	**Aluna:** Daniela América da Silva **Disciplina:** CT208 - Matemática Computacional **Prof. Nei Soma** **Orientação:** Prof. Tasinaffo **Co-orientação:** Prof. Johnny **Data:** 23/Junho/2020

Exercício: Maior Elemento (Resultado Final - atualizado 23-Jun)

1- Introdução

Este relatório apresenta os quatro versionamentos do programa para o cálculo de trocas do maior, menor, segundo maior e segundo menor elemento em uma lista criada a partir da permutação dos números de 1 a 13 [3].
A primeira versão do programa apresenta o processamento de trocas para a lista de permutação do número 1 ao 11, e houve falta de memória para o cálculo do número 11 ao 13.
A segunda versão possibilitou o processamento até o número 13 utilizando o cálculo por recorrências e houve pouca utilização de memória. O programa possui dois passos: (i) armazenamento das trocas para a lista de permutação de 1 até 10; (ii) cálculo das trocas de 11 a 13 a partir das recorrências.
A terceira versão do programa calcula adicionalmente o número de trocas para o menor número e também para o segundo maior e o segundo menor para a lista de permutações do número 1 ao 13, utilizando a recorrência e portanto sem melhoria no tempo do algoritmo.
A quarta versão do programa, utiliza *divide and conquer* [4][7][9] e portanto reduzindo o tempo do algoritmo para 3/2n + log(n).
O relatório contém também a demonstração matemática das funções geratrizes conforme demonstrado nas aulas [1][2][3][10].
Para as simulações o código foi desenvolvido utilizando a linguagem Python e o Google Colab para execução do programa. As bibliotecas Python utilizadas são *itertools* para cálculo das permutações e *numpy* para utilização de matriz para o armazenamento dos dados. O cálculo das trocas, pelo método da recorrência, e pela estratégia de divisão e conquista foram realizados sem uso de bibliotecas.

O código desenvolvido em Python está disponível no Github
https://github.com/dasamerica/ct208/tree/master/aula_2805_maior_elemento

Nota: em algumas demonstrações foi utilizado o n de 1 a 15 apenas para testes durante a simulação.

2. Montar o experimento

> Monte um experimento que faça o mesmo tipo de análise realizada até agora.
> Explique o que vem a ser: média (aritmética) e variância. ◁

2.1 - Simulador Python

Execução realizada no Google Colab pois propicia até 12Gb de memória.
A permutação para a geração da lista de elementos é realizada via biblioteca itertools.
A função que conta as trocas até determinar o maior elemento não utiliza biblioteca phyton.
É possível executar o programa de 1 até 10 elementos em uma única execução que consome até 1Gb de memória RAM.
Para 11 elementos é necessário a execução individual consumindo até 7 Gb memória RAM.
Para 12 e 13 elementos não há memória suficiente.

Código Fonte:
https://github.com/dasamerica/ct208/blob/master/aula_2805_maior_elemento/ct208_maior_elemento.py

Programa Principal	Funções

`# Programa phyton para dado um numero gera uma lista de suas permutações`
`# Verifica quantas iterações necessárias para achar o maior elemento da lista`
`# Daniela América da Silva`
`# Disciplina: ct208`
 	
`#processa lista de permutações do número 1 ao 10`
`x = 1`
`while x < 11:`
`  # inicializa lista de trocas`
`  trocas = []`
 	
`  numeros = cria_lista (x)`
 	
`  #inicializa contador trocas`
`  conta_trocas = set_contador(len(numeros))`
 	
`  # realiza as permutações`
`  perm = permutations(numeros)`
 	
`  # Conta as trocas`
`  for i in list(perm):`
	`# Programa phyton para dado um numero gera uma lista de suas permutações`
	`# Verifica quantas iterações necessárias para achar o maior elemento da lista`
	`# Daniela América da Silva`
	`# Disciplina: ct208`
	
	`# Importa library de permutações`
	`from itertools import permutations`
	
	`#seta vetor contador de trocas`
	`def set_contador (col):`
	`  n = 0`
	`  conta = []`
	`  while n < col:`
	`  conta.append(0)`
	`  n+=1`
	`  return (conta)`
	
	`#cria a lista de números`
	`def cria_lista (num):`
	`  n = 0`
	`  lista = []`
	`  while n < num:`

```
    #print (i)
    number = maior_trocas (i)
    #imprime
    #print (i, '(', number ,')')
    #trocas.append(number)
    conta_trocas[number]+=1


  #print (trocas)
  print ('numero:', x, conta_trocas)
  x+=1
```

```
      lista.append(n+1)
      n+=1
    return (lista)

#conta numero de trocas
def maior_trocas (elements):
  m = elements[0]
  n = 1
  t = 0
  while n < (len(elements)):
    if m < elements[n]:
       m = elements[n]
       t+=1
    n+=1
  return (t)
```

Resultados para lista de permutações até 10 elementos:

```
numero: 1 [1]
numero: 2 [1, 1]
numero: 3 [2, 3, 1]
numero: 4 [6, 11, 6, 1]
numero: 5 [24, 50, 35, 10, 1]
numero: 6 [120, 274, 225, 85, 15, 1]
numero: 7 [720, 1764, 1624, 735, 175, 21, 1]
numero: 8 [5040, 13068, 13132, 6769, 1960, 322, 28, 1]
numero: 9 [40320, 109584, 118124, 67284, 22449, 4536, 546, 36, 1]
numero: 10 [362880, 1026576, 1172700, 723680, 269325, 63273, 9450, 870, 45, 1]
```

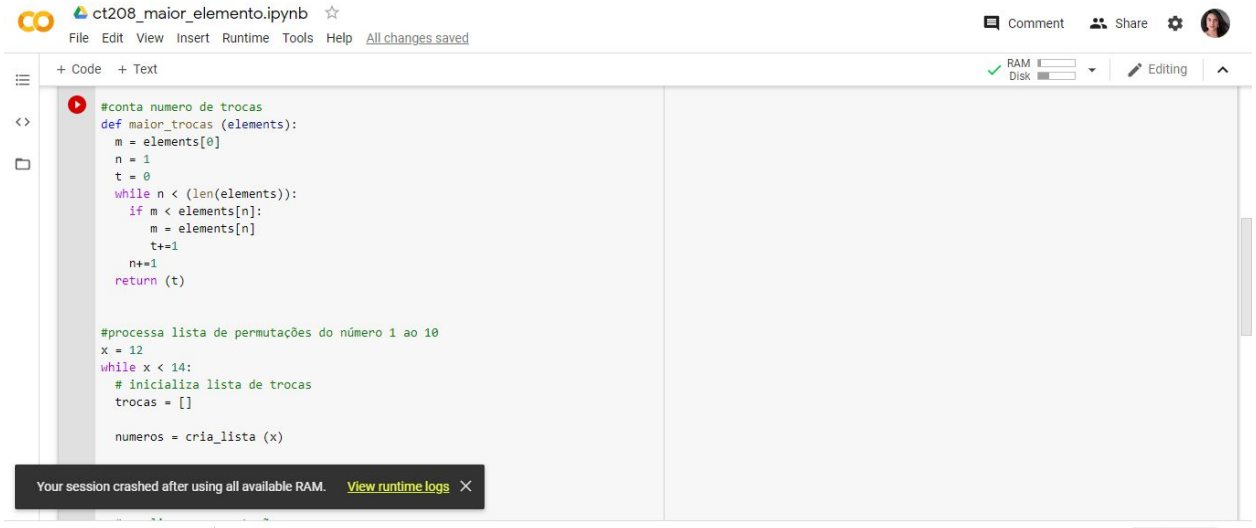
Resultados para lista de permutações com 11 elementos, aproximadamente 7Gb memória
** reduziu o loop para apenas 11 elementos

```
#processa lista de permutações do número 1 ao 10
x = 11
```

```
while x < 12:
  # inicializa lista de trocas
  trocas = []
```

```
numero: 11 [3628800, 10628640, 12753576, 8409500, 3416930, 902055, 157773, 18150, 1320, 55, 1]
```

Resultados para lista de permutações com 12 ou 13 elementos, não há memória suficiente, acima de 12 Gb necessários.



2.2 - Definição de Média Aritmética e Variância

Média aritmética é definida como a “somatório de todos os elementos da série divididos pelo número de elementos”.

Média aritmética = $\frac{\sum_{i=1}^{n} X_i}{n}$

Variância é definida como a “média dos quadrados das diferenças, entre os valores em relação à sua própria média”.

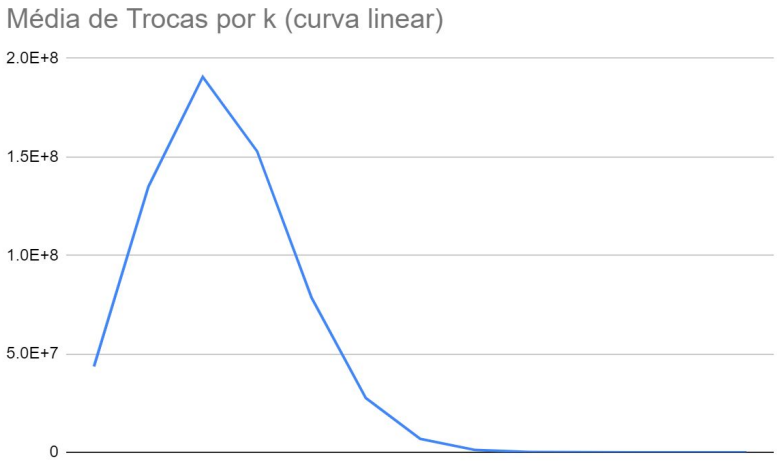
$$\text{Variância} = \frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{n - 1}$$

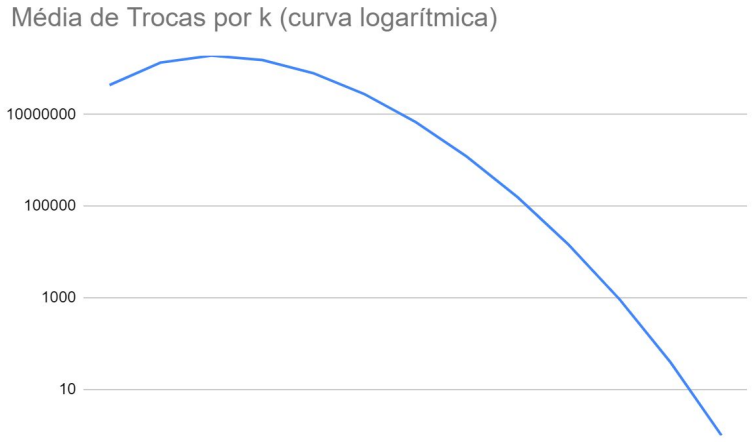
Para a tabela 3, a média e a variância são apresentadas a seguir.

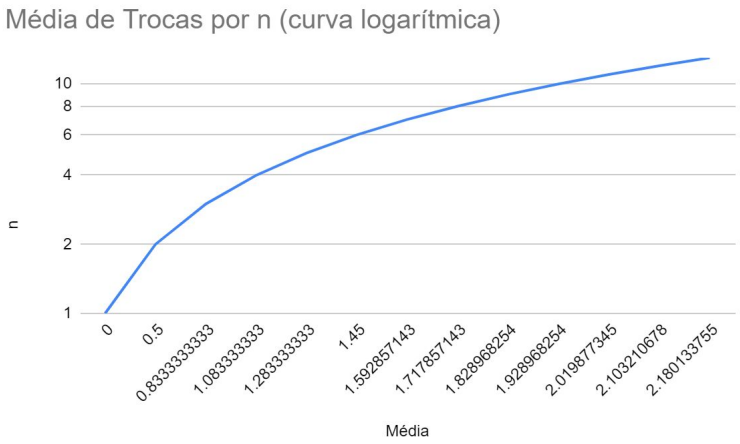
	Qtd trocas	$(X_i - \overline{X})^2$
1 2 3	2	1.361111111
1 3 2	1	0.02777777778
2 1 3	1	0.02777777778
2 3 1	1	0.02777777778
3 1 2	0	0.6944444444
3 2 1	0	0.6944444444
	0.8333333333	0.5666666667
	Média	Variância

Segue o cálculo da média de trocas para as permutações de 1 a 13, e o respectivo gráfico reproduzido no Excel. Disponivel no github: https://github.com/dasamerica/ct208/blob/master/aula_2805_maior_elemento/Gr%C3%A1ficos_trocas.xlsx

	Trocas													Permutações	N	Média
	0	1	2	3	4	5	6	7	8	9	10	11	12			
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
2	1	1	0	0	0	0	0	0	0	0	0	0	0	2	2	0.5
3	2	3	1	0	0	0	0	0	0	0	0	0	0	6	3	0.8333333333
4	6	11	6	1	0	0	0	0	0	0	0	0	0	24	4	1.083333333
5	24	50	35	10	1	0	0	0	0	0	0	0	0	120	5	1.283333333
6	120	274	225	85	15	1	0	0	0	0	0	0	0	720	6	1.45
7	720	1764	1624	735	175	21	1	0	0	0	0	0	0	5040	7	1.592857143
8	5040	13068	13132	6769	1960	322	28	1	0	0	0	0	0	40320	8	1.717857143
9	40320	109584	118124	67284	22449	4536	546	36	1	0	0	0	0	362880	9	1.828968254
10	362880	1026576	1172700	723680	269325	63273	9450	870	45	1	0	0	0	3628800	10	1.928968254
11	3628800	10628640	12753576	8409500	3416930	902055	157773	18150	1320	55	1	0	0	39916800	11	2.019877345
12	39916800	120543840	150917976	105258076	45995730	13339535	2637558	357423	32670	1925	66	1	0	479001600	12	2.103210678
13	479001600	1486442880	1931559552	1414014888	657206836	206070150	44990231	6926634	749463	55770	2717	78	1	6227020800	13	2.180133755


Média de Trocas por k (curva linear)
2.0E+8
1.5E+8
1.0E+8
5.0E+7
0


Média de Trocas por k (curva logarítmica)
10000000
100000
1000
10


Média de Trocas por n (curva logarítmica)
10
8
6
4
2
1
n
0
0.5
0.8333333333
1.083333333
1.283333333
1.45
1.592857143
1.717857143
1.828968254
1.928968254
2.019877345
2.103210678
2.180133755
Média

3 - Programa recursivo $T_{n,k}$

> **Faça um programa recursivo utilizando $T_{n,k}$ como acima. Gere a tabela 1 a partir da sua recursão e estenda seu resultado até $n = 13$.**
> **Seu programa é linear em n? Note que k é no máximo n. (Justifique)**

3.1 Cálculo por Recorrência

Por ser possível executar o programa de 1 até 10 elementos em uma única execução que consome até 1Gb de memória RAM, os valores foram salvos em uma matriz, e a partir desta matriz foram calculadas os valores através da recorrência $T_{n,k}$

$$T_{n,k} = T_{n-1,k-1} + (n-1)\, T_{n-1,k}$$

A utilização de recorrências é linear em n, pois o número de células da matriz de recorrência que armazenam os valores das trocas serão lidas de forma linear n. O valor de k é no máximo n, pois corresponde ao número de células por recorrência que serão calculadas. Ou seja, n varia de 1 a 13, e k de 0 a 12 trocas.

Código Fonte:
https://github.com/dasamerica/ct208/blob/master/aula_2805_maior_elemento/ct208_maior_elemento_recursivo.py

Programa Principal	Funções

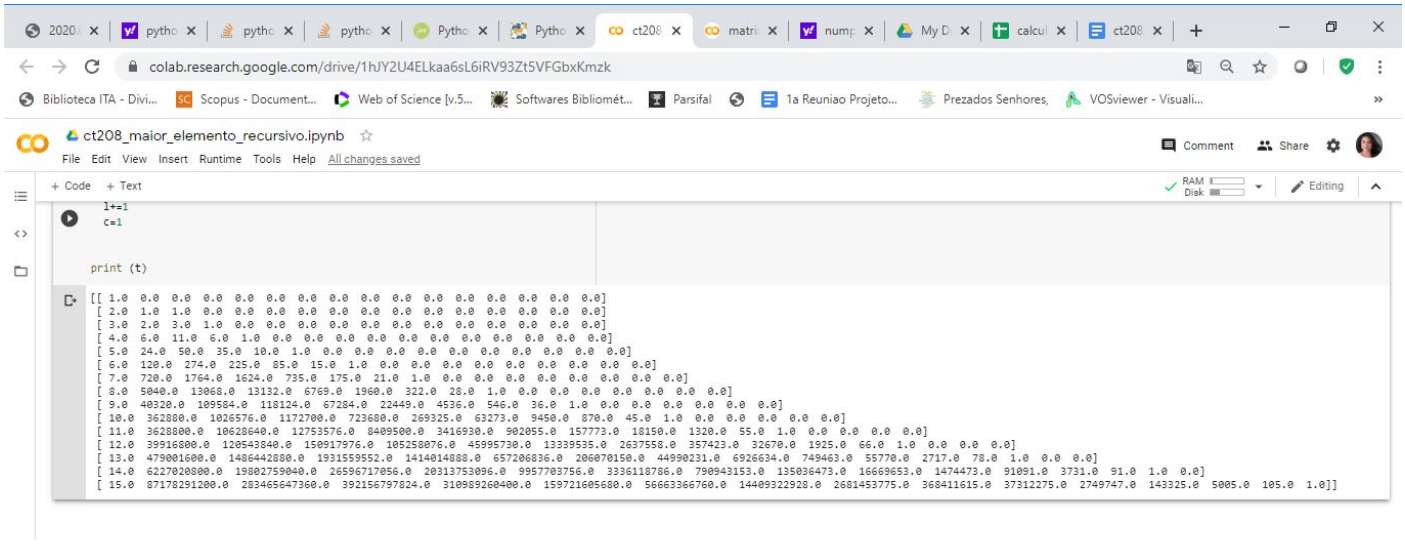
`# Programa phyton para dado um numero gera uma lista de suas permutações`
`# Verifica quantas iterações necessárias para achar o maior elemento da lista`
`# Daniela América da Silva`
`# Disciplina: ct208`
 	
`# inicializa matriz t que armazena resultados das trocas`
`t = np.zeros((15, 16))`
`t[0][0] = 1`
 	
`#processa lista de permutações do número 2 ao 10`
`x = 2`
`while x < 11:`
`  # inicializa lista de trocas`
	`# Programa phyton para dado um numero gera uma lista de suas permutações`
	`# Verifica quantas iterações necessárias para achar o maior elemento da lista`
	`# Daniela América da Silva`
	`# Disciplina: ct208`
	
	`# Importa library de permutações`
	`from itertools import permutations`
	
	`import numpy as np`
	
	`#seta vetor contador de trocas`
	`def set_contador (col):`
	`  n = 0`
	`  conta = []`
	`  while n < col:`
	`  conta.append(0)`

```
  trocas = []

  numeros = cria_lista (x)

  #inicializa contador trocas
  conta_trocas = set_contador(len(numeros))

  # realiza as permutações
  perm = permutations(numeros)

  # Conta as trocas
  for i in list(perm):
    #print (i)
    number = maior_trocas (i)
    #imprime
    #print (i, '(', number ,')')
    conta_trocas[number]+=1


  #print (trocas)
  #print ('numero:', x, conta_trocas)

  #atualiza matriz trocas
  j = 0
  t[x-1][j]=x
  j = 1
  while j < x+1:
    t[x-1][j] = conta_trocas[j-1]
    j+=1

  x+=1

#np.set_printoptions(precision=10,
suppress=True, linewidth=10000)

np.set_printoptions(formatter={'float':
lambda x: ' ' + str(x)})


#processa lista de permutações do número 11
ao 15, utilizando a recorrencia
l=10
c=1
while l < 15:
  t[l][0] = l+1
  t[l][c] = l*t[l-1][c]
  while c < l:
     c+=1
     t[l][c] = t[l-1][c-1] + l*t[l-1][c]
     n+=1
   return (conta)

#cria a lista de números
def cria_lista (num):
   n = 0
   lista = []
   while n < num:
     lista.append(n+1)
     n+=1
   return (lista)

#conta numero de trocas
def maior_trocas (elements):
  m = elements[0]
  n = 1
  t = 0
  while n < (len(elements)):
    if m < elements[n]:
       m = elements[n]
       t+=1
    n+=1
  return (t)
```

```
  c+=1
  t[l][c] = t[l-1][c-1] + l*t[l-1][c]
  l+=1
  c=1


print (t)
```

Resultados até 15 elementos

```
[[ 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 2.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 3.0  2.0  3.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 4.0  6.0  11.0  6.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 5.0  24.0  50.0  35.0  10.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 6.0  120.0  274.0  225.0  85.0  15.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 7.0  720.0  1764.0  1624.0  735.0  175.0  21.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 8.0  5040.0  13068.0  13132.0  6769.0  1960.0  322.0  28.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 9.0  40320.0  109584.0  118124.0  67284.0  22449.0  4536.0  546.0  36.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 10.0  362880.0  1026576.0  1172700.0  723680.0  269325.0  63273.0  9450.0  870.0  45.0  1.0  0.0  0.0  0.0  0.0  0.0]
 [ 11.0  3628800.0  10628640.0  12753576.0  8409500.0  3416930.0  902055.0  157773.0  18150.0  1320.0  55.0  1.0  0.0  0.0  0.0
0.0]
 [ 12.0  39916800.0  120543840.0  150917976.0  105258076.0  45995730.0  13339535.0  2637558.0  357423.0  32670.0  1925.0  66.0
1.0  0.0  0.0  0.0]
 [ 13.0  479001600.0  1486442880.0  1931559552.0  1414014888.0  657206836.0  206070150.0  44990231.0  6926634.0  749463.0
55770.0  2717.0  78.0  1.0  0.0  0.0]
 [ 14.0  6227020800.0  19802759040.0  26596717056.0  20313753096.0  9957703756.0  3336118786.0  790943153.0  135036473.0
16669653.0  1474473.0  91091.0  3731.0  91.0  1.0  0.0]
 [ 15.0  87178291200.0  283465647360.0  392156797824.0  310989260400.0  159721605680.0  56663366760.0  14409322928.0
2681453775.0  368411615.0  37312275.0  2749747.0  143325.0  5005.0  105.0  1.0]]
```



4. Funções Geratrizes

Utilizando a recorrência e as funções geratrizes dadas acima, prove que a quantidade média de trocas é dada por $H_n - 1$.

Sobre as funções geratrizes foi explicado em aula no dia 10-Junho, no item 1 sobre a dedução do caso médio do maior.
Antes da aula foi realizado a visualização do vídeo do Prof. Don Knuth e verificado as páginas de 96-104 do livro *The art of computer programming*. Vol. 3.

1 Dedução do caso médio do maior

Vamos resolver o problema dado na aula passada, da determinação da quantidade média de trocas da variável maior, seja ela representada por $\bar{m}$ é dada por:

$$\bar{m}_n = \frac{1}{n!}\sum_{k=0}^{n-1} k,$$

O cálculo de $\bar{m}$ vem diretamente da definição de média aritmética, ou seja, para um dado n ela é a quantidade de total trocas pela quantidade de casos. Para evitar carregar na notação, vamos definir a probabilidade de que com que n números haja k trocas:

$$p_{n,k} = \frac{T(n,k)}{n!}.$$

Explicação da notação usando o caso $n = 3$ Considere a Tabela 2. Podemos escreve-la como:

qtd trocas	$T(3,k)$	casos
0	2	3, 2, 1 3, 2, 1
1	3	1, 3, 2 2, 1, 3 2, 3, 1
2	1	1, 2, 3

Para calcular o valor $\bar{m}_3$, temos que:

$$\bar{m}_3 = \frac{(0+0)+(1+1+1)+2}{6} = 0\frac{2}{6} + 1\frac{3}{6} + 2\frac{1}{6} = \frac{5}{6}.$$

$$\text{Ou, } \bar{m}_3 = 0\frac{T_{3,0}}{6} + 1\frac{T_{3,1}}{6} + 2\frac{T_{3,2}}{6} = 0p_{3,0} + 1p_{3,1} + 2p_{3,2}.$$

Lembrando agora que:

$$T_{n,k} = T_{(n-1),(k-1)} + (n-1)T_{(n-1),k}. \tag{1}$$

E usando a definição de $p_{n,k}$:

$$p_{n,k} = \frac{T_{n,k}}{n!}, \quad p_{(n-1),(k-1)} = \frac{T_{(n-1),(k-1)}}{(n-1)!}, \quad p_{(n-1),k} = \frac{T_{(n-1),k}}{(n-1)!}.$$

Assim, acertando os termos, chegamos em:

$$p_{n,k} = \frac{(n-1),(k-1)}{n} + \frac{n-1}{n} p_{(n-1,k)}. \quad (2)$$

Usaremos, agora, a função geratriz definida antes.

$$P_n(z) = p_{n,0} + p_{n,1}z + p_{n,2}z^2 + \cdots = \sum_{k=0}^{\infty} p_{n,k} z^k \text{ e } P_n(1) = 1.$$

Como duas séries são iguais se e somente se seus coeficientes forem iguais, sabemos que o termo de potência k na série acima é $p_{n,k}$. Usando este fato e 1 obtemos:

$$p_{n,k=\frac{p(n-1),(k-1)}{n}} z + \frac{(n-1)}{n} p_{(n-1),k},$$

ou seja:

$$P_n(z) = \frac{z}{n} P_{n-1}(z) + \frac{n-1}{n} P_{n-1}(z).$$

Assim, concluímos que:

$$P_n(z) = \frac{z+n-1}{n} P_{n-1}(z). \quad (3)$$

Utilizando a ideia de função geratriz e (3), para o cálculo da média devemos calcular $dP_n(z)/dz$ e avaliar no ponto $z = 1$. Assim,

$$\frac{dP_n(z)}{dz} = \frac{P_{n-1}(z)}{n} + (z+n-1)\frac{dP_{n-1}(z)}{dz}.$$

Para terminar o caso médio o cálculo deve ser no valor $z = 1$, ou seja:

$$\bar{m}_n = \frac{dP_n(1)}{dz} = \frac{P_{n-1}(1)}{n} + \frac{(1+n-1)}{n}\frac{dP_{n-1}(1)}{dz}.$$

Também, $P_{n-1}(1) = 1$, logo:

$$\frac{dP_n(1)}{dz} = \frac{1}{n} + \frac{dP_{n-1}(1)}{dz}, \quad \text{e} \quad \frac{\mathrm{dP_1(1)}}{\mathrm{dz}} = 0.$$

$$\bar{m}_n = \frac{1}{n} + \bar{m}_{n-1}, \quad \text{e} \quad \bar{m}_1 = 0.$$

$$\left.\begin{array}{lcl} \bar{m}_n & = & \frac{1}{n} + \bar{m}_{n-1} \\ \bar{m}_{n-1} & = & \frac{1}{n-1} + \bar{m}_{n-2} \\ \bar{m}_{n-2} & = & \frac{1}{n-2} + \bar{m}_{n-3} \\ \vdots & \vdots & \vdots \\ \bar{m}_1 & = & 0 \end{array}\right\} \Longrightarrow \bar{m}_n = H_n - 1 = \mathcal{O}(\lg n).$$

5. Maior e Menor e os dois maiores

Se você tivesse que determinar os 2 maiores e os 2 menores, $n \geq 4$, você poderia usar o algoritmo do Maior e do menor junto com o do dois Maiores diretamente e ter um tempo limitado por $3/2n + \mathcal{O}(\lg n)$?

Através da estratégia da divisão e conquista é possível ter um tempo limitado por 3/2 n + O(log n), pois é possível quebrar a lista em duas, depois recursivamente realizar a ordenação de cada metade, e então juntar as sublistas já ordenadas [9].

Analisando a demonstração do divide and conquer representada no TutorialsPoint [9]:

Analysis

Let us consider, the running time of Merge-Sort as ***T(n)***. Hence,

$$T(n) = \begin{cases} c & if\ n \leqslant 1 \\ 2\,x\,T(\frac{n}{2}) + d\,x\,n & otherwise \end{cases}$$ where c and d are constants

Therefore, using this recurrence relation,

$$T(n) = 2^i T(\frac{n}{2^i}) + i.d.n$$

As, $i = log\,n$, $T(n) = 2^{log\,n} T(\frac{n}{2^{log\,n}}) + log\,n.d.n$

$$= c.n + d.n.log\,n$$

Therefore, $T(n) = O(n\,log\,n)$

Adicionalmente teremos, n/2 + (n/2 - 1) + (n/2 - 1) buscas, e portanto uma constante (3/2n - 2) + O (nlogn). E utilizando os elementos de menor ordem pode ser escrito como 3/2n + O(log n).

Exemplo do TutorialsPoint [9].

←---------- n/2 ----------→

←----------(n/2 -1)----------→←----------(n/2 - 1)----------→

Divide and Merge operations step by step:

32	14	15	27	31	7	23	26
32	14	15	27	31	7	23	26
32	14	15	27	31	7	23	26
32	14	15	27	31	7	23	26
14	32	15	27	7	31	23	26
14	15	27	32	7	23	26	31
7	14	15	23	26	27	31	32

Seguem as simulações:

- no item 5.1 por recorrência em que não há melhoria no tempo de processamento e,
- no item 5.2 pela estratégia de divisão e conquista com a redução do tempo.

5.1- Cálculo do maior, menor, segundo maior e segundo menor, por recorrência

Realizando a implementação a partir do cálculo por recorrências, não há alteração na performance, e no número de trocas, uma vez que encontrar o menor elemento significa apenas um *if* adicional na função que calcula as trocas, e encontrar o segundo maior ou segundo menor elemento é apenas uma atribuição de duas novas variáveis.

A seguir o código e a demonstração dos resultados.

Código fonte:
Apenas maior e menor elemento:
https://github.com/dasamerica/ct208/blob/master/aula_2805_maior_elemento/ct208_maior_menor_elemento_recursivo.py
Com maior/menor/segundo maior/segundo menor elemento:
https://github.com/dasamerica/ct208/blob/master/aula_2805_maior_elemento/ct208_maior_menor_segundo_elemento_recursivo.py

Programa Principal	Funções

`# Programa phyton para dado um numero gera uma lista de suas permutações`
`# Verifica quantas iterações necessárias para achar o maior, menor, segundo maior e segundo menor elemento da lista`
`# Daniela América da Silva`
`# Disciplina: ct208`
 	
`# inicializa matriz t que armazena resultados das trocas`
`t_maior = np.zeros((15, 16))`
`t_maior[0][0] = 1`
 	
`t_menor = np.zeros((15, 16))`
`t_menor[0][0] = 1`
 	
`t_maior_segundo = np.zeros((15, 16))`
`t_maior_segundo[0][0] = 1`
 	
`t_menor_segundo = np.zeros((15, 16))`
`t_menor_segundo[0][0] = 1`
 	
`#processa lista de permutações do número 2 ao 10`
`x = 2`
`while x < 11:`
`  # inicializa lista de trocas`
`  trocas = []`
	`# Programa phyton para dado um numero gera uma lista de suas permutações`
	`# Verifica quantas iterações necessárias para achar o maior, menor, segundo maior e segundo menor elemento da lista`
	`# Daniela América da Silva`
	`# Disciplina: ct208`
	
	`# Importa library de permutações`
	`from itertools import permutations`
	
	`import numpy as np`
	
	`#seta vetor contador de trocas`
	`def set_contador (col):`
	`  n = 0`
	`  conta = []`
	`  while n < col:`
	`  conta.append(0)`
	`  n+=1`
	`  return (conta)`
	
	`#cria a lista de números`
	`def cria_lista (num):`
	`  n = 0`
	`  lista = []`
	`  while n < num:`
	`  lista.append(n+1)`
	`  n+=1`

```
  numeros = cria_lista (x)

  #inicializa contador trocas
  conta_trocas_maior =
set_contador(len(numeros))
  conta_trocas_menor =
set_contador(len(numeros))
  conta_trocas_maior_segundo =
set_contador(len(numeros))
  conta_trocas_menor_segundo =
set_contador(len(numeros))


  # realiza as permutações
  perm = permutations(numeros)

  # Conta as trocas
  for i in list(perm):
    #print (i)
    number = maior_menor_trocas (i)
    #imprime
    #print (i, '(', number ,')')
    conta_trocas_maior[number[0]]+=1
    conta_trocas_menor[number[1]]+=1
    conta_trocas_maior_segundo[number[2]]+=1
    conta_trocas_menor_segundo[number[3]]+=1


  #print (trocas)
  #print ('numero:', x, conta_trocas)

  #atualiza matriz trocas
  j = 0
  t_maior[x-1][j]=x
  t_menor[x-1][j]=x
  t_maior_segundo[x-1][j]=x
  t_menor_segundo[x-1][j]=x

  j = 1
  while j < x+1:
    t_maior[x-1][j] = conta_trocas_maior[j-1]
    t_menor[x-1][j] = conta_trocas_menor[j-1]
    t_maior_segundo[x-1][j] =
conta_trocas_maior_segundo[j-1]
    t_menor_segundo[x-1][j] =
conta_trocas_menor_segundo[j-1]
    j+=1

  x+=1
```

```
   return (lista)

#conta numero de trocas incluindo o segundo
elemento
def maior_menor_trocas (elements):
  me = elements[0]
  ma = elements[0]
  ma_2 = elements[0]
  me_2 = elements[0]
  n = 1
  t= [0,0,0,0]
  while n < (len(elements)):
    # conta trocas maior
    if ma < elements[n]:
       ma_2 = ma
       ma = elements[n]
       t[0]+=1
       t[2]+=1
    # conta trocas menor
    if me > elements [n]:
       me_2 = me
       me = elements[n]
       t[1]+=1
       t[3]+=1
    n+=1
  return (t)
```

```
#np.set_printoptions(precision=10,
suppress=True, linewidth=10000)

np.set_printoptions(formatter={'float':
lambda x: ' ' + str(x)})


#processa lista de permutações do número 11
ao 15, utilizando a recorrencia
l=10
c=1
while l < 15:
  t_maior[l][0] = l+1
  t_maior[l][c] = l*t_maior[l-1][c]
  t_menor[l][0] = l+1
  t_menor[l][c] = l*t_menor[l-1][c]
  t_maior_segundo[l][0] = l+1
  t_maior_segundo[l][c] =
l*t_maior_segundo[l-1][c]
  t_menor_segundo[l][0] = l+1
  t_menor_segundo[l][c] =
l*t_menor_segundo[l-1][c]


  while c < l:
     c+=1
     t_maior[l][c] = t_maior[l-1][c-1] +
l*t_maior[l-1][c]
     t_menor[l][c] = t_menor[l-1][c-1] +
l*t_menor[l-1][c]
     t_maior_segundo[l][c] =
t_maior_segundo[l-1][c-1] +
l*t_maior_segundo[l-1][c]
     t_menor_segundo[l][c] =
t_menor_segundo[l-1][c-1] +
l*t_menor_segundo[l-1][c]

  c+=1
  t_maior[l][c] = t_maior[l-1][c-1] +
l*t_maior[l-1][c]
  t_menor[l][c] = t_menor[l-1][c-1] +
l*t_menor[l-1][c]
  t_maior_segundo[l][c] =
t_maior_segundo[l-1][c-1] +
l*t_maior_segundo[l-1][c]
  t_menor_segundo[l][c] =
t_menor_segundo[l-1][c-1] +
l*t_menor_segundo[l-1][c]
```

```
  l+=1
  c=1


print ('CONTA TROCAS PARA MAIOR NUMERO')
print (t_maior)
print ('CONTA TROCAS PARA MENOR NUMERO')
print (t_menor)

print ('CONTA TROCAS PARA SEGUNDO MAIOR
NUMERO')
print (t_maior_segundo)
print ('CONTA TROCAS PARA SEGUNDO MENOR
NUMERO')
print (t_menor_segundo)
```

Resultados:

CONTA TROCAS PARA MAIOR NUMERO

```
[[ 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 2.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 3.0  2.0  3.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 4.0  6.0  11.0  6.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 5.0  24.0  50.0  35.0  10.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 6.0  120.0  274.0  225.0  85.0  15.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 7.0  720.0  1764.0  1624.0  735.0  175.0  21.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 8.0  5040.0  13068.0  13132.0  6769.0  1960.0  322.0  28.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 9.0  40320.0  109584.0  118124.0  67284.0  22449.0  4536.0  546.0  36.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 10.0  362880.0  1026576.0  1172700.0  723680.0  269325.0  63273.0  9450.0  870.0  45.0  1.0  0.0  0.0  0.0  0.0  0.0]
 [ 11.0  3628800.0  10628640.0  12753576.0  8409500.0  3416930.0  902055.0  157773.0  18150.0  1320.0  55.0  1.0  0.0  0.0  0.0
0.0]
 [ 12.0  39916800.0  120543840.0  150917976.0  105258076.0  45995730.0  13339535.0  2637558.0  357423.0  32670.0  1925.0  66.0
1.0  0.0  0.0  0.0]
 [ 13.0  479001600.0  1486442880.0  1931559552.0  1414014888.0  657206836.0  206070150.0  44990231.0  6926634.0  749463.0
55770.0  2717.0  78.0  1.0  0.0  0.0]
 [ 14.0  6227020800.0  19802759040.0  26596717056.0  20313753096.0  9957703756.0  3336118786.0  790943153.0  135036473.0
16669653.0  1474473.0  91091.0  3731.0  91.0  1.0  0.0]
 [ 15.0  87178291200.0  283465647360.0  392156797824.0  310989260400.0  159721605680.0  56663366760.0  14409322928.0
2681453775.0  368411615.0  37312275.0  2749747.0  143325.0  5005.0  105.0  1.0]]
```

CONTA TROCAS PARA MENOR NUMERO

```
[[ 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 2.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 3.0  2.0  3.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 4.0  6.0  11.0  6.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 5.0  24.0  50.0  35.0  10.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 6.0  120.0  274.0  225.0  85.0  15.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 7.0  720.0  1764.0  1624.0  735.0  175.0  21.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 8.0  5040.0  13068.0  13132.0  6769.0  1960.0  322.0  28.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 9.0  40320.0  109584.0  118124.0  67284.0  22449.0  4536.0  546.0  36.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 10.0  362880.0  1026576.0  1172700.0  723680.0  269325.0  63273.0  9450.0  870.0  45.0  1.0  0.0  0.0  0.0  0.0  0.0]
```

```
 [ 11.0  3628800.0  10628640.0  12753576.0  8409500.0  3416930.0  902055.0  157773.0  18150.0  1320.0  55.0  1.0  0.0  0.0  0.0
0.0]
 [ 12.0  39916800.0  120543840.0  150917976.0  105258076.0  45995730.0  13339535.0  2637558.0  357423.0  32670.0  1925.0  66.0
1.0  0.0  0.0  0.0]
 [ 13.0  479001600.0  1486442880.0  1931559552.0  1414014888.0  657206836.0  206070150.0  44990231.0  6926634.0  749463.0
55770.0  2717.0  78.0  1.0  0.0  0.0]
 [ 14.0  6227020800.0  19802759040.0  26596717056.0  20313753096.0  9957703756.0  3336118786.0  790943153.0  135036473.0
16669653.0  1474473.0  91091.0  3731.0  91.0  1.0  0.0]
 [ 15.0  87178291200.0  283465647360.0  392156797824.0  310989260400.0  159721605680.0  56663366760.0  14409322928.0
2681453775.0  368411615.0  37312275.0  2749747.0  143325.0  5005.0  105.0  1.0]]
```

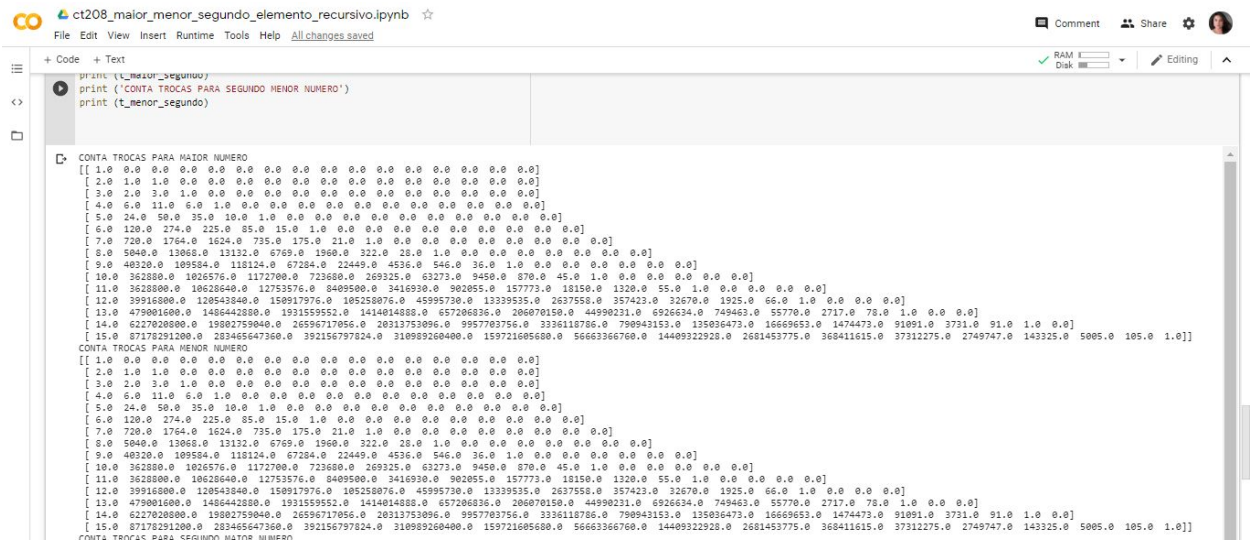
CONTA TROCAS PARA SEGUNDO MAIOR NUMERO

```
[[ 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 2.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 3.0  2.0  3.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 4.0  6.0  11.0  6.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 5.0  24.0  50.0  35.0  10.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 6.0  120.0  274.0  225.0  85.0  15.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 7.0  720.0  1764.0  1624.0  735.0  175.0  21.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 8.0  5040.0  13068.0  13132.0  6769.0  1960.0  322.0  28.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 9.0  40320.0  109584.0  118124.0  67284.0  22449.0  4536.0  546.0  36.0  1.0  0.0  0.0  0.0  0.0  0.0]
 [ 10.0  362880.0  1026576.0  1172700.0  723680.0  269325.0  63273.0  9450.0  870.0  45.0  1.0  0.0  0.0  0.0  0.0]
 [ 11.0  3628800.0  10628640.0  12753576.0  8409500.0  3416930.0  902055.0  157773.0  18150.0  1320.0  55.0  1.0  0.0  0.0  0.0
0.0]
 [ 12.0  39916800.0  120543840.0  150917976.0  105258076.0  45995730.0  13339535.0  2637558.0  357423.0  32670.0  1925.0  66.0
1.0  0.0  0.0  0.0]
 [ 13.0  479001600.0  1486442880.0  1931559552.0  1414014888.0  657206836.0  206070150.0  44990231.0  6926634.0  749463.0
55770.0  2717.0  78.0  1.0  0.0  0.0]
 [ 14.0  6227020800.0  19802759040.0  26596717056.0  20313753096.0  9957703756.0  3336118786.0  790943153.0  135036473.0
16669653.0  1474473.0  91091.0  3731.0  91.0  1.0  0.0]
 [ 15.0  87178291200.0  283465647360.0  392156797824.0  310989260400.0  159721605680.0  56663366760.0  14409322928.0
2681453775.0  368411615.0  37312275.0  2749747.0  143325.0  5005.0  105.0  1.0]]
```

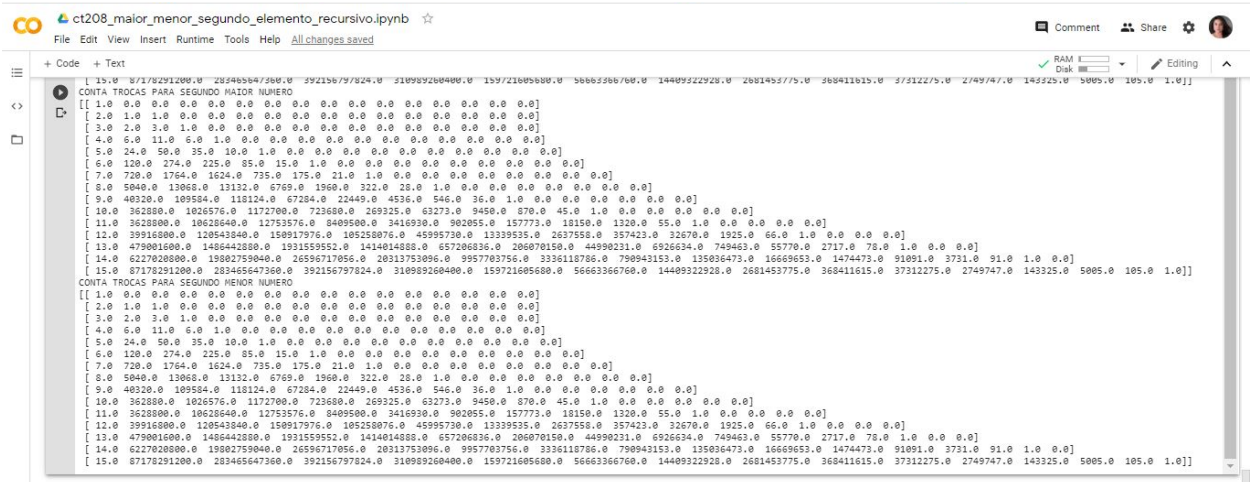
CONTA TROCAS PARA SEGUNDO MENOR NUMERO

```
[[ 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 2.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 3.0  2.0  3.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 4.0  6.0  11.0  6.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 5.0  24.0  50.0  35.0  10.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 6.0  120.0  274.0  225.0  85.0  15.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 7.0  720.0  1764.0  1624.0  735.0  175.0  21.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 8.0  5040.0  13068.0  13132.0  6769.0  1960.0  322.0  28.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 9.0  40320.0  109584.0  118124.0  67284.0  22449.0  4536.0  546.0  36.0  1.0  0.0  0.0  0.0  0.0  0.0]
 [ 10.0  362880.0  1026576.0  1172700.0  723680.0  269325.0  63273.0  9450.0  870.0  45.0  1.0  0.0  0.0  0.0  0.0]
 [ 11.0  3628800.0  10628640.0  12753576.0  8409500.0  3416930.0  902055.0  157773.0  18150.0  1320.0  55.0  1.0  0.0  0.0  0.0
0.0]
 [ 12.0  39916800.0  120543840.0  150917976.0  105258076.0  45995730.0  13339535.0  2637558.0  357423.0  32670.0  1925.0  66.0
1.0  0.0  0.0  0.0]
 [ 13.0  479001600.0  1486442880.0  1931559552.0  1414014888.0  657206836.0  206070150.0  44990231.0  6926634.0  749463.0
55770.0  2717.0  78.0  1.0  0.0  0.0]
 [ 14.0  6227020800.0  19802759040.0  26596717056.0  20313753096.0  9957703756.0  3336118786.0  790943153.0  135036473.0
16669653.0  1474473.0  91091.0  3731.0  91.0  1.0  0.0]
 [ 15.0  87178291200.0  283465647360.0  392156797824.0  310989260400.0  159721605680.0  56663366760.0  14409322928.0
2681453775.0  368411615.0  37312275.0  2749747.0  143325.0  5005.0  105.0  1.0]]
```

Trocas maior e menor



Trocas segundo maior e segundo menor



5.2 - Divisão e Conquista

A quarta versão do programa, utiliza a estratégia da divisão e conquista para o cálculo do maior/segundo maior e menor/segundo menor. Para a execução é utilizado em torno de 1Gb de memória, referente ao armazenamento da lista de permutações de cada número.

Código Fonte:
https://github.com/dasamerica/ct208/blob/master/aula_2805_maior_elemento/ct208_maior_menor_segundo_elemento_DCM.py

Programa Principal	Funções

```
# Programa phyton para dado um numero gera
uma lista de suas permutações
# Verifica quantas iterações necessárias para
achar o maior, menor, segundo maior e segundo
menor elemento da lista
# Daniela América da Silva
# Disciplina: ct208


# inicializa matriz t que armazena resultados
das trocas
t_maior = np.zeros((15, 16))
t_maior[0][0] = 1

t_menor = np.zeros((15, 16))
t_menor[0][0] = 1

t_maior_segundo = np.zeros((15, 16))
t_maior_segundo[0][0] = 1

t_menor_segundo = np.zeros((15, 16))
t_menor_segundo[0][0] = 1

#processa lista de permutações do número 2 ao
10
x = 2
while x < 11:
  # inicializa lista de trocas
  trocas = []

  numeros = cria_lista (x)

  #inicializa contador trocas
  conta_trocas_maior = set_contador(15)
  conta_trocas_menor = set_contador(15)
  conta_trocas_maior_segundo =
set_contador(15)
  conta_trocas_menor_segundo =
set_contador(15)


  # realiza as permutações
  perm = permutations(numeros)

  # Conta as trocas
  for i in list(perm):
    #print (i)
    number = maior_menor_trocas (i)
    #imprime
```

```
# Programa phyton para dado um numero gera
uma lista de suas permutações
# Verifica quantas iterações necessárias para
achar o maior, menor, segundo maior e segundo
menor elemento da lista
# Daniela América da Silva
# Disciplina: ct208

# Importa library de permutações
from itertools import permutations
# Importa library para matriz
import numpy as np
# Importa library para truncate
import math

#seta vetor contador de trocas
def set_contador (col):
   n = 0
   conta = []
   while n < col:
     conta.append(0)
     n+=1
   return (conta)

#cria a lista de números
def cria_lista (num):
   n = 0
   lista = []
   while n < num:
     lista.append(n+1)
     n+=1
   return (lista)

#divide and conquer min
def two_min(arr, iter):
    n = len(arr)
    m=math.trunc(n/2)

    #print(arr, n, iter)
    if n==2: # Oops, we don't consider this
as comparison, right?

        if arr[0]<arr[1]:                    #
Line 1
            iter+=1
            return (arr[0], arr[1], iter)
        else:
            iter+=1
            return (arr[1], arr[0], iter)
```

```
    #print (i, '(', number ,')')
    conta_trocas_maior[number[0]]+=1
    conta_trocas_menor[number[1]]+=1
    conta_trocas_maior_segundo[number[2]]+=1
    conta_trocas_menor_segundo[number[3]]+=1


  #print (trocas)
  #print ('numero:', x, conta_trocas)

  #atualiza matriz trocas
  j = 0
  t_maior[x-1][j]=x
  t_menor[x-1][j]=x
  t_maior_segundo[x-1][j]=x
  t_menor_segundo[x-1][j]=x

  j = 1
  while j < x+1:
    t_maior[x-1][j] = conta_trocas_maior[j-1]
    t_menor[x-1][j] = conta_trocas_menor[j-1]
    t_maior_segundo[x-1][j] =
conta_trocas_maior_segundo[j-1]
    t_menor_segundo[x-1][j] =
conta_trocas_menor_segundo[j-1]
    j+=1

  x+=1


#utilizando dividde and conquer o número de
iterações é o fatorial do número
#é possível utilizar memoization para
calcular o fatorial a partir do número 11

#processa lista de permutações do número 11
ao 15, utilizando a recorrencia
l=10
while l < 15:
  t_maior[l][0] = t_menor[l][0] =
t_maior_segundo[l][0] = t_menor_segundo[l][0]
= l+1
  t_maior[l][l+1] = t_menor[l][l+1] =
t_maior_segundo[l][l+1] =
t_menor_segundo[l][l+1] = factorial(l+1)
  l+=1

#np.set_printoptions(precision=10,
suppress=True, linewidth=10000)
```

```
    if m == 1:
       if arr[0]<arr[1]:
          (least_left, sec_least_left) =
(arr[0],arr[1])
       else:
          (least_left, sec_least_left) =
(arr[1],arr[0])
    else: # always compare at least pairs
      (least_left, sec_least_left, iter) =
two_min(arr[0:m], iter)
    (least_right, sec_least_right, iter) =
two_min(arr[m:n], iter)
    if least_left < least_right:            #
Line 2
        iter+=1
        least = least_left
        if least_right < sec_least_left:    #
Line 3
            return (least, least_right, iter)
        else:
            return (least, sec_least_left,
iter)
    else:
        iter+=1
        least = least_right
        if least_left < sec_least_right:    #
Line 4
            return (least, least_left, iter)
        else:
            return (least, sec_least_right,
iter)


#divide and conquer max
def two_max(arr, iter):
    n = len(arr)
    m=math.trunc(n/2)


    #print(arr, n, iter)
    if n==2: # Oops, we don't consider this
as comparison, right?

        if arr[0]>arr[1]:                   #
Line 1
            iter+=1
            return (arr[0], arr[1], iter)
        else:
            iter+=1
```

```
np.set_printoptions(formatter={'float':
lambda x: ' ' + str(x)},linewidth=10000 )


print ('CONTA TROCAS PARA MAIOR NUMERO')
print (t_maior)
print ('CONTA TROCAS PARA MENOR NUMERO')
print (t_menor)

print ('CONTA TROCAS PARA SEGUNDO MAIOR
NUMERO')
print (t_maior_segundo)
print ('CONTA TROCAS PARA SEGUNDO MENOR
NUMERO')
print (t_menor_segundo)
```

```
            return (arr[1], arr[0], iter)
    if m == 1:
       if arr[0]>arr[1]:
          (biggest_left, sec_biggest_left) =
(arr[0],arr[1])
       else:
          (biggest_left, sec_biggest_left) =
(arr[1],arr[0])
    else: # always compare at least pairs
      (biggest_left, sec_biggest_left, iter)
= two_max(arr[0:m], iter)
    (biggest_right, sec_biggest_right, iter)
= two_max(arr[m:n], iter)
    if biggest_left > biggest_right:
# Line 2
        iter+=1
        biggest = biggest_left
        if biggest_right > sec_biggest_left:
# Line 3
            return (biggest, biggest_right,
iter)
        else:
            return (biggest,
sec_biggest_left, iter)
    else:
        iter+=1
        biggest = biggest_right
        if biggest_left > sec_biggest_right:
# Line 4
            return (biggest, biggest_left,
iter)
        else:
            return (biggest,
sec_biggest_right, iter)

#conta numero de trocas incluindo o segundo
elemento
#utiliza divide and conquer
def maior_menor_trocas (elements):
  me = 0
  ma = 0
  ma_2 = 0
  me_2 = 0
  t_ma = 0
  t_me = 0
  t= [0,0,0,0]


  (ma, ma_2, t_ma) = two_max(elements,-1)
```

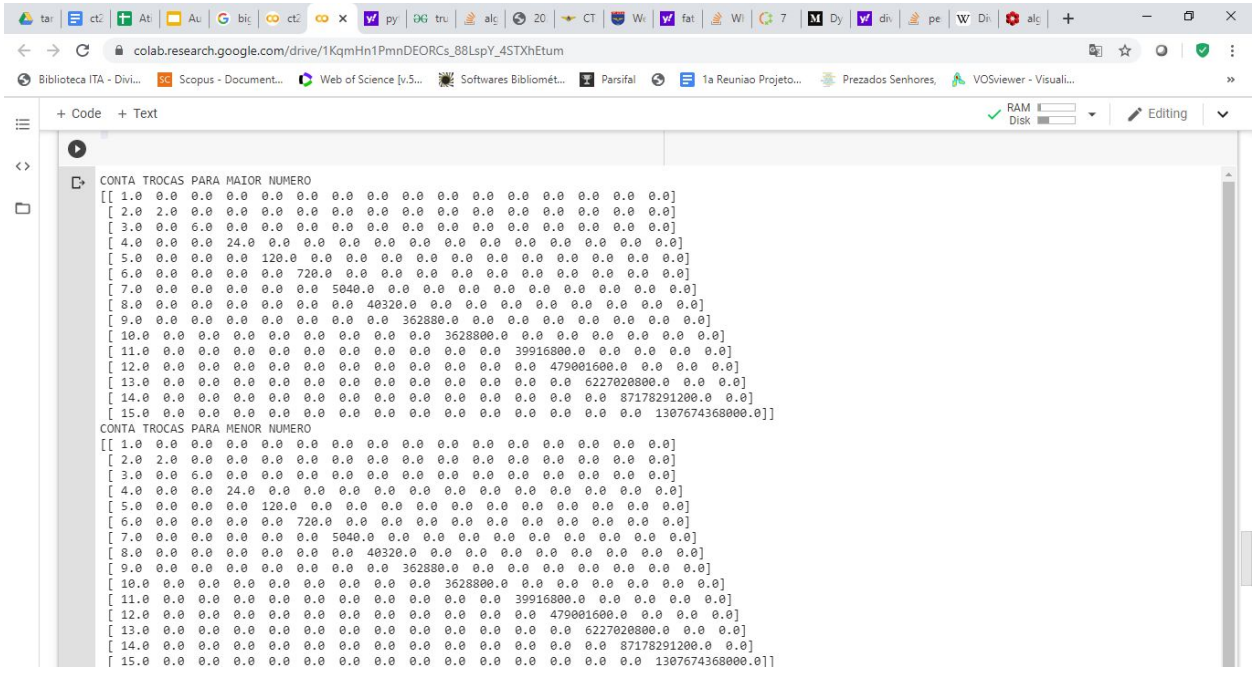
```
  t[0]=t_ma
  t[2]=t_ma

  (me, me_2, t_me) = two_min(elements,-1)
  t[1]=t_me
  t[3]=t_me

  return (t)

#calculo do fatorial utilizando memoization
factorial_memo = {}
def factorial(k):
    if k < 2: return 1
    if k not in factorial_memo:
        factorial_memo[k] = k *
factorial(k-1)
    return factorial_memo[k]
```

Resultados para números de 1 à 15 utilizando divide and conquer.

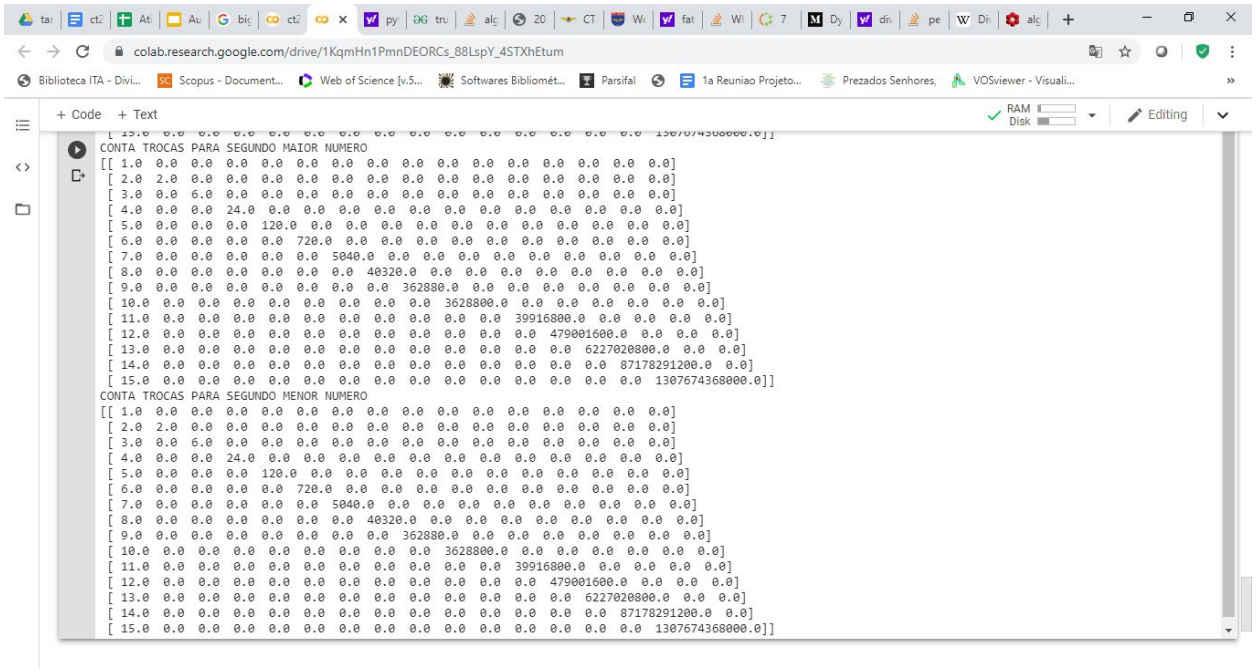
```
CONTA TROCAS PARA MAIOR NUMERO
[[ 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 2.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 3.0  0.0  6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 4.0  0.0  0.0  24.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 5.0  0.0  0.0  0.0  120.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 6.0  0.0  0.0  0.0  0.0  720.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 7.0  0.0  0.0  0.0  0.0  0.0  5040.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 8.0  0.0  0.0  0.0  0.0  0.0  0.0  40320.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 9.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  362880.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 10.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  3628800.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 11.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  39916800.0  0.0  0.0  0.0  0.0]
 [ 12.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  479001600.0  0.0  0.0  0.0]
 [ 13.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  6227020800.0  0.0  0.0]
 [ 14.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  87178291200.0  0.0]
 [ 15.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1307674368000.0]]
CONTA TROCAS PARA MENOR NUMERO
[[ 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 2.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 3.0  0.0  6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 4.0  0.0  0.0  24.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 5.0  0.0  0.0  0.0  120.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 6.0  0.0  0.0  0.0  0.0  720.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 7.0  0.0  0.0  0.0  0.0  0.0  5040.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 8.0  0.0  0.0  0.0  0.0  0.0  0.0  40320.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 9.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  362880.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 10.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  3628800.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 11.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  39916800.0  0.0  0.0  0.0  0.0]
 [ 12.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  479001600.0  0.0  0.0  0.0]
 [ 13.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  6227020800.0  0.0  0.0]
 [ 14.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  87178291200.0  0.0]
 [ 15.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1307674368000.0]]
CONTA TROCAS PARA SEGUNDO MAIOR NUMERO
[[ 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 2.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 3.0  0.0  6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
```

```
 [ 4.0  0.0  0.0  24.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 5.0  0.0  0.0  0.0  120.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 6.0  0.0  0.0  0.0  0.0  720.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 7.0  0.0  0.0  0.0  0.0  0.0  5040.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 8.0  0.0  0.0  0.0  0.0  0.0  0.0  40320.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 9.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  362880.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 10.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  3628800.0  0.0  0.0  0.0  0.0  0.0]
 [ 11.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  39916800.0  0.0  0.0  0.0  0.0]
 [ 12.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  479001600.0  0.0  0.0  0.0]
 [ 13.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  6227020800.0  0.0  0.0]
 [ 14.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  87178291200.0  0.0]
 [ 15.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1307674368000.0]]
CONTA TROCAS PARA SEGUNDO MENOR NUMERO
[[ 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 2.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 3.0  0.0  6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 4.0  0.0  0.0  24.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 5.0  0.0  0.0  0.0  120.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 6.0  0.0  0.0  0.0  0.0  720.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 7.0  0.0  0.0  0.0  0.0  0.0  5040.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 8.0  0.0  0.0  0.0  0.0  0.0  0.0  40320.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 9.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  362880.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 10.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  3628800.0  0.0  0.0  0.0  0.0  0.0  0.0]
 [ 11.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  39916800.0  0.0  0.0  0.0  0.0]
 [ 12.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  479001600.0  0.0  0.0  0.0]
 [ 13.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  6227020800.0  0.0  0.0]
 [ 14.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  87178291200.0  0.0]
 [ 15.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1307674368000.0]]
```

Trocas Maior e segundo maior



Trocas Menor e segundo menor



7- Programas criados para testes

Para adquirir conhecimento sobre as diferentes formas de calcular Fibonacci.
Código fonte:
https://github.com/dasamerica/ct208/blob/master/aula_2805_maior_elemento/ct208_fibonacci.py

Para adquirir conhecimento sobre divide and conquer e memoization.
Código fonte:
https://github.com/dasamerica/ct208/blob/master/aula_2805_maior_elemento/ct208_divide_conquer_memoization.py

Referências

1. Knuth, D, Stanford, Stanford Lecture - Don Knuth: The Analysis of Algorithms (2015, recreating 1969), 2015, acessado de https://www.youtube.com/watch?v=vkUNH9r6UCI) em 08 Junho 2020
2. Knuth, Donald Ervin. *The art of computer programming*. Vol. 3. Pearson Education, 1997, páginas de 96-104
3. Soma, N.Y., ITA, Matemática Computacional, CT208 Notas da Aula de 28 de Maio 2020 - Maior elemento, 2020

4. Sanches, C.A.A, ITA, Estruturas de Dados, Análise de Algoritmos e Complexidade Estrutural, 2020, acessado de http://www.comp.ita.br/~alonso/ensino/CT234/CT234-Cap10.pdf em 08 Junho 2020.
5. Tech with Tim, You tube, Recursion and Memoization Tutorial Python, 2017, acessado de https://www.youtube.com/watch?v=sCecRPSQg6Y em 08 junho 2020
6. StackOverflow, StackOverflow, Memoization Fibonacci Algorithm in Python, acessado de https://stackoverflow.com/questions/29570870/memoization-fibonacci-algorithm-in-python em 08 Junho 2020
7. StackOverflow, StackOverflow, Finding the second smallest number from the given list using divide-and-conquer, acessado de https://stackoverflow.com/questions/19415821/finding-the-second-smallest-number-from-the-given-list-using-divide-and-conquer/19424799#19424799 em 08 Junho 2020
8. StackOverflow, StackOverflow, What is memoization and how can I use it in Phyton, acessado de https://stackoverflow.com/questions/1988804/what-is-memoization-and-how-can-i-use-it-in-python em 08 Junho 2020
9. TutorialsPoint, DAA merge and sort, acessado de https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_merge_sort.htm em 23 Junho 2020.
10. Soma, N.Y., ITA, Matemática Computacional, CT208 Notas da Aula de 10 de Junho 2020 - Caso médio do maior, 2020