sort (4)

Incremento de 4




## Algoritmo Shell Sort

```
def shellSort(uma_lista):
    contador_sublista = len(uma_lista)//2
    while contador_sublista > 0:

      for posicao_inicial in range(contador_sublista):
        InsertionSortComGap(uma_lista,posicao_inicial,contador_sublis

      print("Após incrementos de tamanho",contador_sublista,
                                "a lista é",uma_lista)

      contador_sublista = contador_sublista // 2

def InsertionSortComGap(uma_lista,inicio,gap):
    for i in range(inicio+gap,len(uma_lista),gap):

        currentvalue = uma_lista[i]
        position = i

        while position>=gap and uma_lista[position-gap]>currentvalue:
            uma_lista[position]=uma_lista[position-gap]
            position = position-gap

        uma_lista[position]=currentvalue
```

## Análise de desempenho Shell Sort

> Apesar de ser baseado no insertion sort, e executar o insertion sort tradicional (incremento 1) no final, ele não faz muitas comparações ou trocas, pois a lista está pré-ordenada.

- Entre O(n) and O(n^2)
- Para o incremento [4, 2, 1] => O(n^2)
- Para o incremento [(2^k)□1 (1, 3, 7, 15, 31, …)] => O(n^(3/2))

## Questão Shell Sort

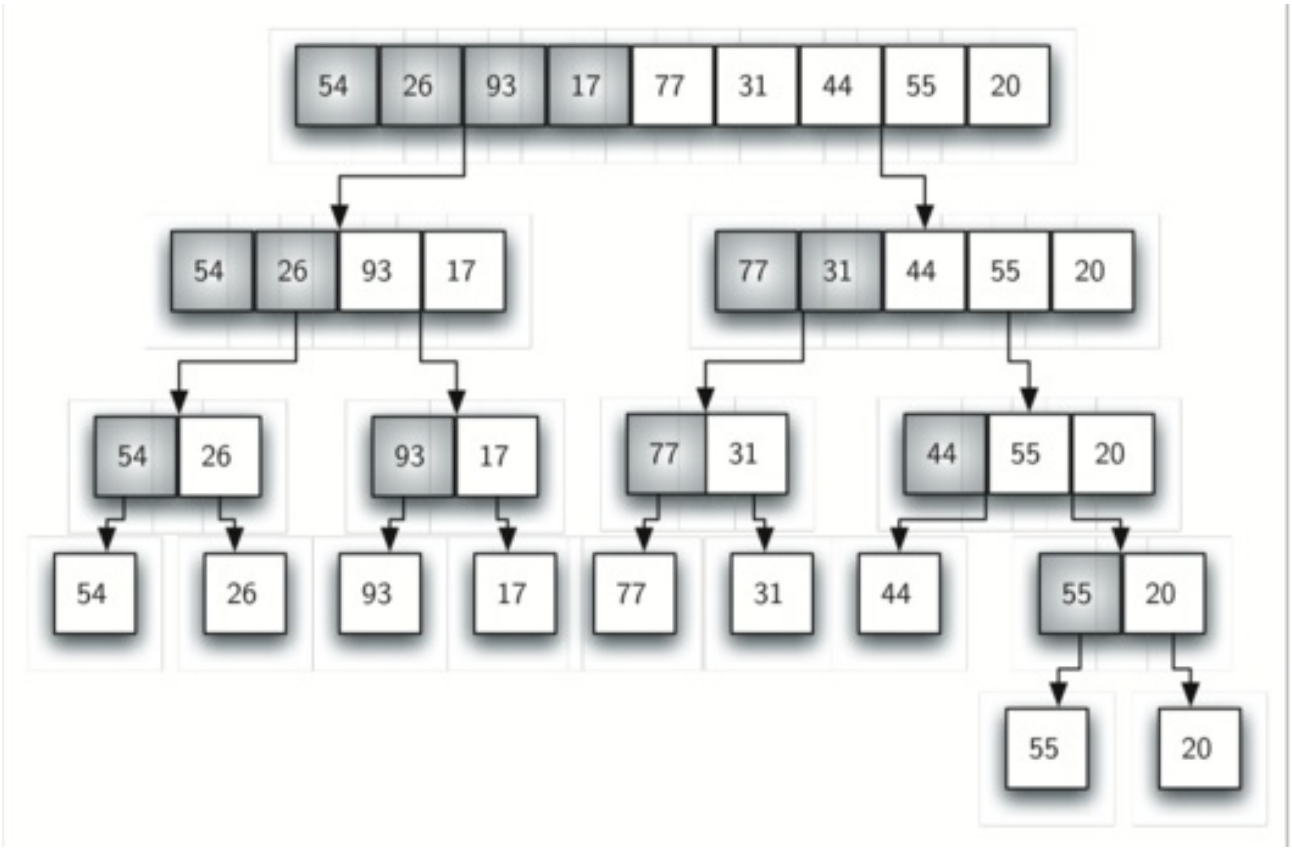
Dada a seguinte lista de números: [5, 16, 20, 12, 3, 8, 9, 17, 19, 7] qual das respostas abaixo ilustra o conteúdo da lista após todas as trocas par um incremente (gap) de tamanho 3?

(A) [5, 3, 8, 7, 16, 19, 9, 17, 20, 12]
(B) [3, 7, 5, 8, 9, 12, 19, 16, 20, 17]
(C) [3, 5, 7, 8, 9, 12, 16, 17, 19, 20]
(D) [5, 16, 20, 3, 8, 12, 9, 17, 20, 7]

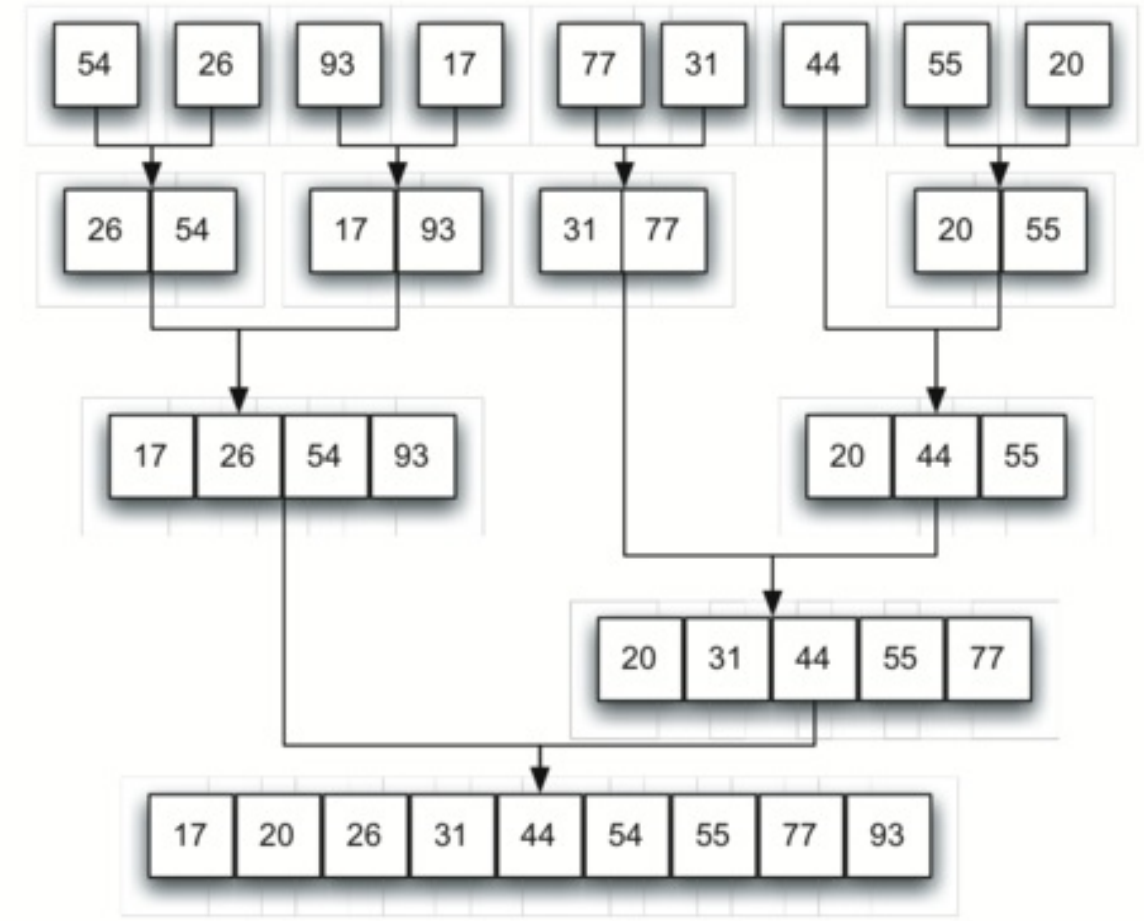
## Merge Sort (1)

- Usa a estratégia dividar para conquistar como forma de melhorar o desempenho
- Algoritmo recursivo
- continuamente divide a lista na metade
- se a lista está vazia ou com um item, está ordenada
- uma vez que as listas estão ordenadas, ele as une mesclando (merge) de forma ordenada

## Merge Sort (2)




## Merge Sort (3)




## Algoritmo Merge Sort - Parte I

```
def mergeSort(uma_lista):
    print("Splitting ",uma_lista)
    if len(uma_lista)>1:
        meio = len(uma_lista)//2
        esquerda = uma_lista[:meio]
        direita = uma_lista[meio:]

        mergeSort(esquerda)
        mergeSort(direita)

        ## aqui vai o merge. ver próximo slide.

    print("Merging ",uma_lista)

uma_lista = [54,26,93,17,77,31,44,55,20]
mergeSort(uma_lista)
print(uma_lista)
```

## Algoritmo Merge Sort - Parte II

```
def mergeSort(uma_lista):
    #...
    i=0
    j=0
    k=0

    while i < len(esquerda) and j < len(direita):
        if esquerda[i] < direita[j]:
            uma_lista[k]=esquerda[i]
            i=i+1
        else:
            uma_lista[k]=direita[j]
            j=j+1
        k=k+1

    while i < len(esquerda):
        uma_lista[k]=esquerda[i]
        i=i+1
        k=k+1

    while j < len(direita):
        uma_lista[k]=direita[j]
        j=j+1
        k=k+1
  print("Merging ",uma_lista)
```

## Análise Merge Sort

- merge sort é um algoritmo O(n log(n))

## Questões Merge Sort

Data a seguinte lista de números: [21, 1, 26, 45, 29, 28, 2, 9, 16, 49, 39, 27, 43, 34, 46, 40] qual das resposta abaixo ilustra a lista a ser ordenada após 3 chamadas recursivas do mergesort? (A) [16, 49, 39, 27, 43, 34, 46, 40]
(B) [21,1]
(C) [21, 1, 26, 45]
(D) [21]

Data a seguinte lista de números: [21, 1, 26, 45, 29, 28, 2, 9, 16, 49, 39, 27, 43, 34, 46, 40] qual das resposta abaixo ilustra as duas primeiras lista a serem unidas/mescladas (merged)?

(A) [21, 1] e [26, 45]
(B) [1, 2, 9, 21, 26, 28, 29, 45] e [16, 27, 34, 39, 40, 43, 46, 49]
(C) [21] e [1]
(D) [9] e [16]

## Comparação dos algorítmos

- Comparação dos Algoritmos - Wikipedia

## Para estudar

- Ordenação
- Seções 5.6 a 5.12 do livro [5] <https://panda.ime.usp.br/pythonds/static/pythonds_pt/05-OrdenacaoBusca/toctree.html
- https://en.wikipedia.org/wiki/Sorting_algorithm#Comparison_of_algorithms]

# Referências

1. Tradução do livro *How to Think Like a Computer Scientist: Interactive Version*, de Brad Miller e David Ranum. link: https://panda.ime.usp.br/pensepy/static/pensepy/index.html
2. Allen Downey, Jeff Elkner and Chris Meyers. *Aprenda Computação com Python 3.0*. link: https://chevitarese.files.wordpress.com/2009/09/aprendacomputaocompython3k.pdf
3. SANTOS, A. C. *Algoritmo e Estrutura de Dados I*. 2014. Disponível em http://educapes.capes.gov.br/handle/capes/176522
4. SANTOS, A. C. *Algoritmo e Estrutura de Dados II*. 2014. Disponível em 2014. Disponível em https://educapes.capes.gov.br/handle/capes/176557
5. Tradução do livro [6] *Problem Solving with Algorithms and Data Structures using Python* de Brad Miller and David Ranum. link: https://panda.ime.usp.br/pythonds/static/pythonds_pt/index.html
6. Brad Miller and David Ranum. *Problem Solving with Algorithms and Data Structures using Python* link: https://runestone.academy/ns/books/published//pythonds/index.html
7. Caelum. Algoritmos e Estruturas Dados em Java. Disponível em https://www.caelum.com.br/download/caelum-algoritmos-estruturas-dados-java-cs14.pdf

# That’s all Folks