AI VIETNAM
AI Course 2023

Interpolation and Its Applications (Exercises)

Vinh Dinh Nguyen
PhD in Computer Science

Year 2023


WHAT
HAVE YOU
LEARNED

❖ Image Processing

Grayscale Image

(Height, Width)

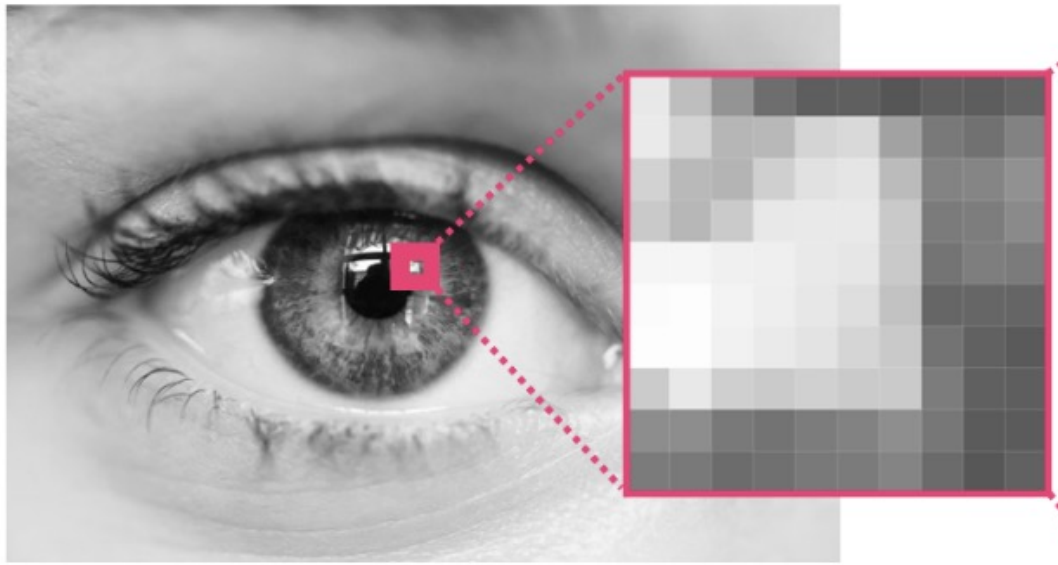
Color Image

(Height, Width, Channel)

Image Data

❖ Grayscale images

Height



Width

230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97

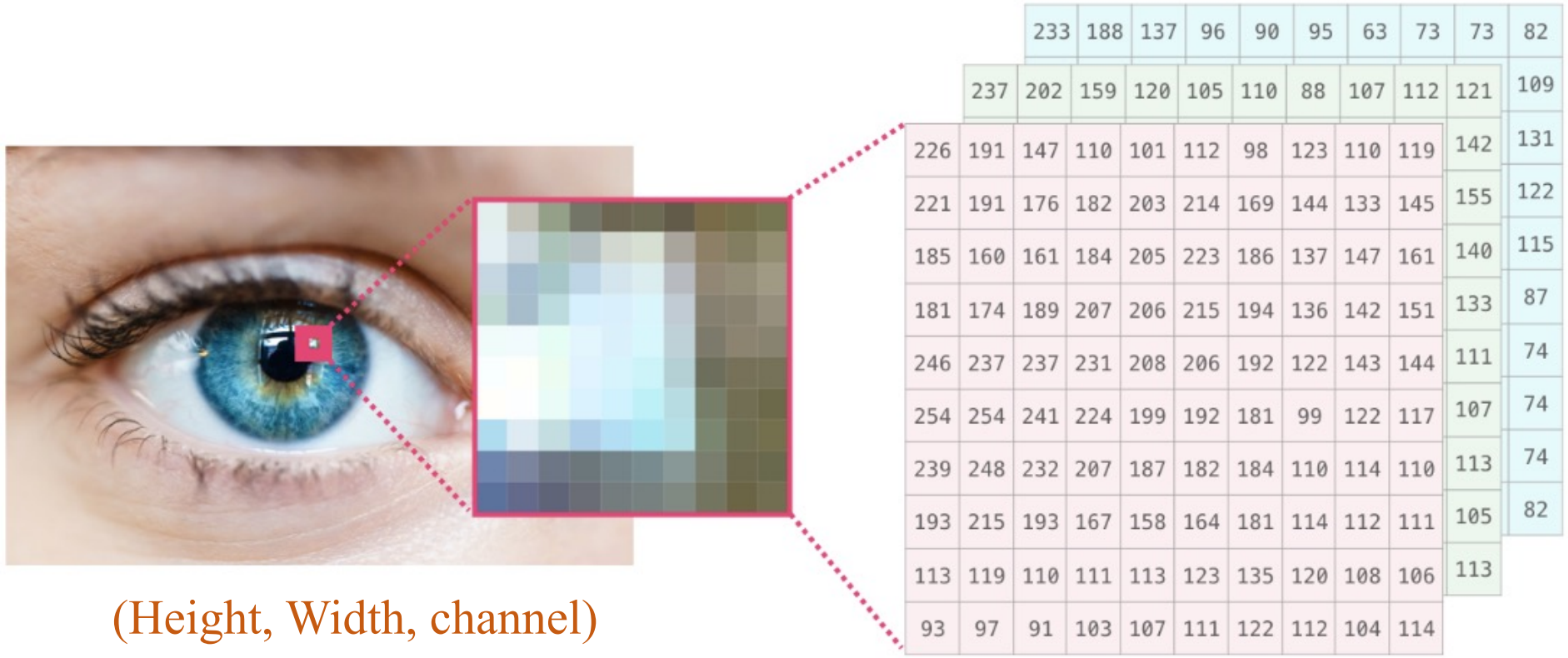
Pixel p = scalar

$0 \leq p \leq 255$

Resolution: #pixels
Resolution = HeightxWidth

Image Data

❖ Color images



(Height, Width, channel)

RGB color image

Pixel $p = \begin{bmatrix} r \\ g \\ b \end{bmatrix}$

$0 \leq r,g,b \leq 255$

Resolution: #pixels
Resolution = HeightxWidth

Read Image by OpenCV

❖ Load an image

```python
import cv2

# read a grayscale image
img = cv2.imread('nature.jpg', 0)

# show the image
cv2.imshow('img', img)

# copy them when showing an image
cv2.waitKey()
cv2.destroyAllWindows()
```

```python
import cv2

# read a grayscale image
img = cv2.imread('nature.jpg', 0)

# save the image
cv2.imwrite('processed_image.jpg', img)
```

```python
# get image info

import numpy as np
import cv2

# read a grayscale image
img = cv2.imread('nature.jpg', 0)

shape = img.shape
print(shape)
```

(500, 1200)

(Height, Width)

Read Image by OpenCV

❖ Load an image

```python
import cv2

# read a color image
img = cv2.imread('nature.jpg', 1)

# show the image
cv2.imshow("img", img)

# copy them when showing an image
cv2.waitKey()
cv2.destroyAllWindows()
```

```python
import cv2

# read a color image
img = cv2.imread('nature.jpg', 1)

# save the image
cv2.imwrite('processed_image.jpg', img)
```

```python
# get image info

import numpy as np
import cv2

# read a grayscale image
img = cv2.imread('nature.jpg', 1)

shape = img.shape
print(shape)
```

(500, 1200, 3)

(Height, Width, Channel)

Image Resize

❖ How ???

Copy and interpolation

Just using simple ideas

Mapping (scale is an integer)

(500, 1200)

(1000, 2400)

Image Resize

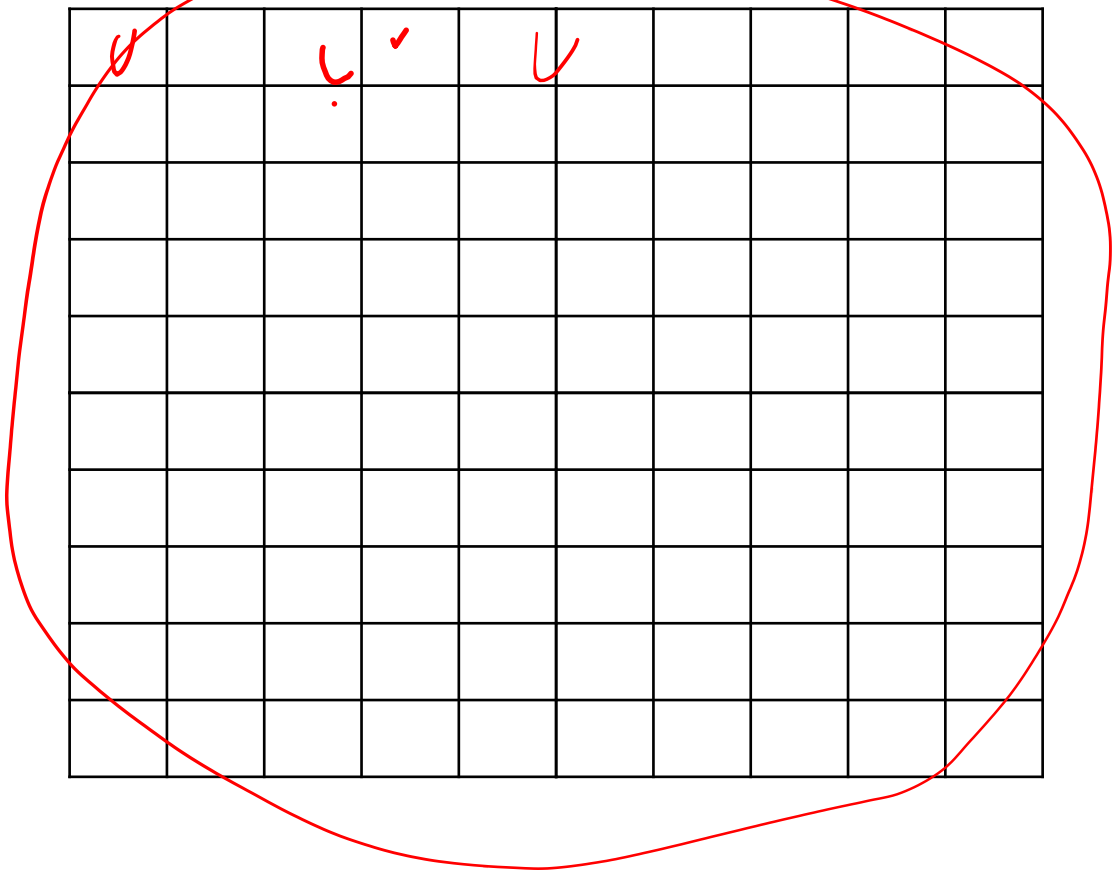
❖ **How ???** **Mapping (scale is an integer)**

Old resolution: (5, 5)

0	2	4	6	8
1	3	5	7	9
0	2	4	6	8
1	3	5	7	9
0	2	4	6	8

Copy and interpolation

new resolution: (10, 10)



Just using simple ideas

AI VIETNAM
AI Course 2022

Image Resize

❖ How ???

Mapping (scale is an integer)

Old resolution: (5, 5)

0	2	4	6	8
1	3	5	7	9
0	2	4	6	8
1	3	5	7	9
0	2	4	6	8

Copy and interpolation

new resolution: (10, 10)

0		2		4		6		8	
1		3		5		7		9	
0		2		4		6		8	
1		3		5		7		9	
0		2		4		6		8	

Just using simple ideas

How to fill in value in:

0		2

Missing Data

Formatted Date	Temperature (C)
2006-01-01 00:00:00.000 +0100	0.577778
2006-01-01 01:00:00.000 +0100	1.161111
2006-01-01 02:00:00.000 +0100	1.666667
2006-01-01 03:00:00.000 +0100	1.711111
2006-01-01 04:00:00.000 +0100	1.183333
2006-01-01 05:00:00.000 +0100	1.205556
2006-01-01 06:00:00.000 +0100	2.222222
2006-01-01 07:00:00.000 +0100	2.072222
2006-01-01 08:00:00.000 +0100	2.2
2006-01-01 09:00:00.000 +0100	NaN
2006-01-01 10:00:00.000 +0100	2.788889
2006-01-01 11:00:00.000 +0100	NaN
2006-01-01 12:00:00.000 +0100	4.911111
2006-01-01 13:00:00.000 +0100	6.205556
2006-01-01 14:00:00.000 +0100	NaN

Fill data

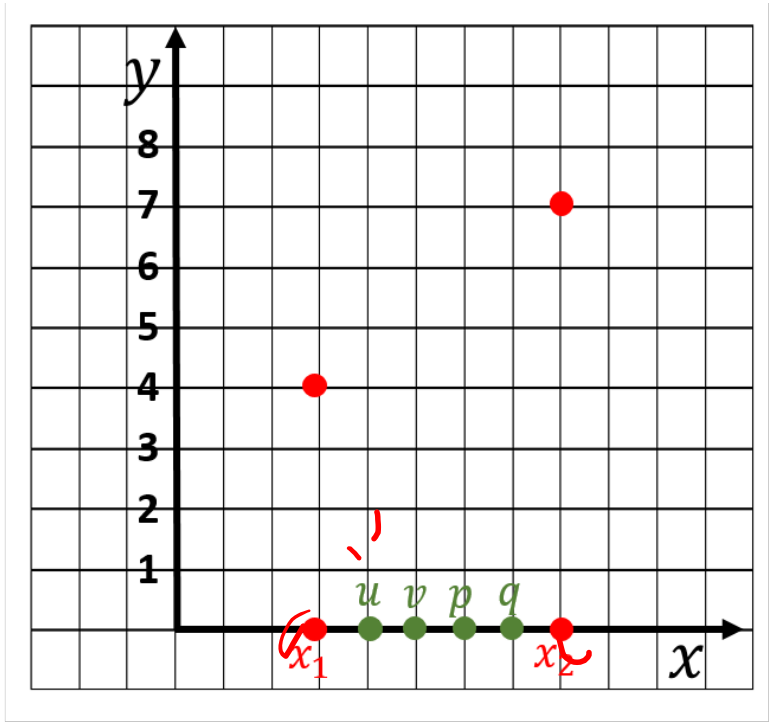
Formatted Date	Temperature (C)
2006-01-01 00:00:00.000 +0100	0.577778
2006-01-01 01:00:00.000 +0100	1.161111
2006-01-01 02:00:00.000 +0100	1.666667
2006-01-01 03:00:00.000 +0100	1.711111
2006-01-01 04:00:00.000 +0100	1.183333
2006-01-01 05:00:00.000 +0100	1.205556
2006-01-01 06:00:00.000 +0100	2.222222
2006-01-01 07:00:00.000 +0100	2.072222
2006-01-01 08:00:00.000 +0100	2.2
2006-01-01 09:00:00.000 +0100	???
2006-01-01 10:00:00.000 +0100	2.788889
2006-01-01 11:00:00.000 +0100	???
2006-01-01 12:00:00.000 +0100	4.911111
2006-01-01 13:00:00.000 +0100	6.205556
2006-01-01 14:00:00.000 +0100	???

How to fill in value in **NaN**

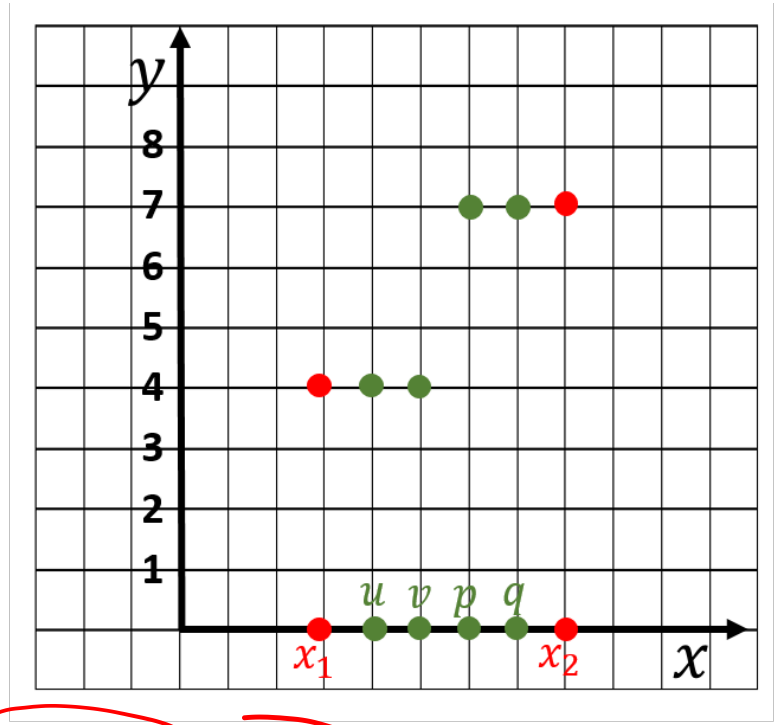
2.2
NaN
2.788889

Interpolation

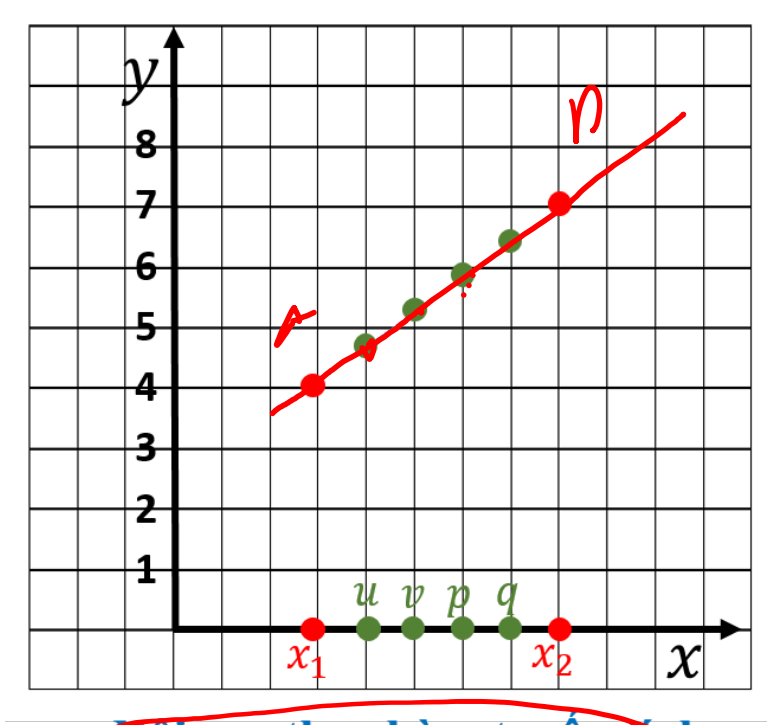
❖ Two simplest techniques



Tìm giá trị cho các vị trí u, v, p và q

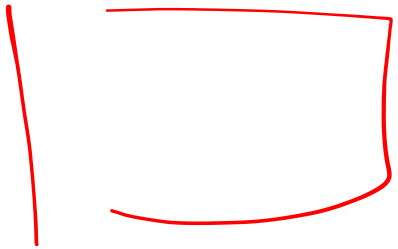


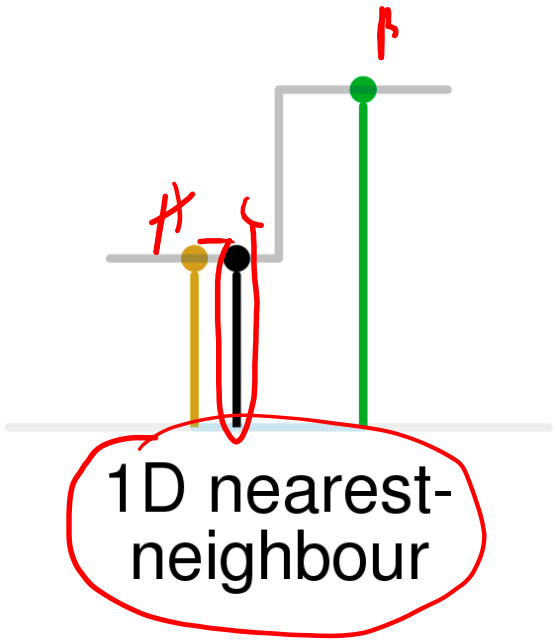
Nearest neighbor: Tính khoảng cách đến x_1 và x_2, và lấy giá trị của x gần hơn

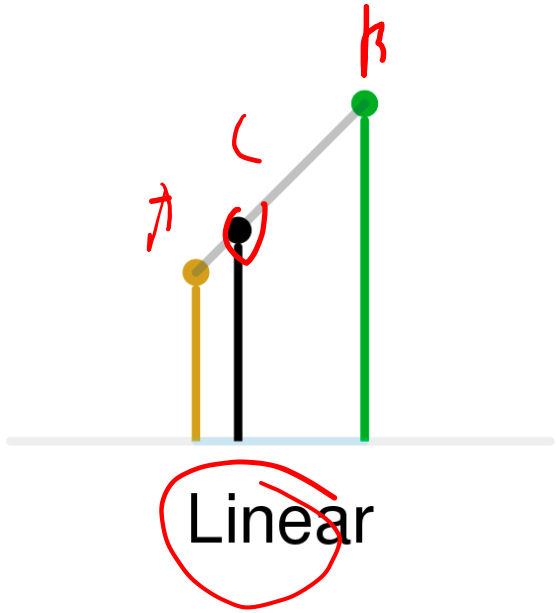


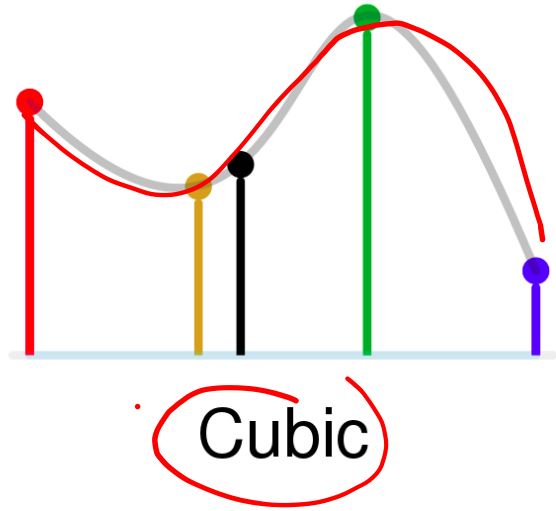
Nội suy theo hàm tuyến tính

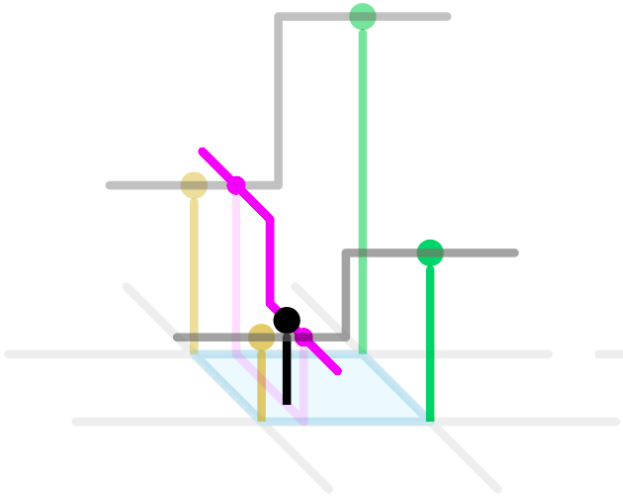
Interpolation

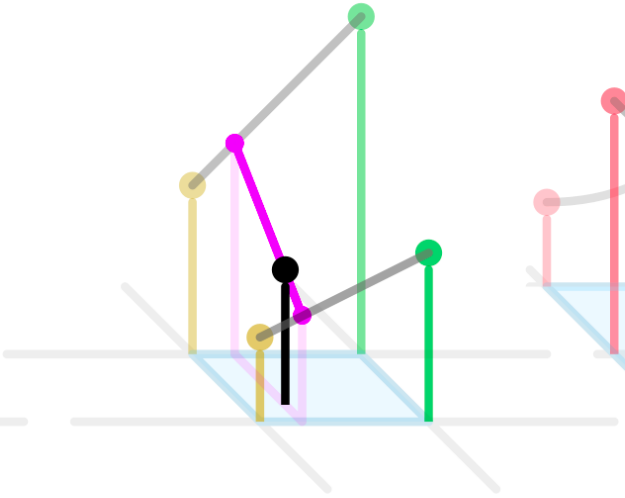


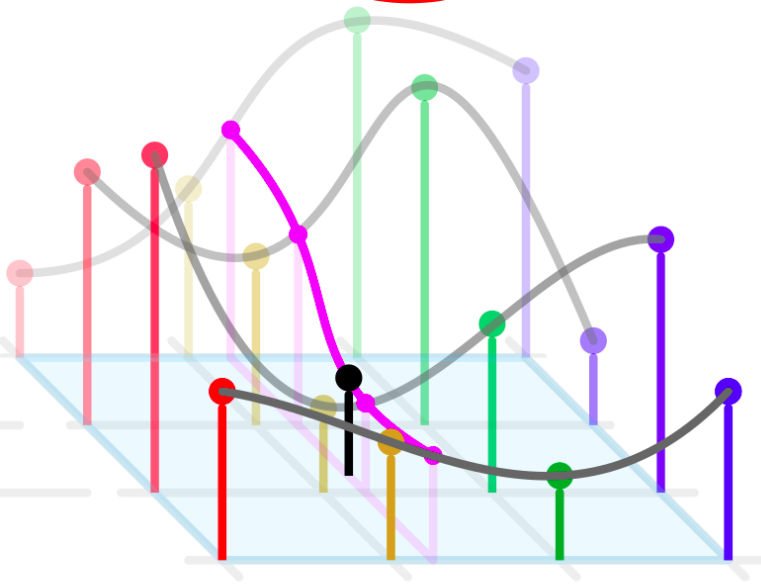






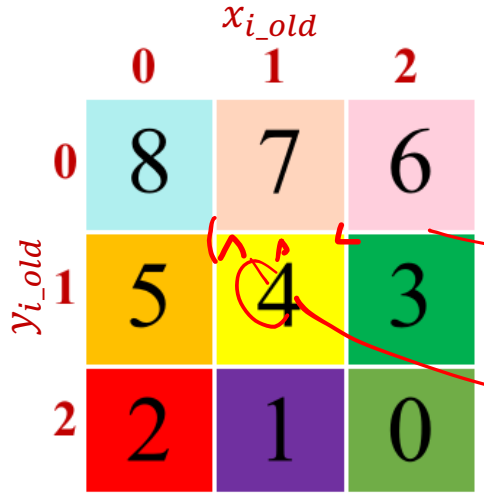




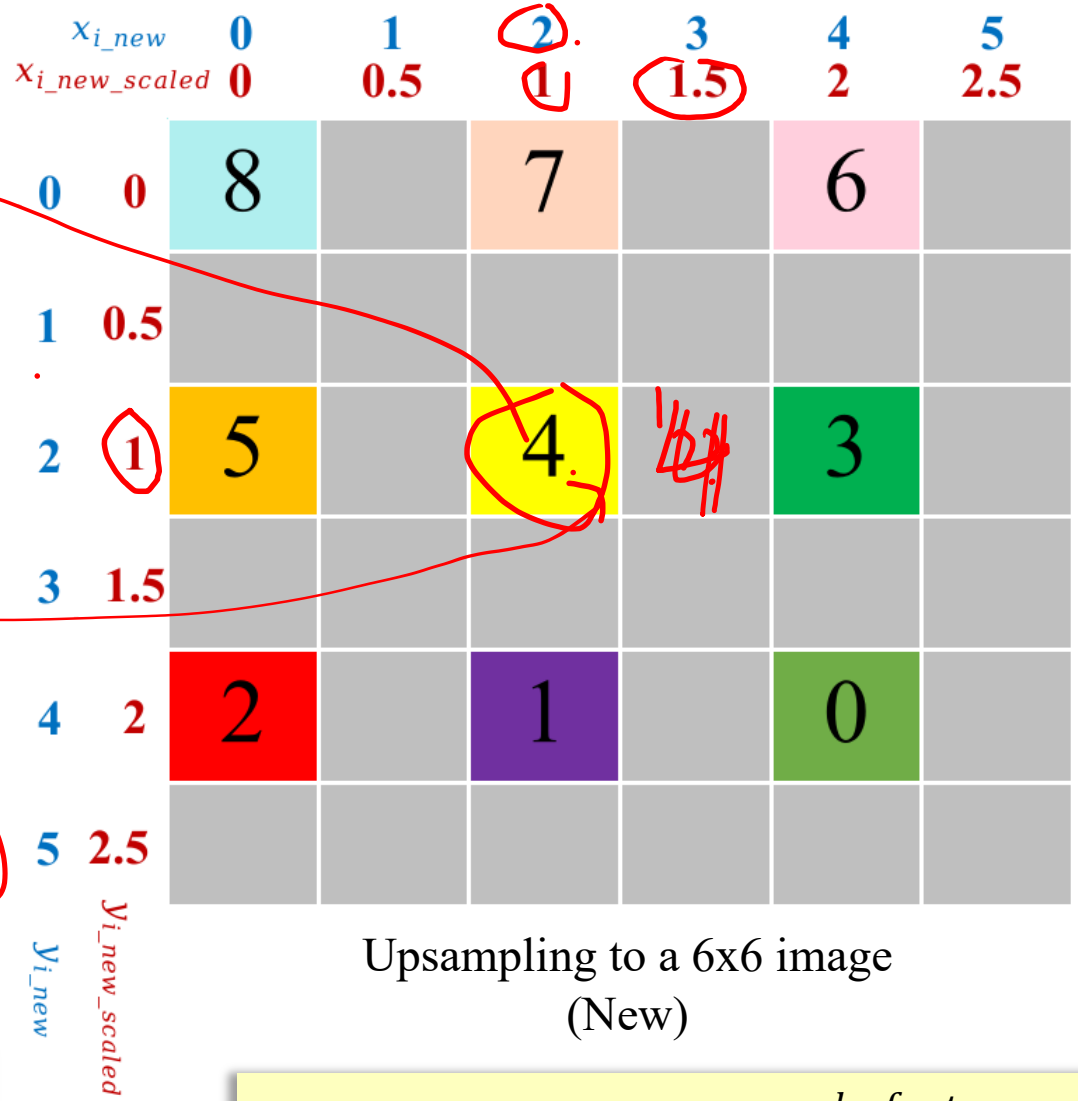


https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation

Nearest-neighbor interpolation



A 3x3 Image
(Old)



Upsampling to a 6x6 image
(New)

old_w: $(width\ of\ Old)\ =\ 3$

old_h: $(hieght\ of\ Old)\ =\ 3$

new_w: $(width\ of\ New)\ =\ 6$

new_h: $(height\ of\ New)\ =\ 6$

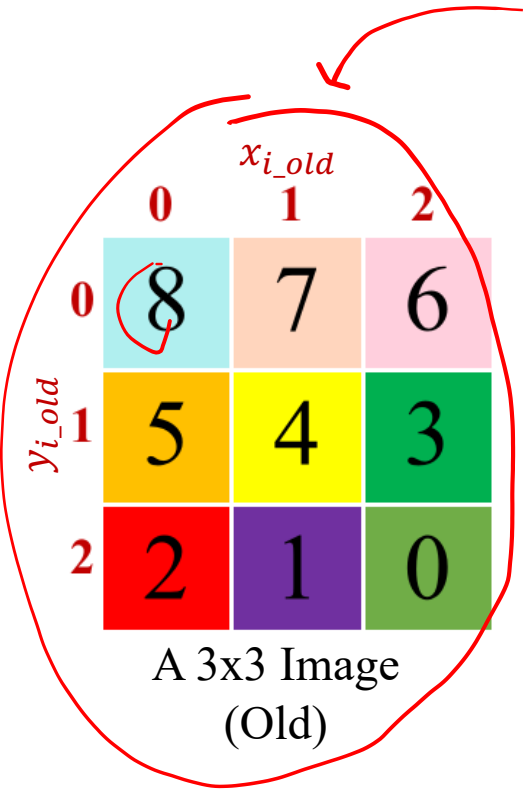
$$w\_scale\_factor = \frac{old\_w}{new\_w}$$

$$h\_scale\_factor = \frac{old\_h}{new\_h}$$

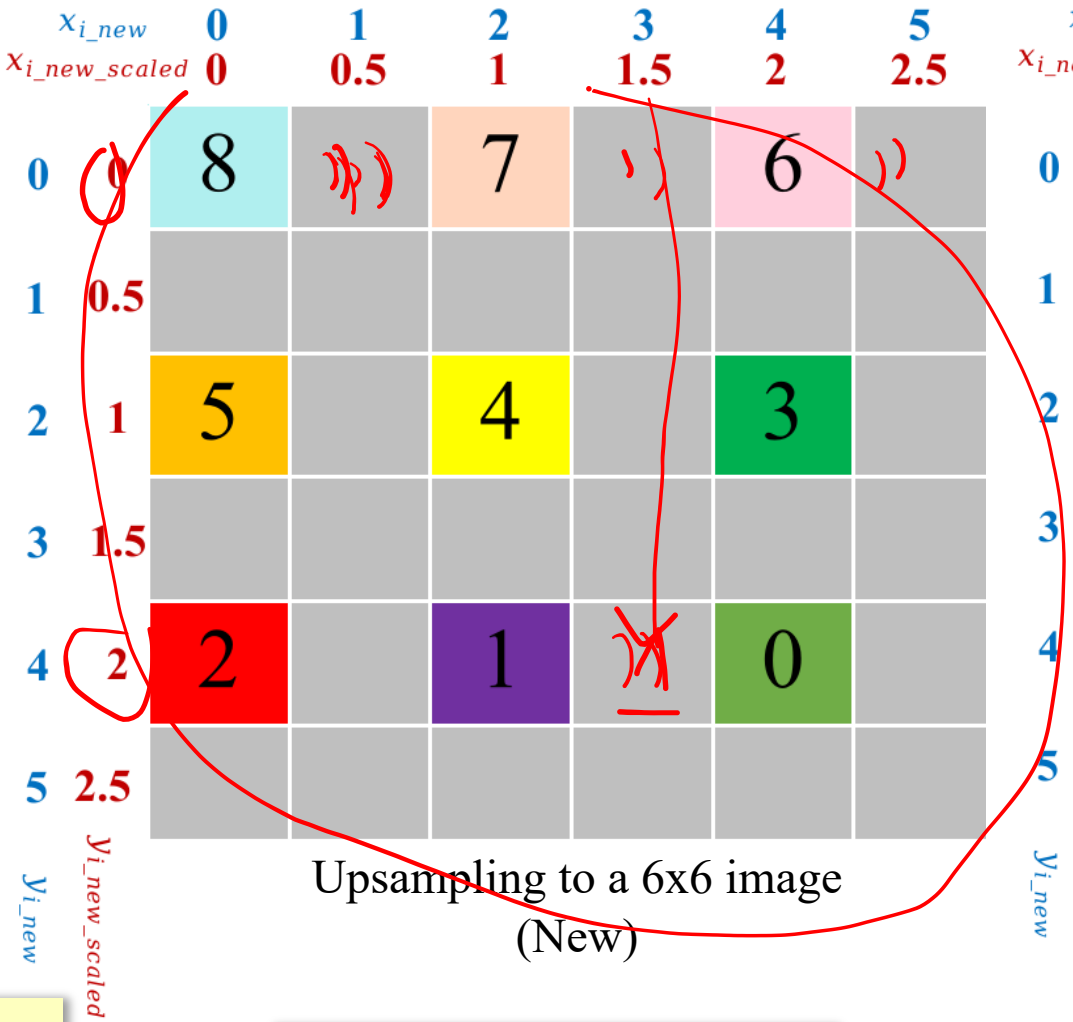
$$x_{i\_new\_scaled} = x_{i\_new} * w\_scale\_factor$$

$$y_{i\_new\_scaled} = y_{i\_new} * h\_scale\_factor$$

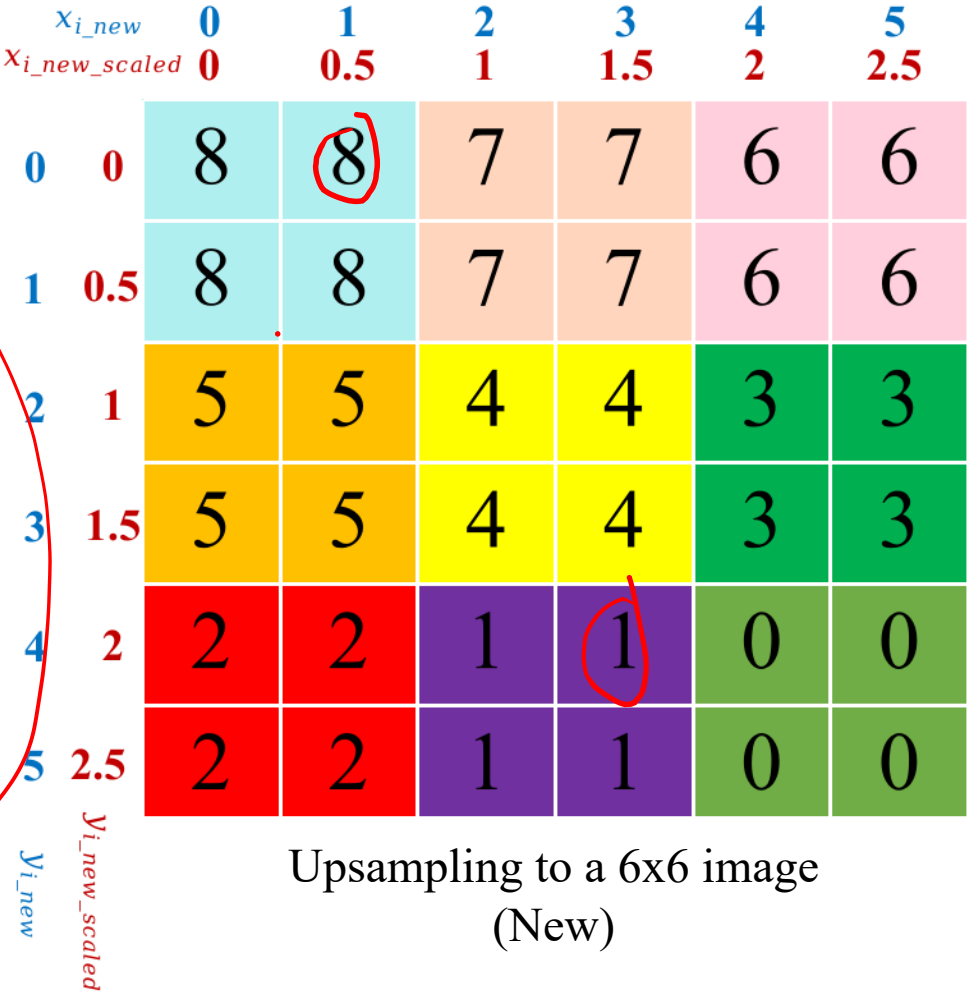
Nearest-neighbor interpolation



A 3x3 Image
(Old)



Upsampling to a 6x6 image
(New)



Upsampling to a 6x6 image
(New)

old_w: $(width\ of\ Old) = 3$

old_h: $(hieght\ of\ Old) = 3$

new_w: $(width\ of\ New) = 6$

new_h: $(height\ of\ New) = 6$

$$w\_scale\_factor = \frac{old\_w}{new\_w}$$

$$h\_scale\_factor = \frac{old\_h}{new\_h}$$

$$x_{i\_new\_scaled} = x_{i\_new} * w\_scale\_factor$$

$$y_{i\_new\_scaled} = y_{i\_new} * h\_scale\_factor$$

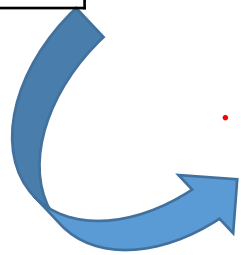
Nearest-neighbor interpolation

Mapping (scale is an integer)

Old resolution: (5, 5)

0	2	4	6	8
1	3	5	7	9
0	2	4	6	8
1	3	5	7	9
0	2	4	6	8

Copy and interpolation



New resolution: (10, 10)

0		2		4		6		8	
1		3		5		7		9	
0		2		4		6		8	
1		3		5		7		9	
0		2		4		6		8	

Just using simple ideas

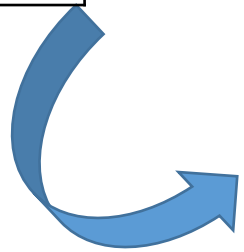
Nearest-neighbor interpolation

Mapping (scale is an integer)

Old resolution: (5, 5)

0	2	4	6	8
1	3	5	7	9
0	2	4	6	8
1	3	5	7	9
0	2	4	6	8

Copy and interpolation



New resolution: (10, 10)

0	0	2	2	4	4	6	6	8	8
0	0	2	2	4	4	6	6	8	8
1	1	3	3	5	5	7	7	9	9
1	1	3	3	5	5	7	7	9	9
0	0	2	2	4	4	6	6	8	8
0	0	2	2	4	4	6	6	8	8
1	1	3	3	5	5	7	7	9	9
1	1	3	3	5	5	7	7	9	9
0	0	2	2	4	4	6	6	8	8
0	0	2	2	4	4	6	6	8	8

Just using simple ideas

Nearest-neighbor interpolation

Mapping (scale is an integer)

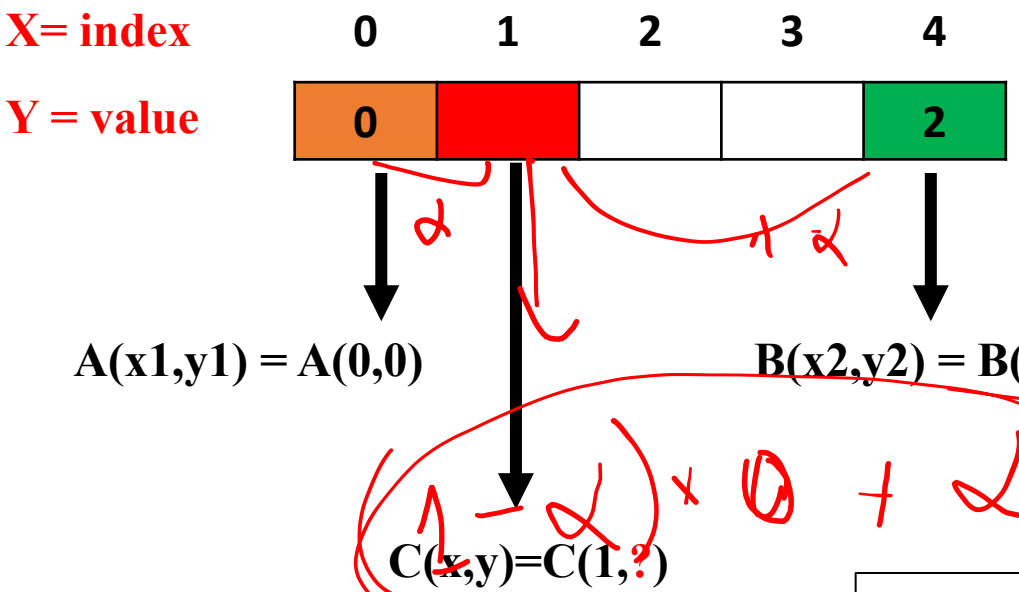
(3, 3)

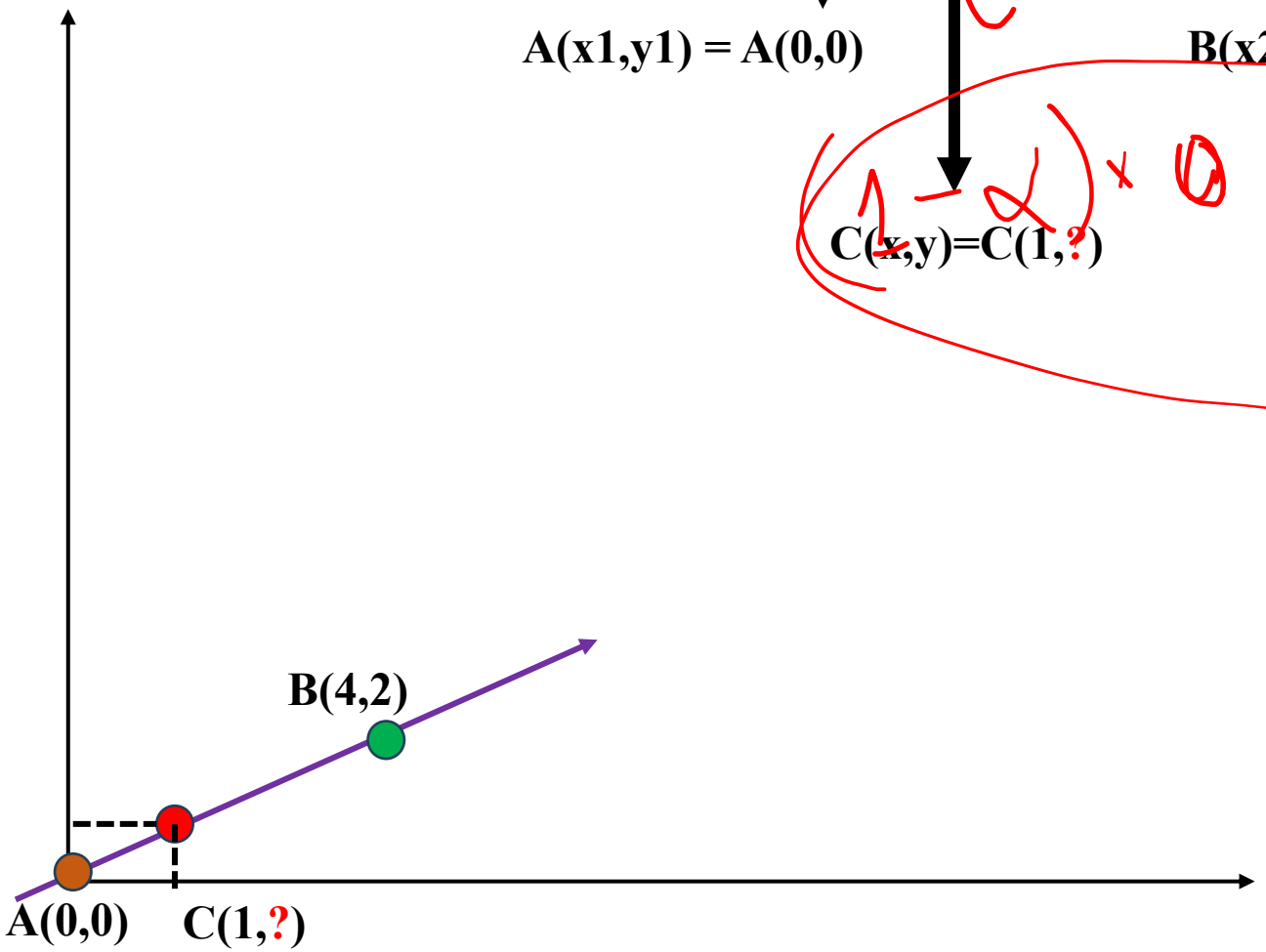
0	2	4
1	3	5
0	2	4

Copy and interpolation

(12, 12)

0				2				4			
1				3				5			
0				2				4			





Linear Interpolation Formula

$$\text{Linear Interpolation Formula (y)} = y_1 + \frac{(x-x_1)(y_2-y_1)}{x_2-x_1}$$

where,

- x_1 and y_1 are the first coordinates
- x_2 and y_2 are the second coordinates
- x is the point to perform the interpolation
- y is the interpolated value

Given $A(x_1, y_1)$ and $B(x_2, y_2)$

Line equation: $y = ax + b$ (1)

From A and B, we have $a = \frac{y_2 - y_1}{x_2 - x_1}$ (2)

Replace A into (1), we have $b = y_1 - \frac{y_2 - y_1}{x_2 - x_1} x_1$ (3)

From (1), (2), and (3) $\Rightarrow y = \frac{y_2 - y_1}{x_2 - x_1} x + \left(y_1 - \frac{y_2 - y_1}{x_2 - x_1} x_1 \right)$ (4)

$$(4) \Rightarrow y = \frac{(y_2 - y_1)x}{x_2 - x_1} + \frac{y_1(x_2 - x_1) - x_1(y_2 - y_1)}{x_2 - x_1}$$

$$= \frac{y_2 x - y_1 x + y_1 x_2 - y_1 x_1 - y_2 x_1 + y_1 x_1}{x_2 - x_1}$$

$$= \frac{x_2 - x}{x_2 - x_1} y_1 + \frac{x - x_1}{x_2 - x_1} y_2$$

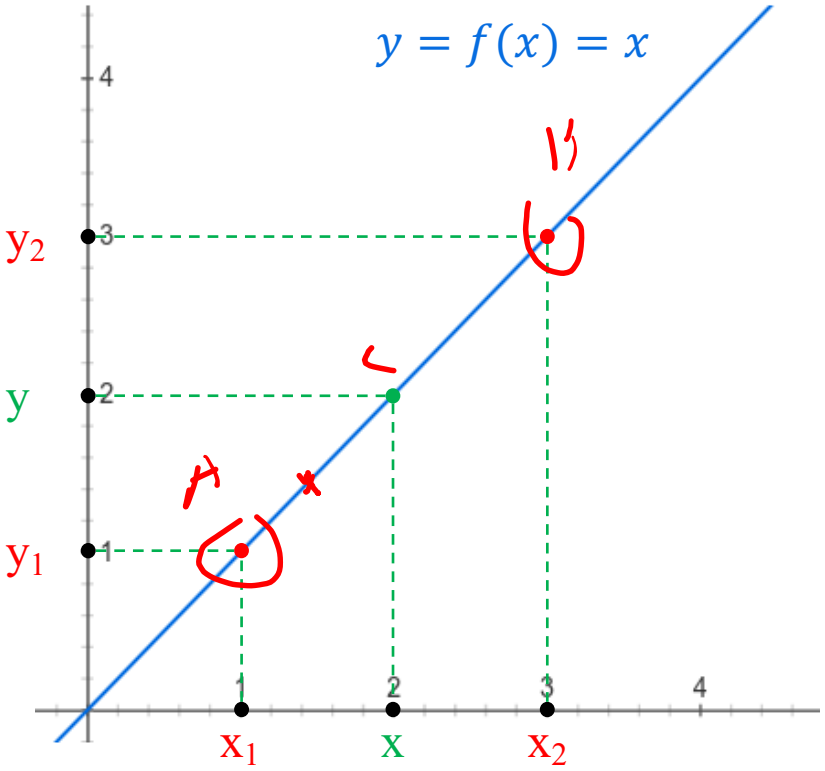
α $\quad$ $1 - \alpha$

$$\frac{x_2 - x}{x_2 - x_1} + \frac{x - x_1}{x_2 - x_1} = 1$$

Line segment equation

$$v = \alpha v_1 + (1 - \alpha) v_2$$

Linear interpolation



Giả xử không biết đường thẳng $y = x$, chỉ biết hai điểm điểm màu đỏ (x_1=1, y_1=1) và (x_2=3, y_2=3). Tìm điểm màu xanh lá có y=? khi biết x=2.

$$y \approx \frac{x_2 - x}{x_2 - x_1} y_1 + \frac{x - x_1}{x_2 - x_1} y_2 = \frac{3-2}{3-1} * 1 + \frac{2-1}{3-1} * 3 = 2$$

Tỉ lệ y phụ thuộc vào y_1 | Tỉ lệ y phụ thuộc vào y_2

(x=1.1, y=?) $y \approx \frac{3-1.1}{3-1} * 1 + \frac{1.1-1}{3-1} * 3 \approx 0.95 * 1 + 0.05 * 3 \approx 1.1$

Càng gần x_1, y phụ thuộc nhiều vào y_1 và ít phụ thuộc y_2

(x=2.9, y=?) $y \approx \frac{3-2.9}{3-1} * 1 + \frac{2.9-1}{3-1} * 3 \approx 0.05 * 1 + 0.95 * 3 \approx 2.9$

Càng gần x_2, y phụ thuộc nhiều vào y_2 và ít phụ thuộc y_1

Bilinear interpolation

Mapping (scale is an integer)

(3, 3)

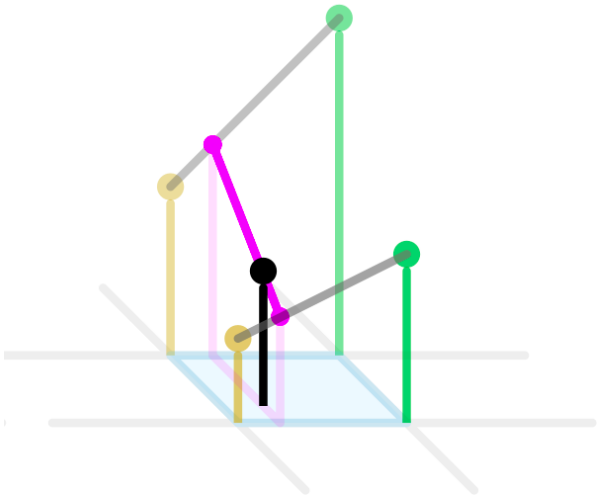
0	2	4
1	3	5
0	2	4

Copy and interpolation

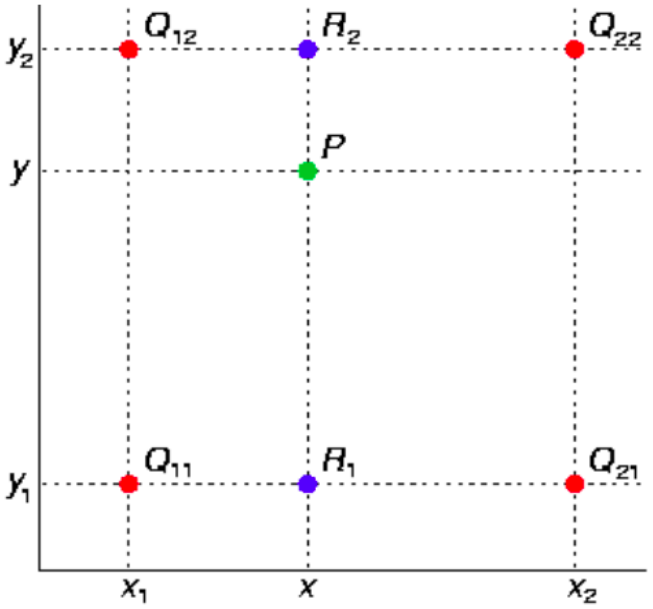
(12, 12)

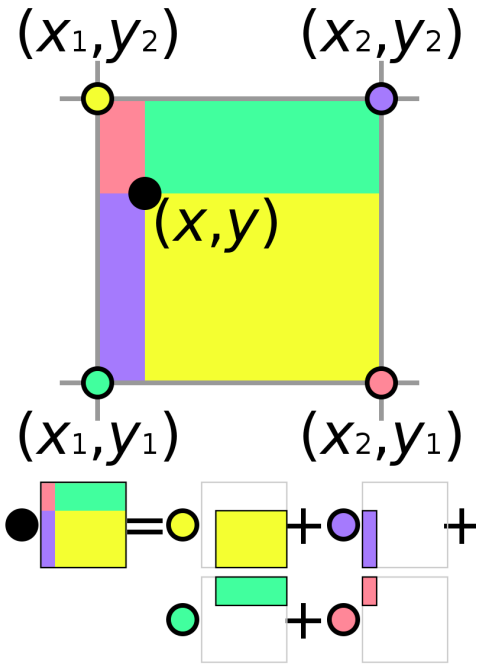
Scaled index	0	1/4	2/4	3/4	4/4	5/4	6/4	7/4	8/4	9/4	10/4	11/4
0	**0**				**2**				**4**			
1/4												
2/4												
3/4												
4/4	**1**				**3**				**5**			
5/4												
6/4												
7/4												
8/4	**0**				**2**				**4**			
9/4												
10/4												
11/4												

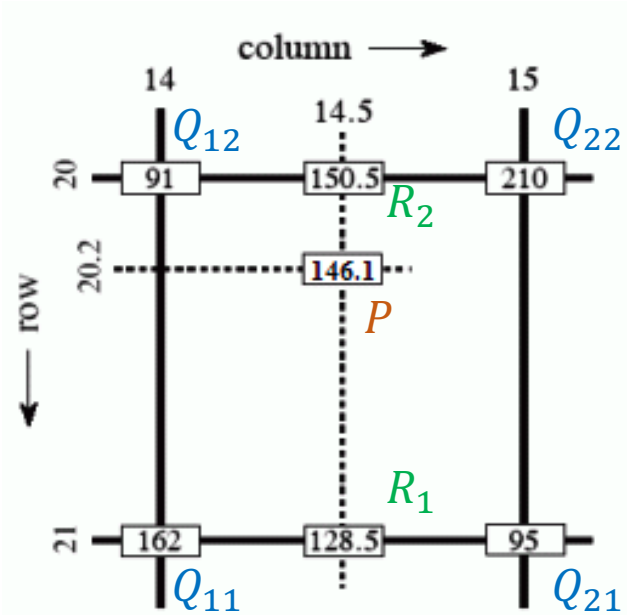
Bilinear Interpolation



https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation







https://en.wikipedia.org/wiki/Bilinear_interpolation

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

$x_1 = 14, x_2 = 15, \quad y_1 = 21, y_2 = 20$

$Q_{11} = (14,21)$, $Q_{21} = (15,21)$

$Q_{12} = (14,20)$, $Q_{22} = (15,20)$

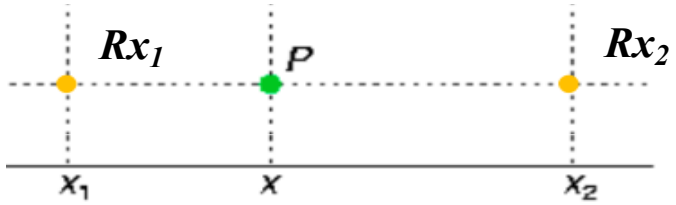
$R_1 = (14.5,21)$, $R_2 = (14.5,20)$, $P = (14.5,20.2)$

$$f(R_1) \approx \frac{15 - 14.5}{15 - 14} 162 + \frac{14.5 - 14}{15 - 14} 95 = 128.5$$

$$f(R_2) \approx \frac{15 - 14.5}{15 - 14} 91 + \frac{14.5 - 14}{15 - 14} 210 = 150.5$$

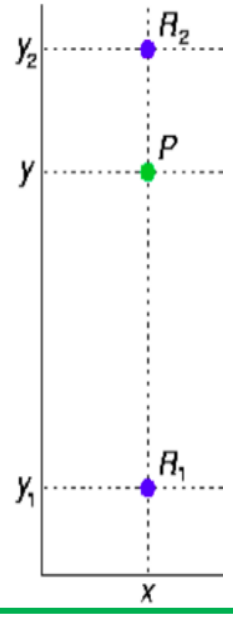
$$f(P) \approx \frac{20 - 20.2}{20 - 21} 128.5 + \frac{20.2 - 21}{20 - 21} 150.5 = 146.1$$

Horizontal Linear Interpolation



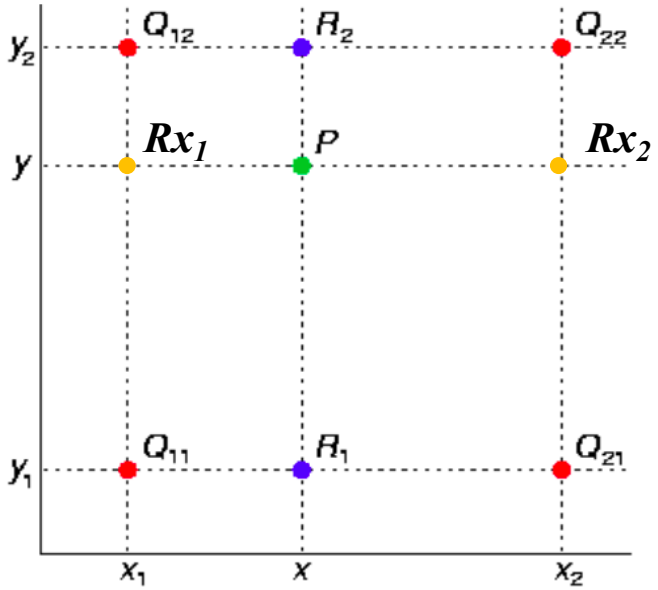
$$f(P) \approx \frac{x_2 - x}{x_2 - x_1} f(Rx_1) + \frac{x - x_1}{x_2 - x_1} f(Rx_2)$$

Vertical Linear Interpolation



$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

Bilinear Interpolation



$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

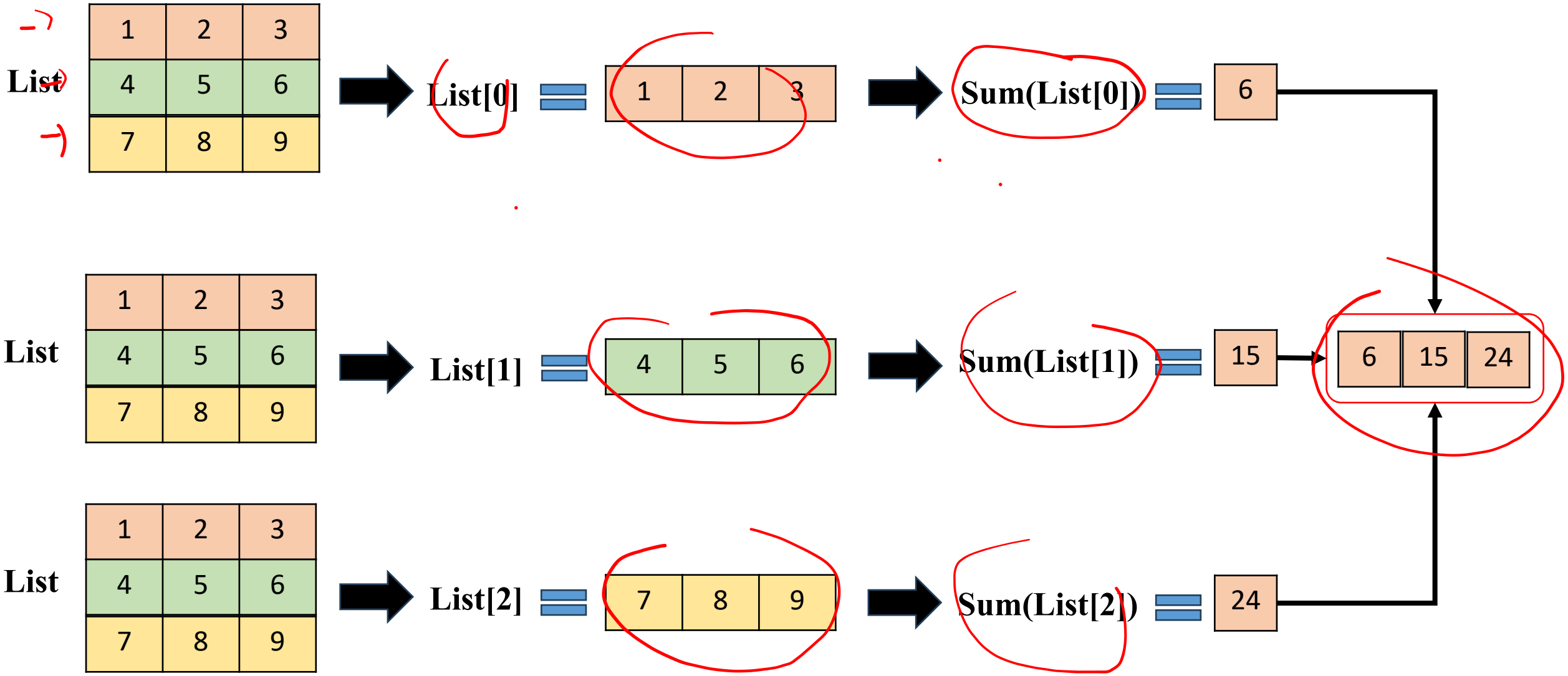

TIME
FOR
ACTION

Problem 1

Câu hỏi 1 (Kỹ thuật thao tác trên dữ liệu danh sách 2 chiều): Cho trước một danh sách các danh sách số tự nhiên được lưu trữ trong danh sách (List) 2 chiều gồm có m dòng và n cột. Phát triển chương trình tính tổng cho mỗi dòng của danh sách. Bên dưới là ví dụ minh hoạ các test case đầu vào và đầu ra của chương trình.

```
Examples
Input:
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]

Output:
result = [6, 15, 24]
--------------------------------
Input:
data = [[1, 2],
        [3, 4],
        [5, 6]]
Output:
[3, 7, 11]
```

Problem Analysis



Solution

```
# implementation
def compute_sum(data):
    result = []

    # your code here

    return result

# test case
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
assert compute_sum(data) == [6, 15, 24]

data = [[1, 2],
        [3, 4],
        [5, 6]]
assert compute_sum(data) == [3, 7, 11]

# multiple choice
data = [[6, 8],
        [3, 1],
        [7, 3]]
print(compute_sum(data))
```

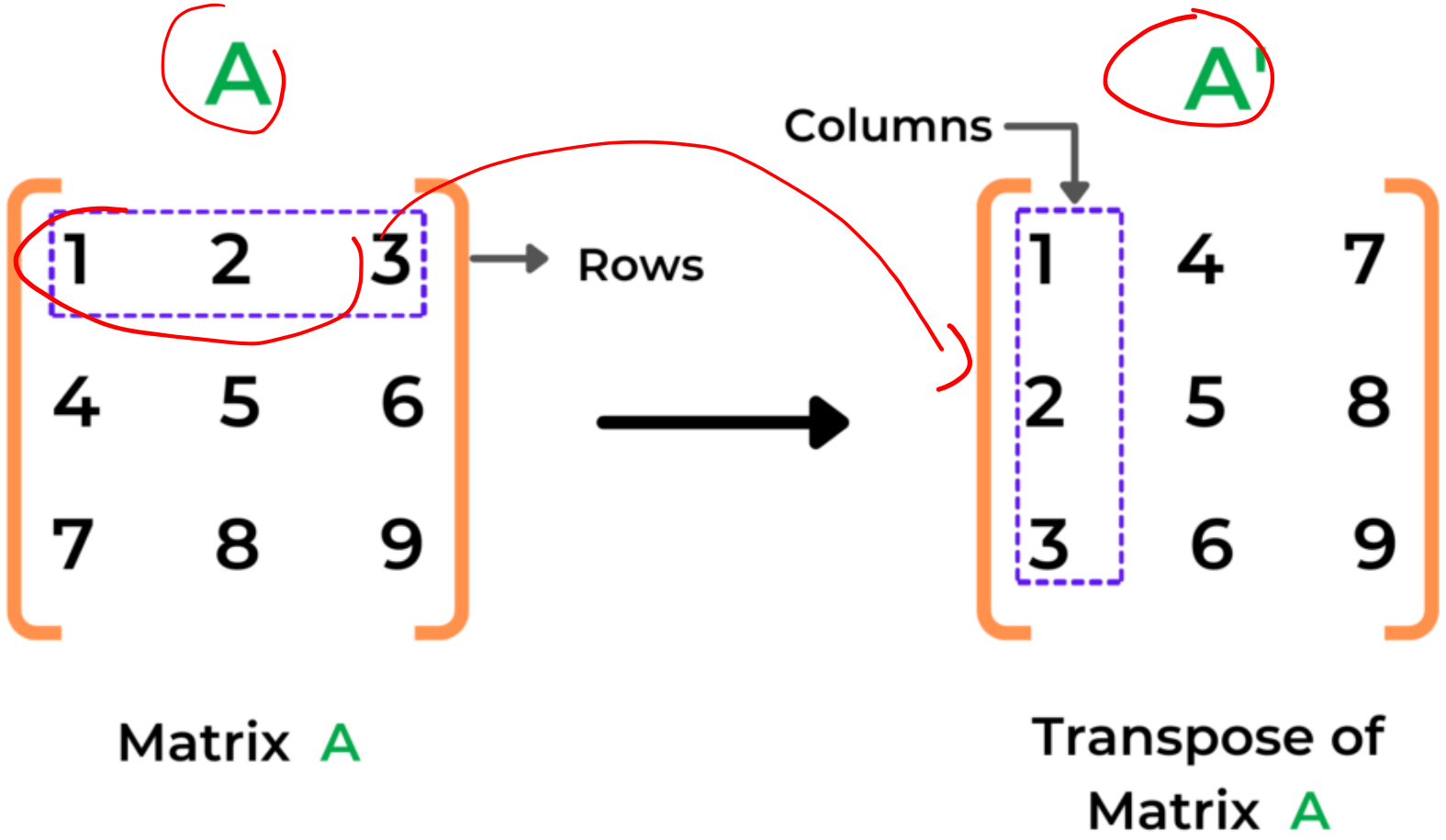
Problem 2

Câu hỏi 2 (Kỹ thuật thao tác trên dữ liệu danh sách 2 chiều):

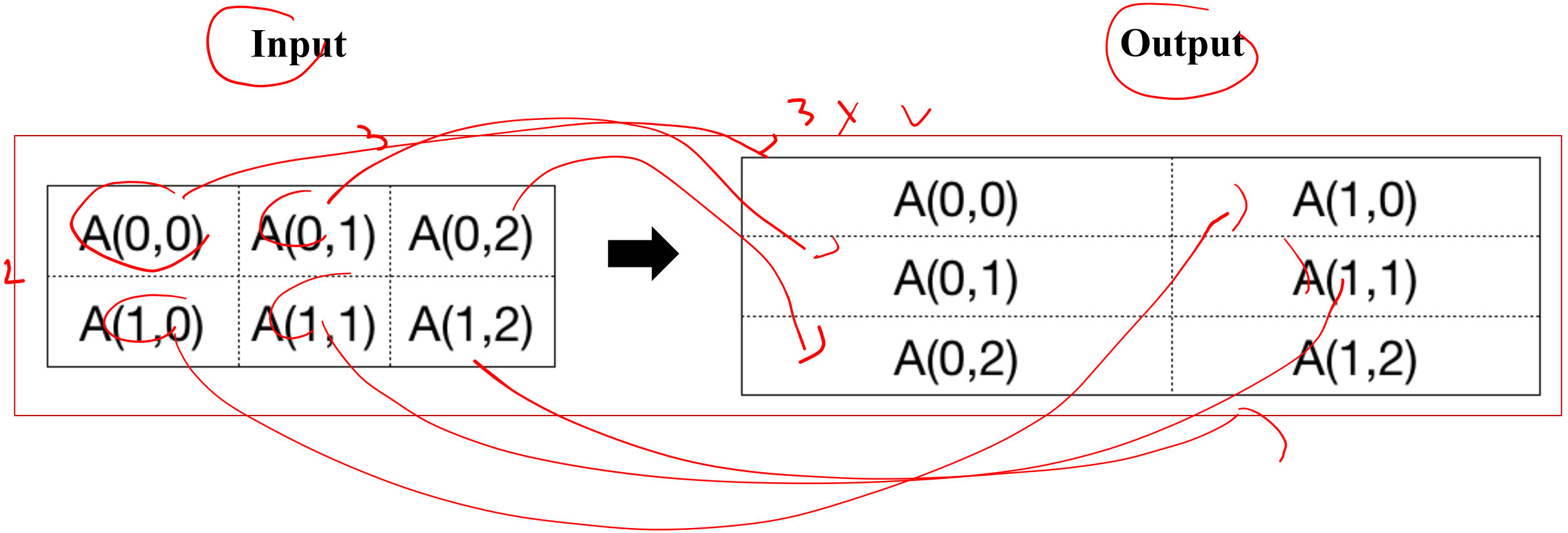
Cho trước một danh sách các danh sách số tự nhiên được lưu trữ trong danh sách (List) 2 chiều gồm có n dòng và n cột (ma trận vuông). Phát triển chương trình hoán vị dữ liệu(thay đổi dòng thành cột và ngược lại) . Bên dưới là ví dụ minh hoạ các test case đầu vào và đầu ra của chương trình.

```
Examples
Input:
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]

Output:
result = [[1, 4, 7],
          [2, 5, 8],
          [3, 6, 9]]
----------------------------
Input:
data = [[1, 2],
        [4, 5]]

Output:
result = [[1, 4],
          [2, 5]]
```

Problem Analysis



Problem Analysis



Solution

```
# implementation
def transpose(data):
    # your code here

    return result

# test case
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
assert transpose(data) == [[1, 4, 7],
                           [2, 5, 8],
                           [3, 6, 9]]

data = [[1, 2],
        [4, 5]]
assert transpose(data) == [[1, 4],
                           [2, 5]]

# multiple choice
data = [[7, 2],
        [3, 5]]
print(transpose(data))
```

Problem 3

Câu hỏi 3 (kỹ thuật nội suy liền kề cho dữ liệu 1 chiều (1D)): Cho trước một danh sách các số tự nhiên có chiều dài là n và một tham số k. Phát triển chương trình nội suy danh sách đầu vào thành danh sách mới có độ dài là n*k sử dụng kỹ thuật nội suy liền kề (nearest neighbor interpolation). Bên dưới là ví dụ minh hoạ các test case đầu vào và đầu ra của chương trình.

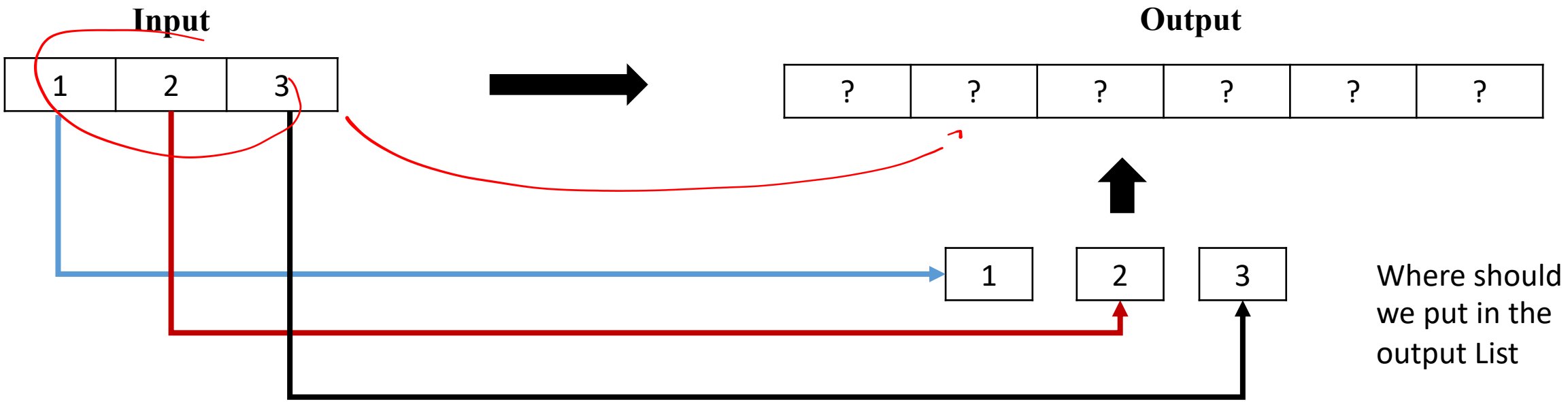
```
Examples
Input:
data = [1, 2, 3]
k = 2

Output:
result = [1, 1, 2, 2, 3, 3]
---------------------------
Input:
data = [1, 2, 3]
k = 3

Output:
result = [1, 1, 1, 2, 2, 2, 3, 3, 3]
```

Problem Analysis

Scale Factor = 3/6 = 1/2 = 0.5

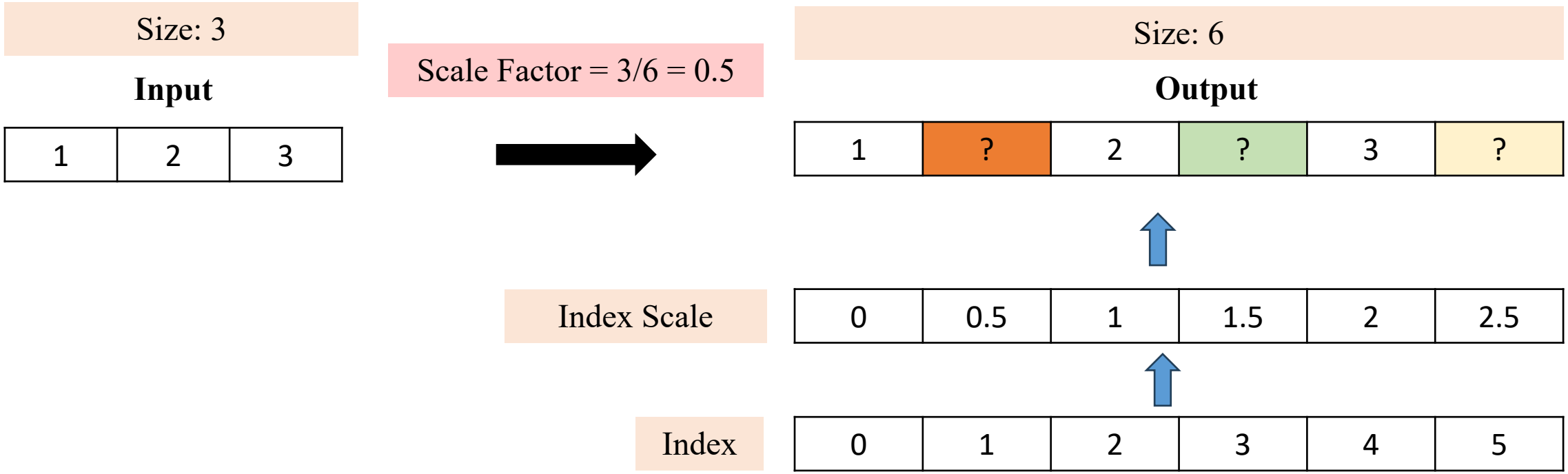
Size: 3

Size: 6

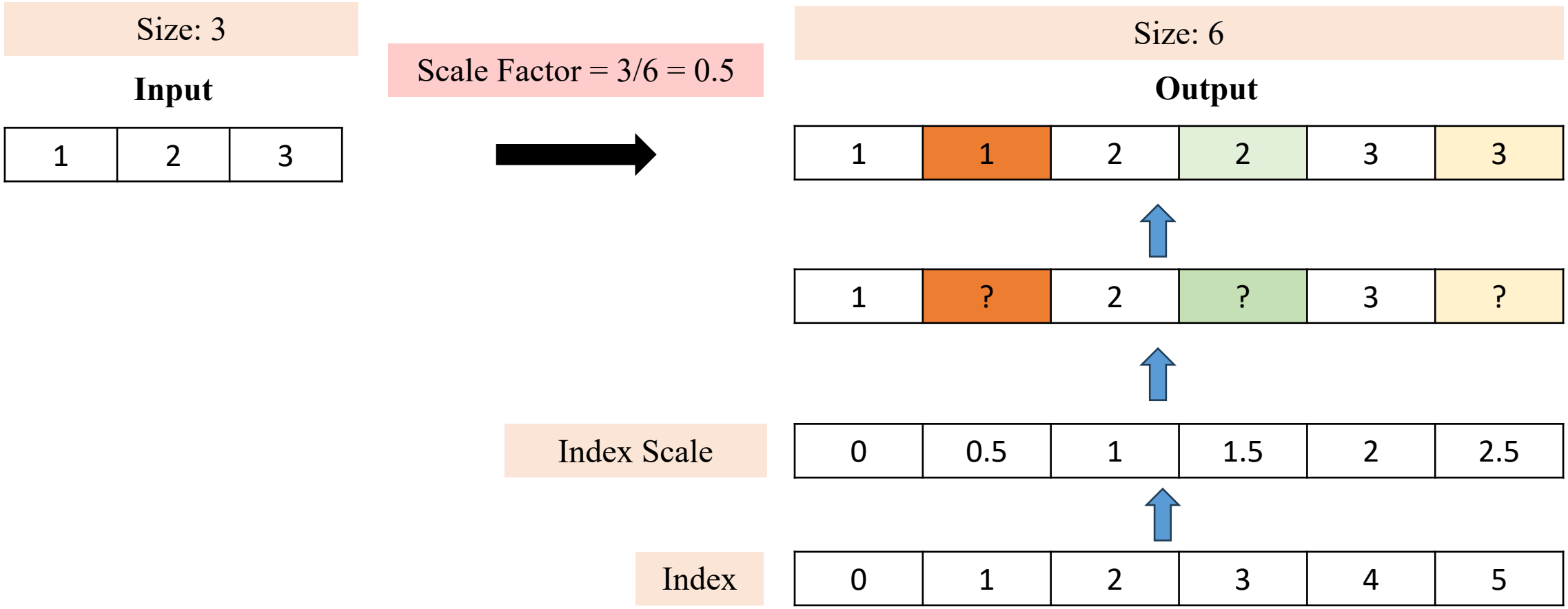


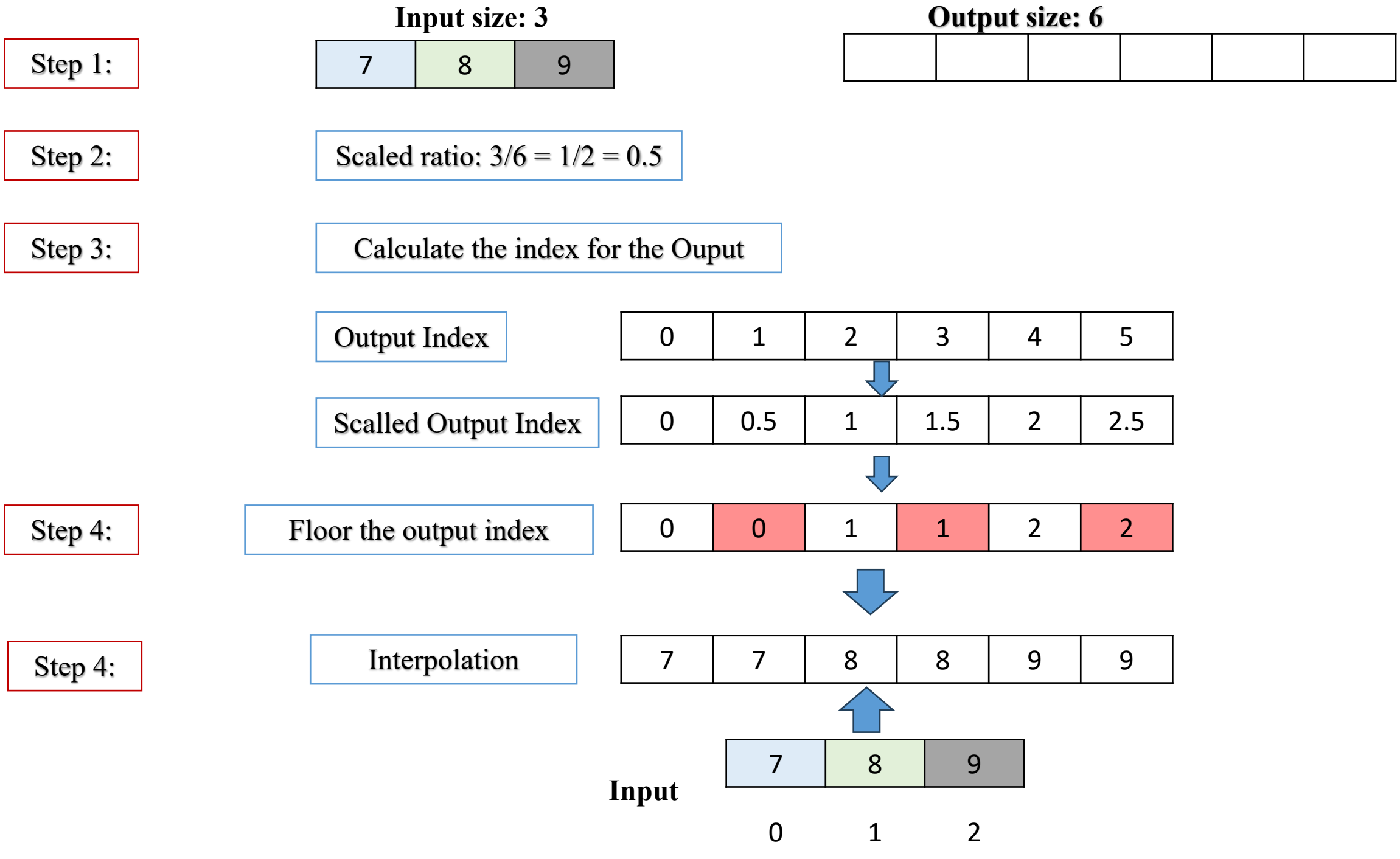
Where should we put in the output List

Problem Analysis



Problem Analysis




Input size: 3
Output size: 6
Step 1:
7
8
9
Step 2:
Scaled ratio: 3/6 = 1/2 = 0.5
Step 3:
Calculate the index for the Ouput
Output Index
0
1
2
3
4
5
Scalled Output Index
0
0.5
1
1.5
2
2.5
Step 4:
Floor the output index
0
0
1
1
2
2
Step 4:
Interpolation
7
7
8
8
9
9
Input
7
8
9
0
1
2

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Floor division is a mathematical operation in Python. This operator will divide the first argument by the second and round the result down to the nearest whole number, making it equivalent to the math. floor() function.

Solution

```python
# implementation
def nni_1d(data, k):
    # your code here

    return result

# test case
data = [1, 2, 3]
k = 2
assert nni_1d(data, k) == [1, 1, 2, 2, 3, 3]

data = [1, 2, 3]
k = 3
assert nni_1d(data, k) == [1, 1, 1, 2, 2, 2, 3, 3, 3]

# multiple choice
data = [5, 7]
k = 3
print(nni_1d(data, k))
```

Problem 4

Câu hỏi 4 (kỹ thuật nội suy liền kề cho dữ liệu 2 chiều (2D)): Cho trước một danh sách các số tự nhiên (m dòng và n cột) và tham số k. Phát triển chương trình nội suy danh sách đầu vào thành danh sách mới (m dòng và n*k cột) sử dụng kỹ thuật nội suy liền kề (nearest neighbor interpolation). Bên dưới là ví dụ minh hoạ các test case đầu vào và đầu ra của chương trình.

```
Examples
Input:
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
k = 2

Output:
result = [[1, 1, 2, 2, 3, 3],
          [4, 4, 5, 5, 6, 6],
          [7, 7, 8, 8, 9, 9]]
-------------------------------
Input:
data = [[1, 2],
        [4, 5],
        [7, 8]]
k = 2

Output:
result = [[1, 1, 2, 2],
          [4, 4, 5, 5],
          [7, 7, 8, 8]]
```

Problem Analysis

Size: 3x3

Input

1	2	3
4	5	6
7	8	9

Scale Factor = 3/6 = 0.5

Index Scale

Size: 3x6

Output

0	0.5	1	1.5	2	2.5

Problem Analysis

Size: 3x3

Input

1	2	3
4	5	6
7	8	9

Scale Factor = 3/6 = 0.5

Size: 3x6

Output

Index Scale

0	0.5	1	1.5	2	2.5

Column Index

0	0.5	1	1.5	2	2.5
0	0.5	1	1.5	2	2.5
0	0.5	1	1.5	2	2.5

Floor(Index Matrix)

Column Index

0	0	1	1	2	2
0	0	1	1	2	2
0	0	1	1	2	2

Problem Analysis

Size: 3x3

Input

1	2	3
4	5	6
7	8	9

Scale Factor = 3/6 = 0.5

Size: 3x6

Output

Index Scale

0	0.5	1	1.5	2	2.5

Column Index

0	0.5	1	1.5	2	2.5
0	0.5	1	1.5	2	2.5
0	0.5	1	1.5	2	2.5

Floor(Index Matrix)

0	0	1	1	2	2
0	0	1	1	2	2
0	0	1	1	2	2

Value

1	1	2	2	3	3
4	4	5	5	6	6
7	7	8	8	9	9

Final result

Simpler Approach

Problem Analysis

Size: 3x3

Input

1	2	3
4	5	6
7	8	9

Scale Factor = 3/6 = 0.5

Index Scale

0
1
2

Size: 3x6

Output

0	0.5	1	1.5	2	2.5
(0,0)	(0,0.5)	(0,1)	(0,1.5)	(0,2)	(0,2.5)
(1,0)	(1,0.5)	(1,1)	(1, 1.5)	(1, 2)	(1, 2.5)
(2,0)	(2,0.5)	(2,1)	(2,1.5)	(2,2)	(2,2.5)

(0,0)	(0,0)	(0,1)	(0,1)	(0,2)	(0,2)
(1,0)	(1,0)	(1,1)	(1, 1)	(1, 2)	(1, 2)
(2,0)	(2,0)	(2,1)	(2,1)	(2,2)	(2,2)

1	1	2	2	3	3
4	4	5	5	6	6
7	7	8	8	9	9

Solution

```python
# implementation
def h_nni_2d(data, k):
    # your code here
    return result

# test case
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
k = 2
assert h_nni_2d(data, k) == [[1, 1, 2, 2, 3, 3],
                             [4, 4, 5, 5, 6, 6],
                             [7, 7, 8, 8, 9, 9]]

data = [[1, 2],
        [4, 5],
        [7, 8]]
k = 2
assert h_nni_2d(data, k) == [[1, 1, 2, 2],
                             [4, 4, 5, 5],
                             [7, 7, 8, 8]]

# multiple choice
data = [[3],
        [4]]
k = 3
print(h_nni_2d(data, k))
```

Problem 5

Câu hỏi 5 (kỹ thuật nội suy liền kề cho dữ liệu 2 chiều (2D)): Cho trước một danh sách các số tự nhiên (m dòng và n cột) và tham số k. Phát triển chương trình nội suy dữ liệu cột của danh sách đầu vào thành danh sách mới (m*k dòng và n cột) sử dụng kỹ thuật nội suy liền kề (nearest neighbor interpolation). Bên dưới là ví dụ minh hoạ các test case đầu vào và đầu ra của chương trình.

```
Examples
Input:
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
k = 2

Output:
result = [[1, 2, 3],
          [1, 2, 3],
          [4, 5, 6]
          [4, 5, 6],
          [7, 8, 9],
          [7, 8, 9]]
-------------------------
Input:
```

Problem Analysis

Size: 3x3

Input

1	2	3
4	5	6
7	8	9

Scale Factor = 3/6 = 0.5

Index Scale

Size: 3x6

Output

Problem Analysis

Size: 3x3

Input

1	2	3
4	5	6
7	8	9

Scale Factor = 3/6 = 0.5

Row index Matrix

Size: 3x6

Output

Index Scale

0
0.5
1
1.5
2
2.5

Problem Analysis

Size: 3x3

Input

1	2	3
4	5	6
7	8	9

Scale Factor = 3/6 = 0.5

Size: 3x6

Output

Index Scale

0	0	0
0.5	0.5	0.5
1	1	1
1.5	1.5	1.5
2	2	2
2.5	2.5	2.5

Row index

Floor(Index Matrix)

Row index

0	0	0
0	0	0
1	1	1
1	1	1
2	2	2
2	2	2

Final result

Value

1	2	3
1	2	3
4	5	6
4	5	6
7	8	9
7	8	9

Simpler Approach

Problem Analysis

Size: 3x3

Input

1	2	3
4	5	6
7	8	9

Scale Factor = 3/6 = 0.5

Output

Size: 3x6

0	1	2	**Index**
(0,0)	(0,1)	(0,2)	0
(0.5,0)	(0.5,1)	(0.5,2)	0.5
(1,0)	(1,1)	(1,2)	1
(1.5,0)	(1.5,1)	(1.5,2)	1.5
(2,0)	(2,1)	(2,2)	2
(2.5,0)	(2.5,1)	(2.5,2)	2.5

(0,0)	(0,1)	(0,2)
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(2,0)	(2,1)	(2,2)

1	2	3
1	2	3
4	5	6
4	5	6
7	8	9
7	8	9

Solution

```
# implementation
def v_nni_2d(data, k):
    # your code here

    return result

# test case
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
k = 2
assert v_nni_2d(data, k) ==  [[1, 2, 3],
                              [1, 2, 3],
                              [4, 5, 6],
                              [4, 5, 6],
                              [7, 8, 9],
                              [7, 8, 9]]
```

Problem 6

Câu hỏi 6 (kỹ thuật nội suy liền kề cho dữ liệu 2 chiều (2D)): Cho trước một danh sách các số tự nhiên (m dòng và n cột) và tham số k. Phát triển chương trình nội suy dữ liệu danh sách đầu vào thành danh sách mới (m*k dòng và n*k cột) sử dụng kỹ thuật nội suy liền kề (nearest neighbor interpolation). Bên dưới là ví dụ minh hoạ các test case đầu vào và đầu ra của chương trình.

```
Examples
Input:
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
k = 2

Output:
result = [[1, 1, 2, 2, 3, 3],
          [1, 1, 2, 2, 3, 3],
          [4, 4, 5, 5, 6, 6],
          [4, 4, 5, 5, 6, 6],
          [7, 7, 8, 8, 9, 9],
          [7, 7, 8, 8, 9, 9]]
------------------------------
Input:
data = [[1, 2],
        [4, 5],
        [7, 8]]
k = 2
```

Problem Analysis

Scale Factor = 3/6 = 0.5

Size: 3x3

Input

1	2	3
4	5	6
7	8	9

column interpolation

Size: 6x6

Output

Column index

0	0.5	1	1.5	2	2.5

Problem Analysis

Scale Factor = 3/6 = 0.5

Size: 3x3

Size: 6x6

Input

0	1	2
1	2	3
4	5	6
7	8	9

column interpolation

Column index

Output

0	0.5	1	1.5	2	2.5
0	0.5	1	1.5	2	2.5
0	0.5	1	1.5	2	2.5
0	0.5	1	1.5	2	2.5

Nearest index

0	0	1	1	2	2
0	0	1	1	2	2
0	0	1	1	2	2

1	1	2	2	3	3
4	4	5	5	6	6
7	7	8	8	9	9

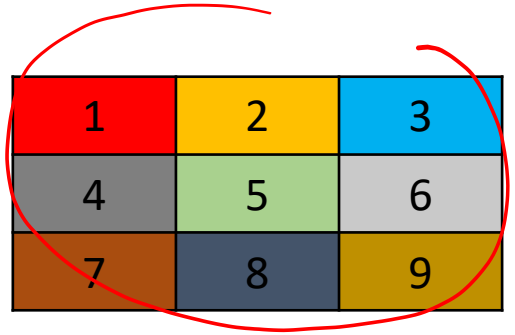
Result

Problem Analysis

Scale Factor = 3/6 = 0.5

Size: 3x3

Input



column interpolation

Size: 6x6

Output

1	1	2	2	3	3
4	4	5	5	6	6
7	7	8	8	9	9

row interpolation

Index Scale

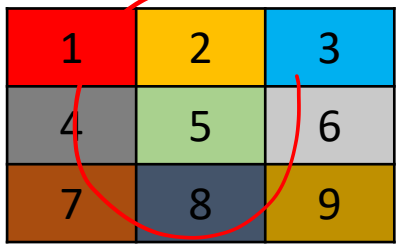
0
0.5
1
1.5
2
2.5

Problem Analysis

Scale Factor = 3/6 = 0.5

Size: 3x3

Input



column interpolation

0	1	1	2	2	3	3
1	4	4	5	5	6	6
2	7	7	8	8	9	9

Size: 6x6

Output

Row Interpolation By Index

0	0	0	0	0	0
0.5	0.5	0.5	0.5	0.5	0.5
1	1	1	1	1	1
1.5	1.5	1.5	1.5	1.5	1.5
2	2	2	2	2	2
2.5	2.5	2.5	2.5	2.5	2.5

Nearest index

0	0	0	0	0	0
0	0	0	0	0	0
1	1	1	1	1	1
1	1	1	1	1	1
2	2	2	2	2	2
2	2	2	2	2	2

1	1	2	2	3	3
1	1	2	2	3	3
4	4	5	5	6	6
4	4	5	5	6	6
7	7	8	8	9	9
7	7	8	8	9	9

Simpler Approach

Problem Analysis

Scale Factor = 3/6 = 0.5

Size: 6x6

Size: 3x3

Input

Column index

Output

0	0.5	1	1.5	2	2.5
(0,0)	(0,0.5)	(0,1)	(0,1.5)	(0,2)	(0,2.5)
(0.5,0)	(0.5,0.5)	(0.5,1)	(0.5,1.5)	(0.5,2)	(0.5,2.5)
(1,0)	(1,0.5)	(1,1)	(1,1.5)	(1,2)	(1,2.5)
(1.5,0)	(1.5,0.5)	(1.5,1)	(1.5,1.5)	(1.5,2)	(1.5,2.5)
(2,0)	(2,0.5)	(2,1)	(2,1.5)	(2,2)	(2,2.5)
(2.5,0)	(2.5,0.5)	(2.5,1)	(2.5,1.5)	(2.5,2)	(2.5,2.5)

Row index

0
0.5
1
1.5
2
2.5

Nearest index

(0,0)	(0,0)	(0,1)	(0,1)	(0,2)	(0,2)
(0,0)	(0,0)	(0,1)	(0,1)	(0,2)	(0.,2)
(1,0)	(1,0)	(1,1)	(1,1)	(1,2)	(1,2)
(1,0)	(1,0)	(1,1)	(1,1)	(1,2)	(1,2)
(2,0)	(2,0)	(2,1)	(2,1)	(2,2)	(2,2)
(2,0)	(2,0)	(2,1)	(2,1)	(2,2)	(2,2)

Problem Analysis

Scale Factor = 3/6 = 0.5

Size: 3x3

Input

1	2	3
4	5	6
7	8	9

Column index

Size: 6x6

Output

0	0.5	1	1.5	2	2.5	
(0,0)	(0,0.5)	(0,1)	(0,1.5)	(0,2)	(0,2.5)	0
(0.5,0)	(0.5,0.5)	(0.5,1)	(0.5,1.5)	(0.5,2)	(0.5,2.5)	0.5
(1,0)	(1,0.5)	(1,1)	(1,1.5)	(1,2)	(1,2.5)	1
(1.5,0)	(1.5,0.5)	(1.5,1)	(1.5,1.5)	(1.5,2)	(1.5,2.5)	1.5
(2,0)	(2,0.5)	(2,1)	(2,1.5)	(2,2)	(2,2.5)	2
(2.5,0)	(2.5,0.5)	(2.5,1)	(2.5,1.5)	(2.5,2)	(2.5,2.5)	2.5

Row index

Nearest index

1	1	2	2	3	3
1	1	2	2	3	3
4	4	5	5	6	6
4	4	5	5	6	6
7	7	8	8	9	9
7	7	8	8	9	9

Solution

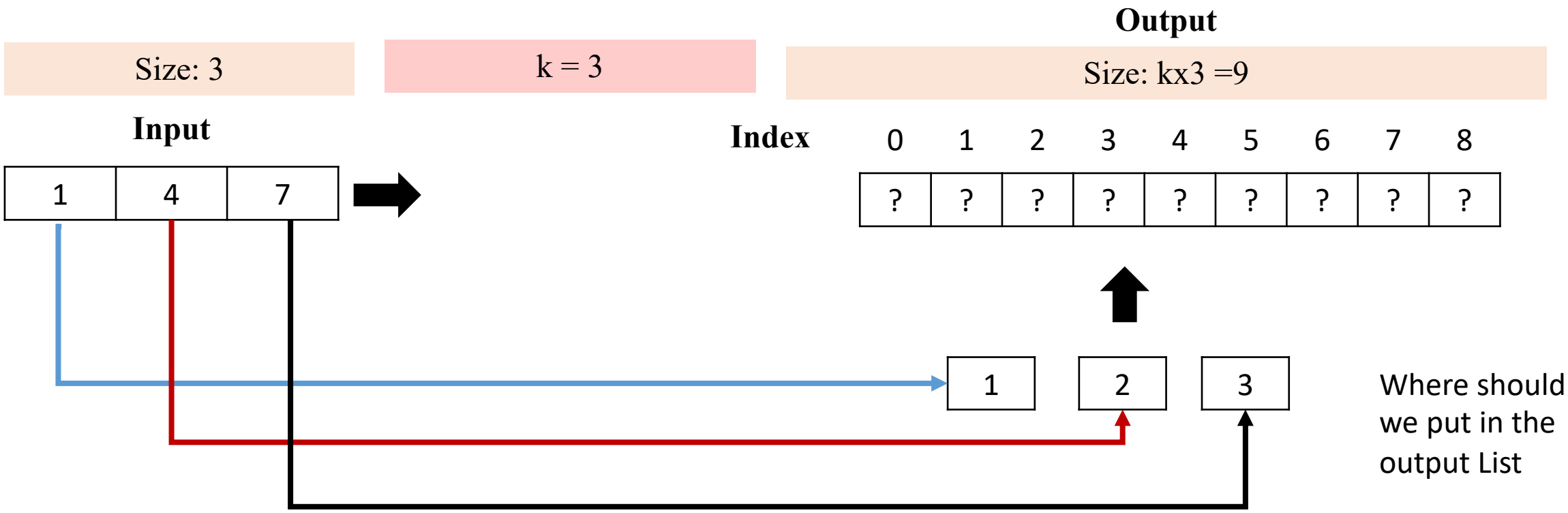
```python
# implementation
def nni_2d(data, k):
    # your code here

    return result

# test case
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
k = 2
assert nni_2d(data, k) == [[1, 1, 2, 2, 3, 3],
                           [1, 1, 2, 2, 3, 3],
                           [4, 4, 5, 5, 6, 6],
                           [4, 4, 5, 5, 6, 6],
                           [7, 7, 8, 8, 9, 9],
                           [7, 7, 8, 8, 9, 9]]

data = [[1, 2],
        [4, 5],
        [7, 8]]
k = 2
assert nni_2d(data, k) == [[1, 1, 2, 2],
                           [1, 1, 2, 2],
                           [4, 4, 5, 5],
                           [4, 4, 5, 5],
                           [7, 7, 8, 8],
                           [7, 7, 8, 8]]
```

Problem 7

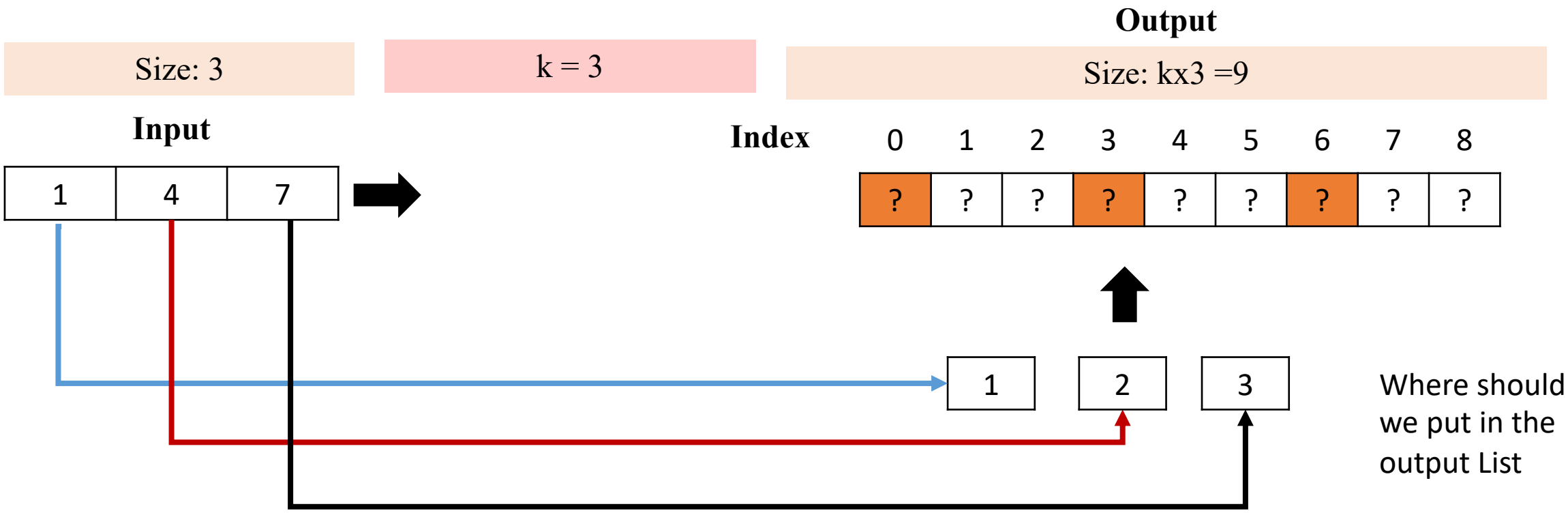
Câu hỏi 7 (Kỹ thuật nội suy linear cho dữ liệu 1 chiều (1D)): Cho trước danh sách số thực có chiều dài n và tham số k. Phát triển chương trình nội suy danh sách đầu vào thành danh sách mới có độ dài là n*k sử dụng kỹ thuật nội suy linear. Lưu ý rằng, để đơn giản, bạn có thể bỏ qua phần tử cuối cùng (k-1). Bên dưới là ví dụ minh hoạ các test case đầu vào và đầu ra của chương trình.

```
Examples
Input:
data = [1, 2, 3]
k = 2

Output:
result = [1.0, 1.5, 2.0, 2.5, 3.0, x]

----------------------------------------
Input:
data = [1, 4, 7]
k = 3

Output:
result = [1, 2.0, 3.0, 4, 5.0, 6.0, 7, x, x]

Here, x is any number.
```

Analyze Problem



Analyze Problem



Analyze Problem

Size: 3

k = 3

Output

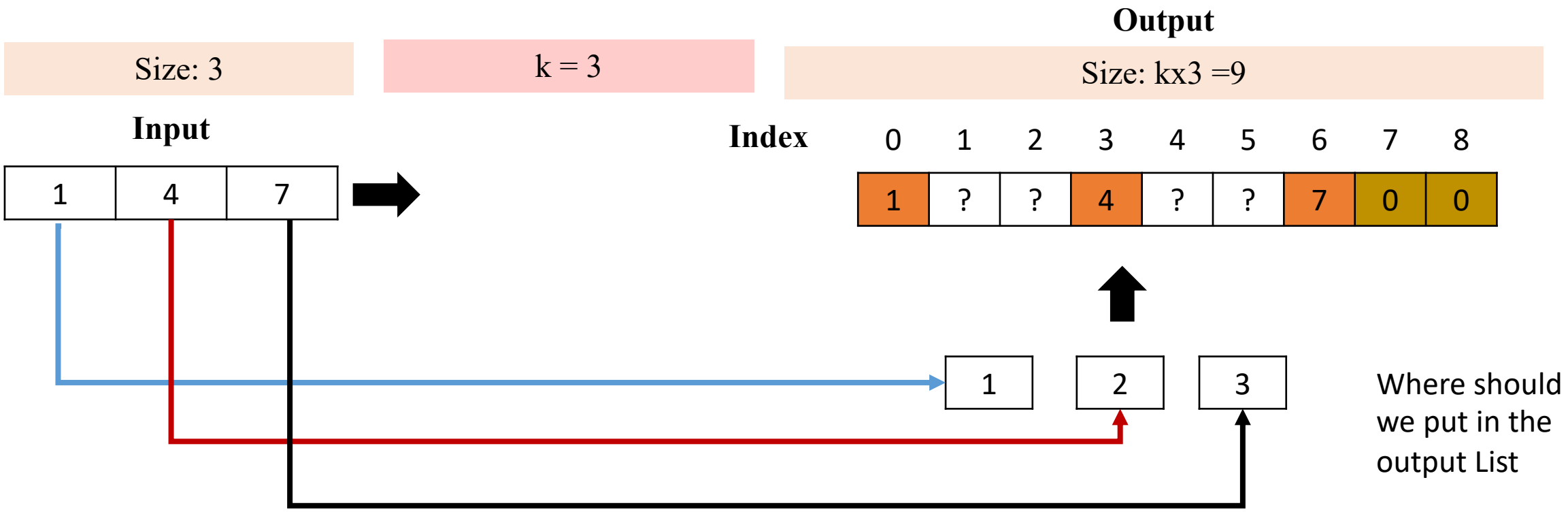
Size: kx3 =9

Input

Where should we put in the output List

```
n = len(data)
result_len = n*k
result = [0]*result_len

for i in range(result_len-k+1):

    if i%k == 0:
        result[i] = data[i//k]
```

Analyze Problem



Analyze Problem

Size: 3

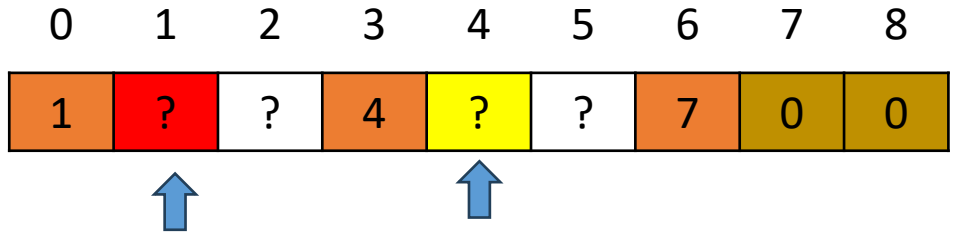
k = 3

Output

Size: kx3 =9

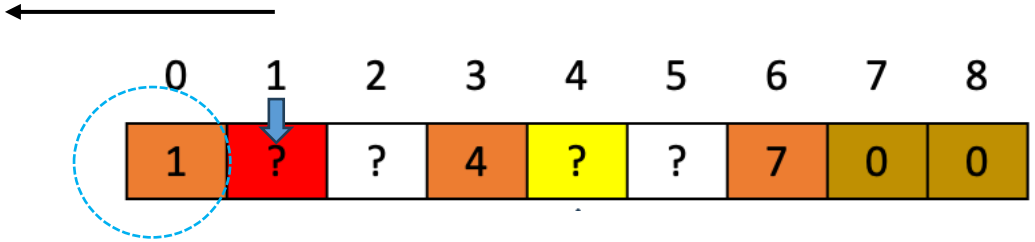
Input

Index

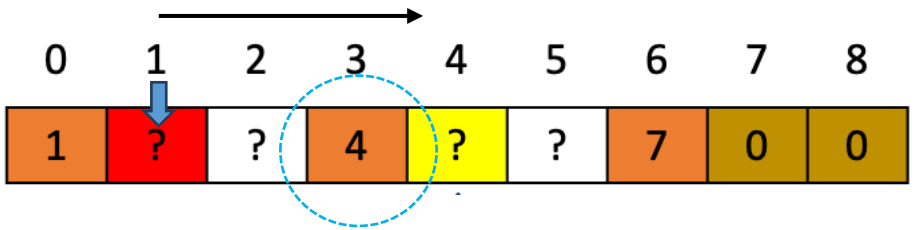


How to fill in the values

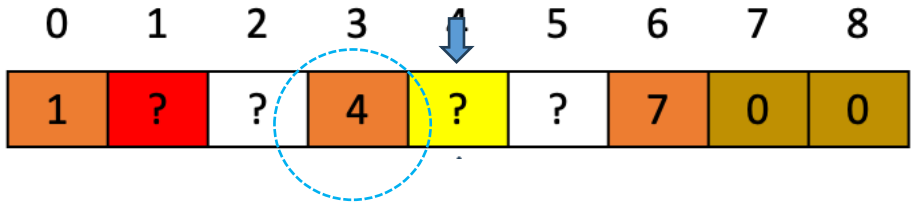
Find left nearest point



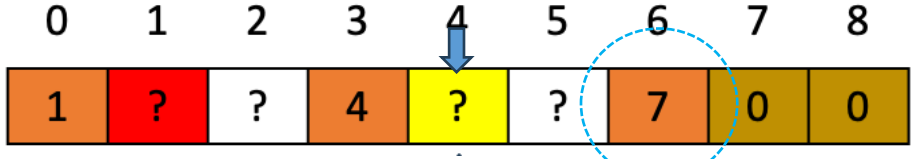
Find right nearest point



Find left nearest point

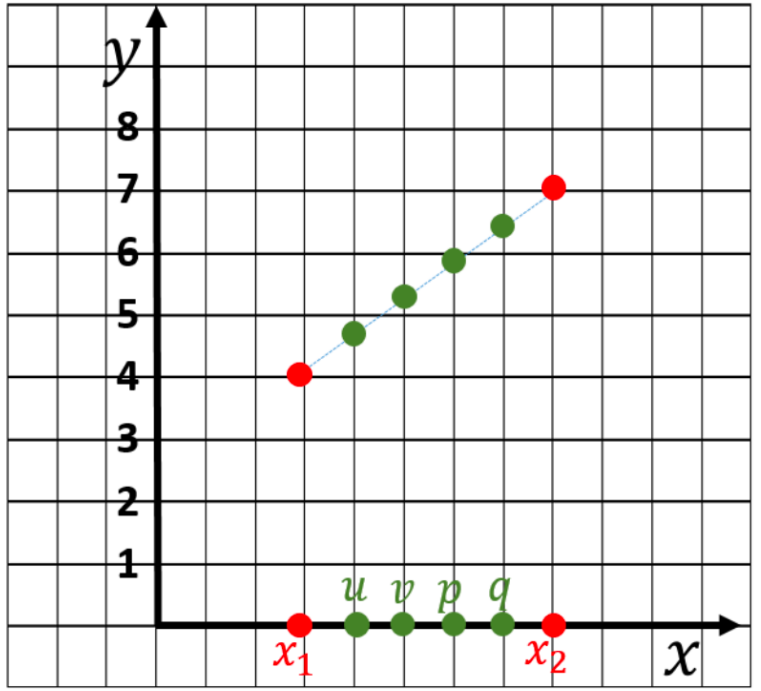


Find right nearest point



Analyze Problem

```
def linear_interpolation(p1, p2, x):
    x1, y1 = p1
    x2, y2 = p2
    if x1 != x2:
        y = ((x2-x)/(x2-x1))*y1 + ((x-x1)/(x2-x1))*y2
    else:
        y = y1
    return y
```



Linear interpolation

Analyze Problem

Size: 3

k = 3

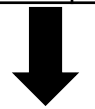
Size: kx3 =9

Input

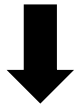
1	4	7

Index

0	1	2	3	4	5	6	7	8
1	?	?	4	?	?	7	0	0



X is index
Y is the value of the list of at X index



A(0,1), B(1,?), C(2,?), D(3,4), E(4,?), F(5,?), G(6,7), H(7,?), I(8,?)

Analyze Problem

Size: 3

k = 3

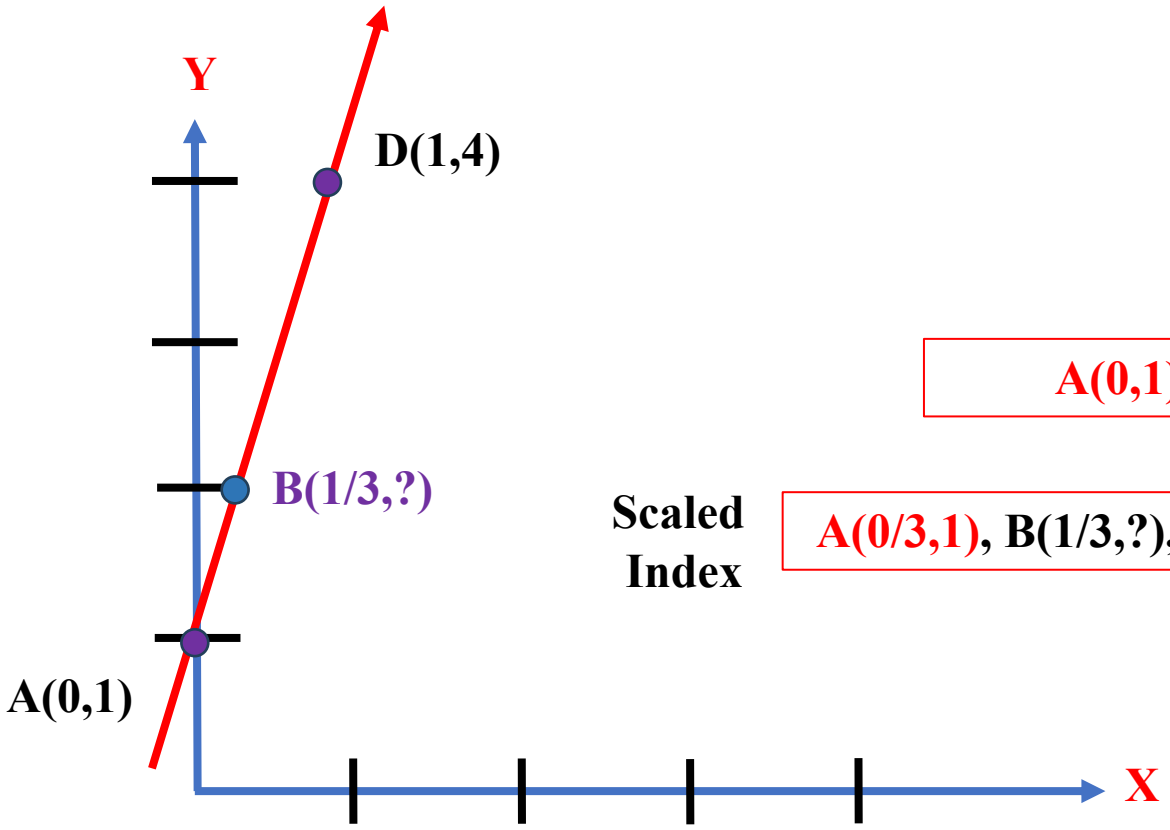
Size: kx3 =9

Input

1	4	7

Index

0	1	2	3	4	5	6	7	8
1	?	?	4	?	?	7	0	0



X is index
Y is the value of the list of at X index

A(0,1), B(1,?), C(2,?), D(3,4), E(4,?), F(5,?), G(6,7), H(7,?), I(8,?)

Scaled Index

A(0/3,1), B(1/3,?), C(2/3,?), D(3/3,4), E(4/3,?), F(5/3,?), G(6/3,7), H(7/3,?), I(8/3,?)

A(0/3,1), B(1/3,?), C(2/3,?), D(3/3,4)

A(0,1), B(1/3,?), C(2/3,?), D(1,4)

Analyze Problem

Size: 3

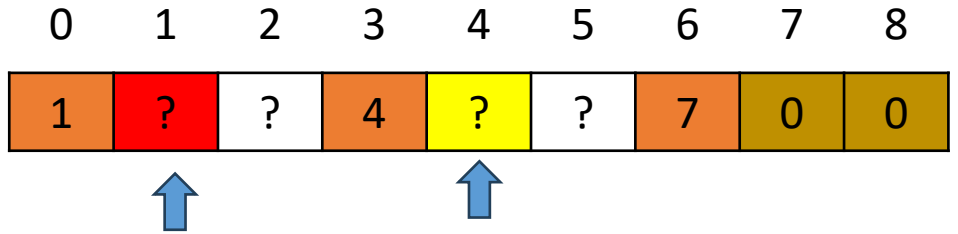
k = 3

Output

Size: kx3 =9

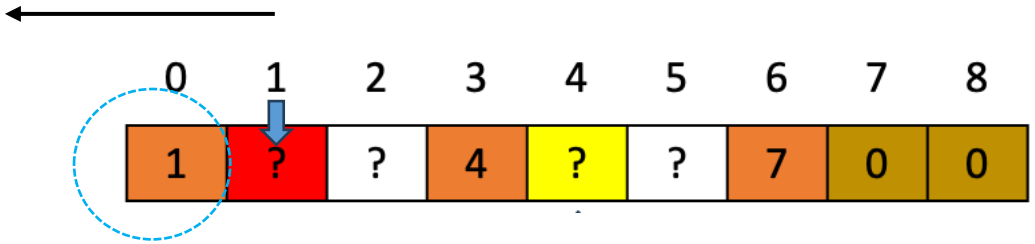
Input

Index

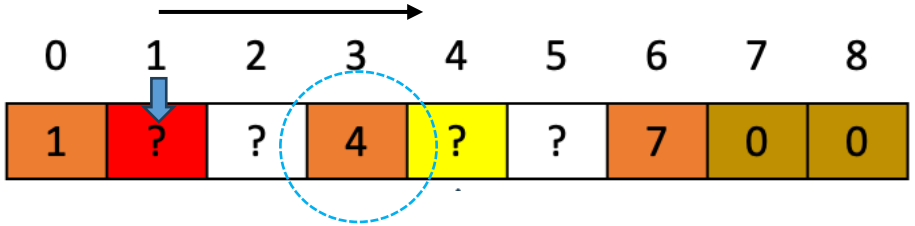


How to fill in the values

Find left nearest point



Find right nearest point




```python
def li_1d(data, k):
    n = len(data)
    result_len = n*k
    result = [0]*result_len

    for i in range(result_len-k+1):

        if i%k == 0:
            result[i] = data[i//k]
        else:
            begin = math.floor(i/k)
            end   = math.ceil(i/k)
            p1 = (begin, data[begin])
            p2 = (end, data[end])
            y = linear_interpolation(p1=p1, p2=p2, x=i/k)
            result[i] = y

    return result
```

Solution

```python
# implementation
import math

def linear_interpolation(p1, p2, x):
    x1, y1 = p1
    x2, y2 = p2
    if x1 != x2:
        y = ((x2-x)/(x2-x1))*y1 + ((x-x1)/(x2-x1))*y2
    else:
        y = y1
    return y

def li_1d(data, k):
    # your code here

    return result

# test case
data = [1, 2, 3]
k = 2
assert li_1d(data, k)[:-1] == [1, 1.5, 2, 2.5, 3, 0][:-1]
```

Problem 8

Câu hỏi 8 (optional): Cho trước ảnh đầu vào (link1 hoặc link2) có kích thước dài 400 và rộng 400 như bên dưới:

Bạn có thể sử dụng thư viện opencv để đọc ảnh đầu và chuyển dữ liệu ảnh đầu vào thành danh sách 2 chiều (400 dòng và 400 cột).

```
import cv2

image = cv2.imread('tree.jpg', 0).tolist()

height = len(image)
width = len(image[0])
```

Ví dụ, *image* có kiểu dữ liệu là 2 chiều (a list of lists). Phát triển chương trình để tăng kích thước ảnh đầu (400x400) và thành ảnh mới (1200 x1200) sử dụng kỹ thuật nội suy nearst-neigbor hoặc linear.

400x400

Nearest-neigbor Interpolation

1200 x 1200

Analyze Problem

400x400

Tree.jpg

Read by

OpenCV

To List

400x400

Solution

```
import cv2
import numpy as np
import math

def resize_nni(source, source_h, source_w, target_h, target_w):
    new_data = [[0]*target_w for _ in range(target_h)]

    #Calculate horizontal and vertical scaling factor
    w_scale_factor = source_w/target_w
    h_scale_factor = source_h/target_h

    #your code here


    return new_data

image = cv2.imread('tree.jpg', 0).tolist()

height = len(image)
width = len(image[0])

new_image = resize_nni(image, height, width, height*3, width*3)
cv2.imwrite('tree_2x.jpg', np.array(new_image))
```

Bilinear interpolation

Mapping (scale is an integer)

(3, 3)

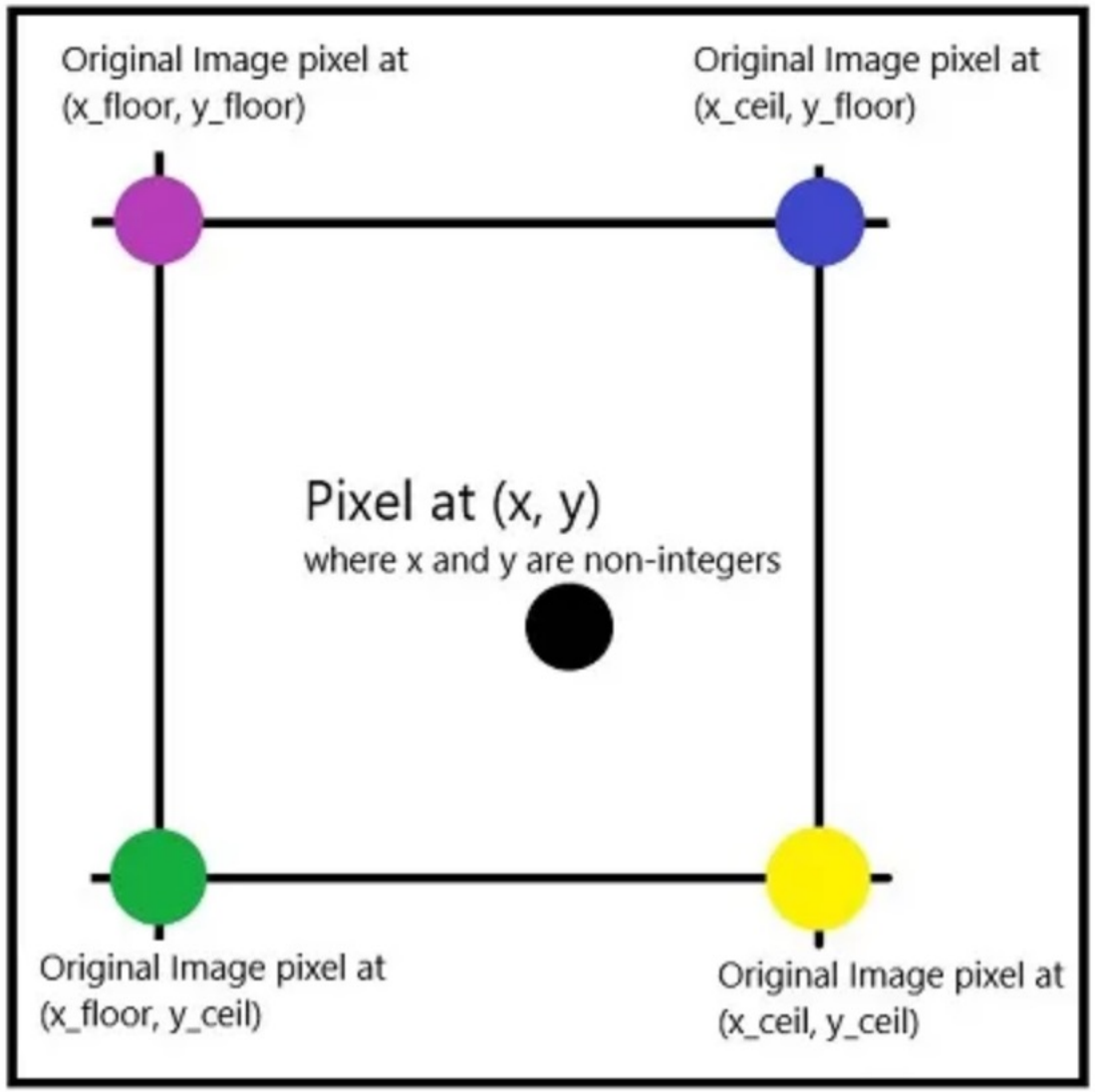
0	2	4
1	3	5
0	2	4

Copy and interpolation

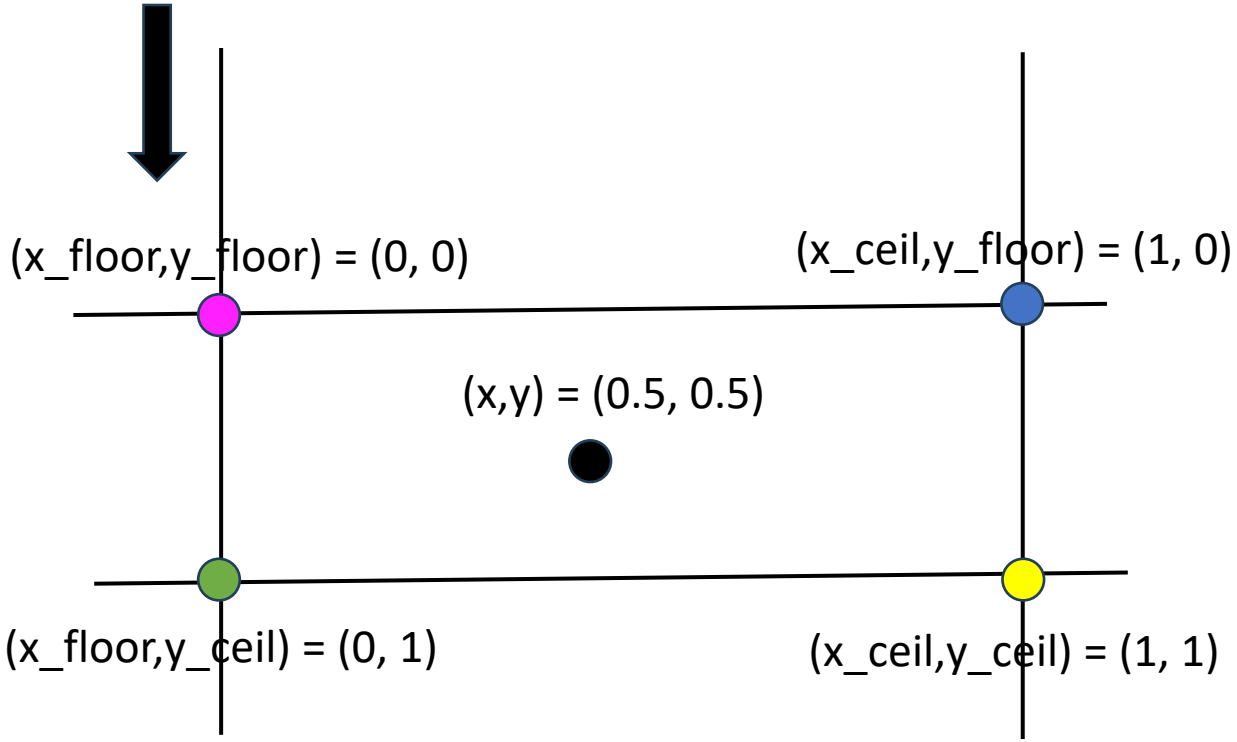
(12, 12)

Scaled index	0	1/4	2/4	3/4	4/4	5/4	6/4	7/4	8/4	9/4	10/4	11/4
0	**0**				**2**				**4**			
1/4												
2/4												
3/4												
4/4	**1**				**3**				**5**			
5/4												
6/4												
7/4												
8/4	**0**				**2**				**4**			
9/4												
10/4												
11/4												

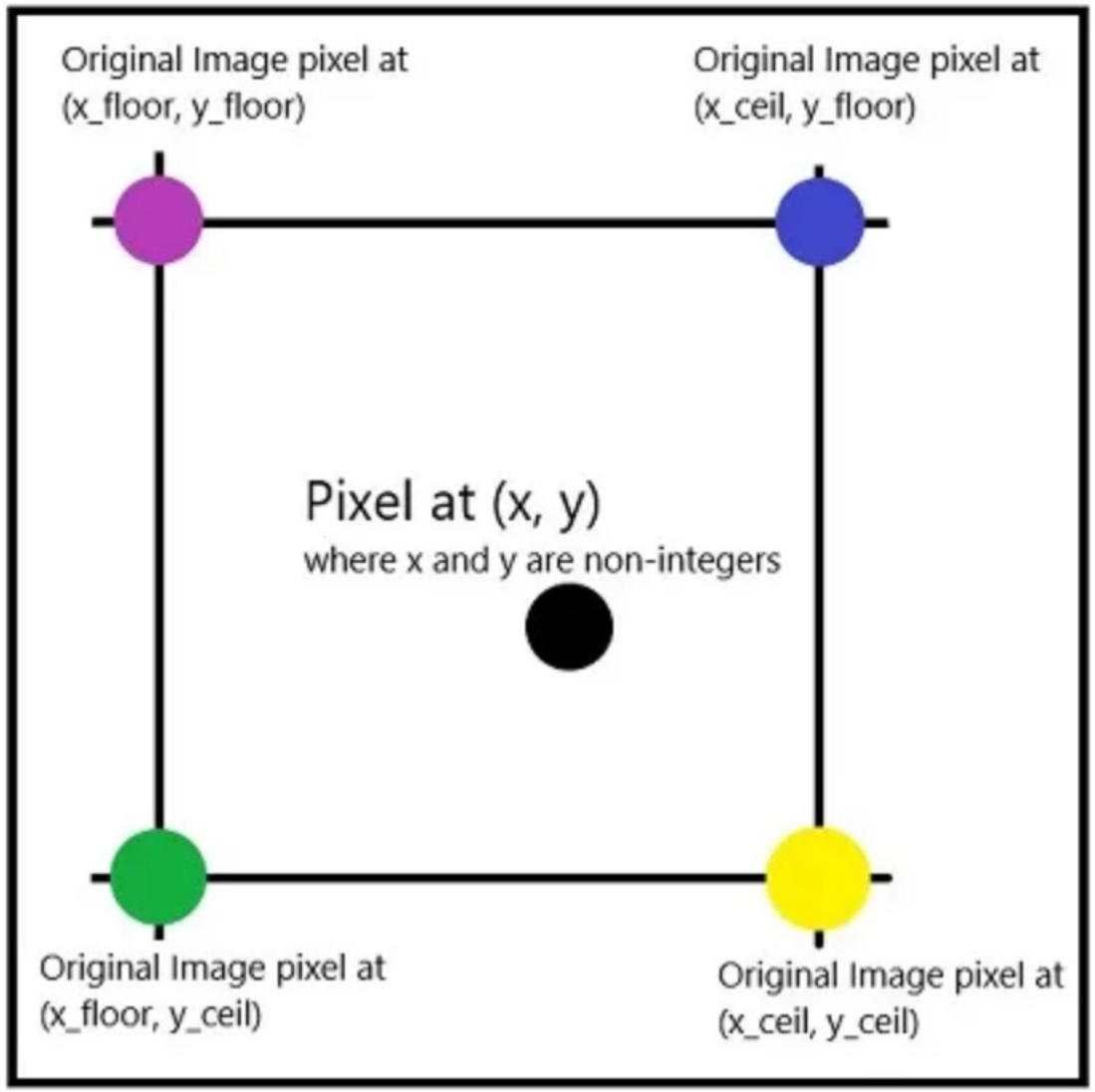
Bi-linear Interpolation for Image Resize



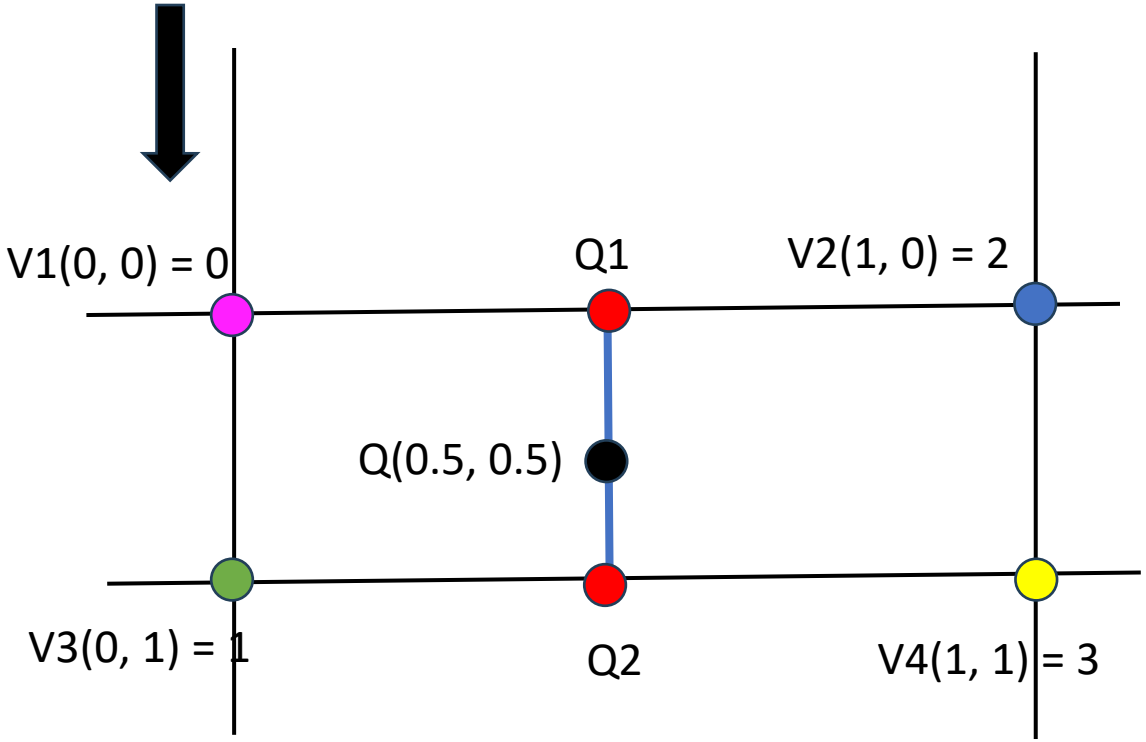
Scaled index	0	1/4	2/4	3/4	4/4
0	0				2
1/4					
2/4					
3/4					
4/4	1				3



Bi-linear Interpolation for Image Resize

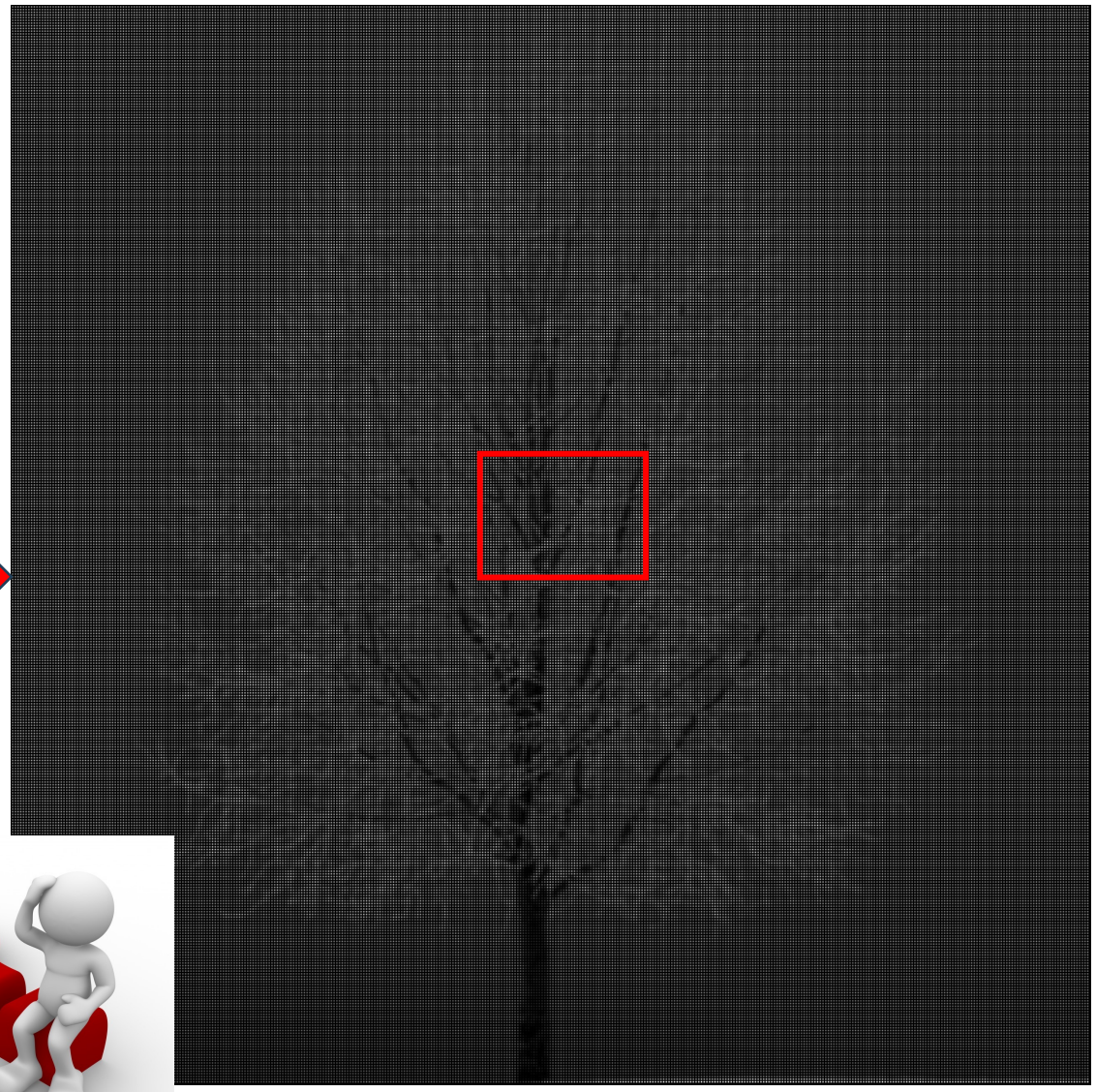


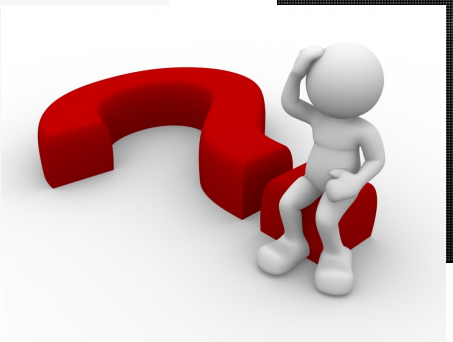
Scaled index	0	1/4	2/4	3/4	4/4
0	0				2
1/4					
2/4					
3/4					
4/4	1				3



Solution

```python
def bl_resize_normalcase(original_img, old_h, old_w, new_h, new_w):
  resized = np.zeros((new_h, new_w))
  #Calculate horizontal and vertical scaling factor
  w_scale_factor = (old_w ) / (new_w ) if new_h != 0 else 0
  h_scale_factor = (old_h ) / (new_h ) if new_w != 0 else 0
  for i in range(new_h):
    for j in range(new_w):
      #map the coordinates back to the original image
      x = i * h_scale_factor
      y = j * w_scale_factor
      #calculate the coordinate values for 4 surrounding pixels.
      x_floor = math.floor(x)
      x_ceil = min( old_h - 1, math.ceil(x))
      y_floor = math.floor(y)
      y_ceil = min(old_w - 1, math.ceil(y))

      v1 = original_img[x_floor][y_floor]
      v2 = original_img[x_ceil][y_floor]
      v3 = original_img[x_floor][y_ceil]
      v4 = original_img[x_ceil] [y_ceil]

      q1 = v1 * (x_ceil - x) + v2 * (x - x_floor)
      q2 = v3 * (x_ceil - x) + v4 * (x - x_floor)
      q = q1 * (y_ceil - y) + q2 * (y - y_floor)

      resized[i,j] = q
  return resized.astype(np.uint8)
```



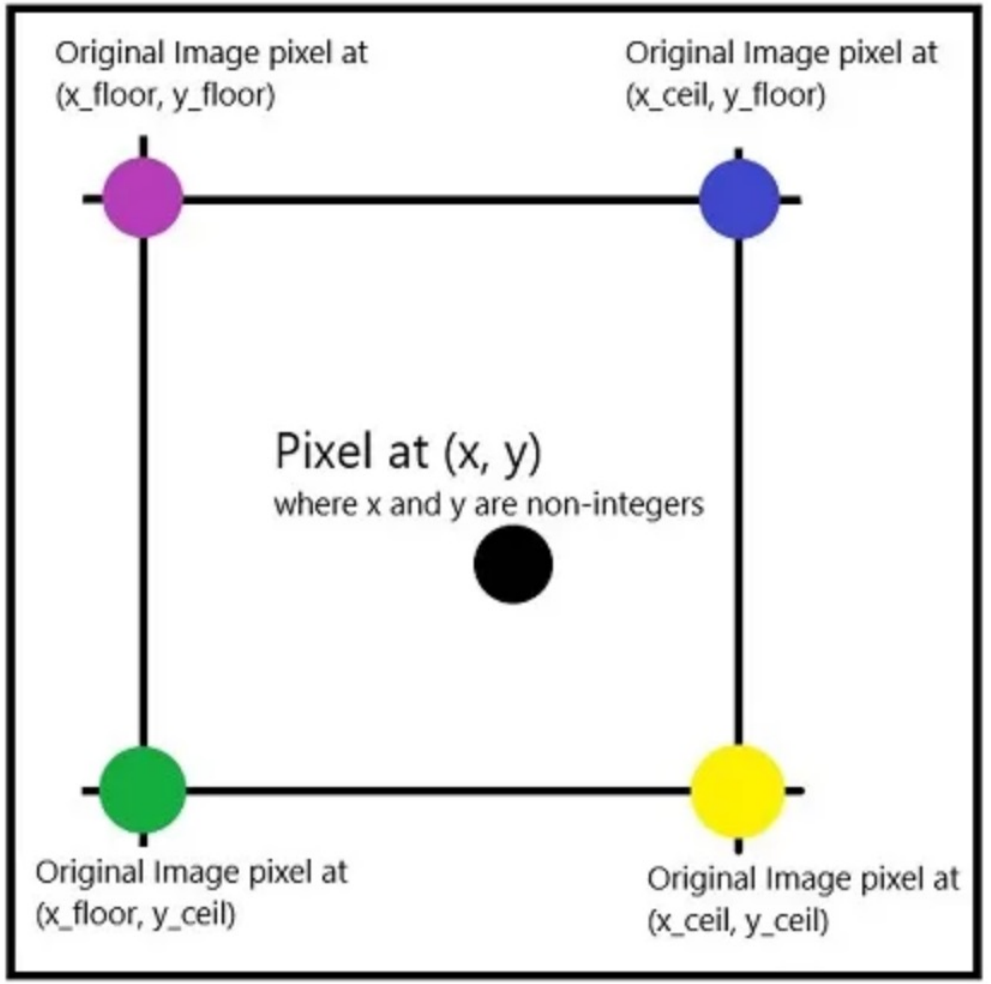


Solution

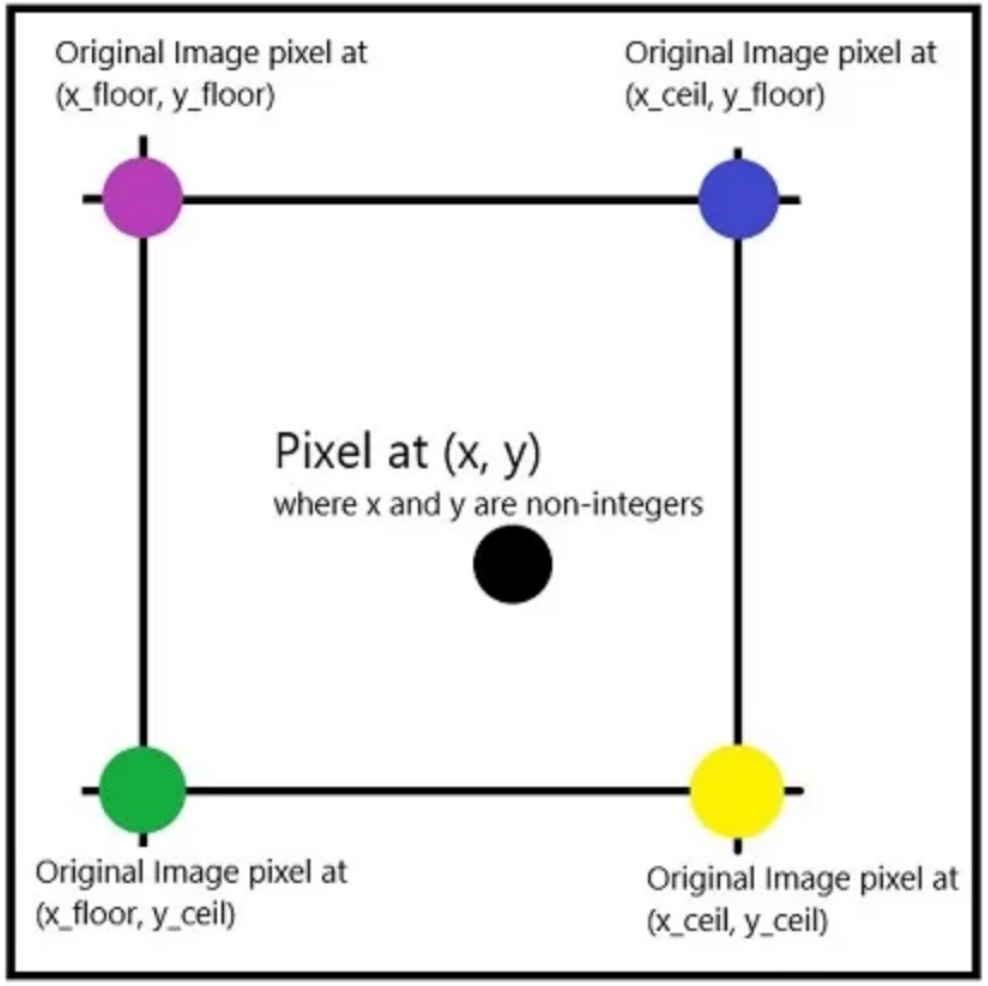
```python
def bl_resize_normalcase(original_img, old_h, old_w, new_h, new_w):
  resized = np.zeros((new_h, new_w))
  #Calculate horizontal and vertical scaling factor
  w_scale_factor = (old_w ) / (new_w ) if new_h != 0 else 0
  h_scale_factor = (old_h ) / (new_h ) if new_w != 0 else 0
  for i in range(new_h):
    for j in range(new_w):
      #map the coordinates back to the original image
      x = i * h_scale_factor
      y = j * w_scale_factor
      #calculate the coordinate values for 4 surrounding pixels.
      x_floor = math.floor(x)
      x_ceil = min( old_h - 1, math.ceil(x))
      y_floor = math.floor(y)
      y_ceil = min(old_w - 1, math.ceil(y))

      v1 = original_img[x_floor][y_floor]
      v2 = original_img[x_ceil][y_floor]
      v3 = original_img[x_floor][y_ceil]
      v4 = original_img[x_ceil] [y_ceil]

      q1 = v1 * (x_ceil - x) + v2 * (x - x_floor)
      q2 = v3 * (x_ceil - x) + v4 * (x - x_floor)
      q = q1 * (y_ceil - y) + q2 * (y - y_floor)

      resized[i,j] = q
  return resized.astype(np.uint8)
```

What's wrong?

This happens when either x or y is an integer resulting in $q=0$

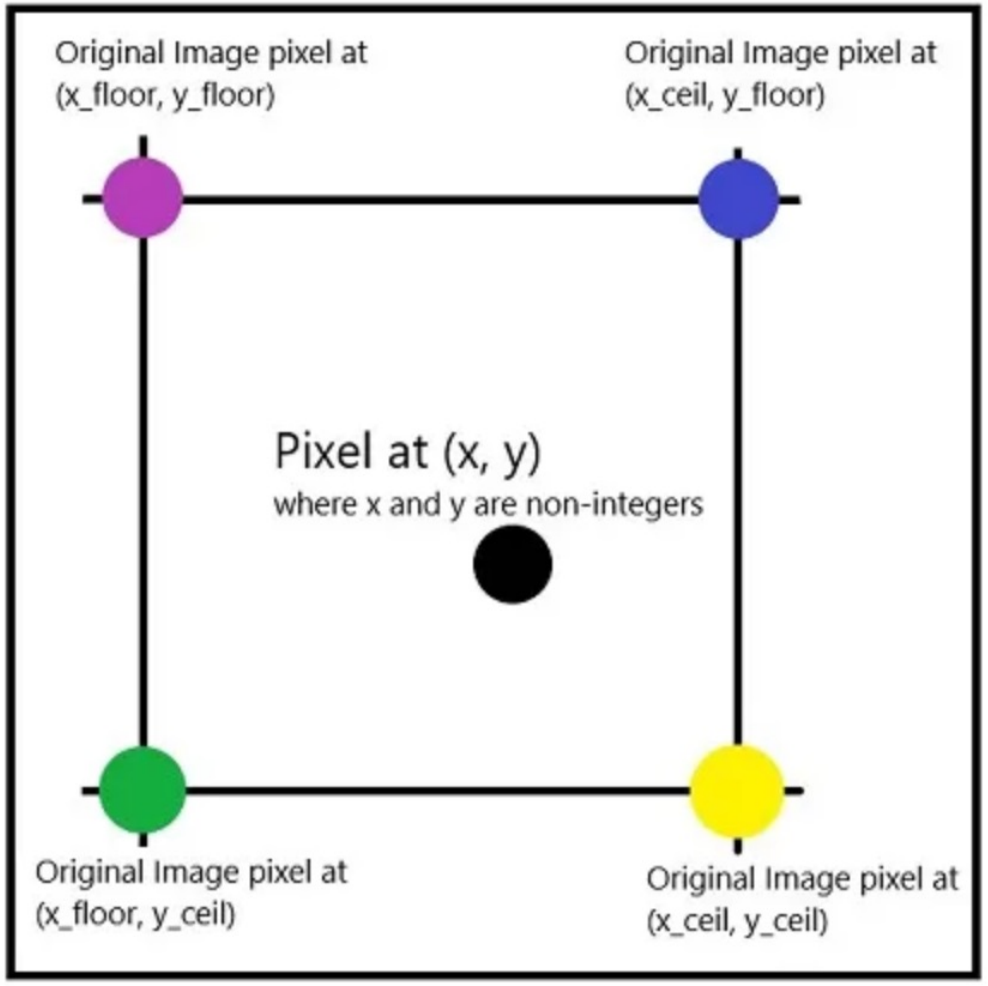


This happens when either x or y is an integer resulting in $q=0$



When x is an integer, x_floor and x_ceil will have the same value as x. As a result, both $q1$ and $q2$ will be 0 and finally, q will be 0 as well

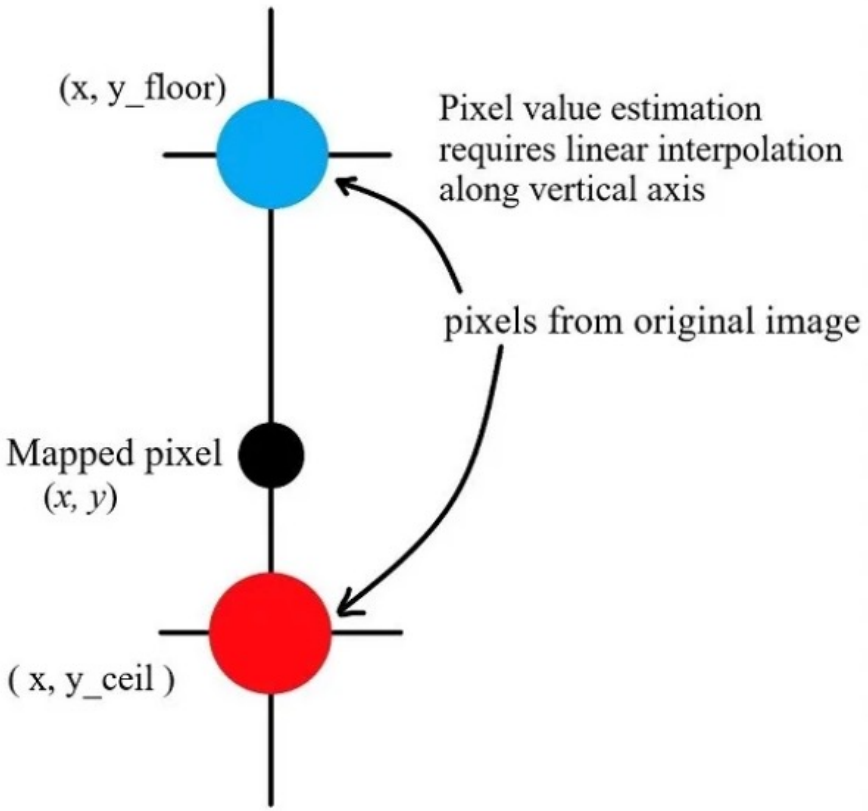
This happens when either *x* or *y* is an integer resulting in *q=0*



if *y* is an integer, *y_floor* and *y_ceil* will have the same value as *y*. Consequently, we will get non-zero values for *q1* and *q2* but *q* will be 0 since *y = y_floor = y_ceil.*

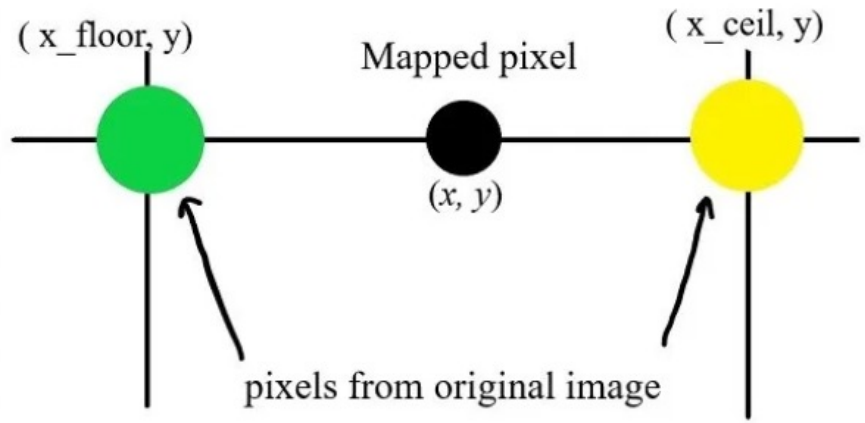
How to solve the problem

When x is an interger :



When y is an integer :

Pixel value estimation requires linear interpolation along horizontal axis.



Solution

```
if (x_ceil == x_floor) and (y_ceil == y_floor):
  q = original_img[int(x)][int(y)]
elif (x_ceil == x_floor):
  q1 = original_img[int(x)][int(y_floor)]
  q2 = original_img[int(x)][int(y_ceil)]
  q = q1 * (y_ceil - y) + q2 * (y - y_floor)
elif (y_ceil == y_floor):
  q1 = original_img[int(x_floor)][int(y)]
  q2 = original_img[int(x_ceil)][int(y)]
  q = (q1 * (x_ceil - x)) + (q2  * (x - x_floor))
else:
  v1 = original_img[x_floor][y_floor]
  v2 = original_img[x_ceil][y_floor]
  v3 = original_img[x_floor][y_ceil]
  v4 = original_img[x_ceil] [y_ceil]

  q1 = v1 * (x_ceil - x) + v2 * (x - x_floor)
  q2 = v3 * (x_ceil - x) + v4 * (x - x_floor)
  q = q1 * (y_ceil - y) + q2 * (y - y_floor)
```