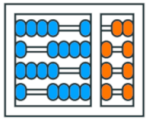


Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 - Aula 23

Algoritmos de Ordenação Recursivos - Merge Sort

Algoritmos e Programação de Computadores

Turmas

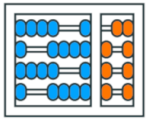
OVXZ

Prof. Lise R. R. Navarrete

lrommel@ic.unicamp.br

Terça-feira, 21 de junho de 2022

21:00h - 23:00h (CB06)



Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 – Algoritmos e Programação de Computadores

Turmas

OVXZ

https://ic.unicamp.br/~mc102/

Site da Coordenação de MC102

Aulas teóricas:

Terça-feira, 21:00h - 23:00h (CB06)

Quinta-feira, 19:00h - 21:00h (CB06)

Conteúdo

- Divisão e Conquista
- O Problema da Ordenação
- Merge Sort
  - Combinando as Listas ordenadas
  - Análise de complexidade do Merge Sort

Divisão e Conquista

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- Esta técnica consiste em dividir um problema maior recursivamente em problemas menores até que ele possa ser resolvido diretamente.
- A solução do problema inicial é dada através da combinação dos resultados de todos os problemas menores computados.
- A técnica soluciona o problema através de três fases:
  - Divisão: o problema maior é dividido em problemas menores.
  - Conquista: cada problema menor é resolvido recursivamente.
  - Combinação: os resultados dos problemas menores são combinados para se obter a solução do problema maior.

https://colab.research.google.com/

```
[1] def problema1(n):
        # caso base
        if n < 2:
            return 1

        # divisão
        subproblema1 = n-1
        subproblema2 = n-2

        # conquista
        solução1 = problema1(subproblema1)
        solução2 = problema1(subproblema2)

        # combinação
        solução = solução1 + solução2

        return solução
```

```
[2] problema1(10)
```

```
89
```

```
for i in range(10):
  print(problema1(i),end=" ")
```

```
1 1 2 3 5 8 13 21 34 55
```

https://colab.research.google.com/

```
[4] def problema2(n):
        # caso base
        if n < 2:
            return 1

        # divisão
        subproblema1 = n-1


        # conquista
        solução1 = problema2(subproblema1)


        # combinação
        solução = n * solução1

        return solução
```

```
[5] problema2(10)
```

```
3628800
```

```
for i in range(10):
  print(problema2(i),end=" ")
```

```
1 1 2 6 24 120 720 5040 40320 362880
```

O Problema da Ordenação

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- Iremos continuar o estudo de algoritmos para o problema de ordenação visto anteriormente:

Definição do Problema

Dada uma coleção de elementos, com uma relação de ordem entre eles, ordenar os elementos da coleção de forma crescente.

- Nos nossos exemplos, a coleção de elementos será representada por uma lista de inteiros.
  - Números inteiros possuem uma relação de ordem entre eles.
- Apesar de usarmos números inteiros, os algoritmos que estudaremos servem para ordenar qualquer coleção de elementos que possam ser comparados entre si.
- Ambos os algoritmos recursivos de ordenação que veremos usam o paradigma de Divisão e Conquista.

Merge Sort

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- O Merge Sort foi proposto por John von Neumann em 1945.
- O algoritmo Merge Sort é baseado em uma operação de intercalação (`merge`) que une duas listas ordenadas para gerar uma terceira lista também ordenada.
- O algoritmo pode ser construído a partir dos seguintes passos:
  - Divisão: a lista é dividida em duas sublistas de tamanhos quase iguais (diferindo em no máximo um elemento).
  - Conquista: cada sublista é ordenada recursivamente.
  - Combinação: as duas sublistas ordenadas são intercaladas para se obter a lista final ordenada.

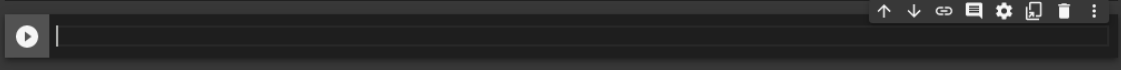
```
[ ]  def problema3(lista, inicio, fim):
        # caso base
        if fim - inicio < 2:
           return

        # divisão:
        meio = ( inicio + fim ) // 2
        inicio1 , fim1 = inicio , meio  # subproblema1
        inicio2 , fim2 = meio   , fim   # subproblema2

        # conquista
        problema3(lista, inicio1, fim1)
        problema3(lista, inicio2, fim2)

        # combinação
        ...
```

https://colab.research.google.com/



```
[ ]
```

https://colab.research.google.com/

```
[9]  def problema3(lista, inicio, fim):
        # caso base
        if fim - inicio < 2:
           return

        # divisão:
        meio = ( inicio + fim ) // 2
        inicio1 , fim1 = inicio , meio  # subproblema1
        inicio2 , fim2 = meio   , fim   # subproblema2

        # conquista
        problema3(lista, inicio1, fim1)
        problema3(lista, inicio2, fim2)

        # combinação
        lista = combinar( lista[inicio:meio] , lista[meio:fim] )
```

```
[10] lista = [4, 9, 2, 11, 27, 13, 17, 8]
     problema3(lista, 0, len(lista))
```

```
     print(lista)
```

```
[4, 9, 2, 11, 27, 13, 17, 8]
```

https://colab.research.google.com/

```
[12] def problema3(lista, inicio, fim):
        # caso base
        if fim - inicio < 2:
           return

        # divisão:
        meio = ( inicio + fim ) // 2
        inicio1 , fim1 = inicio , meio  # subproblema1
        inicio2 , fim2 = meio   , fim   # subproblema2

        # conquista
        problema3(lista, inicio1, fim1)
        problema3(lista, inicio2, fim2)

        # combinação
        lista[inicio:fim] = combinar( lista[inicio:meio] , lista[meio:fim] )
```

```
[13] lista = [4, 9, 2, 11, 27, 13, 17, 8]
     problema3(lista, 0, len(lista))
```

```
print(lista)
```

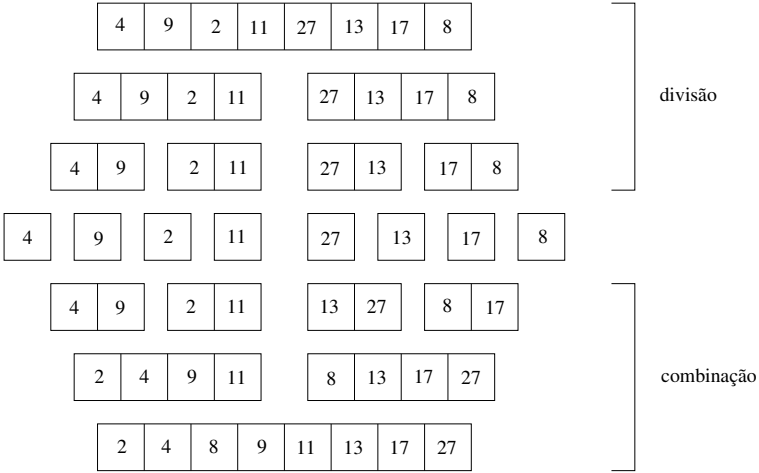
```
[2, 4, 8, 9, 11, 13, 17, 27]
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- Simulação das chamadas recursivas:

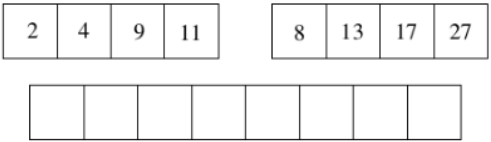
```
[4  9  2  11  27  13  17  8]
[4  9  2  11][27  13  17  8]
[4  9][2  11][27  13][17  8]
[4][9][2][11][27][13][17][8]
[4  9][2  11][13  27][8  17]
[2  4  9  11][8  13  17  27]
[2  4  8  9  11  13  17  27]
```

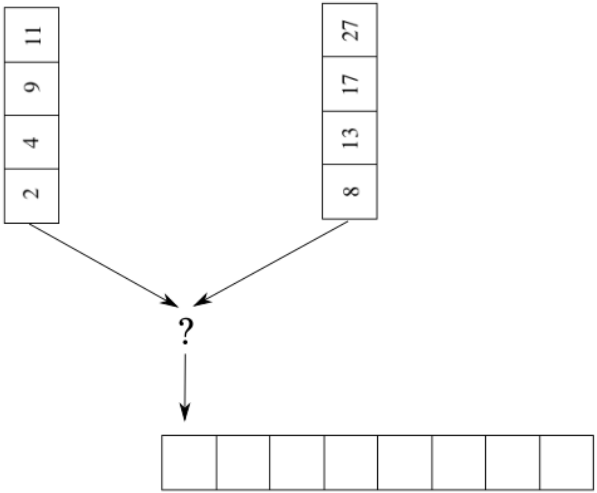
https://ic.unicamp.br/~mc102/aulas/aula13.pdf

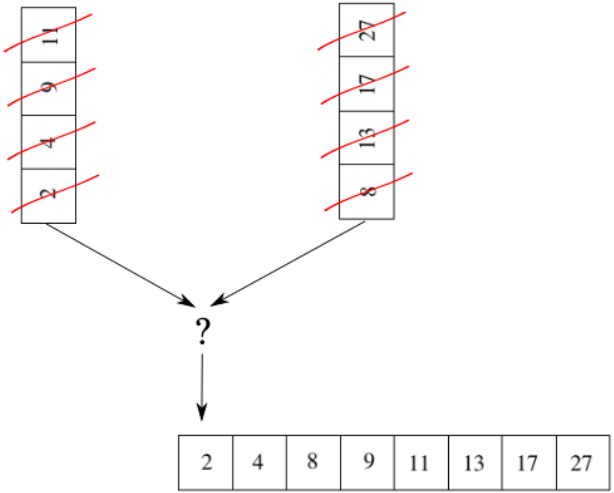


Merge Sort

Combinando as Listas ordenadas


2
4
9
11
8
13
17
27


11
9
4
2
27
17
13
8
?


11
9
4
2
27
17
13
8
?
2
4
8
9
11
13
17
27

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
1  def merge(lista1, lista2):
2    i = j = 0
3    aux = []
4
5    while (i < len(lista1)) and (j < len(lista2)):
6      if lista1[i] < lista2[j]:
7        aux.append(lista1[i])
8        i = i + 1
9      else:
10       aux.append(lista2[j])
11       j = j + 1
12
13   while i < len(lista1):
14     aux.append(lista1[i])
15     i = i + 1
16
17   while j < len(lista2):
18     aux.append(lista2[j])
19     j = j + 1
20
21   return aux
```

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

```
1  def merge(lista1, lista2):
2    i = j = 0
3    aux = []
4
5    while (i < len(lista1)) and (j < len(lista2)):
6      if lista1[i] < lista2[j]:
7        aux.append(lista1[i])
8        i = i + 1
9      else:
10       aux.append(lista2[j])
11       j = j + 1
12
13   aux = aux + lista1[i:]
14
15
16
17   aux = aux + lista2[j:]
18
19
20
21   return aux
```

https://colab.research.google.com/

```
[7]  def merge(lista1, lista2):
       i = j = 0
       aux = []
       while (i < len(lista1)) or (j < len(lista2)):
         if (    (i<len(lista1) and j<len(lista2)) and lista1[i]<lista2[j] ) or \
            (not (i<len(lista1) and j<len(lista2)) and i<len(lista1) ):
           aux.append(lista1[i])
           i = i + 1
         else :
           aux.append(lista2[j])
           j = j + 1
       return aux
```

```
[8]  def merge_sort(lista, inicio, fim):
       if fim - inicio > 1:
         # divisão:
         meio = ( inicio + fim ) // 2
         # conquista
         merge_sort(lista, inicio, meio)
         merge_sort(lista, meio, fim)
         # combinação
         lista[inicio:fim] = merge( lista[inicio:meio] , lista[meio:fim] )
```

```
[9]  lista = [4, 9, 2, 11, 27, 13, 17, 8]
     merge_sort(lista, 0, len(lista))
```

```
     print(lista)
```

```
[2, 4, 8, 9, 11, 13, 17, 27]
```

Merge Sort

Análise de complexidade do Merge Sort

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- Seja $T(n)$ o custo de ordenar uma lista de n elementos usando o Merge Sort.
- Para $n > 1$, temos que o algoritmo executa:
  - A ordenação recursiva dos $\lfloor n/2 \rfloor$ primeiros elementos da lista.
  - A ordenação recursiva dos $\lceil n/2 \rceil$ últimos elementos da lista.
  - Intercala as duas sublistas previamente ordenadas.
- A seguinte recorrência define o tempo de execução do Merge Sort:

$$T(1) = c_1$$
$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + M(n) + c_2$$

- É fácil verificar que $M(n)$, o tempo de execução da função `merge`, é proporcional à função $f(n) = n$.

https://ic.unicamp.br/~mc102/aulas/aula13.pdf

- É possível mostrar que $T(n)$, o tempo de execução do Merge Sort, é proporcional à função $f(n) = n \log n$, tanto no melhor quanto no pior caso.
- Da forma como a função `merge` foi implementada utilizando uma lista auxiliar, o Merge Sort necessita de espaço linear de memória adicional.

Perguntas ....

Referências

- Zanoni Dias, MC102, Algoritmos e Programação de Computadores, IC/UNICAMP, 2021. https://ic.unicamp.br/~mc102/

  Aula Introdutória [slides] [vídeo]
  Primeira Aula de Laboratório [slides] [vídeo]
  Python Básico: Tipos, Variáveis, Operadores, Entrada e Saída [slides] [vídeo]
  Comandos Condicionais [slides] [vídeo]
  Comandos de Repetição [slides] [vídeo]
  Listas e Tuplas [slides] [vídeo]
  Strings [slides] [vídeo]
  Dicionários [slides] [vídeo]
  Funções [slides] [vídeo]
  Objetos Multidimensionais [slides] [vídeo]
  Algoritmos de Ordenação [slides] [vídeo]
  Algoritmos de Busca [slides] [vídeo]
  Recursão [slides] [vídeo]
  Algoritmos de Ordenação Recursivos [slides] [vídeo]
  Arquivos [slides] [vídeo]
  Expressões Regulares [slides] [vídeo]
  Execução de Testes no Google Cloud Shell [slides] [vídeo]
  Numpy [slides] [vídeo]
  Pandas [slides] [vídeo]

- Panda - Cursos de Computação em Python (IME -USP) https://panda.ime.usp.br/

  Como Pensar Como um Cientista da Computação https://panda.ime.usp.br/pensepy/static/pensepy/
  Aulas de Introdução à Computação em Python https://panda.ime.usp.br/aulasPython/static/aulasPython/

- Fabio Kon, Introdução à Ciência da Computação com Python http://bit.ly/FabioKon/
- Socratica, Python Programming Tutorials http://bit.ly/SocraticaPython/
- Google - online editor for cloud-native applications (Python programming) https://shell.cloud.google.com/
- w3schools - Python Tutorial https://www.w3schools.com/python/
- Outros, citados nos Slides.