



Analysis and Design of Algorithms

Lecture 5



Sorting Algorithms II

Dr. Mohamed Loey
Lecturer, Faculty of Computers and Information
Benha University
Egypt

Table of Contents

Counting Sort

Radix Sort

Merge Sort

Quick Sort

Counting Sort

Counting Sort

□ Counting sort is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence.

Counting Sort

□ Algorithm:

- **Step1:** Create a count array to store the count of each unique object
- **Step2 :** Modify count array by adding the previous number.
- **Step3 :** Create output array by decrease count array

Counting Sort

- Example 1 Assume the following Array in range of 0 to 5:

1	4	3	2	3	5	2
---	---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---

0 1 2 3 4 5

0	0	0	0	0	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0 1 2 3 4 5

0	0	0	0	0	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	0	0	0	0	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0 1 2 3 4 5

0	1	0	0	0	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0 1 2 3 4 5

0	1	0	0	0	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0 1 2 3 4 5

0	1	0	0	0	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	0	0	0	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	0	0	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0 1 2 3 4 5

0	1	0	0	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	0	0	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	0	1	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0 1 2 3 4 5

0	1	0	1	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	0	1	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	1	1	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0 1 2 3 4 5

0	1	1	1	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	1	2	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0 1 2 3 4 5

0	1	1	2	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---

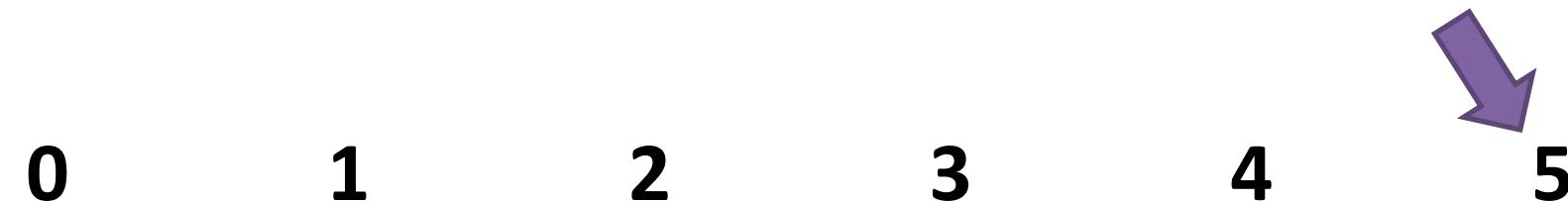


0	1	1	2	1	0
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	1	2	1	1
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



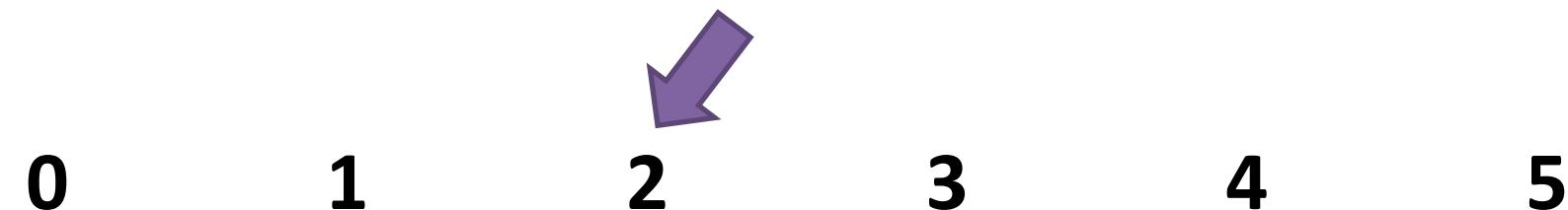
0 1 2 3 4 5

0	1	1	2	1	1
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	1	2	1	1
---	---	---	---	---	---

Counting Sort

- Create a count array to store the count of each unique object:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	2	2	1	1
---	---	---	---	---	---

Counting Sort

- Modify count array by adding the previous number :

1	4	3	2	3	5	2
---	---	---	---	---	---	---

0 1 2 3 4 5

0	1	2	2	1	1
---	---	---	---	---	---

Counting Sort

- Modify count array by adding the previous number :

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	2	2	1	1
---	---	---	---	---	---

Counting Sort

- Modify count array by adding the previous number :

1	4	3	2	3	5	2
---	---	---	---	---	---	---

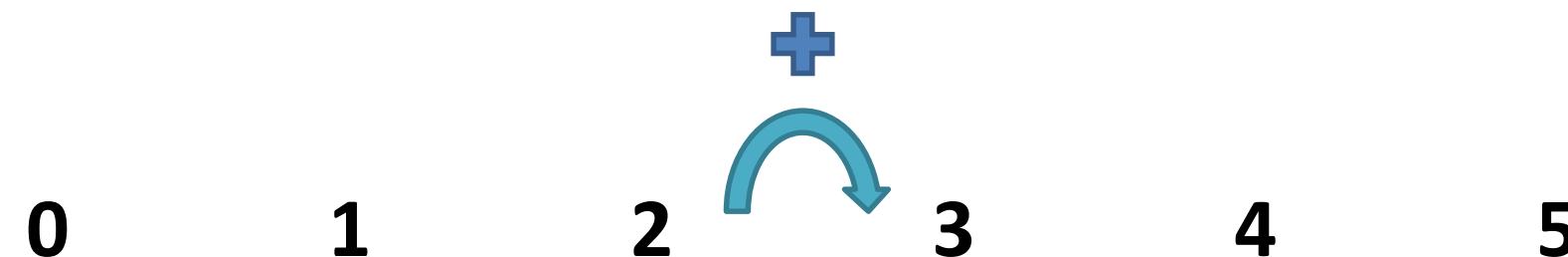


0	1	3	2	1	1
---	---	---	---	---	---

Counting Sort

- Modify count array by adding the previous number :

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	3	5	1	1
---	---	---	---	---	---

Counting Sort

- Modify count array by adding the previous number :

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	3	5	6	1
---	---	---	---	---	---

Counting Sort

- Modify count array by adding the previous number :

1	4	3	2	3	5	2
---	---	---	---	---	---	---



0	1	3	5	6	7
---	---	---	---	---	---

Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:

1	4	3	2	3	5	2
---	---	---	---	---	---	---

0 1 2 3 4 5

0	1	3	5	6	7
---	---	---	---	---	---

Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:

1	4	3	2	3	5	2
---	---	---	---	---	---	---

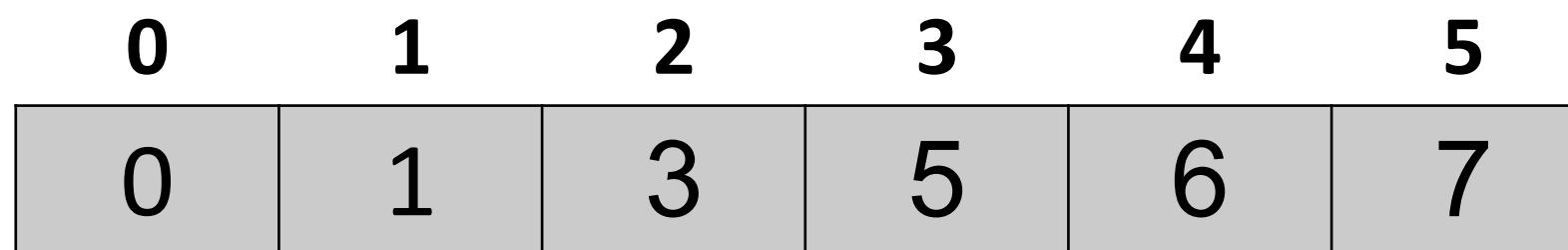
0	1	2	3	4	5
0	1	3	5	6	7

1 2 3 4 5 6 7



Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:

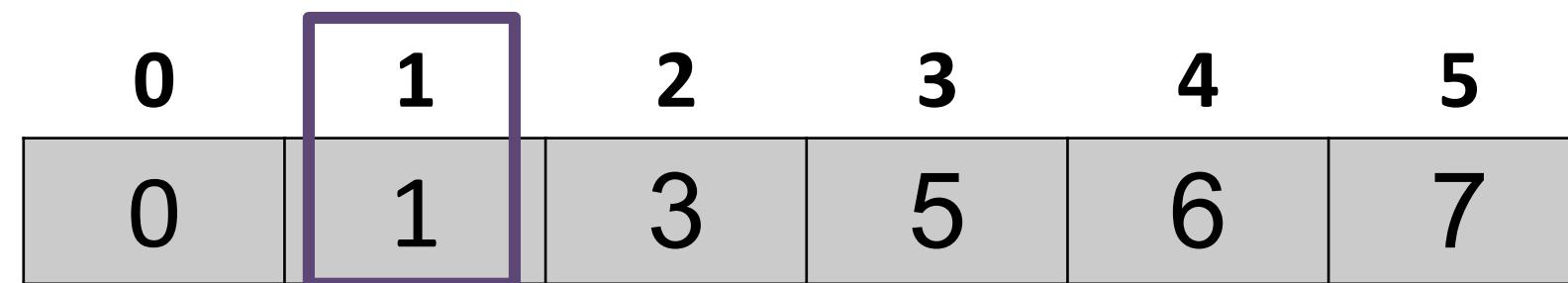


1 2 3 4 5 6 7



Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:

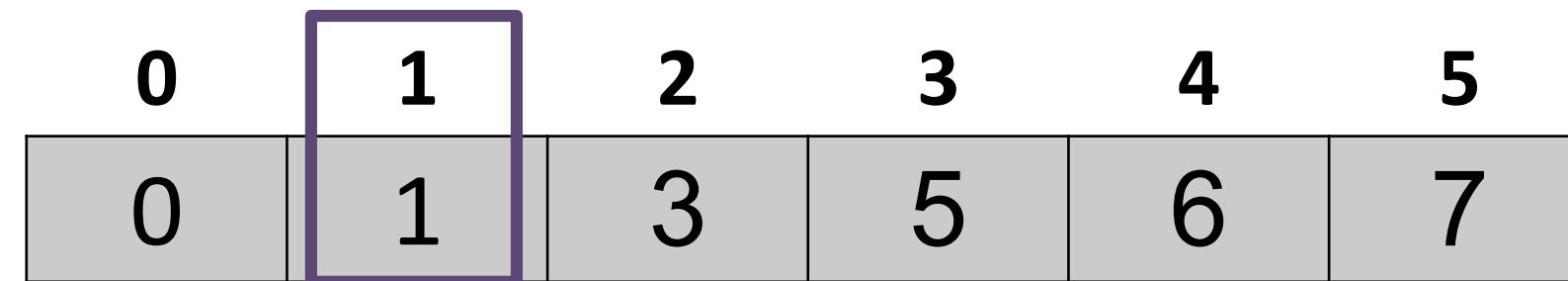


1 2 3 4 5 6 7



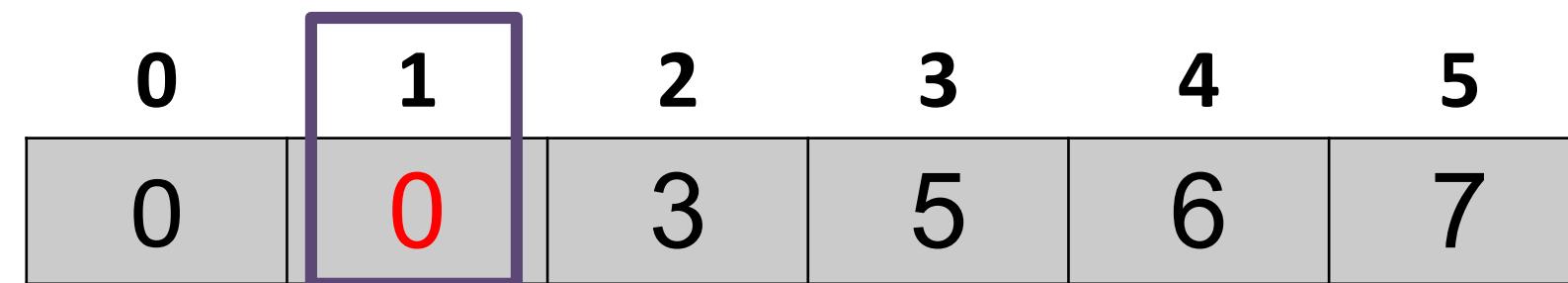
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



Counting Sort

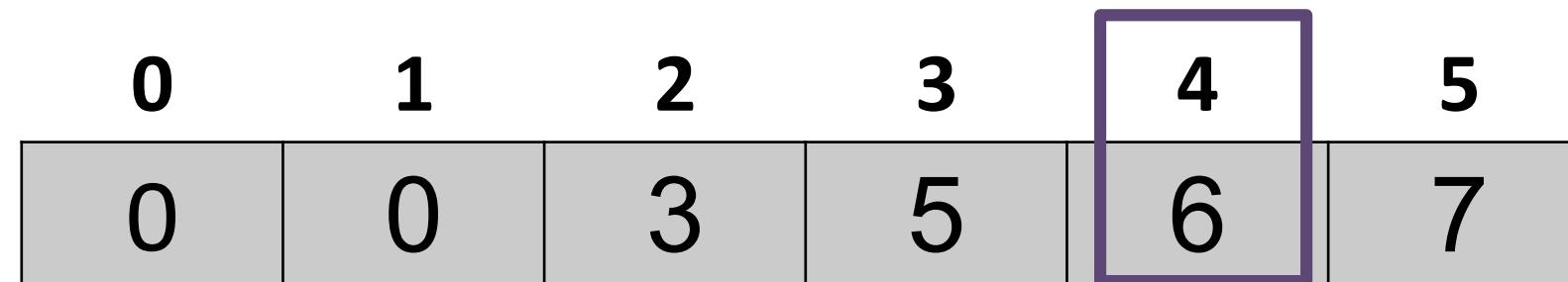
- Output each object from the input sequence followed by decreasing its count by 1:



Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:

1	4	3	2	3	5	2
---	---	---	---	---	---	---

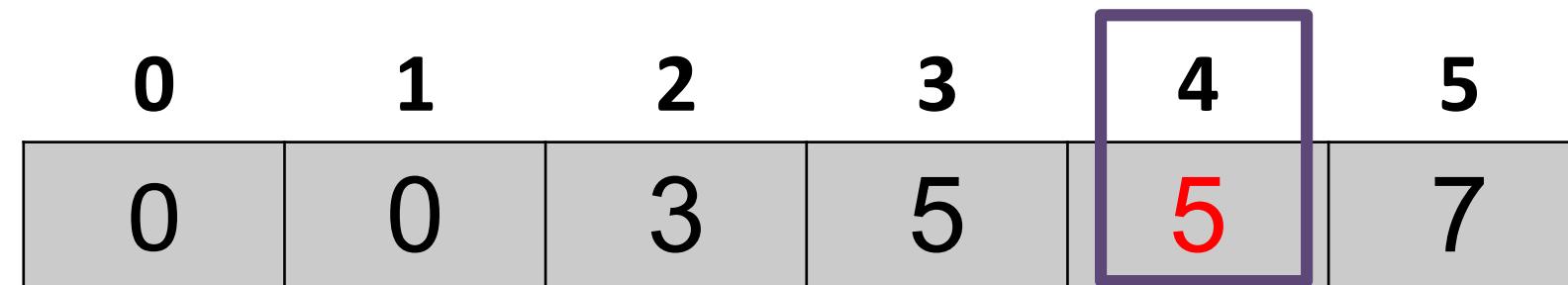


1 2 3 4 5 6 7

1						
---	--	--	--	--	--	--

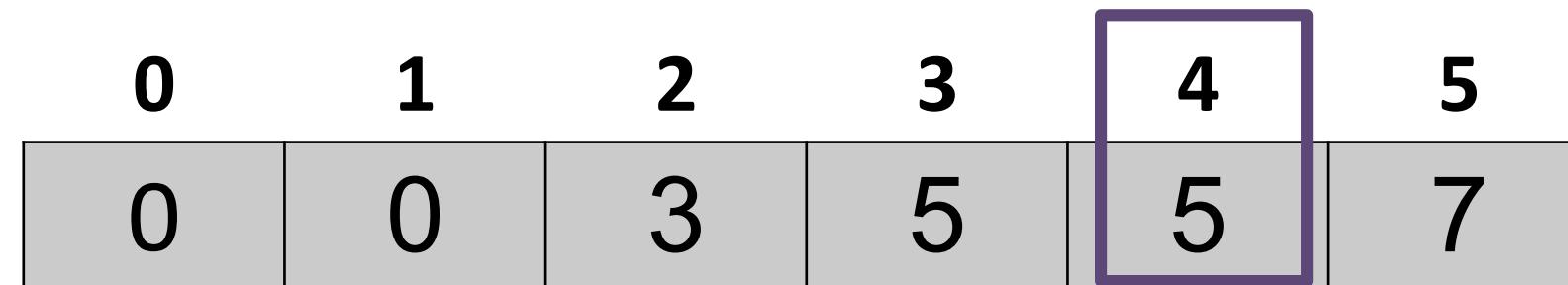
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



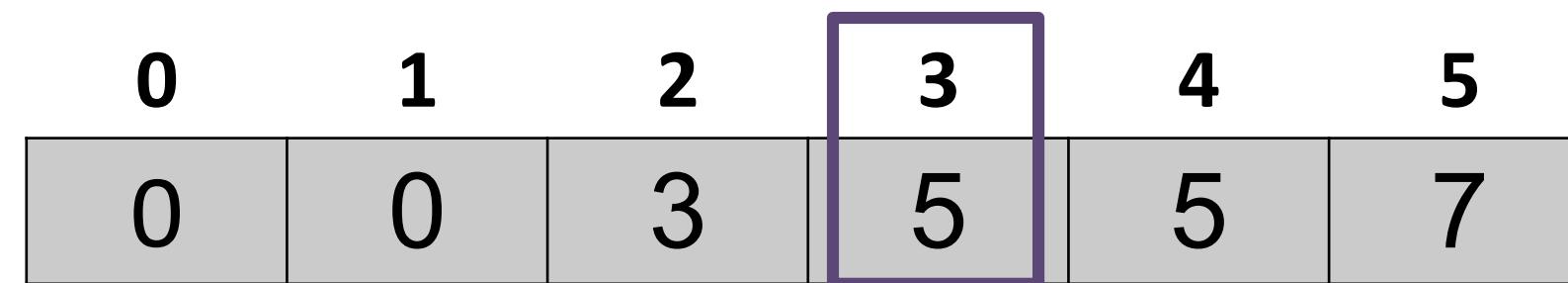
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



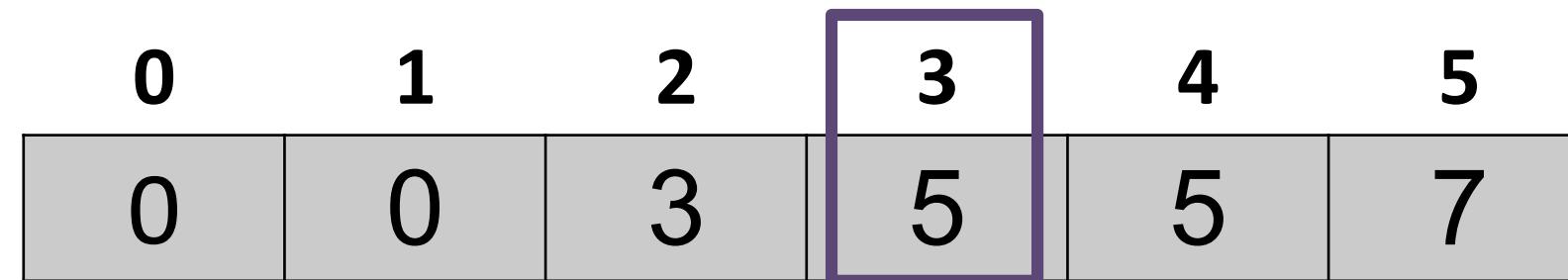
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



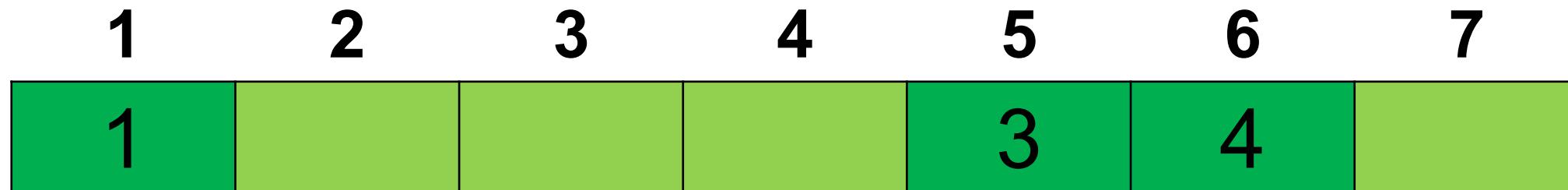
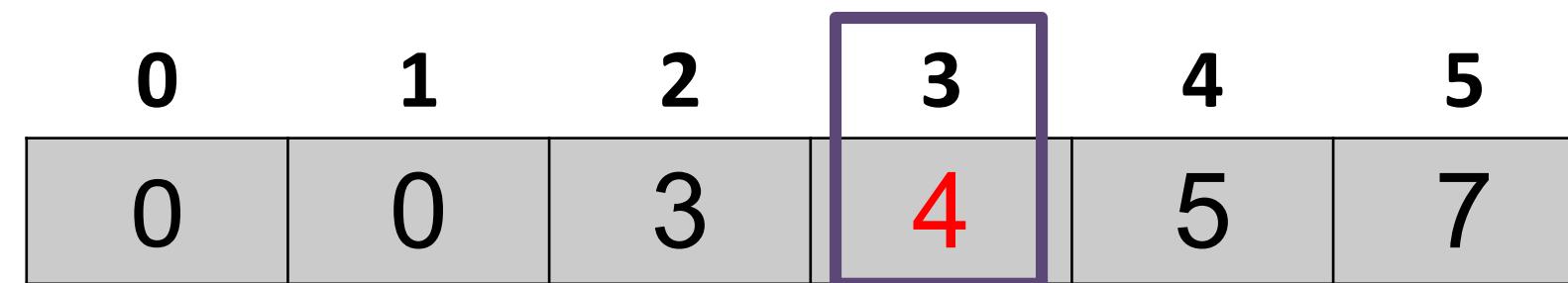
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



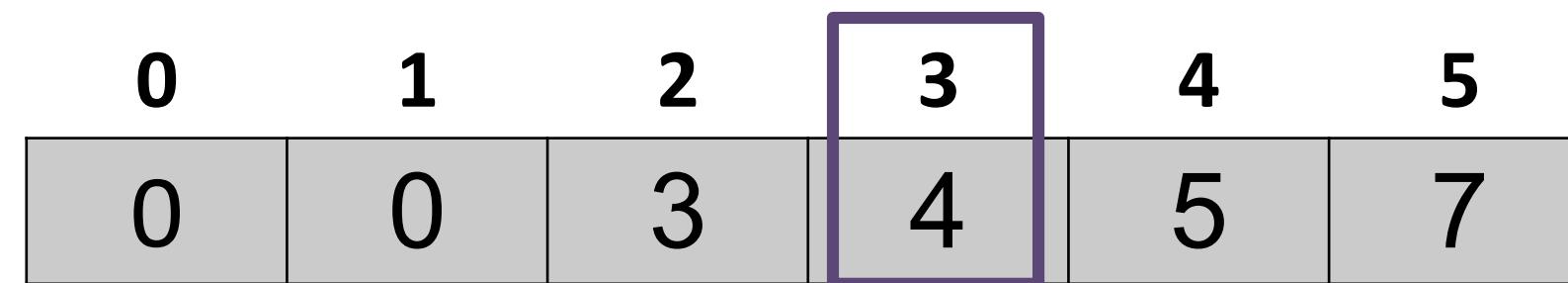
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



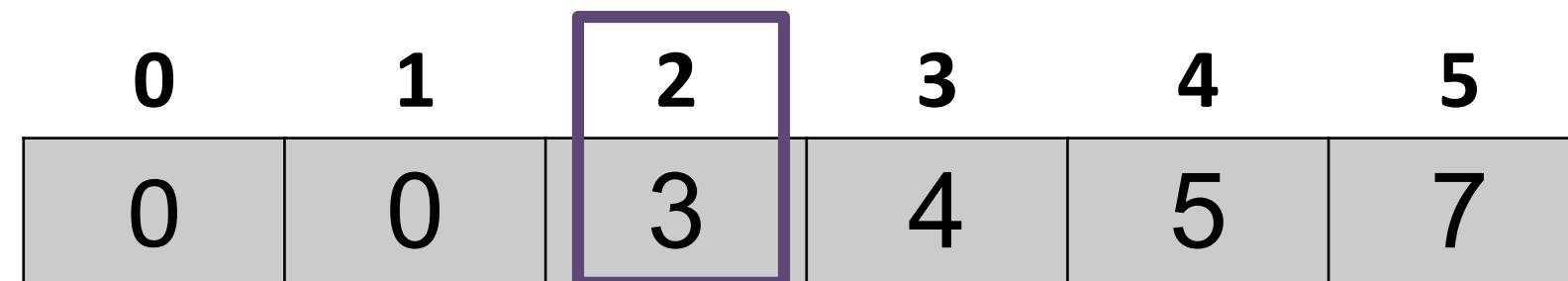
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



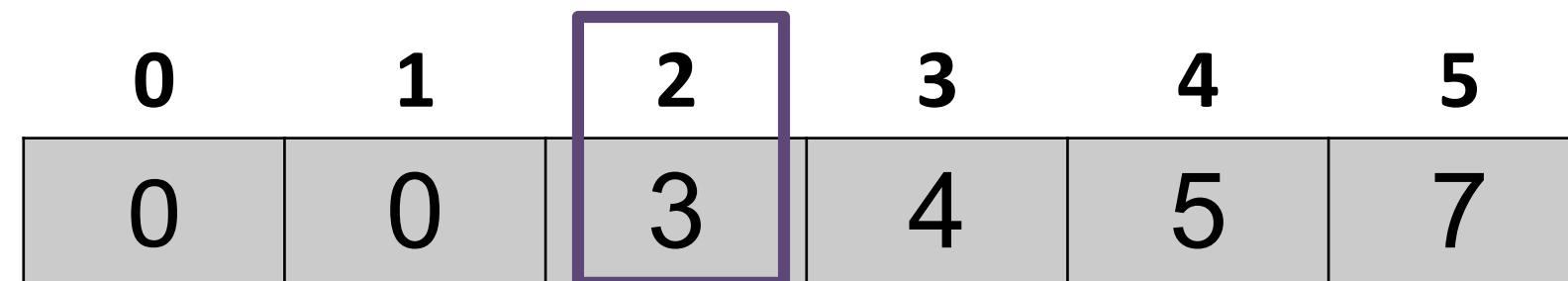
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



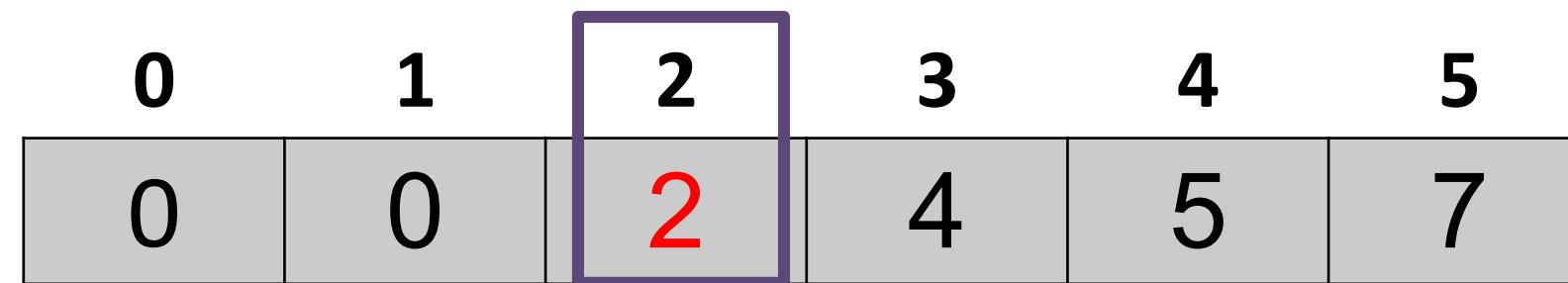
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



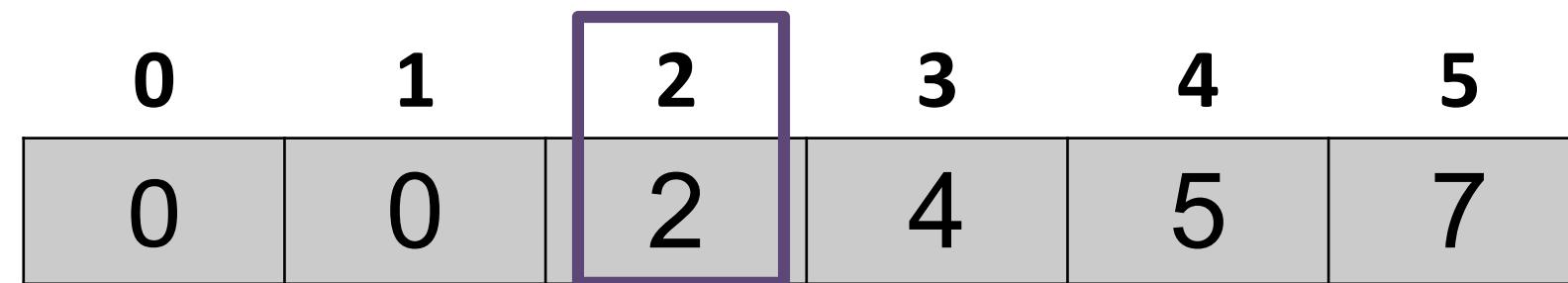
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



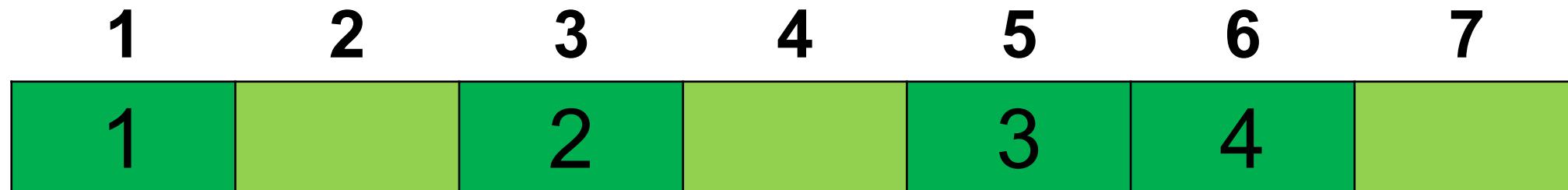
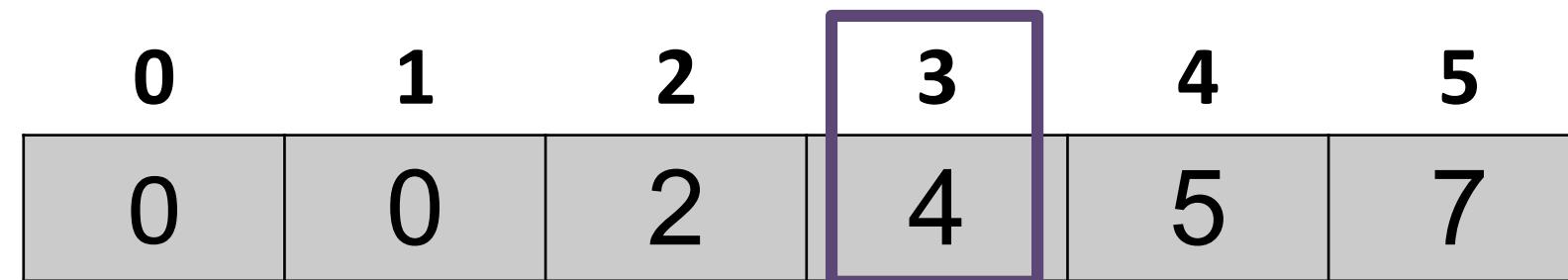
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



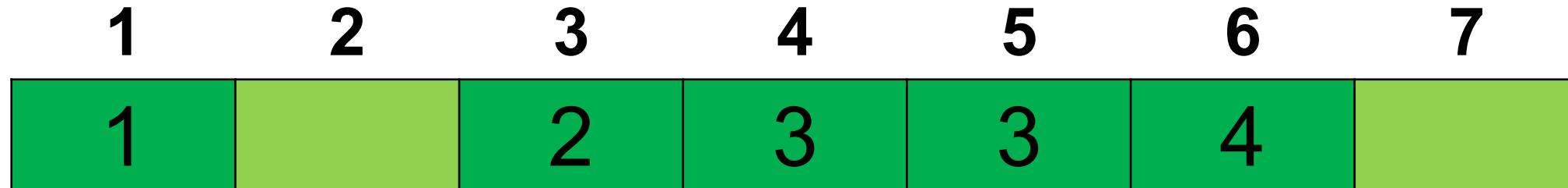
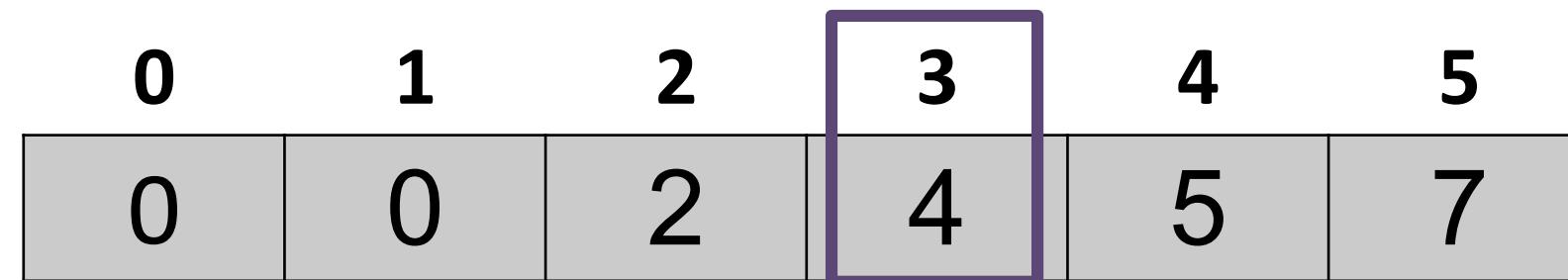
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



Counting Sort

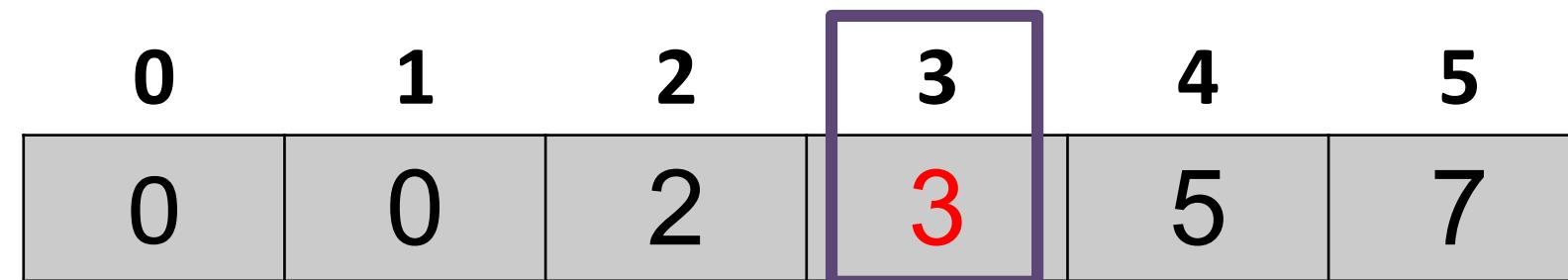
- Output each object from the input sequence followed by decreasing its count by 1:



Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:

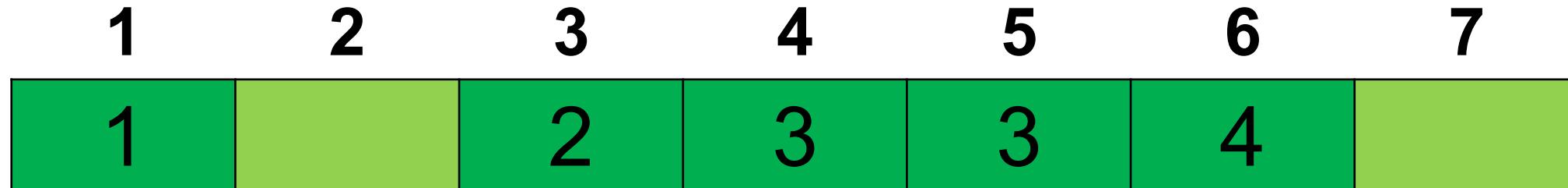
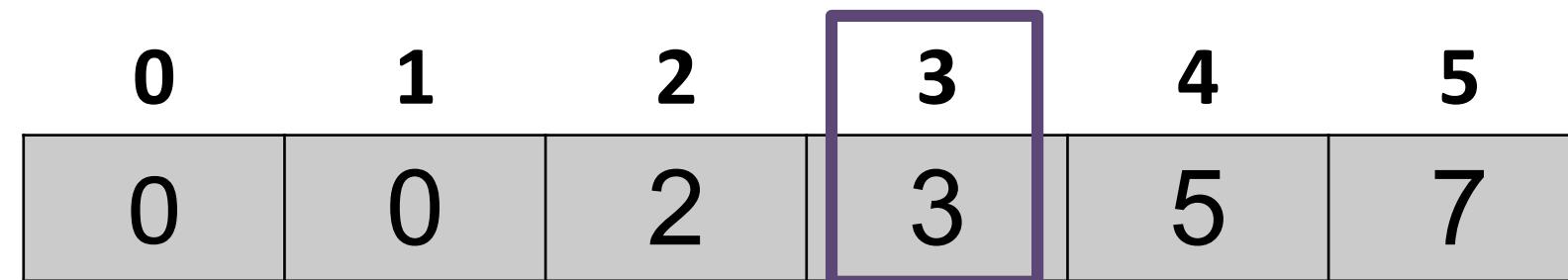
1	4	3	2	3	5	2
---	---	---	---	---	---	---



1	2	3	4	5	6	7
1		2	3	3	4	

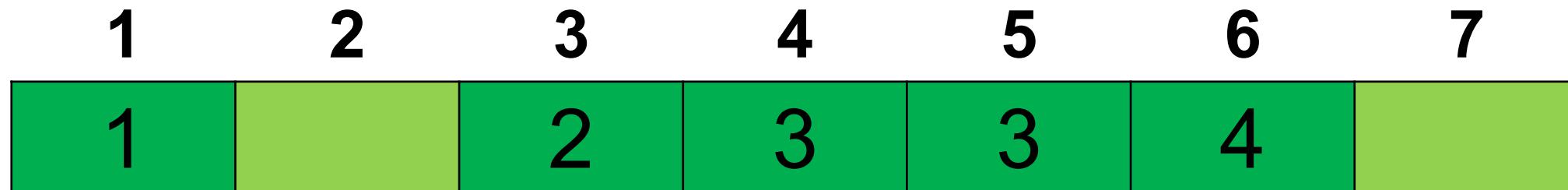
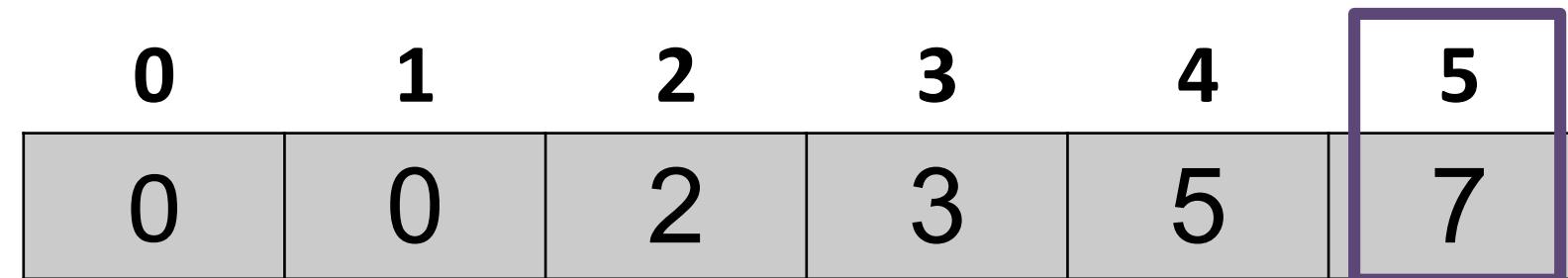
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



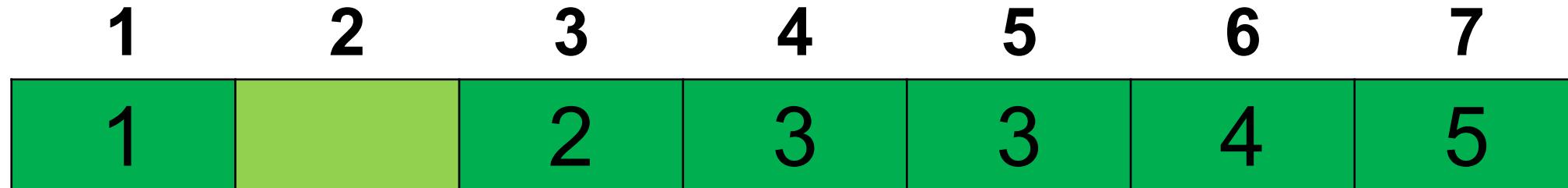
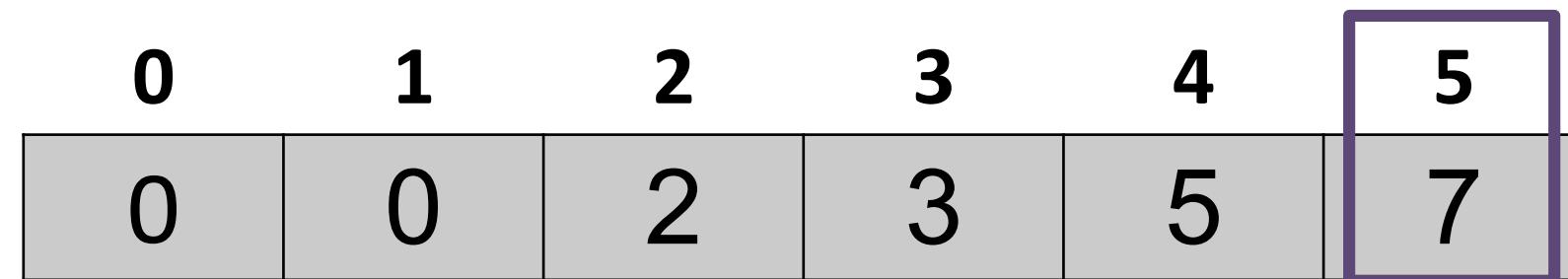
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



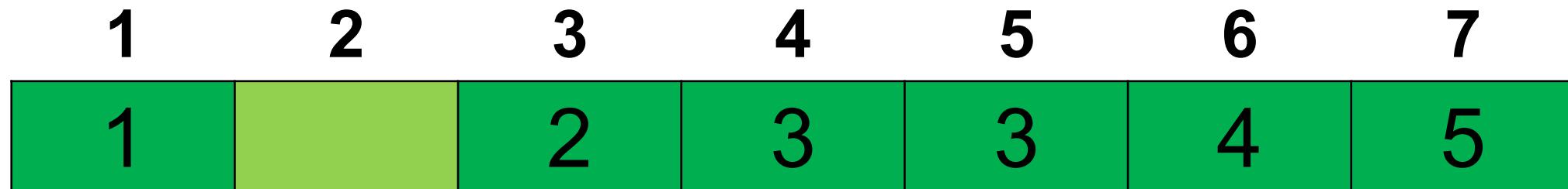
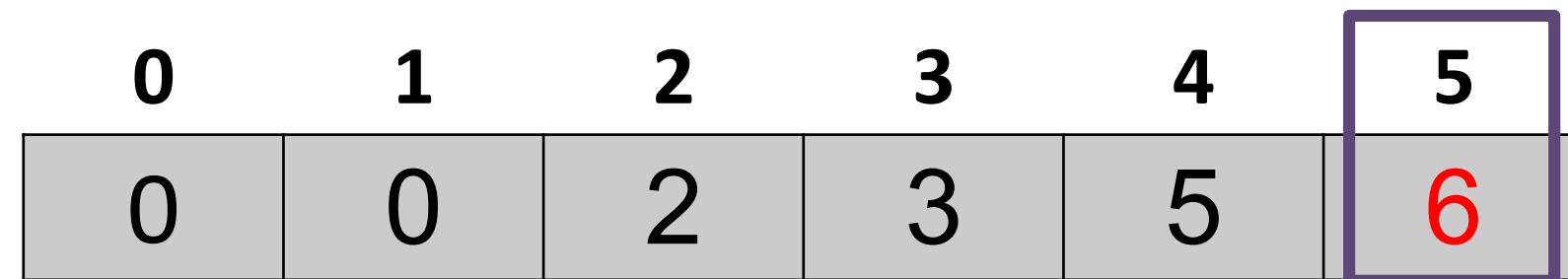
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



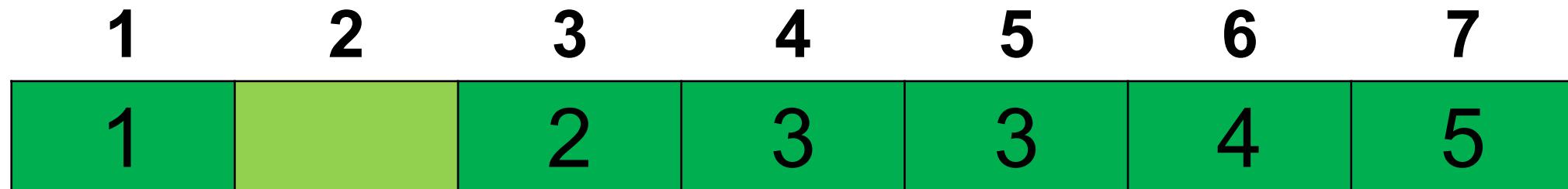
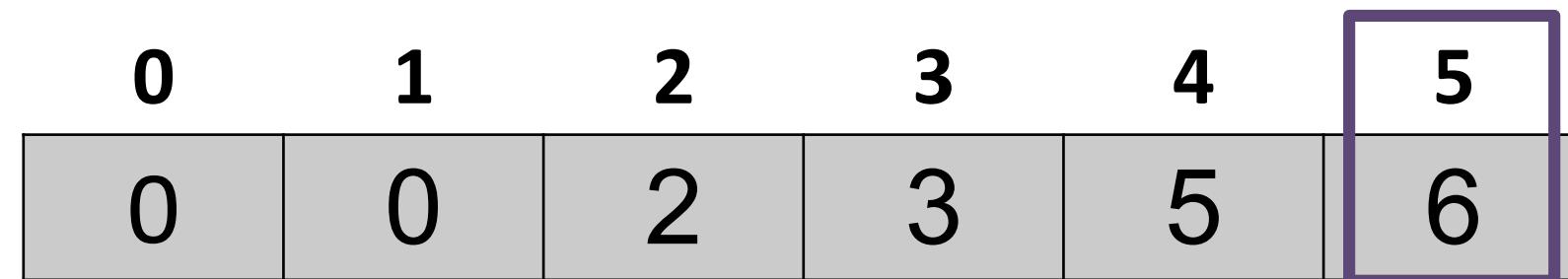
Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:



Counting Sort

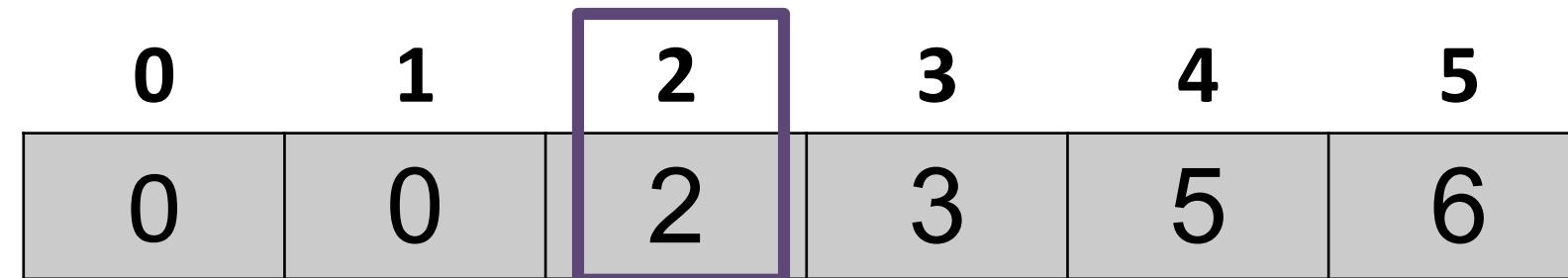
- Output each object from the input sequence followed by decreasing its count by 1:



Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:

1	4	3	2	3	5	2
---	---	---	---	---	---	---

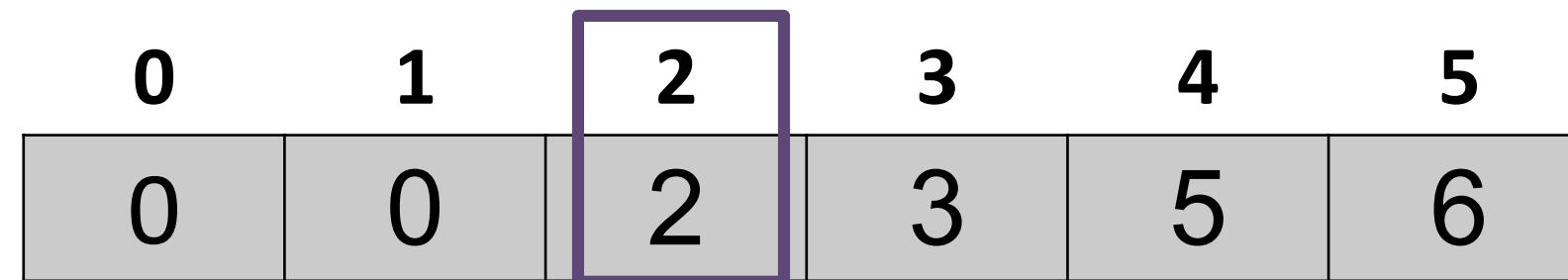


1	2	3	4	5	6	7
1		2	3	3	4	5

Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:

1	4	3	2	3	5	2
---	---	---	---	---	---	---

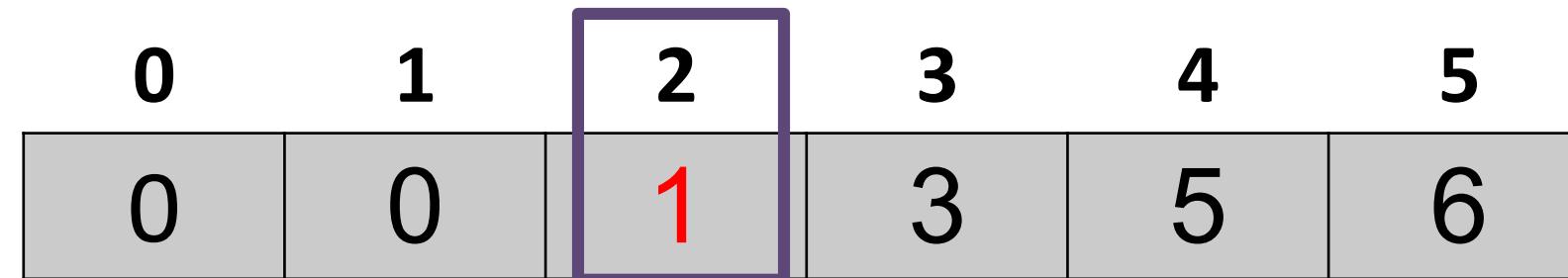


1	2	3	4	5	6	7
1	2	2	3	3	4	5

Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:

1	4	3	2	3	5	2
---	---	---	---	---	---	---

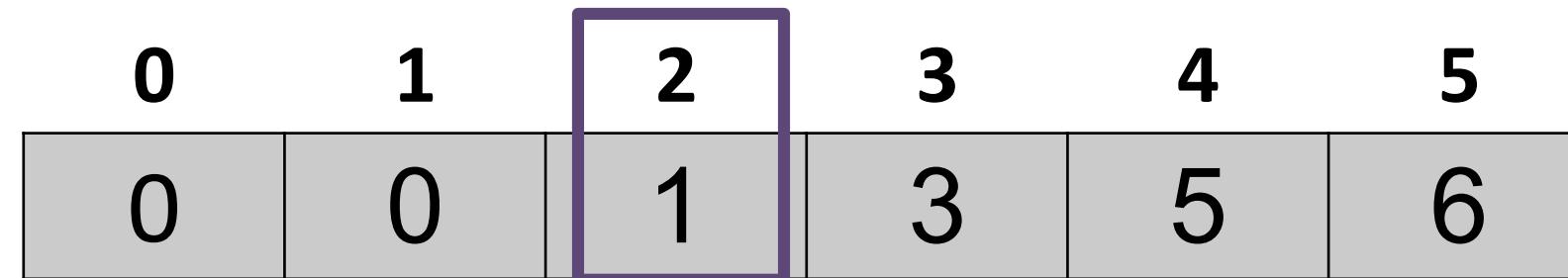


1	2	3	4	5	6	7
1	2	2	3	3	4	5

Counting Sort

- Output each object from the input sequence followed by decreasing its count by 1:

1	4	3	2	3	5	2
---	---	---	---	---	---	---



1	2	3	4	5	6	7
1	2	2	3	3	4	5

Counting Sort

- Array is now sorted

1	2	2	3	3	4	5
---	---	---	---	---	---	---

Counting Sort

Example 2:

Range=[0-4]

Count	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>2</td><td>1</td><td>1</td><td>1</td></tr></table>	0	1	2	3	4	0	2	1	1	1
0	1	2	3	4							
0	2	1	1	1							
Add	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>2</td><td>3</td><td>4</td><td>5</td></tr></table>	0	1	2	3	4	0	2	3	4	5
0	1	2	3	4							
0	2	3	4	5							
Output	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td></td><td></td><td>2</td><td></td><td></td></tr></table>	1	2	3	4	5			2		
1	2	3	4	5							
		2									
Reduce	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>2</td><td>2</td><td>4</td><td>5</td></tr></table>	0	1	2	3	4	0	2	2	4	5
0	1	2	3	4							
0	2	2	4	5							
Output	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td></td><td>2</td><td></td><td></td><td>4</td></tr></table>	1	2	3	4	5		2			4
1	2	3	4	5							
	2			4							

Reduce	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>2</td><td>2</td><td>4</td><td>4</td></tr></table>	0	1	2	3	4	0	2	2	4	4
0	1	2	3	4							
0	2	2	4	4							
Output	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td></td><td>1</td><td>2</td><td></td><td>4</td></tr></table>	0	1	2	3	4		1	2		4
0	1	2	3	4							
	1	2		4							
Reduce	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>1</td><td>2</td><td>4</td><td>4</td></tr></table>	0	1	2	3	4	0	1	2	4	4
0	1	2	3	4							
0	1	2	4	4							
Output	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	0	1	2	3	4	1	1	2	3	4
0	1	2	3	4							
1	1	2	3	4							
Sorted	<table border="1"><tr><td>1</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	1	2	3	4	1	1	2	3	4
1	1	2	3	4							
1	1	2	3	4							

Counting Sort

□ Python

Code

```
def countSort(arr):
    outputarr = [0 for i in range(127)]
    countarr = [0 for i in range(127)]

    for i in arr:
        countarr[ord(i)] += 1

    for i in range(127):
        countarr[i] += countarr[i-1]

    for i in range(len(arr)):
        outputarr[countarr[ord(arr[i])]-1] = arr[i]
        countarr[ord(arr[i])] -= 1

    return outputarr[0:len(arr)]
```

Counting Sort

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(72	48	110	H	104	68	150	h
9	9	11		41	29	51)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	-	127	7F	177	

Counting Sort

```
arr = "mynameiskhan"  
ans = countSort(arr)  
print("".join(ans))
```

Counting Sort

- **Time Complexity:** $O(n+k)$ where n is the number of elements in input array, and k is the range of input.
- Example of worst case
 - Range between 1 to 10K

10	5	10k	5k	200
----	---	-----	----	-----

Radix Sort

Radix Sort

- ❑ Radix sort is an algorithm that sorts numbers by processing digits of each number either starting from the least significant digit (LSD) or starting from the most significant digit (MSD).
- ❑ The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.

□ Algorithm:

- **Step1:** Take the least significant digit of each element
- **Step2 :** Sort the list of elements based on that digit
- **Step3 :** Repeat the sort with each more significant digit

Radix Sort

- ❑ Assume the following Array:

170	45	75	90	802	24	2	66
-----	----	----	----	-----	----	---	----

Radix Sort

- The Sorted list will appear after three steps

170	45	75	90	802	24	2	66
-----	----	----	----	-----	----	---	----

170	90	802	2	24	45	75	66
-----	----	-----	---	----	----	----	----

802	2	24	45	66	170	75	90
-----	---	----	----	----	-----	----	----

2	24	45	66	75	90	170	802
---	----	----	----	----	----	-----	-----

Radix Sort

- Step 1: Sorting by least significant digit (1s place)

1 <u>70</u>	4 <u>5</u>	7 <u>5</u>	9 <u>0</u>	80 <u>2</u>	2 <u>4</u>	2	6 <u>6</u>
-------------	------------	------------	------------	-------------	------------	---	------------

170	90	802	2	24	45	75	66
-----	----	-----	---	----	----	----	----

Radix Sort

- Step2: Sorting by next digit (10s place)

1 <u>7</u> 0	<u>9</u> 0	8 <u>0</u> 2	2	<u>2</u> 4	<u>4</u> 5	<u>7</u> 5	<u>6</u> 6
--------------	------------	--------------	---	------------	------------	------------	------------

802	2	24	45	66	170	75	90
-----	---	----	----	----	-----	----	----

Radix Sort

- Step3: Sorting by most significant digit (100s place)

802	2	24	45	66	170	75	90
-----	---	----	----	----	-----	----	----

2	24	45	66	75	90	170	802
---	----	----	----	----	----	-----	-----

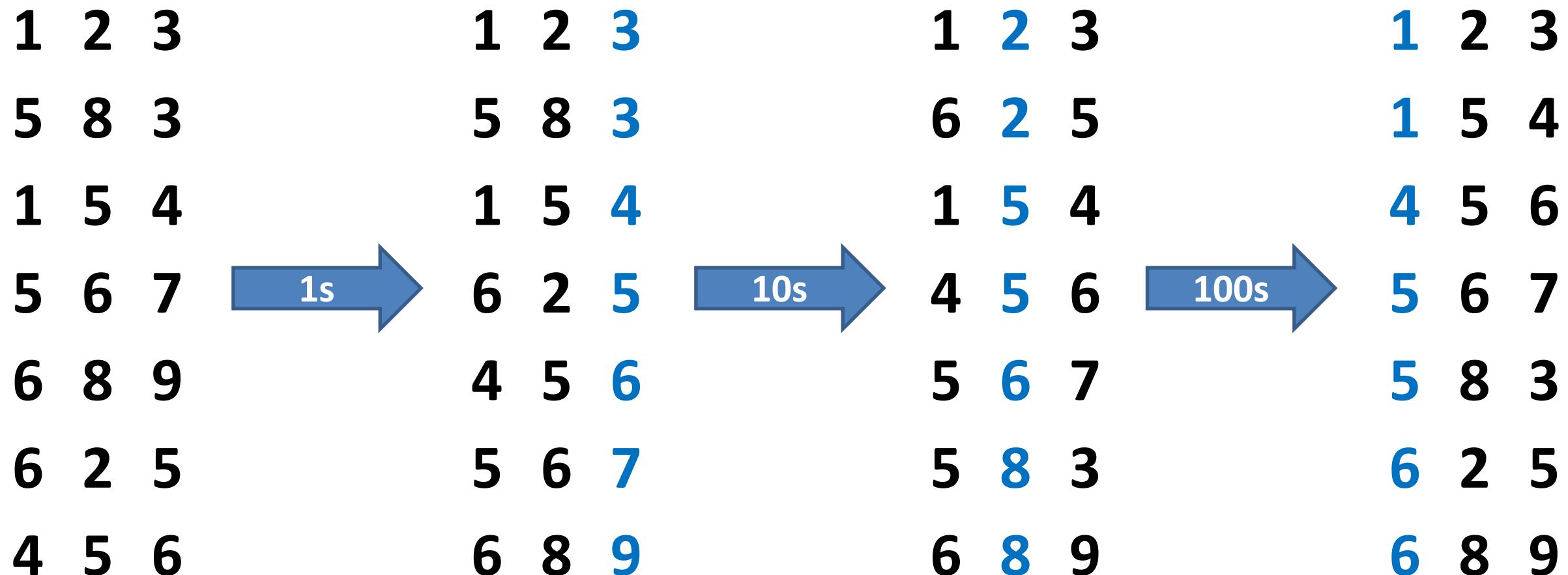
Radix Sort

- Array is now sorted

2	24	45	66	75	90	170	802
---	----	----	----	----	----	-----	-----

Radix Sort

Example 2



Radix Sort

```
def countingSort(arr, count1):
    n = len(arr)
    output = [0] * (n)
    count = [0] * (10)
    for i in range(0, n):
        index = (arr[i]/count1)
        count[ int((index)%10) ] += 1

    for i in range(1,10):
        count[i] += count[i-1]

    i = n-1
    while i>=0:
        index = (arr[i]/count1)
        output[ count[ int((index)%10) ] - 1] = arr[i]
        count[ int((index)%10) ] -= 1
        i -= 1
    return output
```

Radix Sort

□ Python Code

```
def radixSort(arr):
    # Find the maximum number to know number of digits
    maxnum = max(arr)

    count = 1
    while maxnum/count > 0:
        arr=countingSort(arr,count)
        count *= 10
    return arr
```

Radix Sort

□ Python Code

```
arr = [ 170, 45, 75, 90, 802, 24, 2, 66]  
print(radixSort(arr))
```

Radix Sort

❑ **Time Complexity:** $O((n+k)*d)$ where n is the number of elements in input array, k is the range of input, and d is number of digits.

Merge Sort

Merge Sort

- ❑ Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.

❑ Algorithm:

- **Step1:** Divide the list recursively into two halves until it can no more be divided
- **Step2 :** Merge (Conquer) the smaller lists into new list in sorted order

Merge Sort

□ Assume the following Array:

85	24	63	45	17	31	96	50
----	----	----	----	----	----	----	----

Merge Sort

❑ Divide

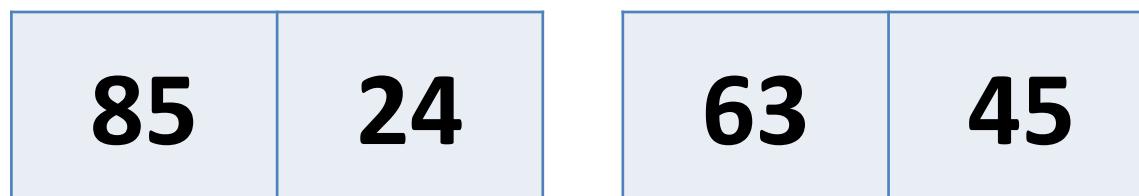
85	24	63	45	17	31	96	50
----	----	----	----	----	----	----	----

85	24	63	45
----	----	----	----

17	31	96	50
----	----	----	----

Merge Sort

❑ Divide



Merge Sort

❑ Divide



Merge Sort

❑ Sort & Merge

85

24

63

45

17

31

96

50

Merge Sort

❑ Sort & Merge



85

24

63

45

17

31

96

50

Merge Sort

□ Sort & Merge

24 85

85 24 63 45 17 31 96 50

Merge Sort

□ Sort & Merge

24 85

85 24 63 45 17 31 96 50

Merge Sort

□ Sort & Merge

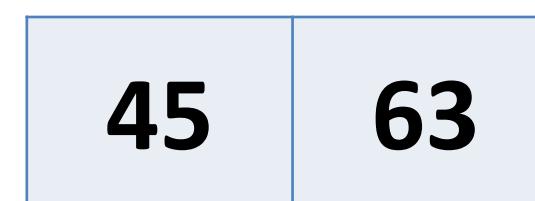
24 85

45 63

85 24 63 45 17 31 96 50

Merge Sort

□ Sort & Merge



Merge Sort

□ Sort & Merge

24	85
----	----

45	63
----	----

17	31
----	----

85	24	63	45	17	31	96	50
----	----	----	----	----	----	----	----

Merge Sort

□ Sort & Merge

24	85
----	----

45	63
----	----

17	31
----	----

--	--

85	24
----	----

63	45
----	----

17	31
----	----

96	50
----	----

Merge Sort

□ Sort & Merge

24	85
----	----

45	63
----	----

17	31
----	----

50	96
----	----

85	24
----	----

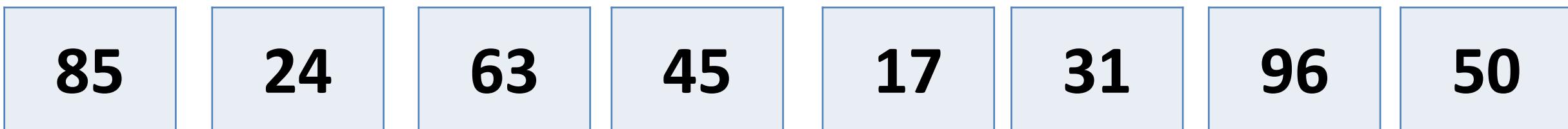
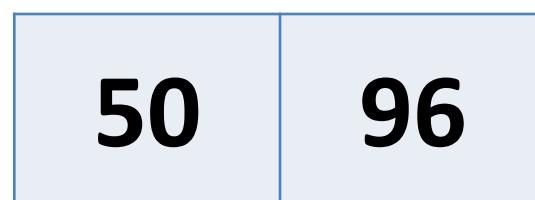
63	45
----	----

17	31
----	----

96	50
----	----

Merge Sort

□ Sort & Merge



Merge Sort

□ Sort & Merge

24

24 85

45 63

17 31

50 96

85 24 63 45 17 31 96 50

Merge Sort

□ Sort & Merge



Merge Sort

□ Sort & Merge



Merge Sort

□ Sort & Merge



Merge Sort

□ Sort & Merge

24	45	63	85
----	----	----	----

17	31	50	96
----	----	----	----

24	85
----	----

45	63
----	----

17	31
----	----

50	96
----	----

85	24	63	45	17	31	96	50
----	----	----	----	----	----	----	----

Merge Sort

□ Sort & Merge



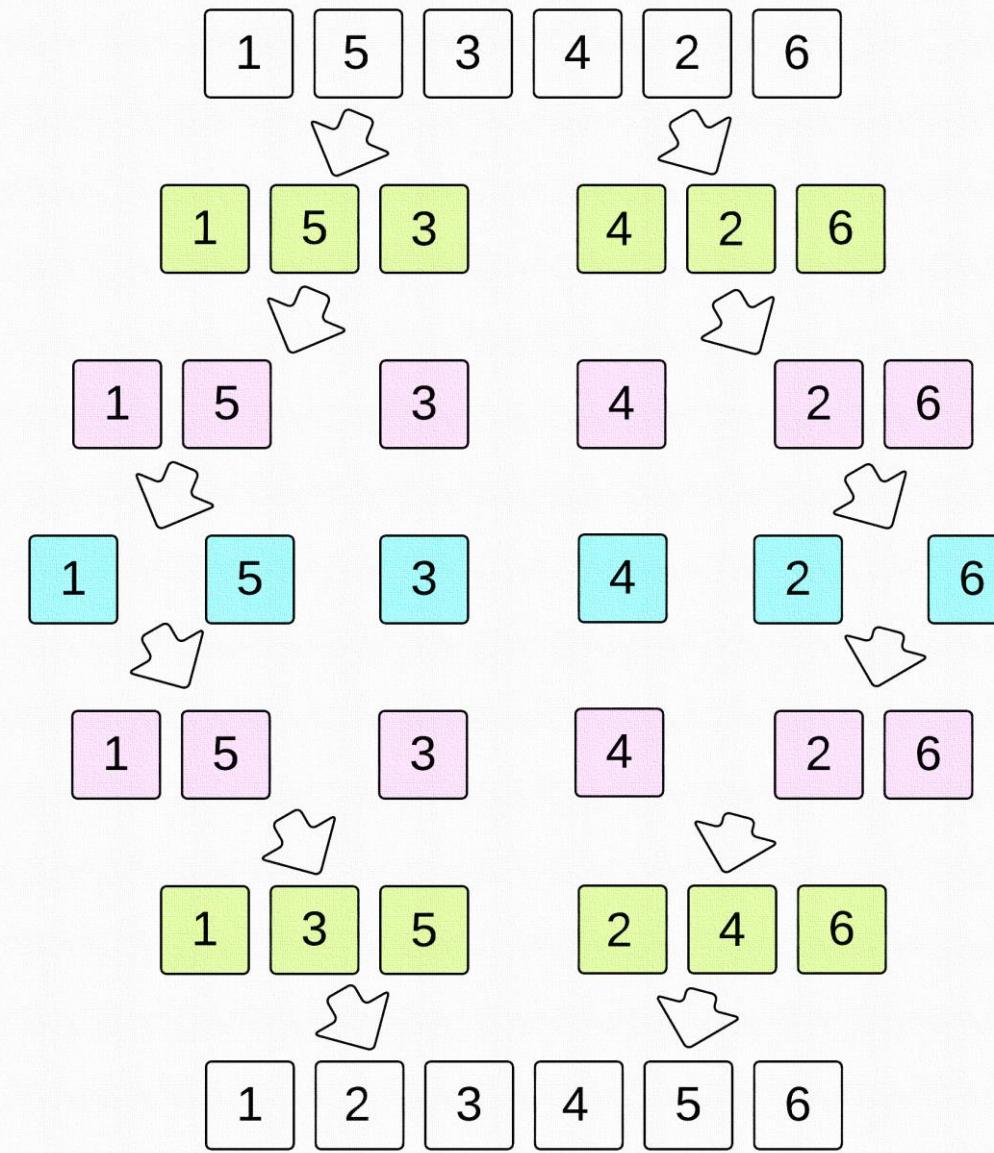
Merge Sort

□ Array is now sorted

17	24	31	45	50	63	85	96
----	----	----	----	----	----	----	----

Merge Sort

Example 2



Merge Sort

```
def mergeSort(arr):
    if len(arr) >1:
        mid = len(arr)//2 #Finding the mid of the array
        L = arr[:mid] # Dividing the array elements
        R = arr[mid:] # into 2 halves
        mergeSort(L) # Sorting the first half
        mergeSort(R) # Sorting the second half
```

Merge Sort

```
i = j = k = 0
# Copy data to temp arrays L[] and R[]
while i < len(L) and j < len(R):
    if L[i] < R[j]:
        arr[k] = L[i]
        i+=1
    else:
        arr[k] = R[j]
        j+=1
    k+=1
```

Merge Sort

```
# Checking if any element was left
while i < len(L):
    arr[k] = L[i]
    i+=1
    k+=1
while j < len(R):
    arr[k] = R[j]
    j+=1
    k+=1
```

Merge Sort

```
def mergeSort(arr):
    if len(arr) >1:
        mid = len(arr)//2 #Finding the mid of the array
        L = arr[:mid] # Dividing the array elements
        R = arr[mid:] # into 2 halves
        mergeSort(L) # Sorting the first half
        mergeSort(R) # Sorting the second half
        i = j = k = 0
        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i+=1
            else:
                arr[k] = R[j]
                j+=1
            k+=1
        # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i+=1
            k+=1
        while j < len(R):
            arr[k] = R[j]
            j+=1
            k+=1
```

Merge Sort

```
arr = [12, 11, 13, 5, 6, 7]
mergeSort(arr)
print(arr)
```

Merge Sort

❑ Time Complexity: $O(n * \log(n))$

Quick Sort

Quick Sort

- Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

Quick Sort

- There are many different versions of Quick Sort that pick pivot in different ways:
 - 1) Always pick first element as pivot.
 - 2) Always pick last element as pivot.
 - 3) Pick a random element as pivot.
 - 4) Pick median as pivot.

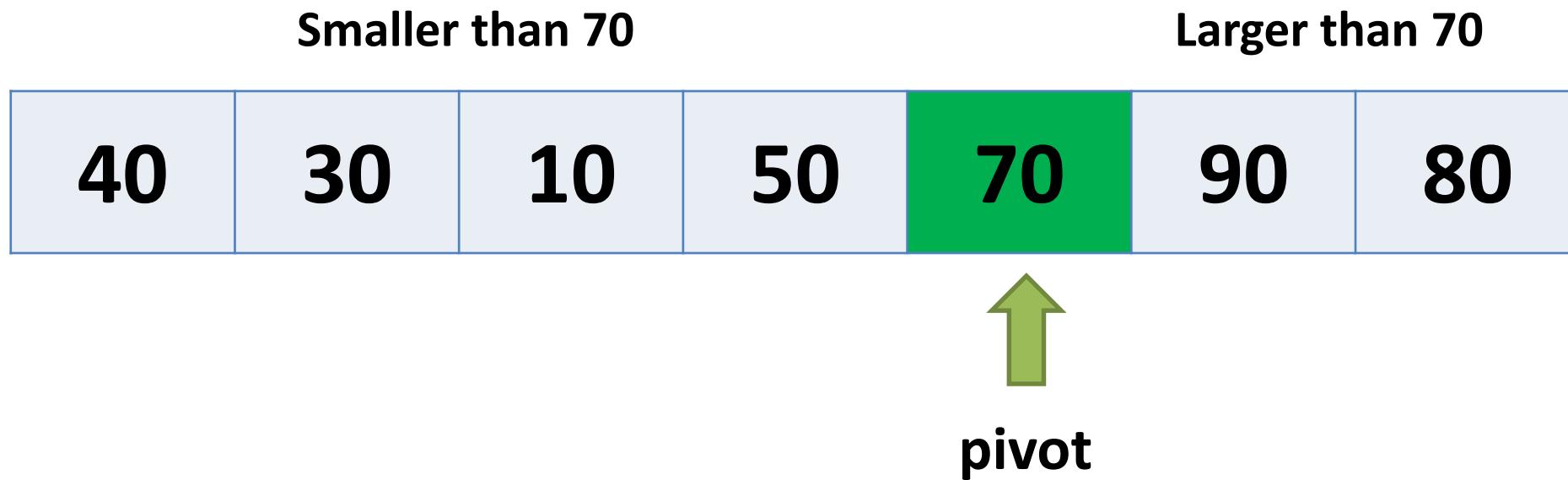
Quick Sort

❑ Algorithm:

- Pick a **pivot** element
- Partition the array into three parts:
 - ❑ First part: all elements in this part is less than the pivot.
 - ❑ Second part: the pivot itself (only one element!)
 - ❑ Third part: all elements in this part is greater than or equal to the pivot
- Apply the quicksort algorithm to the first and the third part. (recursively)

Quick Sort

□ Quick sort Objective



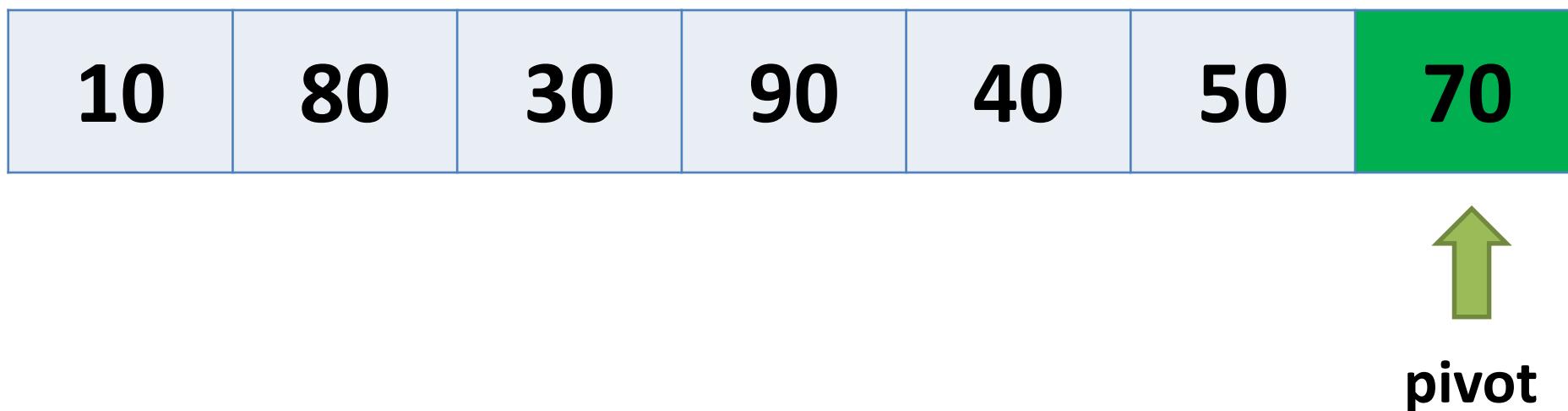
Quick Sort

□ Assume the following Array:

10	80	30	90	40	50	70
----	----	----	----	----	----	----

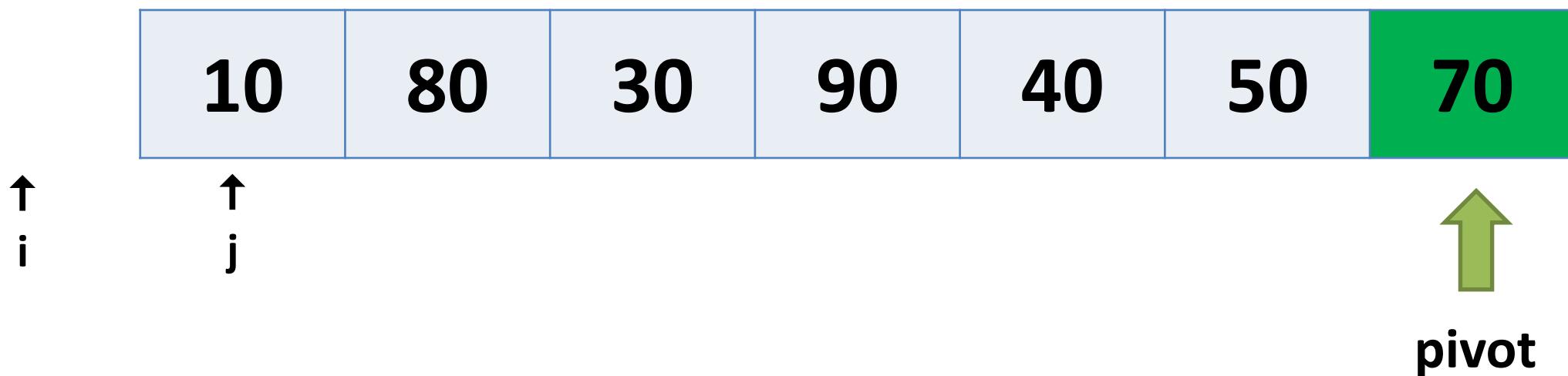
Quick Sort

- Pick a pivot



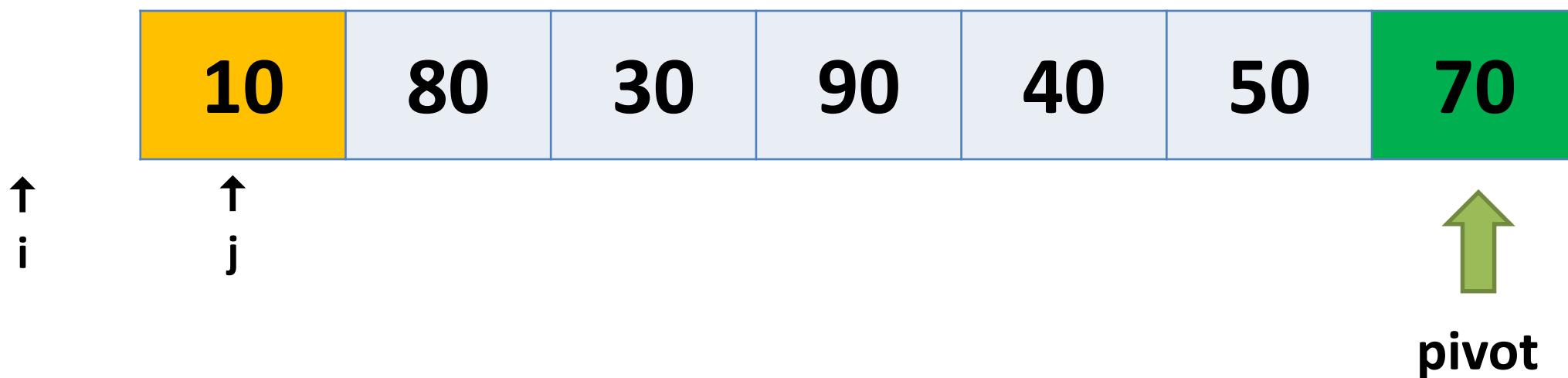
Quick Sort

- i: index of smallest element, j: Loop variable



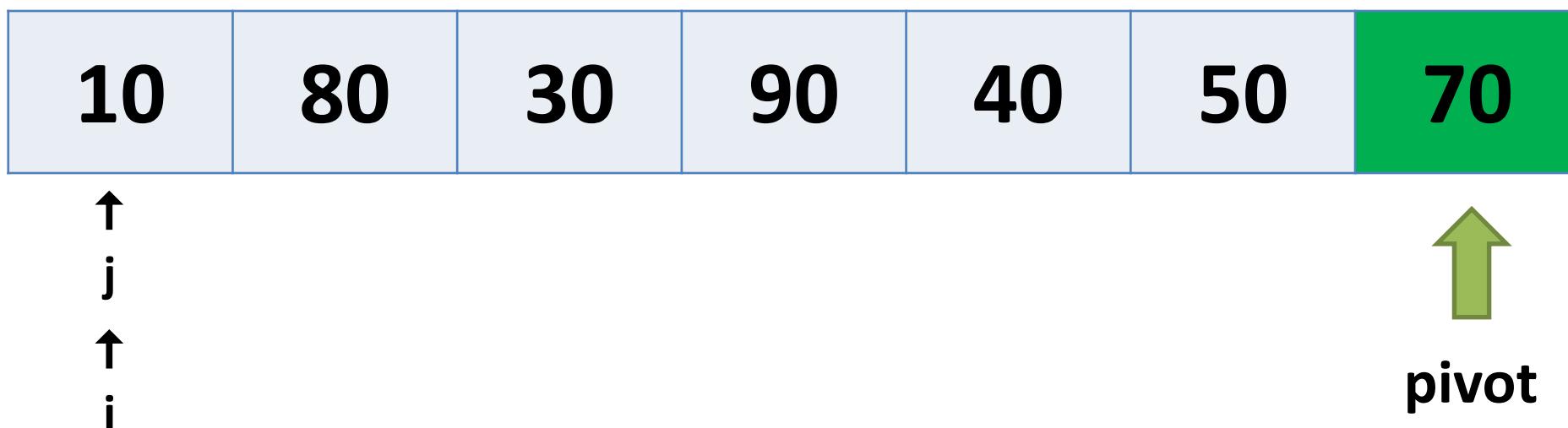
Quick Sort

❑ $\text{arr}[j] < \text{pivot}$



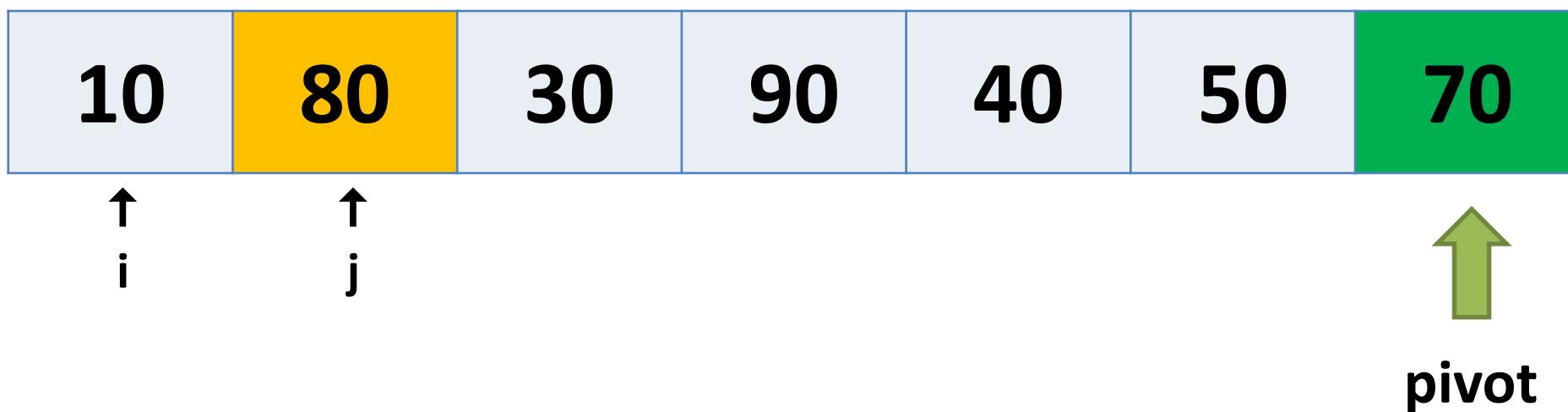
Quick Sort

- `i++`
- `swap(arr[i],arr[j])`



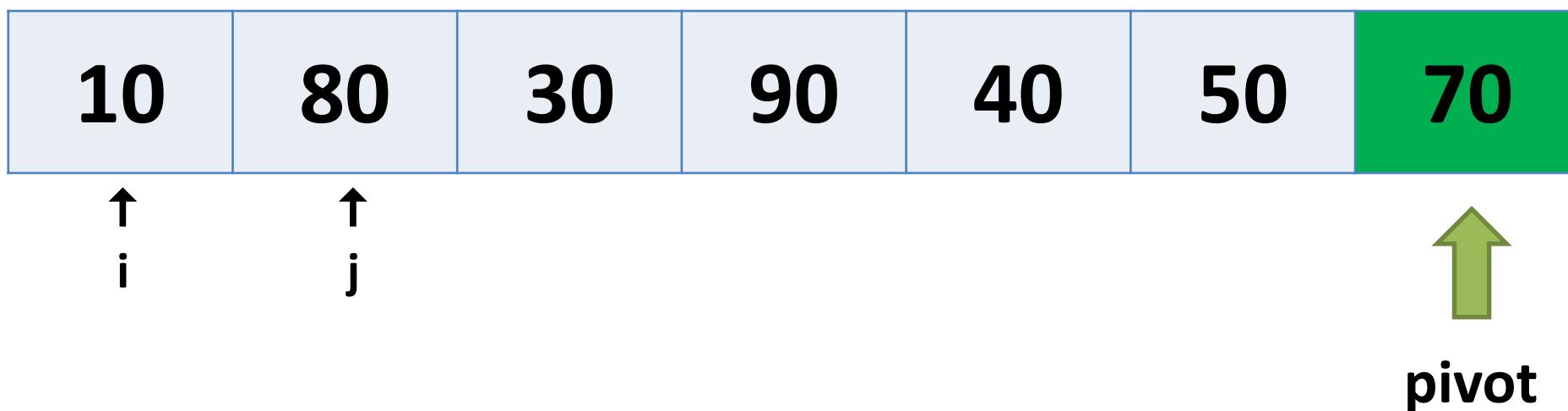
Quick Sort

❑ $\text{arr}[j] < \text{pivot}$



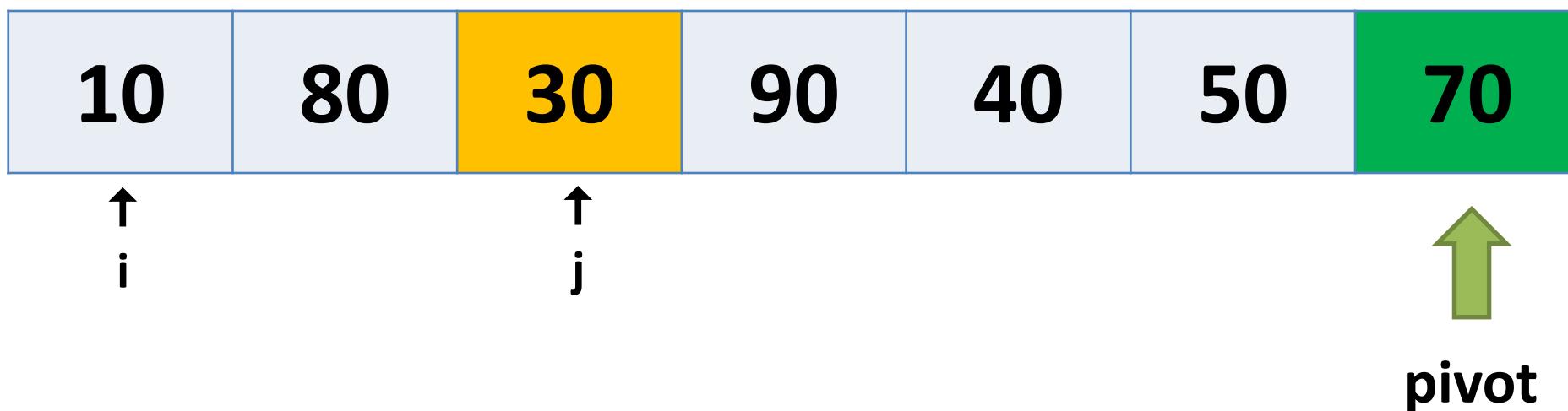
Quick Sort

- No action



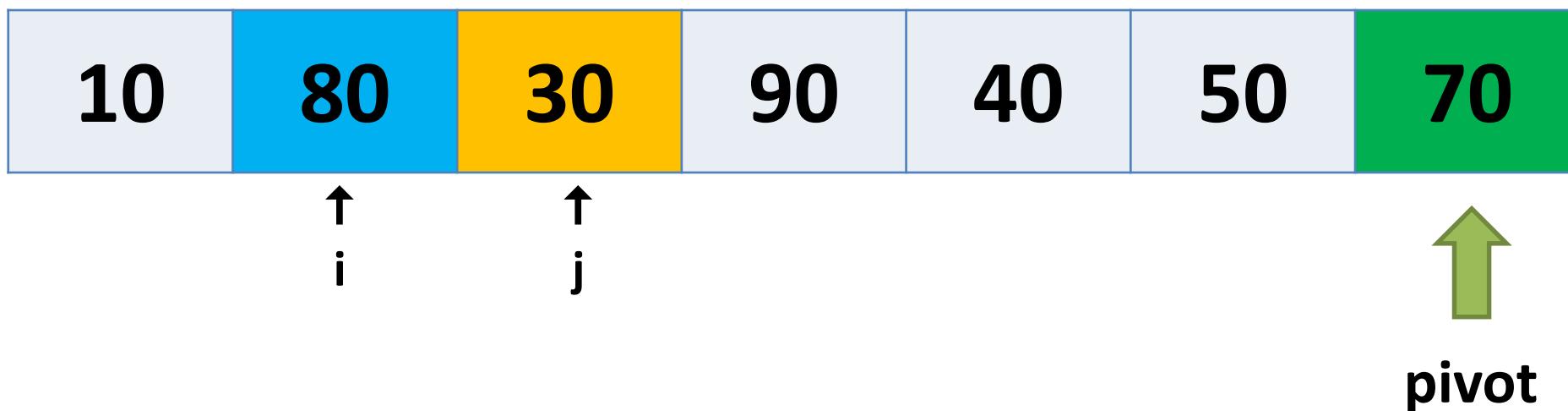
Quick Sort

❑ $\text{arr}[j] < \text{pivot}$



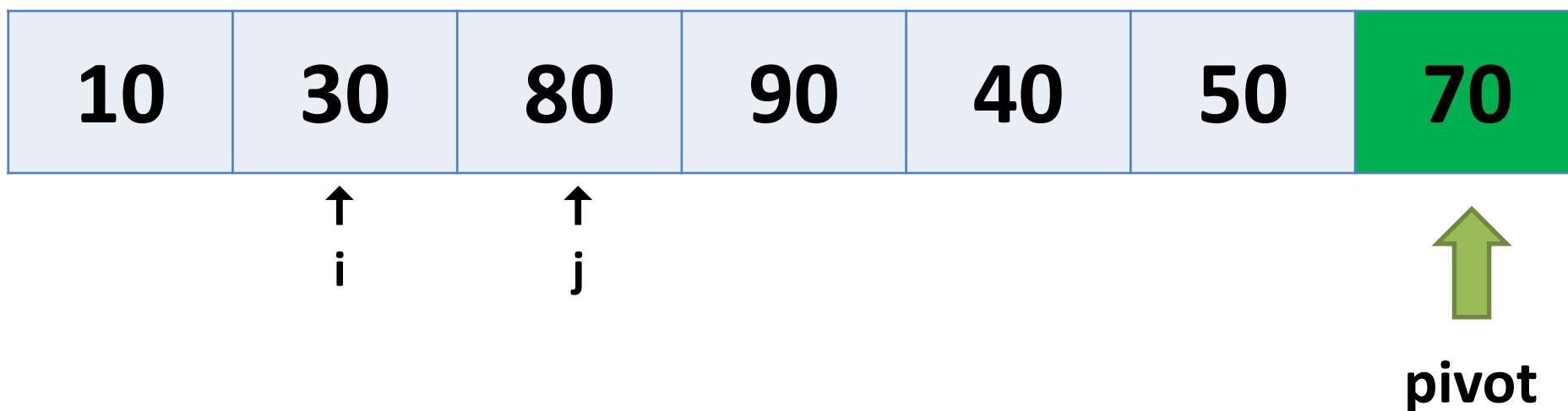
Quick Sort

- ❑ `i++`
- ❑ `swap(arr[i],arr[j])`



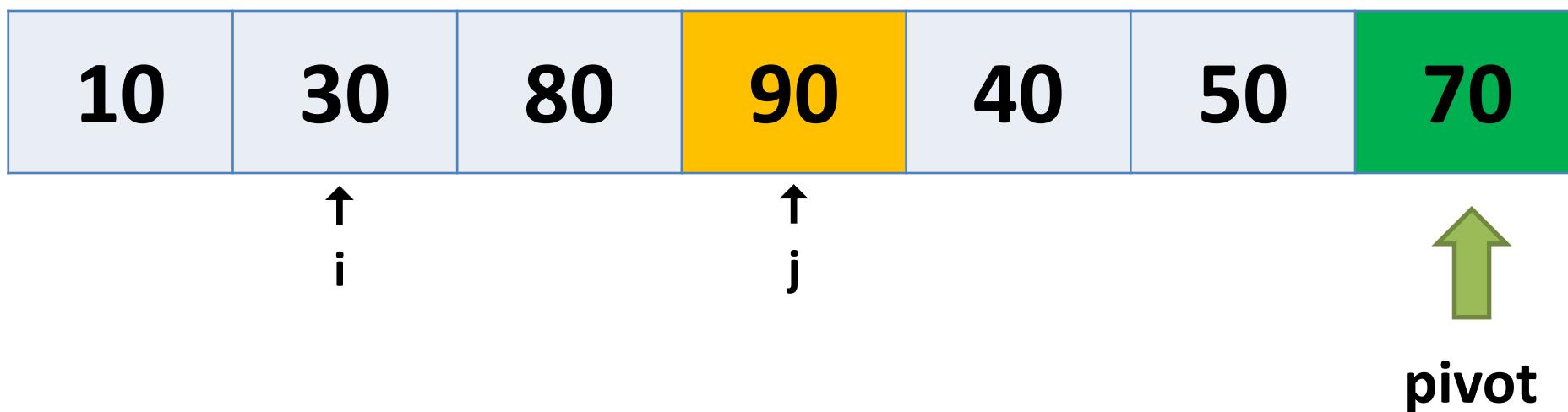
Quick Sort

- ❑ `i++`
- ❑ `swap(arr[i],arr[j])`



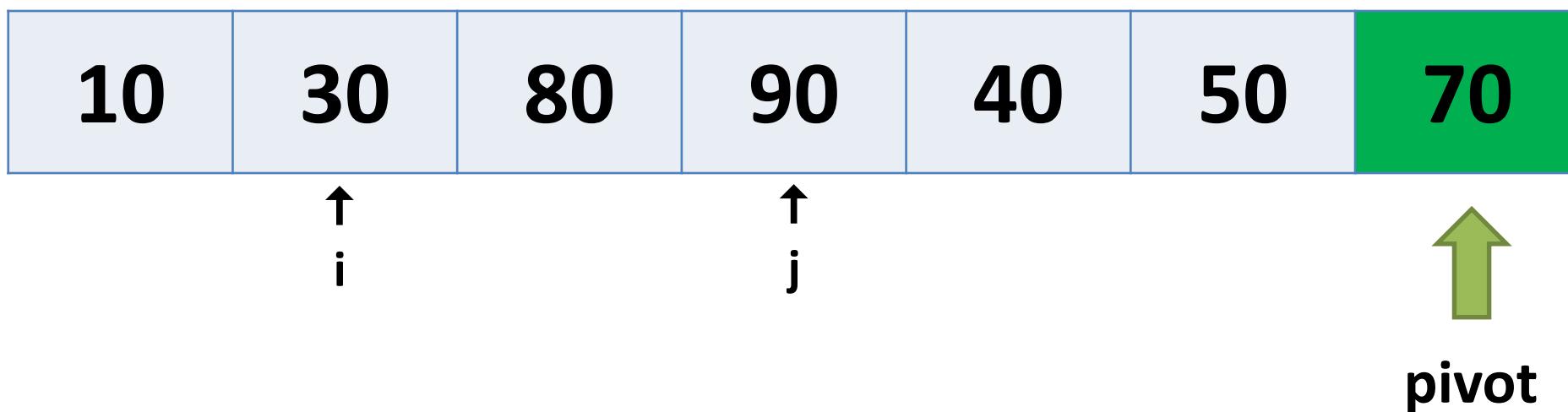
Quick Sort

❑ $\text{arr}[j] < \text{pivot}$



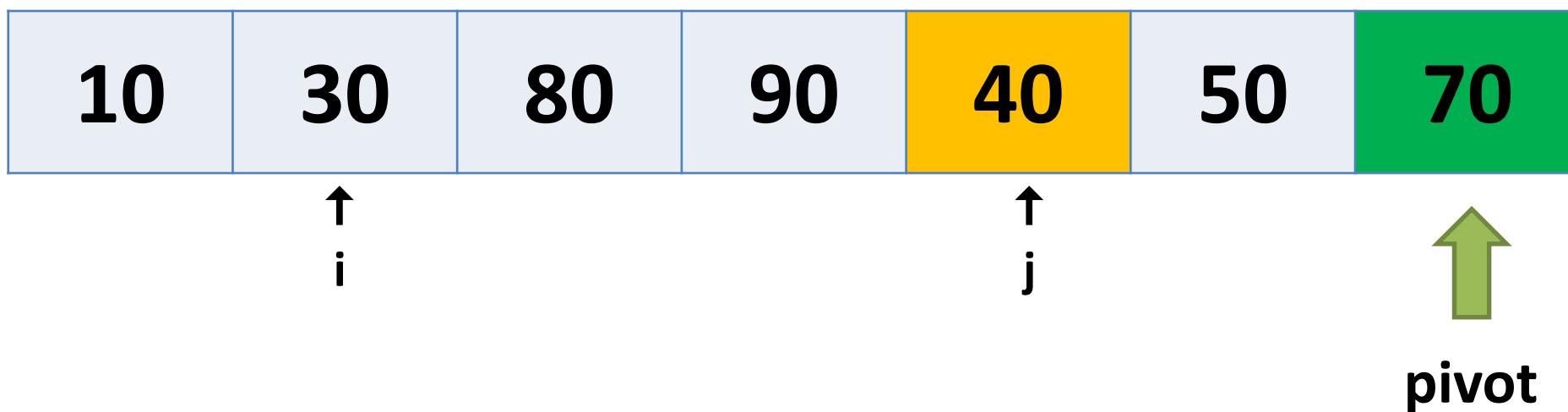
Quick Sort

No action



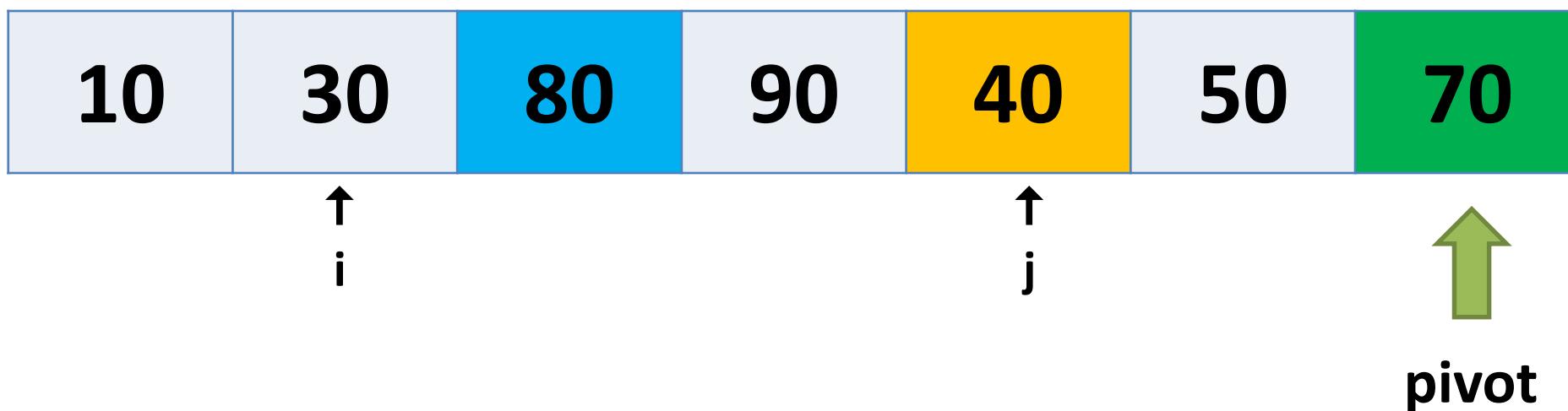
Quick Sort

❑ $\text{arr}[j] < \text{pivot}$



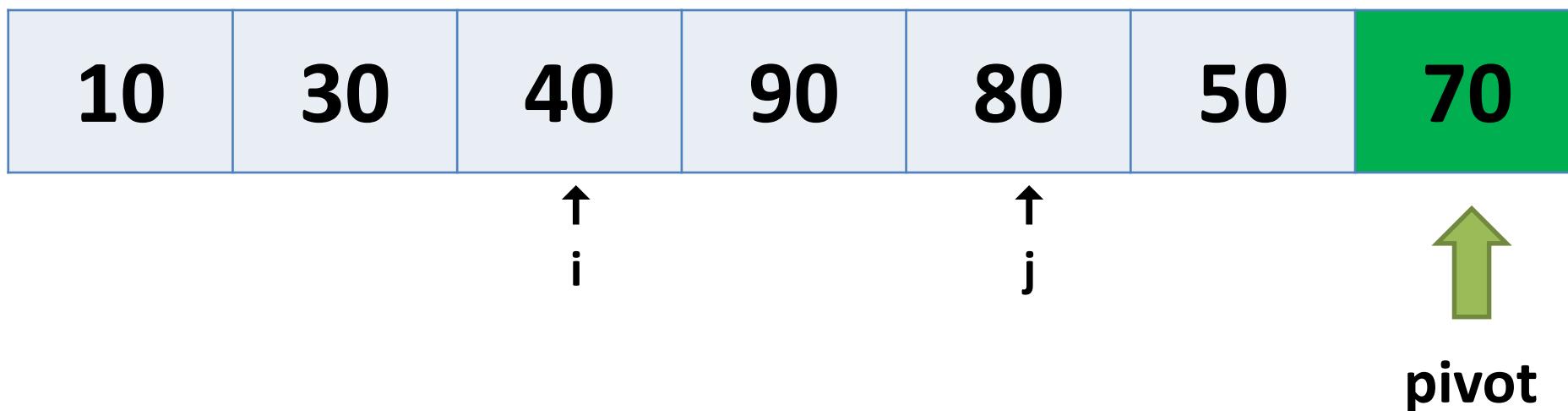
Quick Sort

- ❑ `i++`
- ❑ `swap(arr[i],arr[j])`



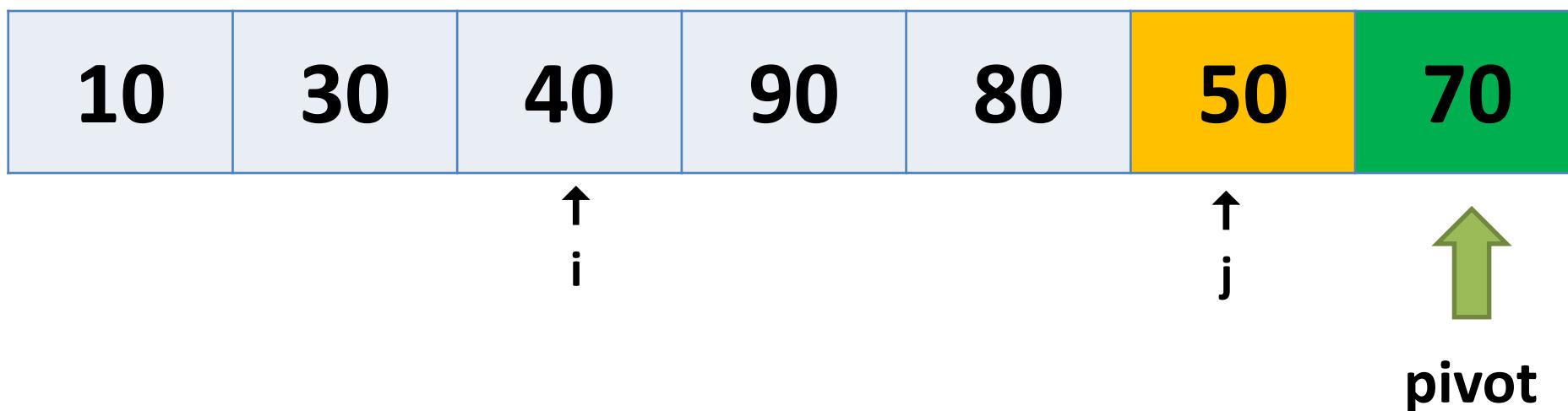
Quick Sort

- ❑ `i++`
- ❑ `swap(arr[i],arr[j])`



Quick Sort

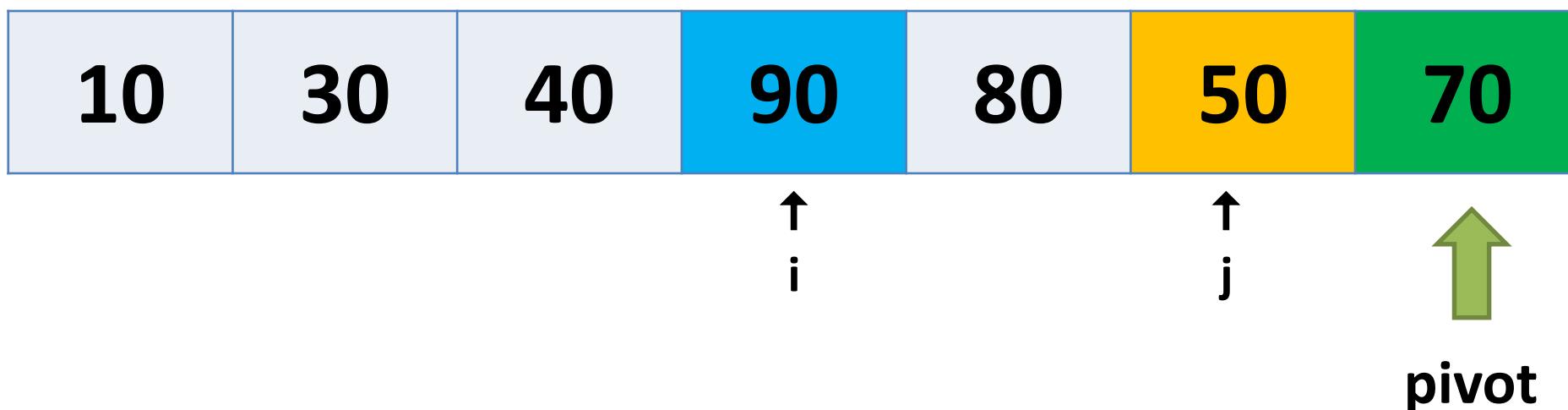
❑ $\text{arr}[j] < \text{pivot}$



Quick Sort

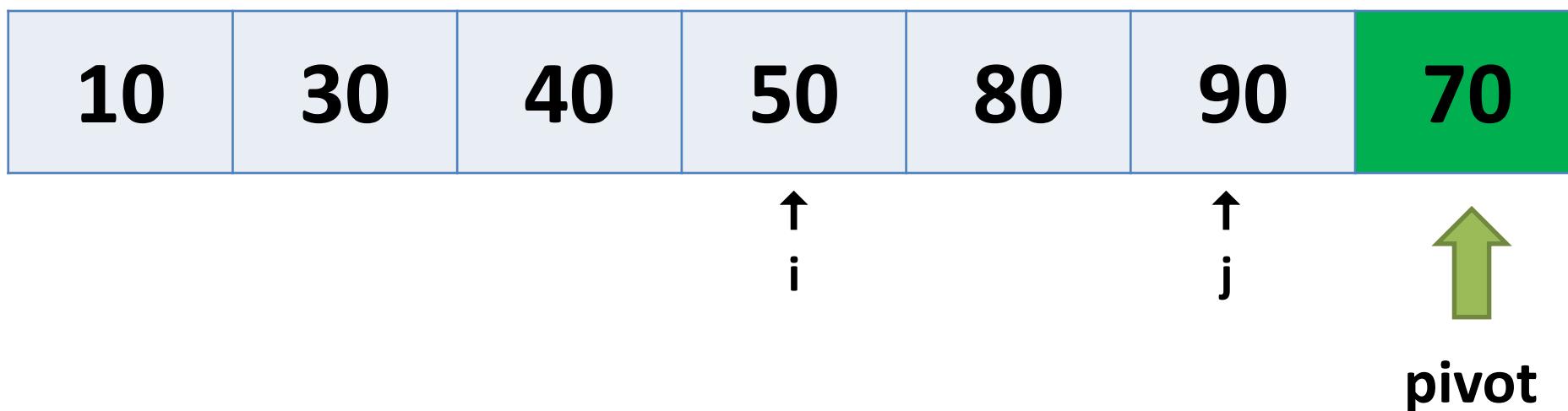
□ `i++`

□ `swap(arr[i],arr[j])`



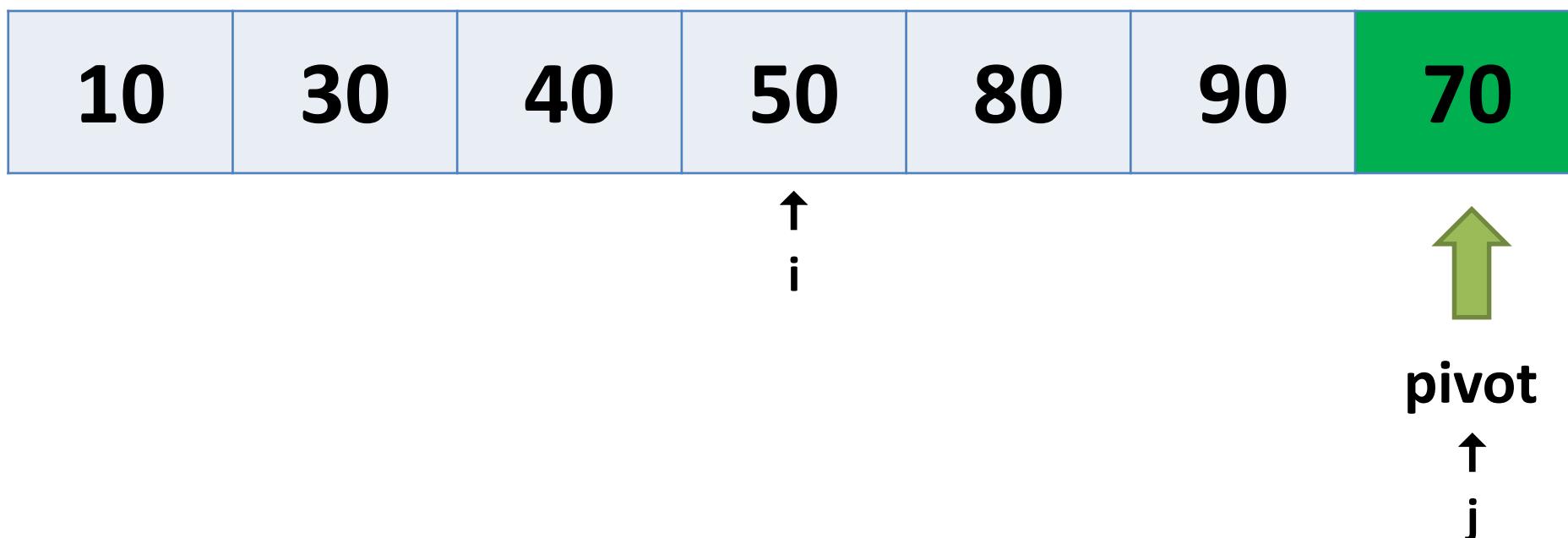
Quick Sort

- ❑ `i++`
- ❑ `swap(arr[i],arr[j])`



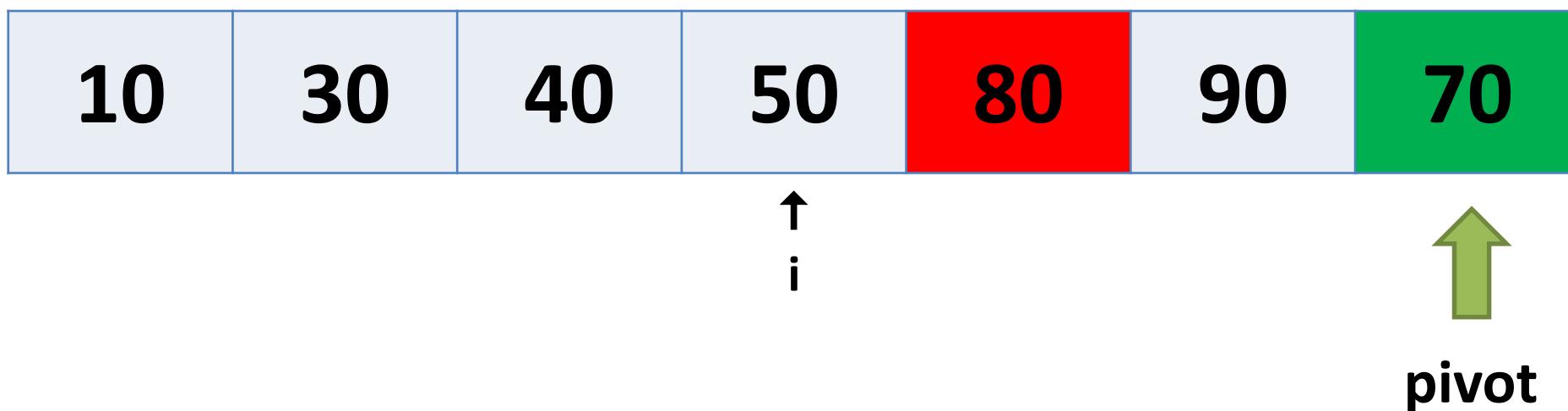
Quick Sort

Stop

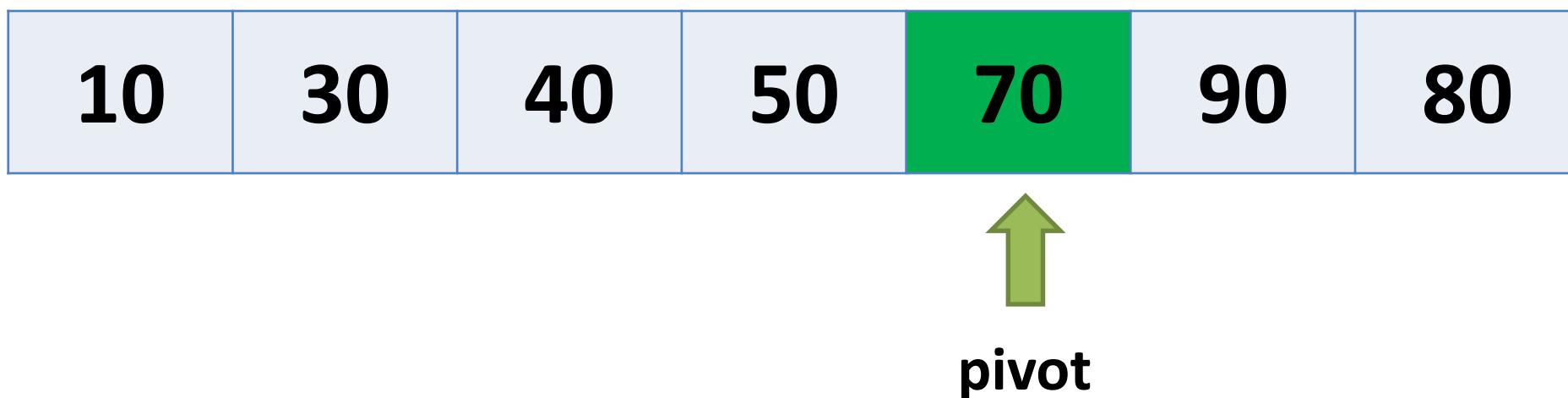


Quick Sort

☐ swap(arr[i+1],arr[end])

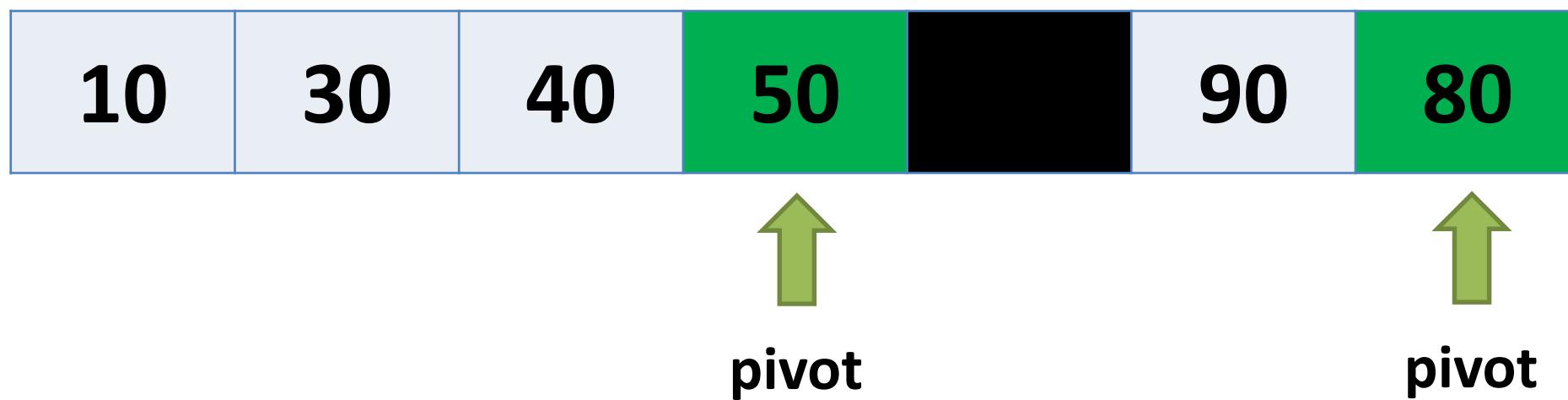


Quick Sort



Quick Sort

- Quick sort left partition recursively
 - Quick sort right partition recursively



Quick Sort

```
def QuickSort(arr,start,end):
    # Partitioning
    i = start-1
    pivot = arr[end]
    for j in range(start , end):
        if arr[j] <= pivot:
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]
    arr[i+1],arr[end] = arr[end],arr[i+1]
    # partition the array using pivot value
    pivotindex= i+1
    # quicksort left partition recursively
    quickSort(arr, start, pivotindex-1)
    # quicksort right partition recursively
    quickSort(arr, pivotindex+1, end)
```

Quick Sort

```
arr = [10, 7, 8, 9, 1, 5]
QuickSort(arr,0,len(arr)-1)
print(arr)
```

Quick Sort

- ❑ **Time Complexity:** $O(n^2)$ because of two recursive calls

Contact Me





THANKS FOR
YOUR TIME