Fundamentos de Algoritmos e Estrutura de Dados

Prof. André Gustavo Hochuli

gustavo.hochuli@pucpr.br
aghochuli@ppgia.pucpr.br

Plano de Aula

- **Apresentação do Professor**
- **Apresentação da Disciplina**
  - **O que esperar da disciplina?**
  - **Plano de Ensino**
  - **Ferramentas**
- **Tópicos:**
  - **Complexidade Computacional**
  - **Introdução a Estrutura de Dados**

  - **PONTO DE ATENÇÃO: Mesmo para fins pessoais, não é permitido:**
    - **GRAVAR AÚDIO E VIDEO**
    - **USAR APLICATIVOS DE ANOTAÇÃO/RESUMO, PRINCIPALMENTE AQUELES BASEADOS EM IA**
    - **MOTIVO: LGPD**

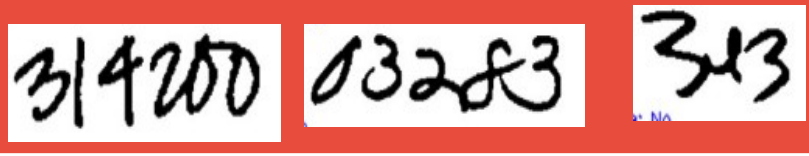
Apresentação do Professor

Prof. André Gustavo Hochuli

- Formação
  - Ciência da Computação [2004, PUCPR]
  - Mestre [2007, PPGIA/PUCPR]
  - Doutor [2018, PPGINF/UFPR]
- Experiência Profissional
  - P&D em Visão Computacional [2008-2013]
  - Professor Universitário [2014 – Atual]
- Linhas de Pesquisa
  - Aprendizagem de Máquina
  - Visão Computacional
  - Deep Learning
  - Reconhecimento de Padrões

Hobbies:
Aviação
Futebol
Tecnologia

Handwritten Recognition



Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Handwritten digit segmentation: Is it still necessary?

A.G. Hochuli [a], L.S. Oliveira [a,*], A.S. Britto Jr [b], R. Sabourin [c]

[a] Federal University of Parana (UFPR), Rua Cel. Francisco H. dos Santos, 100, Curitiba, PR 81531-990, Brazil
[b] Pontifical Catholic University of Parana (PUCPR), R. Imaculada Conceição, 1155, Curitiba, PR 80215-901, Brazil
[c] Ecole de Technologie Superieure, 1100 rue Notre Dame Ouest, Montreal, Quebec, Canada

ARTICLE INFO

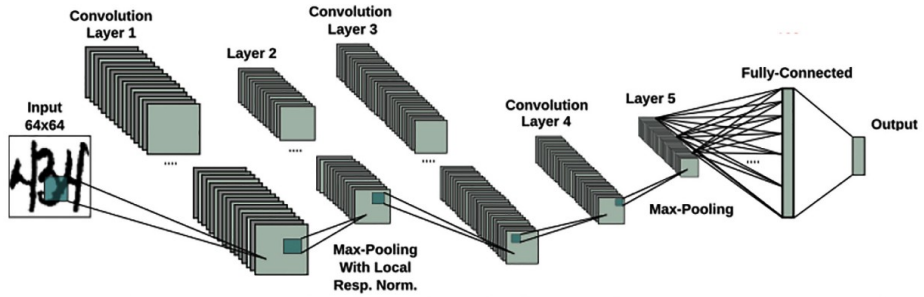
Article history:
Received 3 January 2017
Revised 31 December 2017
Accepted 7 January 2018
Available online 10 January 2018

ABSTRACT

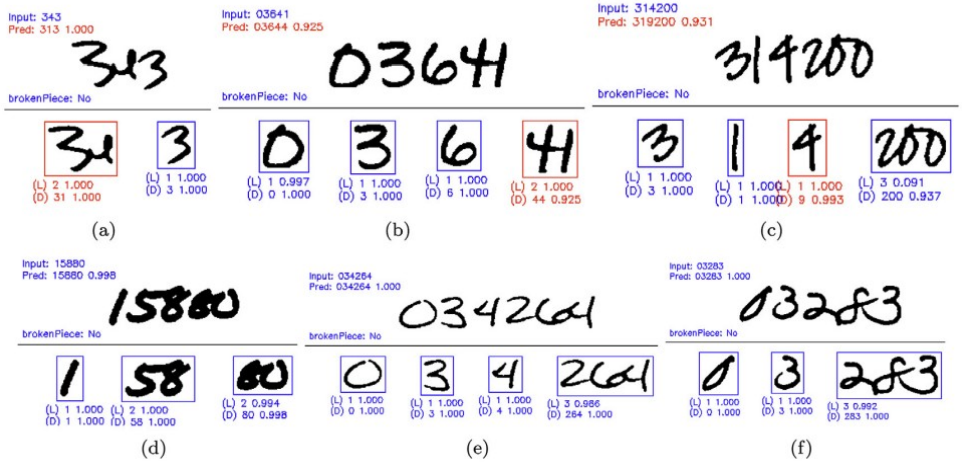
Over the last decades, a great deal of research has been devoted to handwritten digit segmentation. Algorithms based on different features extracted from the background, foreground, and contour of images have been proposed, with those achieving the best results usually relying on a heavy set of heuristics and over-segmentation. Here, the challenge lies in finding a good set of heuristics to reduce the number of segmentation hypotheses. Independently of the heuristic over-segmentation strategy adopted, all algorithms used show their limitations when faced with complex cases such as overlapping digits. In this work, we postulate that handwritten digit segmentation can be successfully replaced by a set of clas-

Table 7
Comparison of the recognition rates on NIST SD19.

Length	Samples	Britto et al. [2]	Oliveira et al. [19]	Oliveira et al. [18]	Sadri et al. [23]	* Sadri et al. [23]	Gatt et a
2	2370	94.8	96.8	97.6	95.5	98.9	99.0
3	2385	91.6	95.3	96.2	91.4	97.2	97.3
4	2345	91.3	93.3	94.2	91.0	96.1	96.5
5	2316	88.3	92.4	94.0	88.0	95.8	95.9
6	2169	89.0	93.1	93.8	88.6	96.1	96.6



Models	Input	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Fully Connected	Output
1-Digit	64x64 Binary	Convolution 7x7@3@64	MaxPooling and Local Resp. Norm. 3x3@2@64	Convolution 3x3@1@128	Convolution 3x3@1@32	MaxPooling 3x3@2@32	128	10
2-Digit	64x64 Binary	Convolution 7x7@3@72	MaxPooling and Local Resp. Norm. 3x3@2@72	Convolution 3x3@1@192	Convolution 3x3@1@64	MaxPooling 3x3@2@64	1024	100
3-Digit	64x64 Binary	Convolution 7x7@1@24	MaxPooling and Local Resp. Norm. 2x2@2@24	Convolution 5x5@1@42	Convolution 5x5@1@32	MaxPooling 2x2@2@32	1200	1000



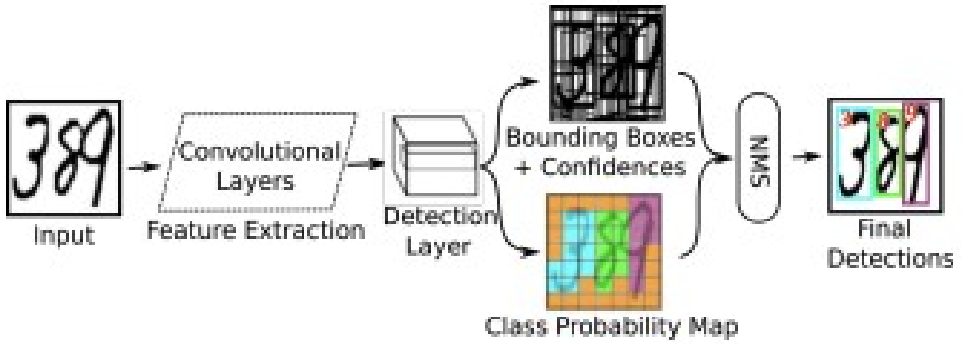
Handwritten Recognition

An End-to-End Approach for Recognition of Modern and Historical Handwritten Numeral Strings

Andre G. Hochuli, Alceu S. Britto Jr., Jean P. Barddal
Graduate Program in Informatics (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba, PR - Brazil
{aghochuli, alceu, jean.barddal}@ppgia.pucpr.br

Luiz E. S. Oliveira
Department of Informatics (DInf)
Universidade Federal do Paraná
Curitiba, PR - Brazil
lesoliveira@inf.ufpr.br

Robert Sabourin
System Engineering Dept. (LIVIA)
École de Technologie Supérieure
Montreal, QC - Canada
robert.sabourin@etsmtl.ca



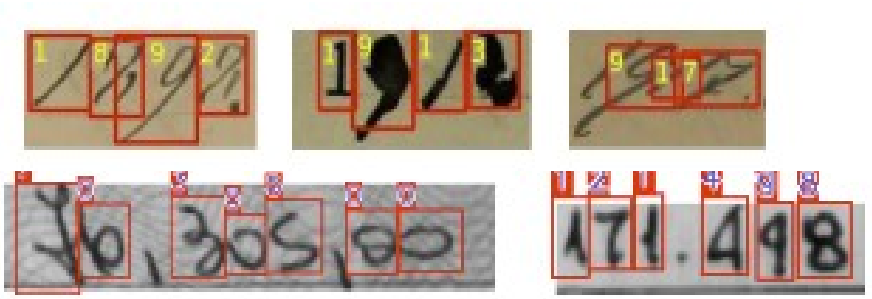


Table V
COMPARISON OF THE ACCURACY (%) ON ORAND-CAR DATASETS REPORTED BY [32]

Methods	CAR-A	CAR-B
Tebessa I[31]	37.05	26.62
Tebessa II[31]	39.72	27.72
Hochuli et al.[5]	50.10	40.20
Singapore[31]	52.30	59.30
Pernanbuco[31]	78.30	75.43
Beijing[31]	80.73	70.13
CRNN[33]	88.01	89.79
Saabni[34]*	85.80	
ResNet-RNN[35]	89.75	91.14
ResNet-RNN[32]	91.89	93.79
YoLo	**96.20**	**96.80**

* Unified CAR-A and CAR-B datasets

Table VI
BENCHMARK FOR ARDIS DATASET IV (SINGLE DIGITS) OF MODELS ON DIFFERENT TRAINNING PROTOCOLS (REPORTED BY [1])

Method	Accuracy (%)			
	Case I	Case II	Case III	Case IV
YoLo	**87.60**	**64.10**	**99.70**	**99.27**
Hochuli et al.[5]	67.20	51.90	83.30	60.55
CNN	58.80	35.44	98.60	99.34
HOG-SVM	56.18	33.18	95.50	98.08
RNN	45.74	28.96	91.12	96.74
kNN	50.15	22.72	89.60	96.63
SVM	43.40	30.62	92.40	96.48
Random Forest	20.12	17.15	87.00	93.12

Parking Lot Monitoring

Evaluation of Different Annotation Strategies for Deployment of Parking Spaces Classification Systems

Andre G. Hochuli, Alceu S. Britto Jr.
Graduate Program in Informatics
Pontifícia Universidade Católica do Paraná
Curitiba, PR - Brazil
{aghochuli, alceu}@ppgia.pucpr.br

Paulo R. L. de Almeida
Department of Informatics
Universidade Federal do Paraná
Curitiba, PR - Brazil
paulorla@ufpr.br

Williams B. S. Alves, Fábio M. C. Cagni
Pontifícia Universidade Católica do Paraná
Curitiba, PR - Brazil
{williams.alves, fabio.cagni}@pucpr.edu.br

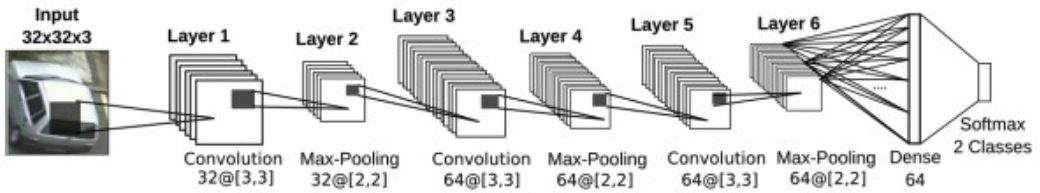


Fig. 1. The 3-convolutional layers architecture used to classify the parking spots.

(a) UFPR04 (b) UFPR05

(c) PUCPR

Fig. 2. The PKLOT dataset comprises three scenarios named a) UFPR04, b) UFPR05 and c) PUCPR.

(a) Polygon (b) Bounding Box (c) Fixed Square

Fig. 3. Types of parking spots location demarcations.

Parking Lot Monitoring

Optimizing Parking Space Classification: Distilling Ensembles into Lightweight Classifiers

Paulo Luza Alves*, André Hochuli†, Luiz Eduardo de Oliveira*, Paulo Lisboa de Almeida*
* Departamento de Informática (DInf), Universidade Federal do Paraná, Curitiba, PR - Brazil
{paulomateus,luiz.oliveira,paulorla}@ufpr.br
† Programa de Pós-Graduação em Informática (PPGIa), Pontifícia Universidade Católica do Paraná, Curitiba, PR - Brazil
aghochuli@ppgia.pucpr.br

***Abstract*—When deploying large-scale machine learning models for smart city applications, such as image-based parking lot monitoring, data often must be sent to a central server to perform classification tasks. This is challenging for the city's infrastructure, where image-based applications require transmitting large volumes of data, necessitating complex network and hardware infrastructures to process the data. To address this issue in image-based parking space classification, we propose creating a robust ensemble of classifiers to serve as Teacher models. These Teacher models are distilled into lightweight and specialized Student models that can be deployed directly on edge devices. The knowledge is distilled to the Student models through pseudo-labeled samples generated by the Teacher model, which are utilized to fine-tune the Student models on the target scenario. Our results show that the Student models, with 26 times fewer parameters than the Teacher models, achieved an average accuracy of 96.6% on the target test datasets, surpassing the Teacher models, which attained an average accuracy of 95.3%.**

I. Introduction

When deploying machine learning models that deal with

Fig. 1. Occupied (red) and empty (blue) parking spaces – PKLot

small-scale deployment with 1,000 cameras. Suppose
camera sends a 1280 × 720 pixels JPEG compressed
to a central server every 30 seconds. In that case, it will
in a transfer of 35 gigabytes of data per hour to the s
not accounting for the network's overheads (estimated
the PKLot dataset [1]). This data transfer can approxir

Deep Single Models vs. Ensembles: Insights for a Fast Deployment of Parking Monitoring Systems

Andre Gustavo Hochuli*
Jean Paul Barddal
Graduate Program in Informatics (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba, PR - Brazil
{aghochuli, jpbarddal}@ppgia.pucpr.br

Gillian Cezar Palhano
Leonardo Matheus Mendes
Pontifícia Universidade Católica do Paraná
Curitiba, PR - Brazil
gillian.palhano@pucpr.edu.br
l.mendes2@pucpr.edu.br

Paulo Ricardo Lisboa de Almeida
Department of Informatics (DInf)
Universidade Federal do Paraná
Curitiba, PR - Brazil
paulorla@ufpr.br

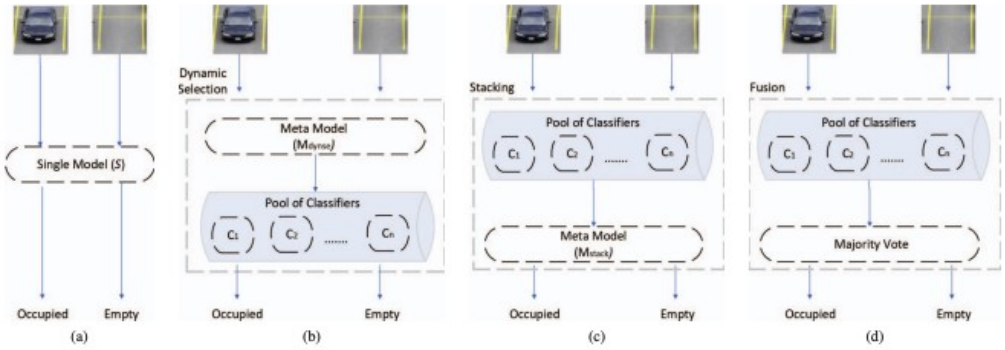


Fig. 1. The proposed strategies assessed for parking lot monitoring systems in cross-dataset scenarios: (a) a single model, and the ensemble-based frameworks named (b) dynamic selection and (c) stacking, and finally (d) the majority vote fusion-based framework.

Facial Expression Recognition

ORIGINAL ARTICLE

Representation ensemble learning applied to facial expression recognition

Bruna Rossetto Delazeri[1] · Andre Gustavo Hochuli[1] · Jean Paul Barddal[1] · Alessandro Lameiras Koerich[2] · Alceu de Souza Britto Jr.[1,3]

Received: 2 February 2024 / Accepted: 3 October 2024 / Published online: 18 November 2024

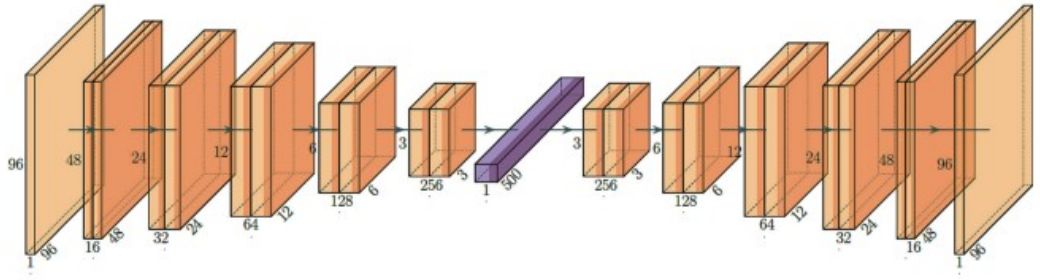
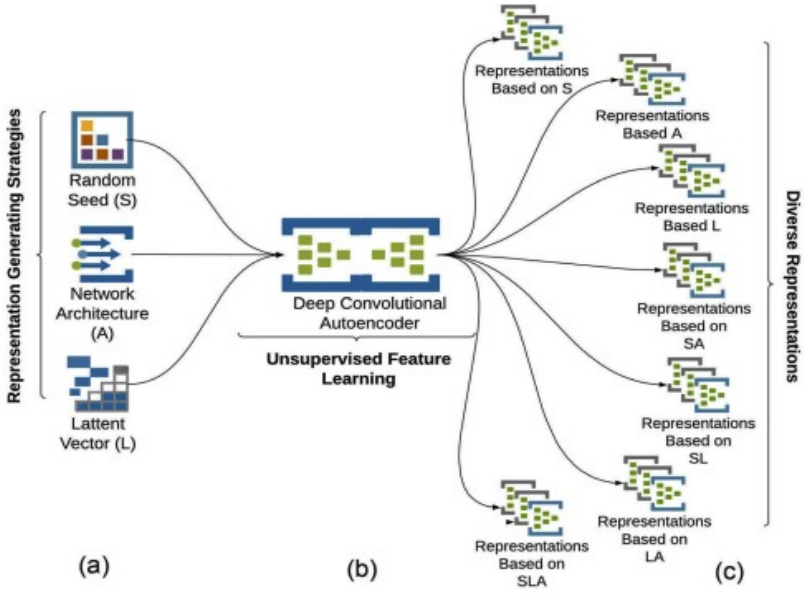
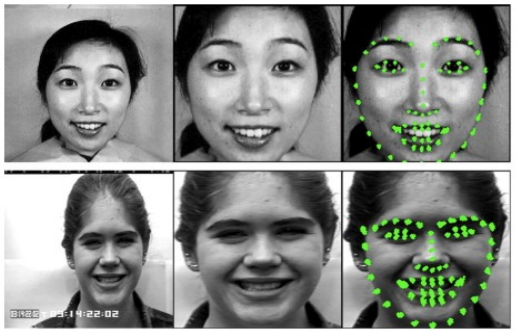


Fig. 3 Default CAE Architecture



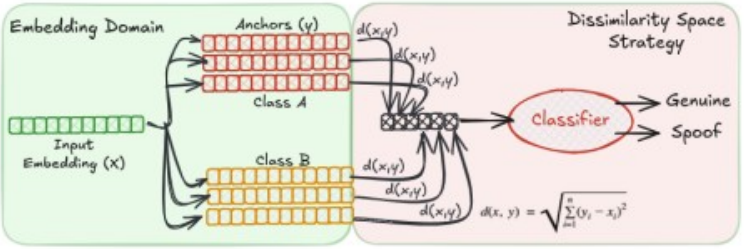


Audio Speaker Verification

A Dissimilarity-Based Countermeasure for Detecting Replay Attacks in Speaker Verification

Maria Eduarda Maciel Pinto
Programa de Pós-Graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba, State of Paraná, Brazil
Email: mariamaciel@ppgia.pucpr.br

Alceu de Souza Britto Jr., Andre Gustavo Hochuli
Programa de Pós-Graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba, State of Paraná, Brazil
Email: {alceu, aghochuli}@ppgia.pucpr.br



similarity Space strategy transforms the embedding (x) by generating a K-dimensional dissimilarity representation, consisting of the Euclidean distances to all anchors ($y_0, ..., y_K$).

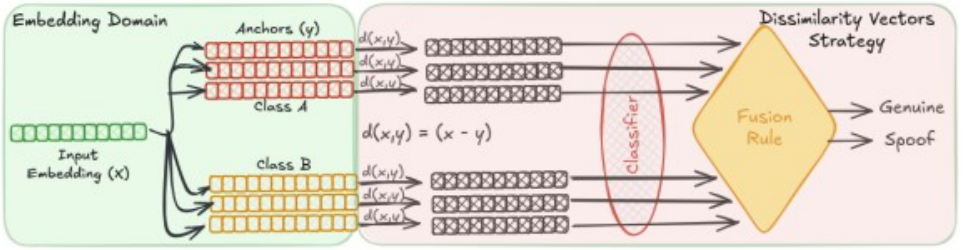


Fig. 3: The Dissimilarity Vectors strategy transforms the embedding (x) into K new representations by calculating the element-wise differences between the input and each of the anchors ($y_0, ..., y_K$).

A. ASVSpoof 2019 Dataset

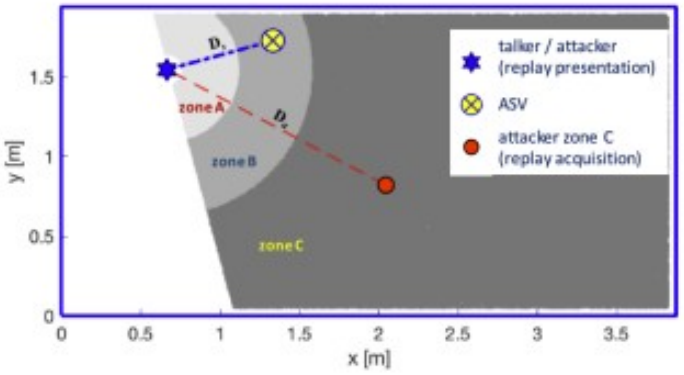


Fig. 4: Proposed Pipeline: a deep architecture performs feature extraction from a MFCC representation, serving as input to the dissimilarity domain.

Plant Species Classification

Combining Muti-Layer Features For Plant Species Classification in a Siamese Network

Matheus Moresco[1], Alceu de S. Britto Jr[1,2], Yandre M. G. Costa[3], Luciano J. Senger[1], Andre G. Hochuli[2]

[1]State University of Ponta Grossa (UEPG), Ponta Grossa (PR), Brazil
Email: {3100120007015, ljsenger, alceubritto}@uepg.br
[2]Graduate Program in Informatics (PPGIa), Pontifícia Universidade Católica do Paraná (PUCPR), Curitiba (PR), Brazil
Email: {aghochuli, alceu}@ppgia.pucpr.br
[3]Department of Informatics (DIN), Universidade Estadual de Maringá (UEM), Maringá (PR), Brazil
Email: {yandre}@din.uem.br

TABLE I
CNN AND SNN PERFORMANCE RATES (%) IN FLAVIA AND MALAYAKEW DATASETS

Architetures	Datasets			
	Flavia		MalayaKew	
	CNN	SNN	CNN	SNN
VGG-16	97.9	97.7	90.9	70.8
MobileNet	**99.5**	98.8	**96.5**	**84.5**
DenseNet201	98.6	**99.4**	95.6	82.9
ResNet50	92.1	74.2	50.2	28.4
Inception v3	98.2	95.7	87.9	66.8

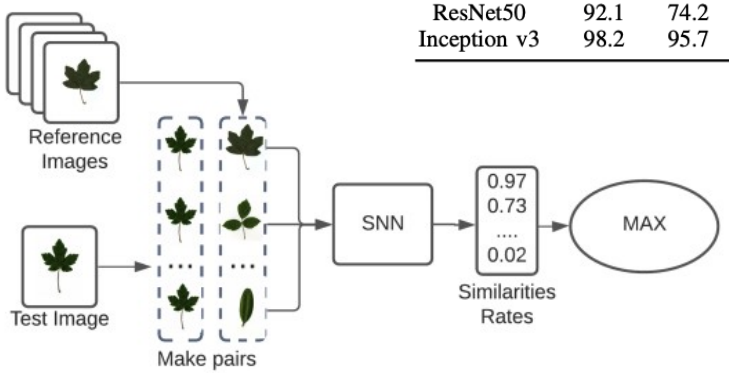


TABLE II
PERFORMANCE (%) OF THE SNN WITH MULTI-LAYER FEATURES IN FLAVIA DATASET

VGG-16 Architecture

Intermediate Layers	Entire Vector	Standard (256)	
	Concatenate	*Concatenate*	*Add*
FC Layer Only	97.3	-	-
B5	92.8	95.9	93.3
B5+B4	96.3	98.4	97.0
B5+B4+B3	97.0	**98.6**	98.2
B5+B4+B3+B2	98.4	98.4	98.2
B5+B4+B3+B2+B1	97.0	97.2	97.9

MobileNet Architecture

Intermediate Layers	Entire Vector	Standard (256)	
	Concatenate	*Concatenate*	*Add*
FC Layer Only	98.7	-	-
B5	99.3	97.2	99.3
B5+B4	98.6	98.9	98.4
B5+B4+B3	98.2	99.4	99.4
B2+B3+B4+B5	98.0	99.3	**99.8**
B1+B2+B3+B4+B5	98.7	98.7	98.2

DenseNet Architecture

Intermediate Layers	Entire Vector	Standard (256)	
	Concatenate	*Concatenate*	*Add*
FC Layer Only	99.4	-	-
B5	98.0	99.1	99.5
B5+B4	98.7	99.4	99.8
B5+B4+B3	98.9	99.3	**100.0**
B2+B3+B4+B5	99.1	99.3	99.4
B1+B2+B3+B4+B5	99.1	99.8	99.8

Research Interests

- Medical Image Analysis
- Edge AI
- Federated Learning
- Mixture of Experts
- Domain Adaptation

Rewarding Experiences

Apresentação da Disciplina

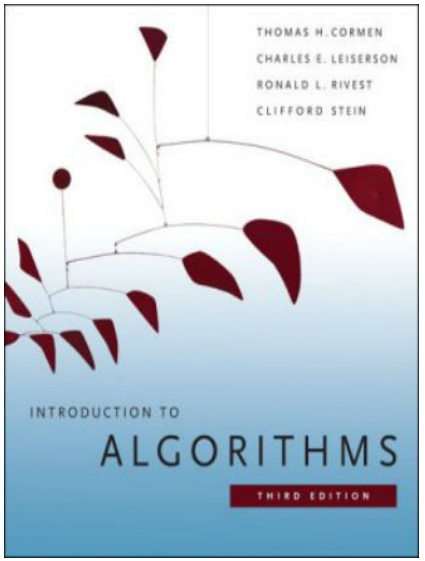
O que esperar da disciplina ?

- Abordagem Teórico – Prática
- Principais Tópicos em Estrutura de Dados
- Material em Inglês - Livros e Artigos
- Resolução de problemas com aplicações no cotidiano computacional
- Espaço para o estudante debater e trazer problemas/dúvidas
- Trabalhos em grupos e avaliações individuais
- Linguagens C e Python

Plano de Ensino

- Introdução e Conceitos Básicos
- Métodos de Ordenação
- Estruturas de Dados Elementares (listas, pilhas, filas)
- Tabela Hash (Hashing)
- Árvores (Binária e AVL) e Métodos de Balanceamento
- Grafos
- Complexidade e Programação Dinâmica
- Avaliação:
  - Trabalhos, Seminários, Avaliações (Provas)
  - Trabalho de Recuperação

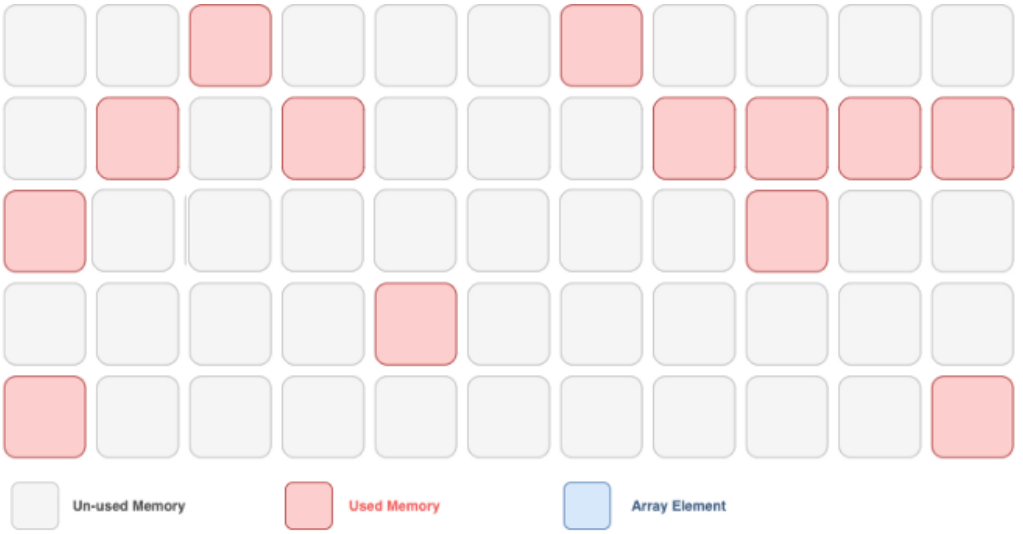
- Média: (Trab + Aval) / N
  - N = Qtd Trab + Qtd de Aval



Tópico 1: Introdução a Estrutura de Dados

Introdução – Tipos Primitivos

- Tipos de Dados Primitivos
  - Inteiros (int)
  - Ponto Flutuante (float/double),
  - caracteres (char),
  - boleanos (bool).....

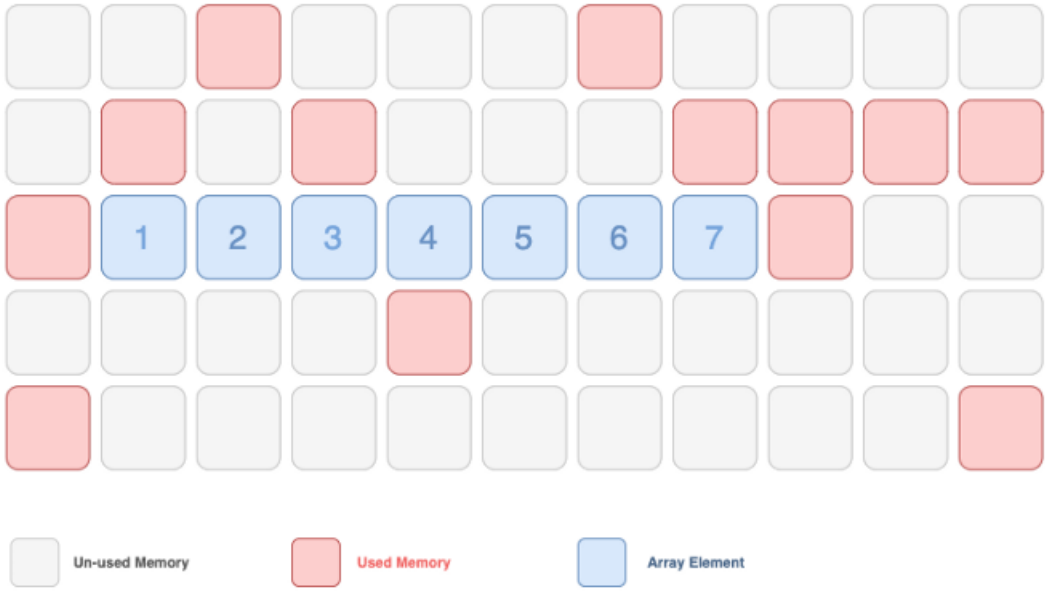


- Representação

- Dados primitivos (variáveis) vs grandes massas de dados

Introdução - Arrays Estáticos

- Arrays Estáticos:
  - Estrutura Indexada
  - Alocação contígua
- Vantagens
  - Acesso é rápido e sequencial
  - Baixo Overhead
  - Requer baixo nível de programaç
- Desvantagem
  - Inviável para grandes massas de dados
  - Fragmentação de mémoria pode impedir a alocação contígua

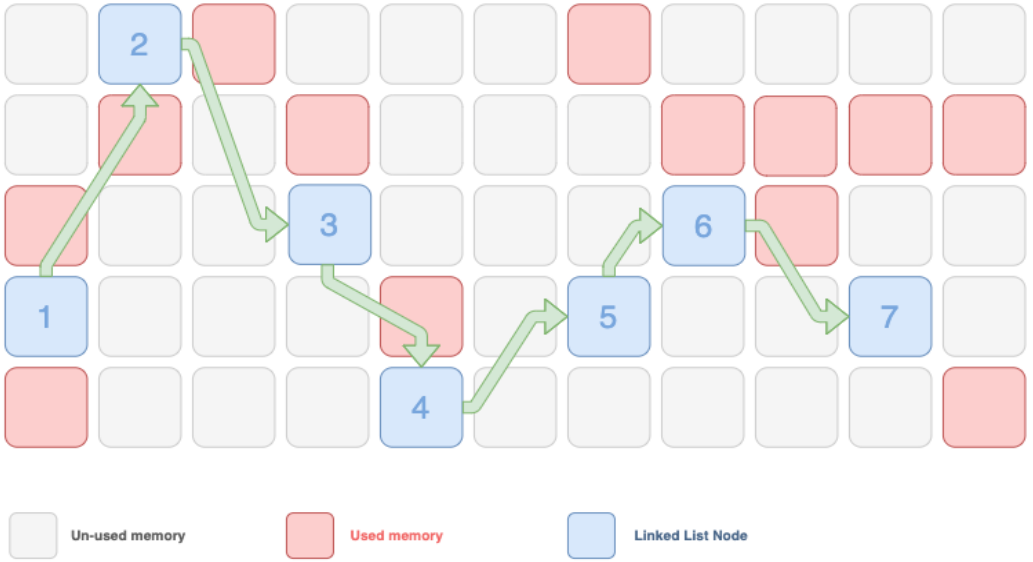


Introdução – Arrays Dinâmicos

- Arrays Dinâmicos:
  - Não Indexado: Ponteiros garantem o encadeamento sequêncial
  - Alocação não-contígua

- Vantagens
  - Armazenar grandes massas de dados
  - Memória física é o limite

- Desvantagem
  - “Desempenho”
  - Alto Overhead
  - Requer alto nível de programação

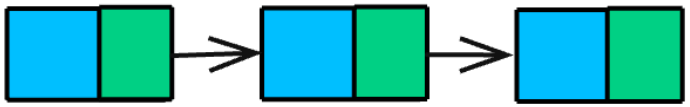


- Tradeoff: Time vs Memory Efficiency

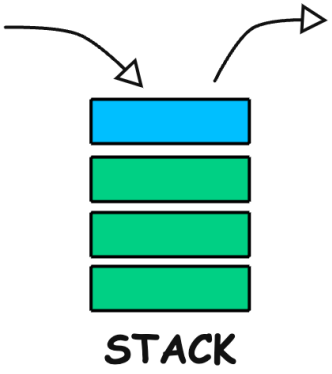
Introdução - Topologias

LINEARES

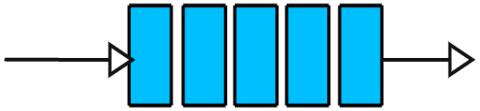
Array



LINKED LIST

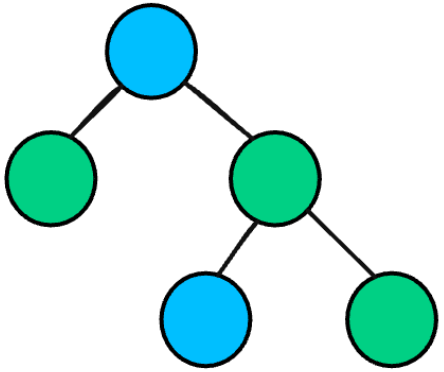


STACK

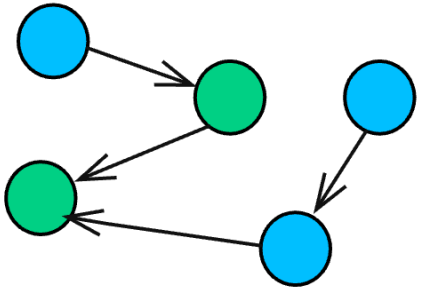


QUEUE

NÃO LINEARES



TREE

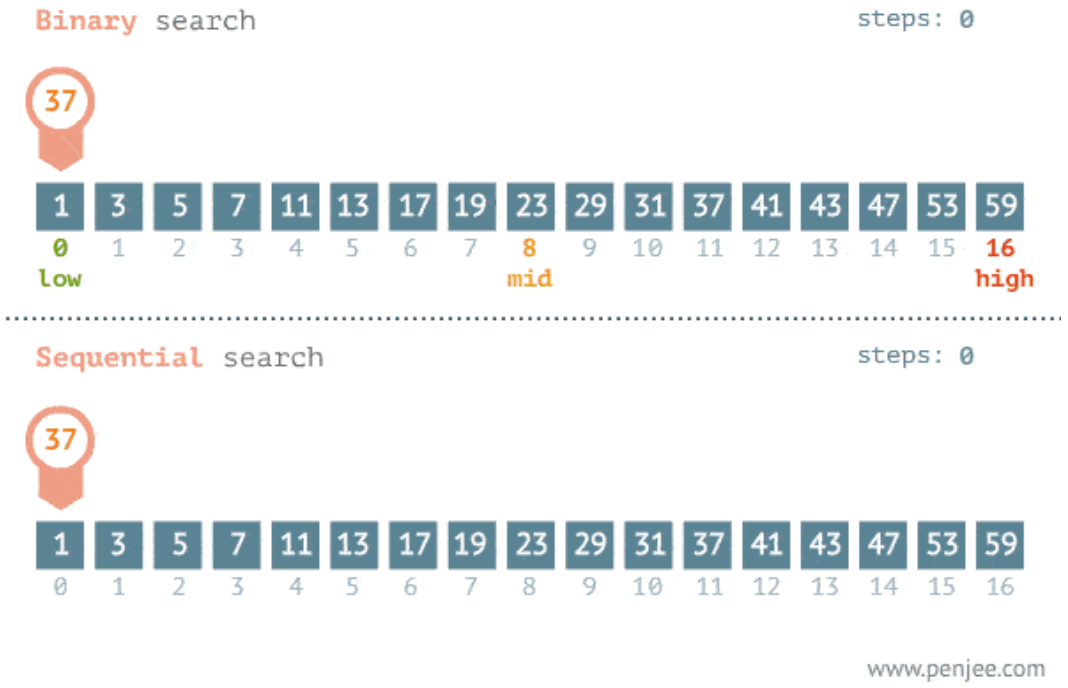


GRAPH

Tópico 2: Algoritmos de Ordenação e Busca (PARTE 1)

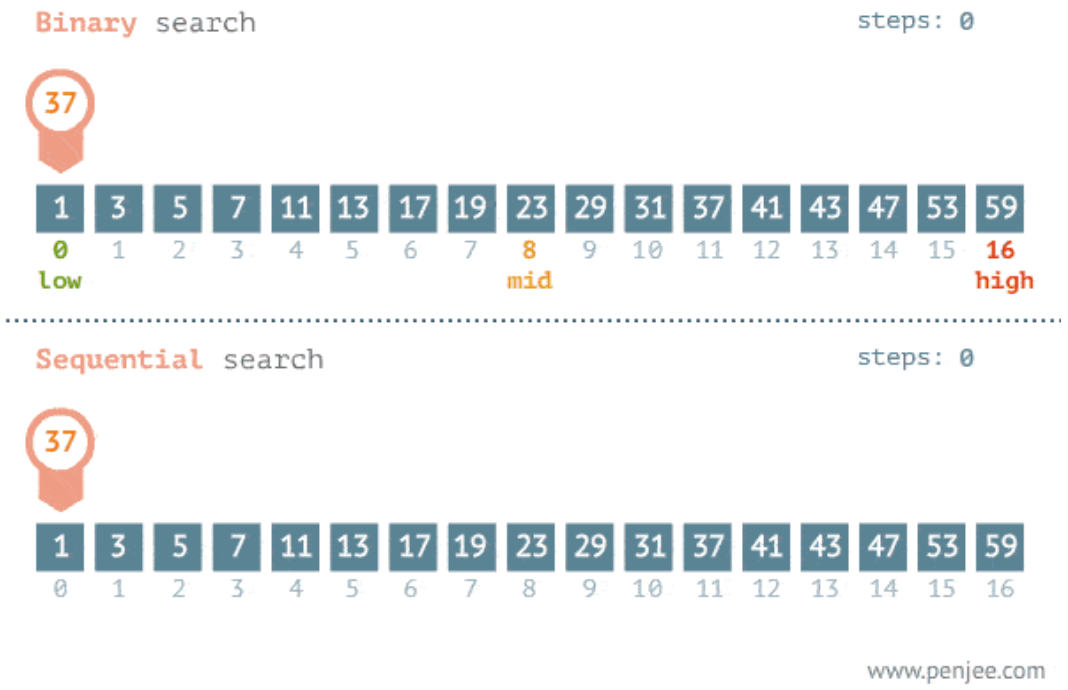
Métodos de Busca

- **Encontrar um elemento em um conjunto de dados**
- **Sequencial ou Linear (arrays ordenados ou não)**
- **Binária (arrays ordenados)**



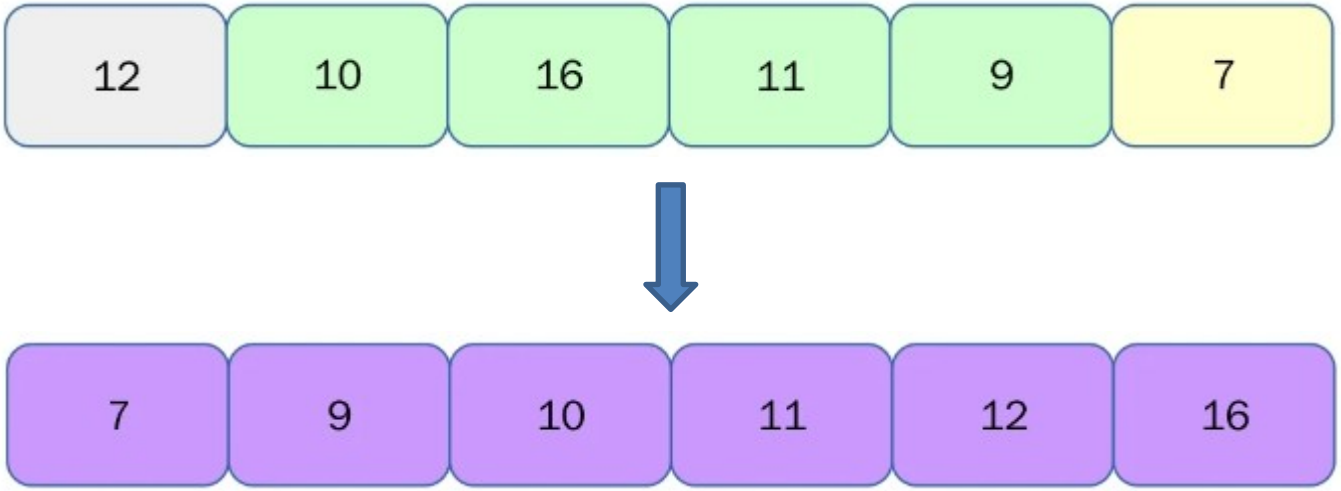
Métodos de Busca

- **Códificação**



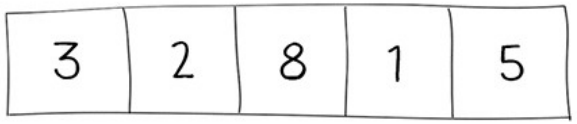
Ordenação

- Organizam os dados
- “Melhoram” a performance de uma busca

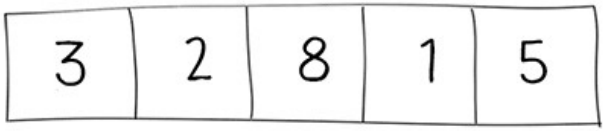


Ordenação

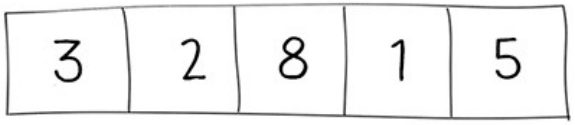
BUBBLE SORT



SELECTION SORT

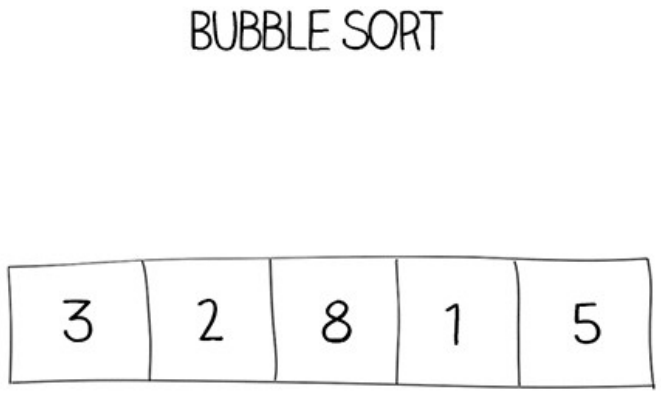


INSERTION SORT

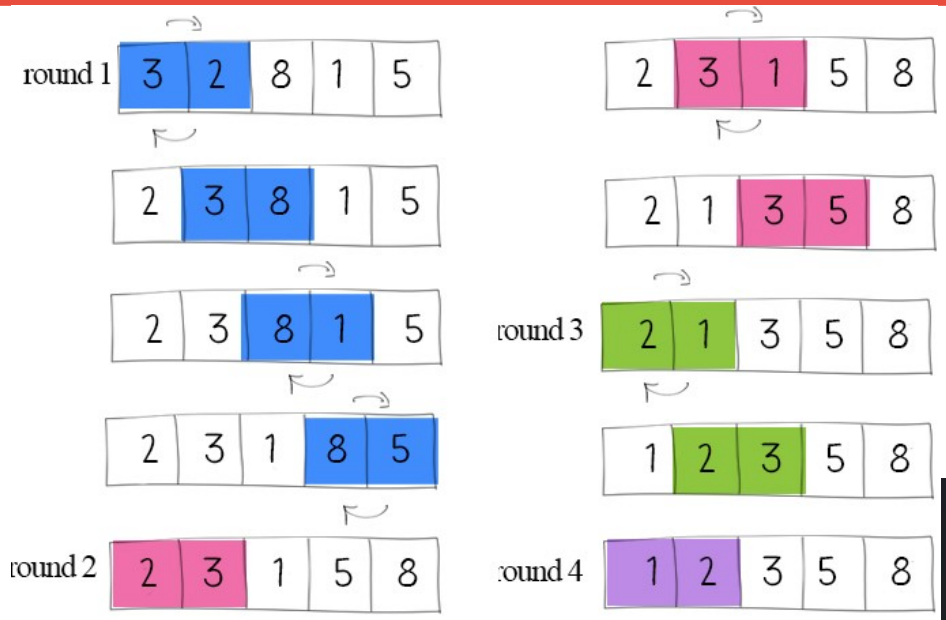


Ordenação – Bubble Sort

- **Pecorre repetidamente trocando elementos adjacentes para ordenar os valores em sequência crescente ou decrescente até que não haja mais trocas.**



Ordenação – Bubble Sort

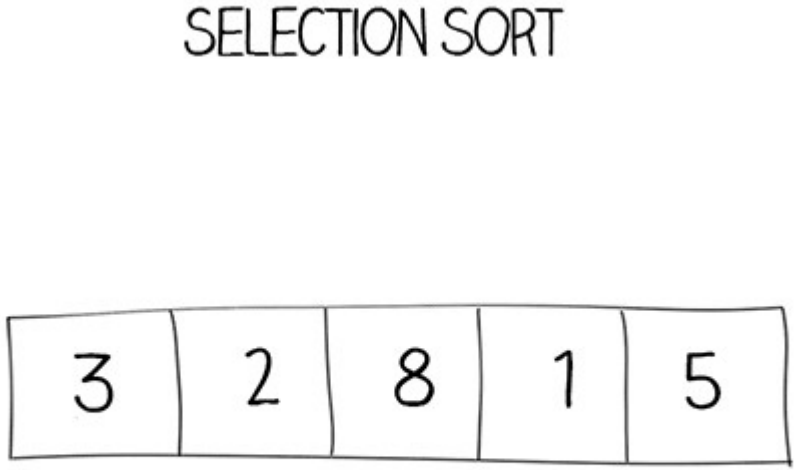


Lets Code !!!

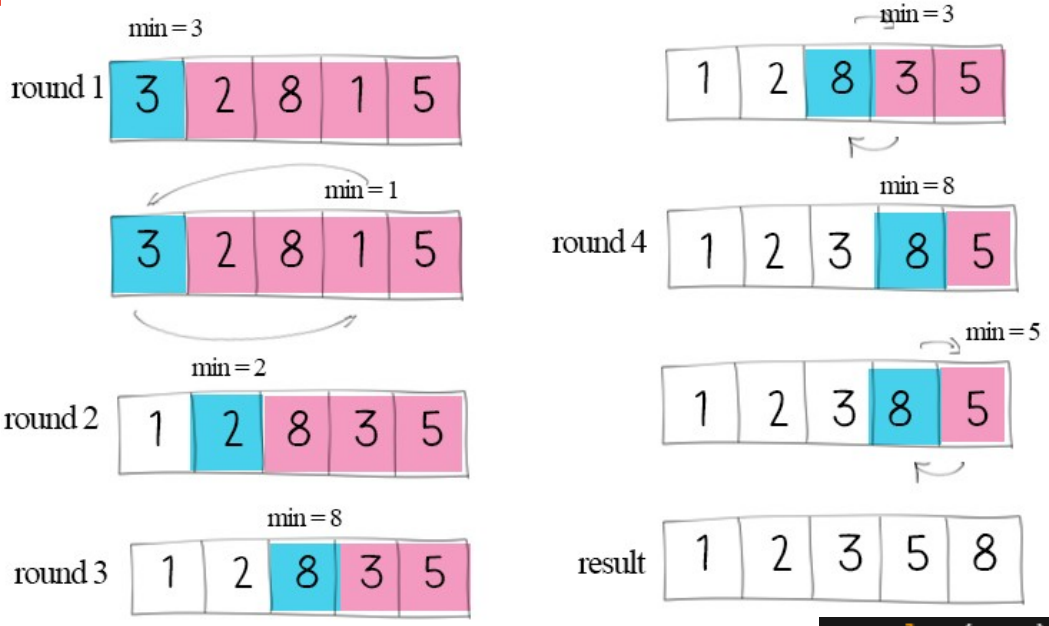
```python
n = len(arr)
swap_count = 0
for i in range(n):
    swapped = False
    for j in range(0, n - i - 1):
        if arr[j] > arr[j + 1]:
            arr[j], arr[j + 1] = arr[j + 1], arr[j]
            swapped = True
            swap_count += 1
    if not swapped:
        break
```

Ordenação – Selection Sort

- **Percorre o vetor iterativamente, selecionando em cada iteração o menor (ou maior, para ordem decrescente) elemento da sublista não ordenada e trocando-o com o elemento na posição inicial dessa sublista, garantindo a ordenação progressiva do vetor até que todos os elementos estejam posicionados corretamente.**



Ordenação – Selection Sort

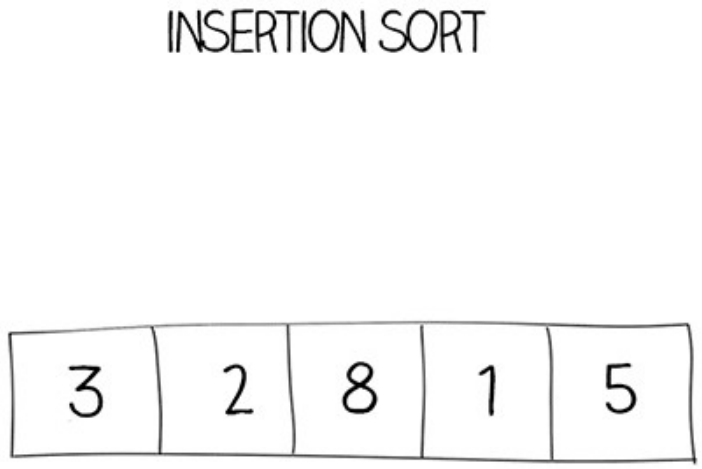



```python
n = len(arr)
for i in range(n):
    min_idx = i
    # Encontrar o índice do menor elemento na sublista arr[i:n]
    for j in range(i + 1, n):
        if arr[j] < arr[min_idx]:
            min_idx = j
    # Troca o elemento atual com o menor encontrado
    arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

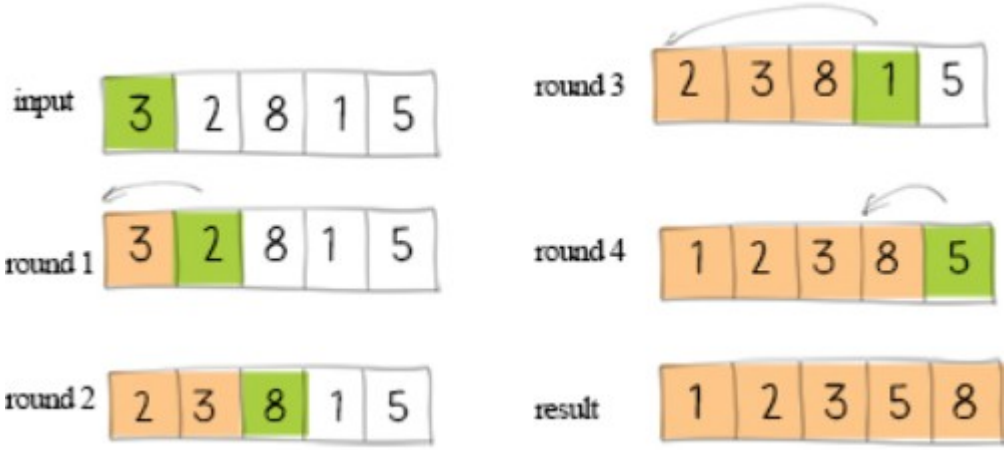
Lets Code !!!

Ordenação – Insertion Sort

- **Percorre iterativamente o vetor a partir do segundo elemento, comparando-o com os elementos anteriores e deslocando-os para a direita até encontrar a posição correta para inserir o elemento atual, garantindo assim a ordenação incremental (crescente ou decrescente) do vetor até que todos os elementos estejam ordenados.**



Ordenação – Insertion Sort



Lets Code !!!

```python
n = len(arr)
for i in range(1, n):
    key = arr[i]
    j = i - 1
    # Desloca elementos maiores (para ordenação crescente) à direita
    while j >= 0 and arr[j] > key:
        arr[j + 1] = arr[j]
        j -= 1
    arr[j + 1] = key
```

Ordenação

- **Outros métodos:**
  - Merge Sort*
  - Quick Sort*
  - Shell Sort*
  - Heap Sort*

- (*) Abordaremos esses algoritmos posteriormente, no contexto de recursão.

Tópico 2: Recursão

Recursão

- Funções que invocam a si mesma (laço)
  - Critério de Parada
  - Incremento ou Decremento

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print(i)
    print_rec(i-1) #increment/decrement
```

Recursão

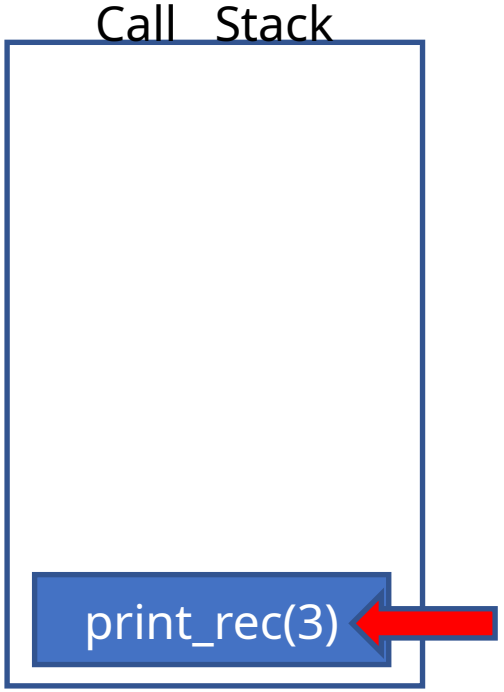
```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print(i)
    print_rec(i-1) #increment/decrement
```

print('main')
print_rec(3)

print('voltei main')

Recursão

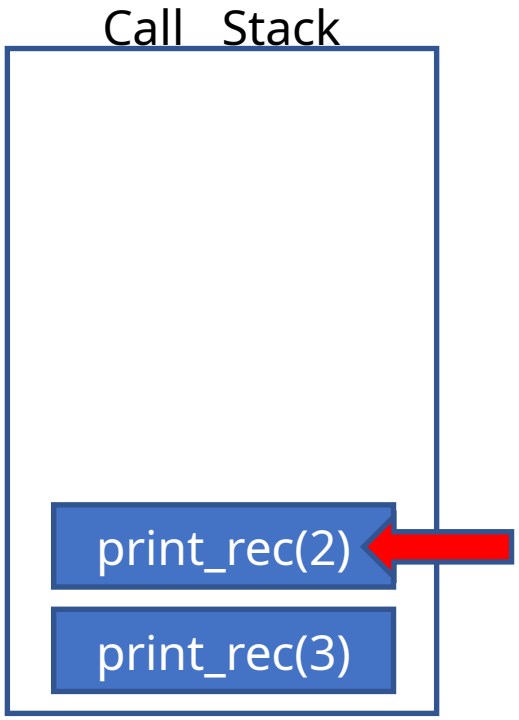
```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```



Recursão

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print(2)
    print_rec(1) #increment/decrement
```

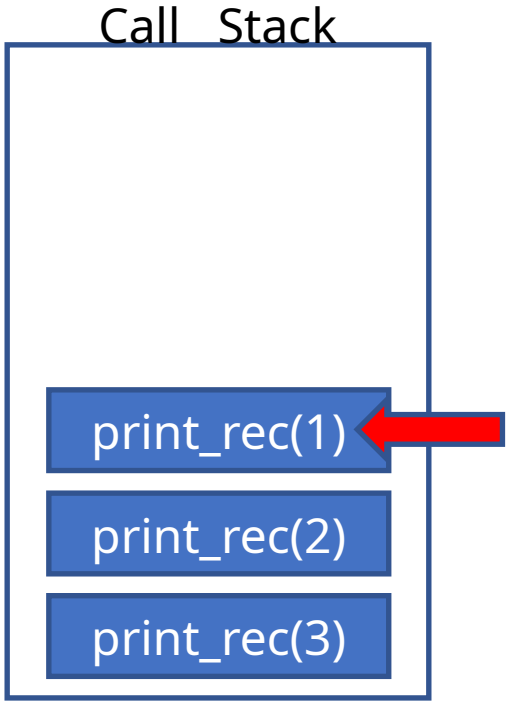


Recursão

```python
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

```python
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print(2)
    print_rec(1) #increment/decrement
```

```python
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print(1)
    print_rec(0) #increment/decrement
```



Recursão

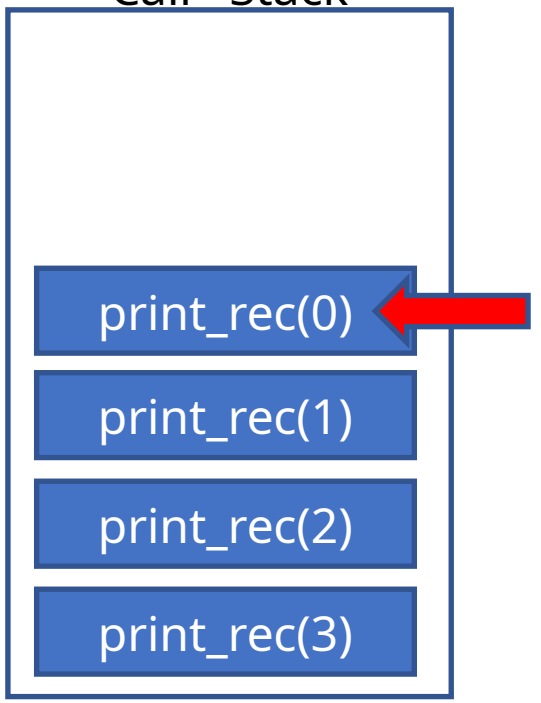
```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print(2)
    print_rec(1) #increment/decrement
```

```
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print(1)
    print_rec(0) #increment/decrement
```

```
def print_rec(0):

    if (0<=0): #Stop Criteria
        return
```

Call Stack



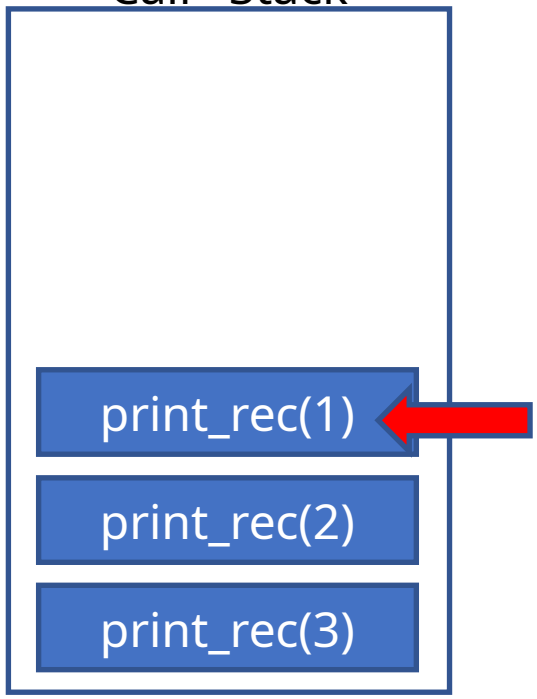
Recursão

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print(2)
    print_rec(1) #increment/decrement
```

```
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print(1)
    print_rec(0) #increment/decrement
```

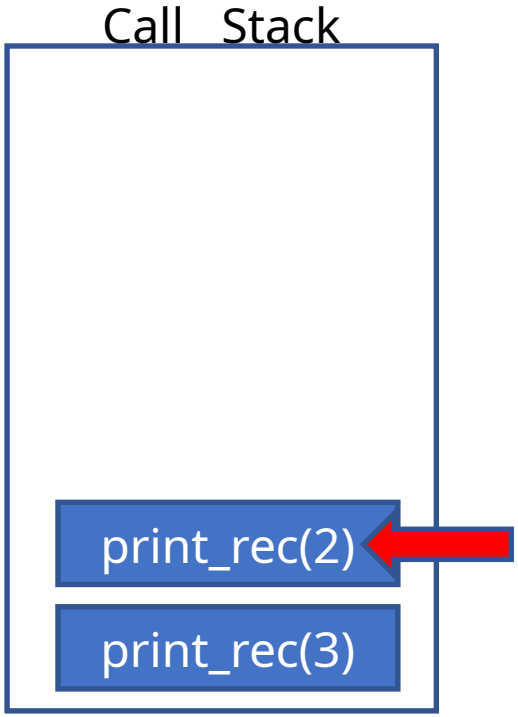
Call Stack



Recursão

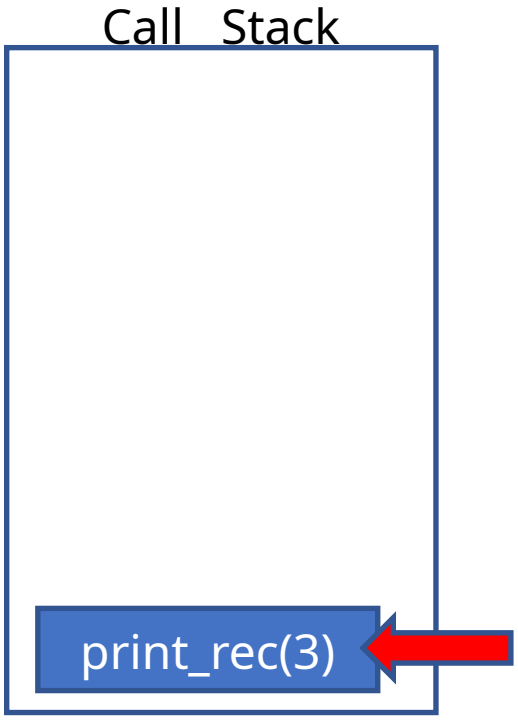
```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print(2)
    print_rec(1) #increment/decrement
```



Recursão

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```



Recursão

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print(i)
    print_rec(i-1) #increment/decrement

print('main')
print_rec(3)
print('voltei main')
```

Recursão

- Qual a diferença entre as duas funções abaixo:

```python
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print(i)
    print_rec(i-1) #increment/decrement
```

```python
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print_rec(i-1) #increment/decrement
    print(i)
```

Recursão

- Qual a diferença entre as duas funções abaixo:

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print(i)
    print_rec(i-1) #increment/decrement
```

print(i) executa no empilhamento

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print_rec(i-1) #increment/decrement
    print(i)
```

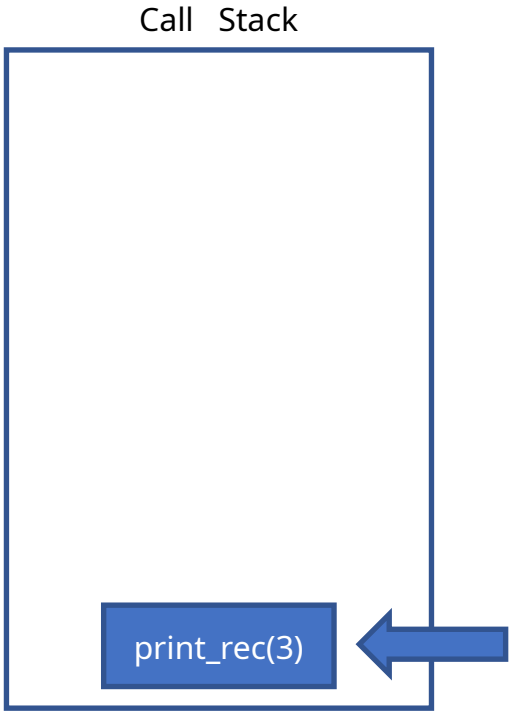
print(i) executa no desempilhamento

Recursão

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print_rec(i-1) #increment/decrement
    print(i)


print_rec(3)
```

Recursão

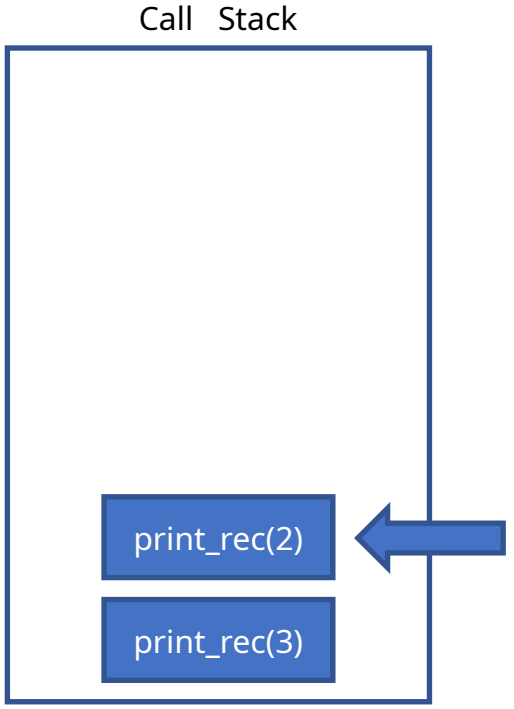
```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```



Recursão

```python
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

```python
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print_rec(1) #increment/decrement
    print(2)
```

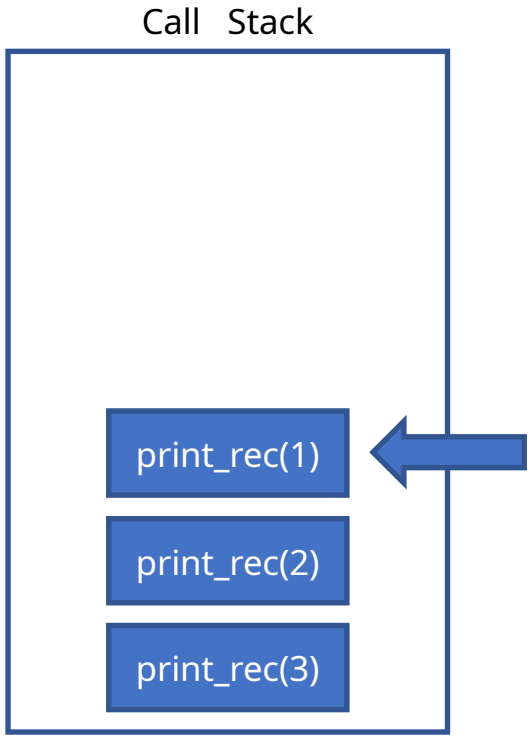


Recursão

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print_rec(1) #increment/decrement
    print(2)
```

```
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print_rec(0) #increment/decrement
    print(1)
```



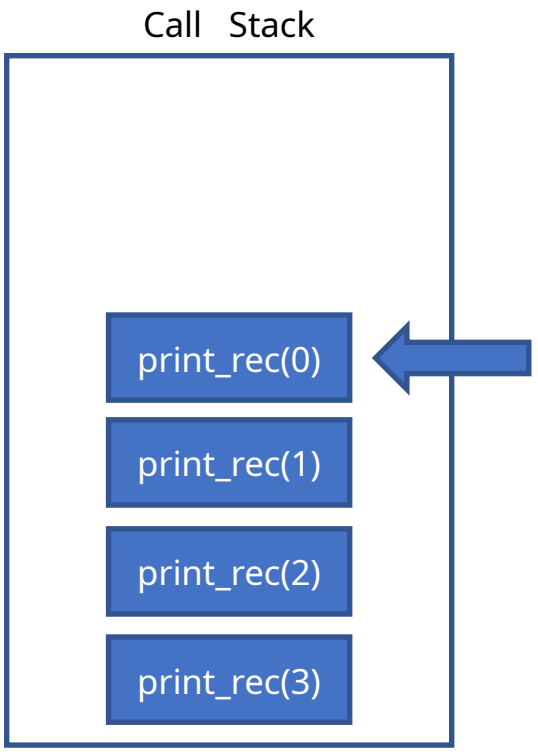
Recursão

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print_rec(1) #increment/decrement
    print(2)
```

```
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print_rec(0) #increment/decrement
    print(1)
```

```
def print_rec(0):

    if (0<=0): #Stop Criteria
        return
```

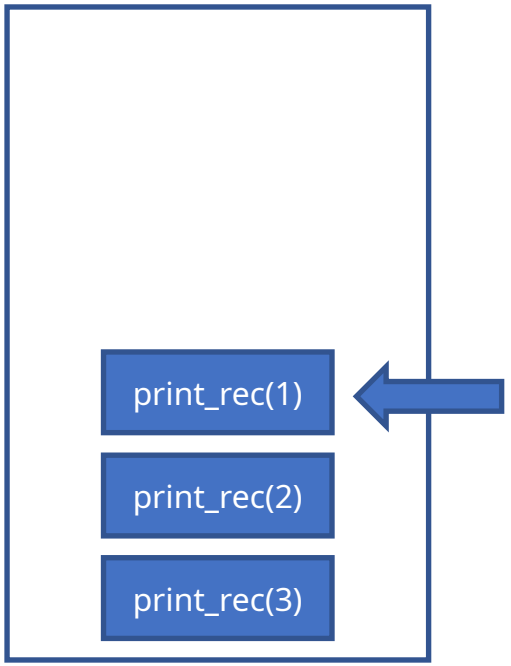


Recursão

```python
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

```python
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print_rec(1) #increment/decrement
    print(2)
```

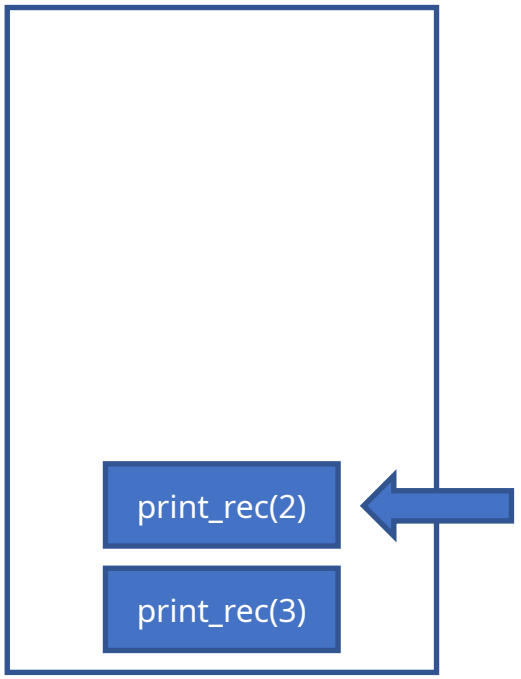
```python
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print_rec(0) #increment/decrement
    print(1)
```



Recursão

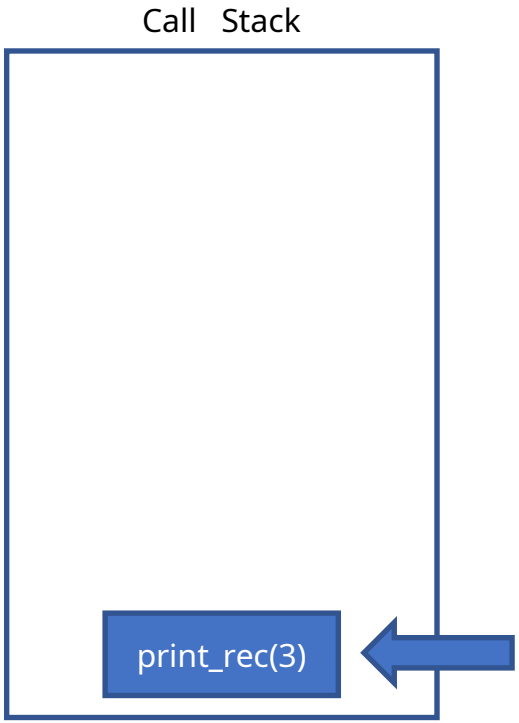
```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print_rec(1) #increment/decrement
    print(2)
```



Recursão

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```



Recursão

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print_rec(i-1) #increment/decrement
    print(i)


print_rec(3)
```

Recursão - Exercícios

- Implemente uma soma recursiva de 0....N
  - N = 3 | Soma = 3 + 2 + 1 ou 1 + 2 + 3
  - N = 3 | Soma = 5 + 4 + 3 + 2 + 1 ou 1 + 2 + 3 + 4 + 5

- Implemente o Fibonacci Iterativo

Recursão vs Iteração

- Cada situação demanda uma abordagem

- Via de regra, a recursão apresenta *overhead* em relação a iterativa
  - No entanto, veremos que em algumas situações a aplicação de recursão é sugestiva

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print_rec(i-1) #increment/decrement
    print(i)
```

```
def print_iter(i):

    for n in range(i)
        print(n)
```