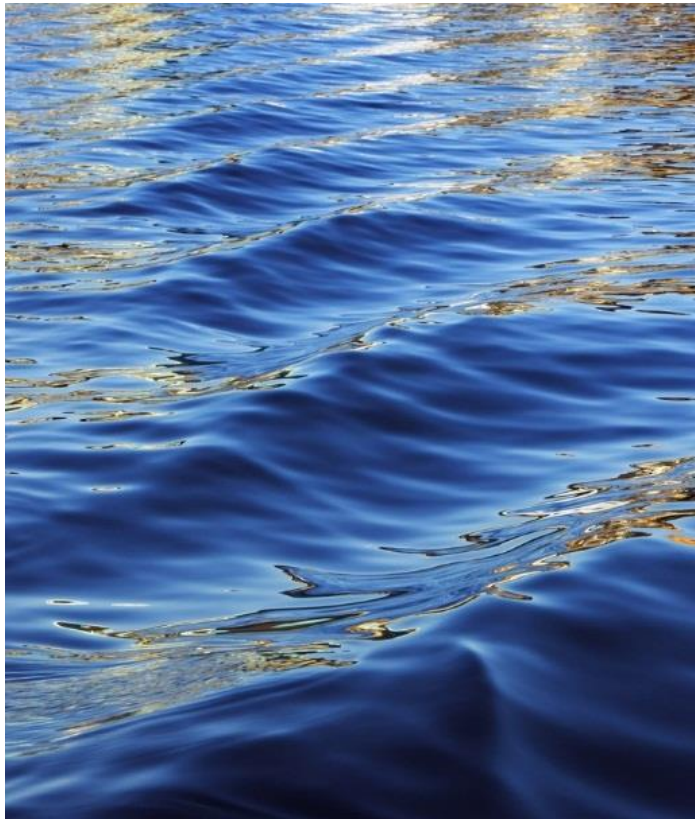
Searching and Sorting

Data structure and Algorithms (310-2101)



Sorting

What is Sorting?

Sorting is the process of placing elements from a collection in order.
For example, a list of words could be sorted alphabetically or by length. A list of cities could be sorted by population, by area, or by zip code.

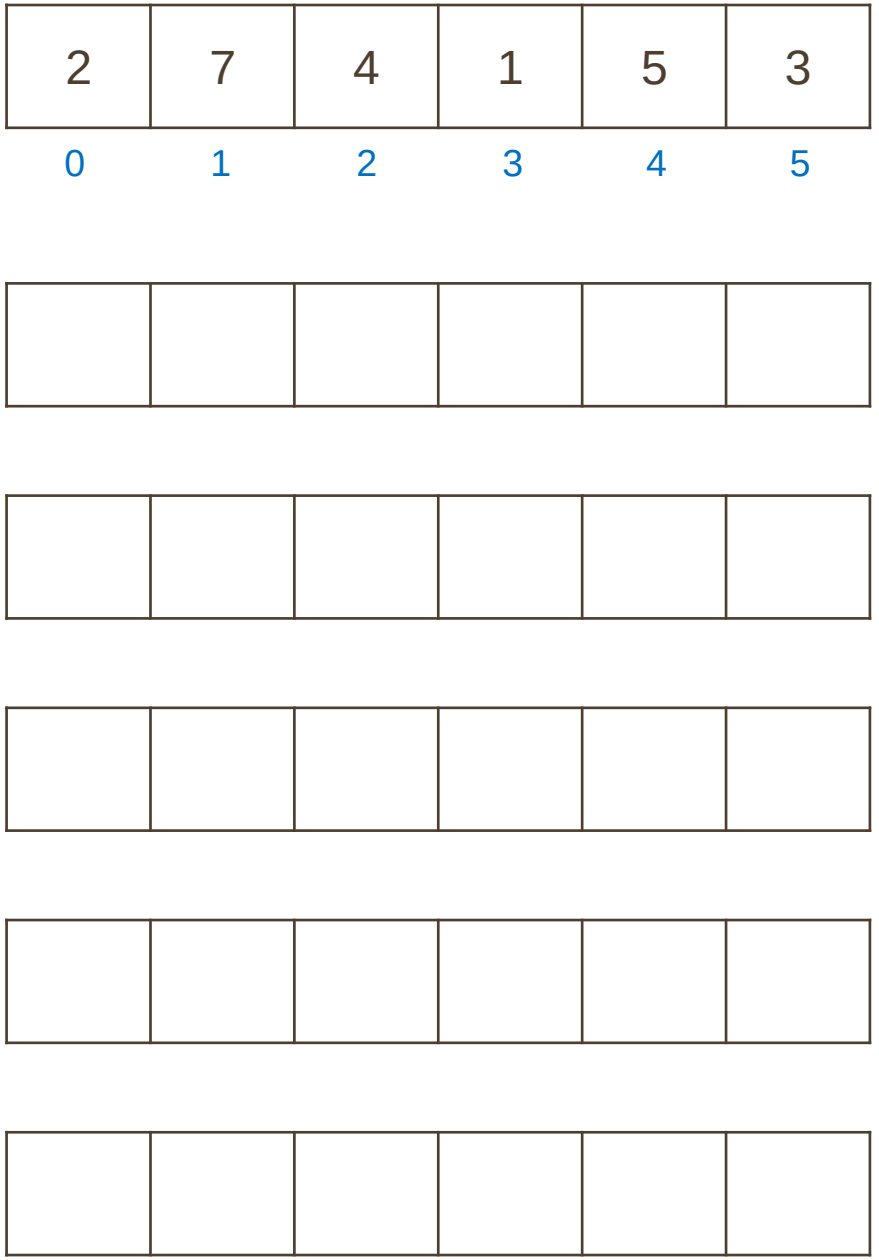
There are many, many sorting algorithms that have been developed and analyzed. This suggests that sorting is an important area of study in computer science. Sorting many items can take a substantial amount of computing resources.

List of Sorting Algorithms in this Class

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Quick Sort

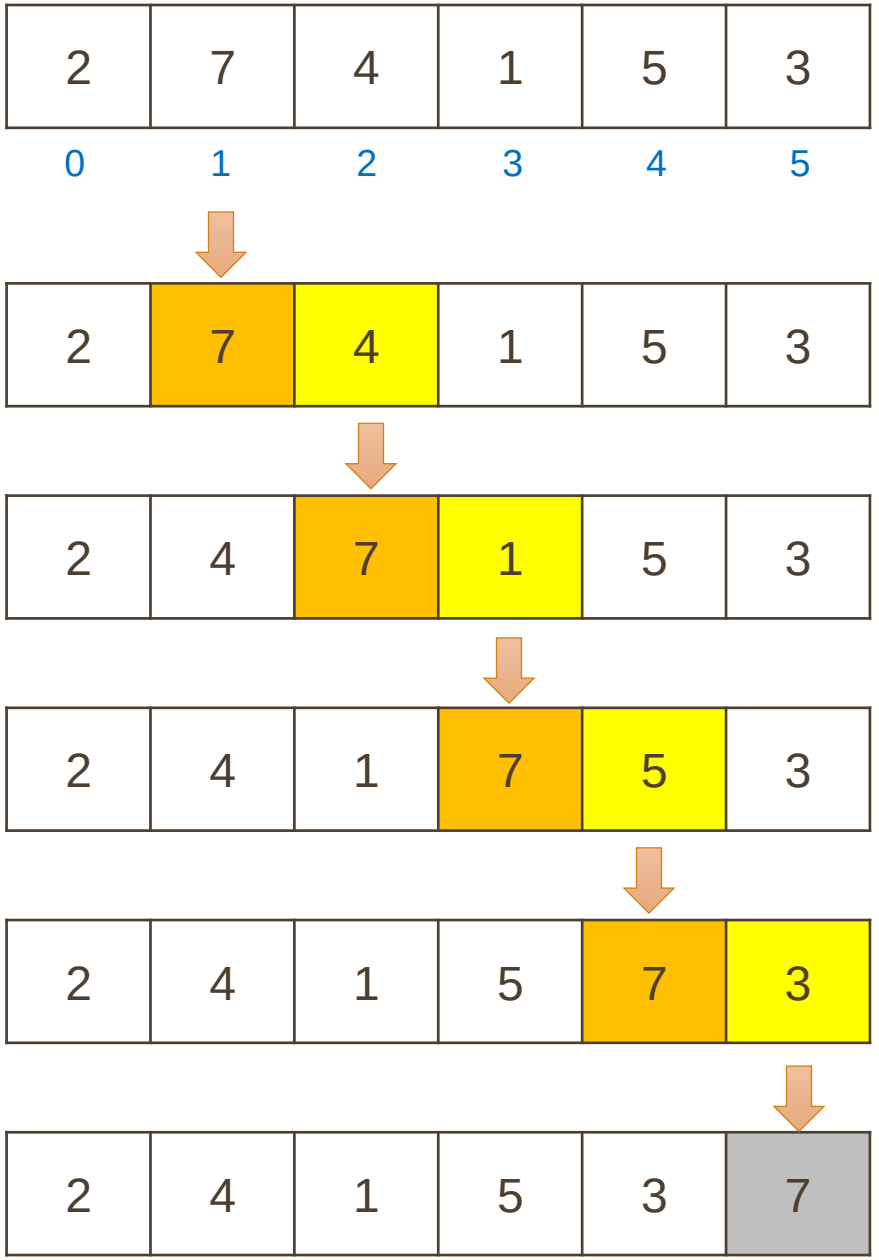
Bubble Sort

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



Bubble Sort

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



Bubble Sort - Code

```
1
2   def BubbleSort(a_list):
3       n = len(a_list)
4       for k in range(1, n):
5           for i in range(0, n - k):
6               if a_list[i] > a_list[i+1]:
7                   tmp = a_list[i]
8                   a_list[i] = a_list[i+1]
9                   a_list[i+1] = tmp
10      return a_list       # Return or Not
11
12  if __name__ == '__main__':
13
14      A = [2, 7, 4, 1, 5, 3]
15      print(A)
16
17      BubbleSort(A)
18      print(A)
19
```

Bubble Sort - Code

```
2   def BubbleSort(a_list):
3       exchanges = True
4       pass_num = len(a_list) - 1
5       while pass_num > 0 and exchanges:
6           exchanges = False
7           for i in range(pass_num):
8               if a_list[i] > a_list[i+1]:
9                   exchanges = True
10                  temp = a_list[i]
11                  a_list[i] = a_list[i+1]
12                  a_list[i+1] = temp
```

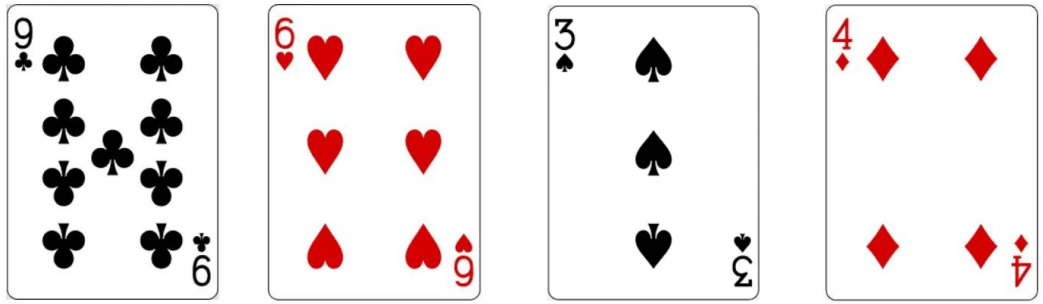
Bubble Sort - Code

```
1
2   def BubbleSort(a_list):
3       n = len(a_list)
4       for k in range(1, n):
5           flag = 0
6           for i in range(0, n - k):
7               if a_list[i] > a_list[i+1]:
8                   tmp = a_list[i]
9                   a_list[i] = a_list[i+1]
10                  a_list[i+1] = tmp
11                  flag = 1
12          if flag == 0:
13              break
14      return a_list       # Return or Not
```

Exercise

1) แก้ไขโปรแกรม ให้นับจำนวนครั้งที่มีการ “สลับ” ตำแหน่งของตัวเลข
2) แก้ไขโปรแกรม ให้เรียงลำดับค่าจาก มาก ไป น้อย

Selection Sort



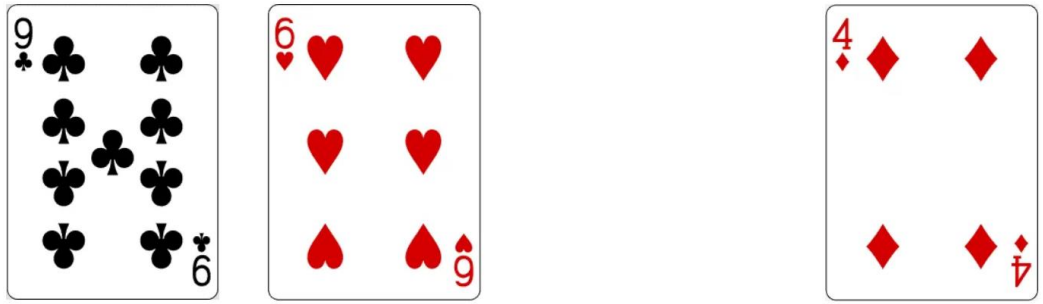
Left Hand

Unsorted

Right Hand

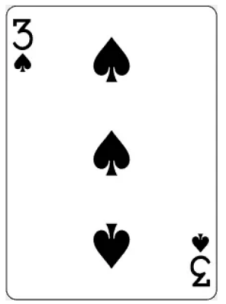
Sorted

Selection Sort



Left Hand

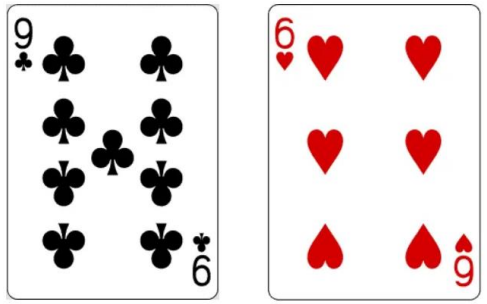
Unsorted



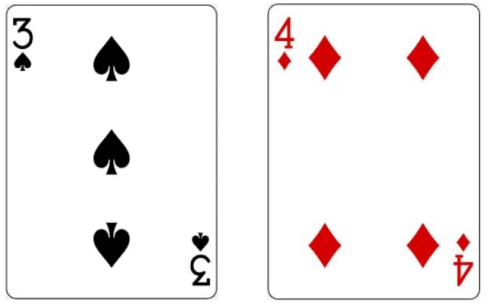
Right Hand

Sorted

Selection Sort

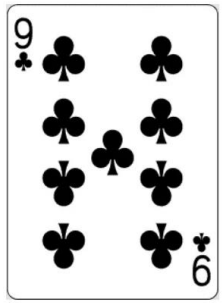


Unsorted

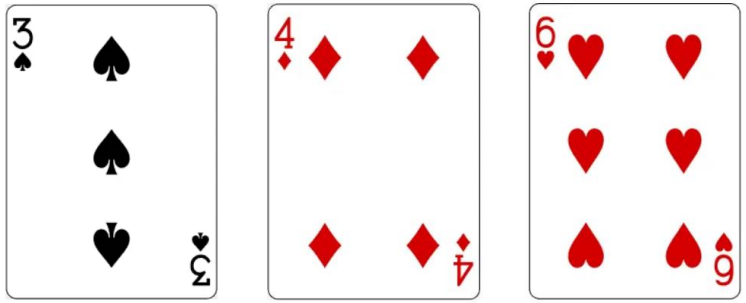


Sorted

Selection Sort



Unsorted

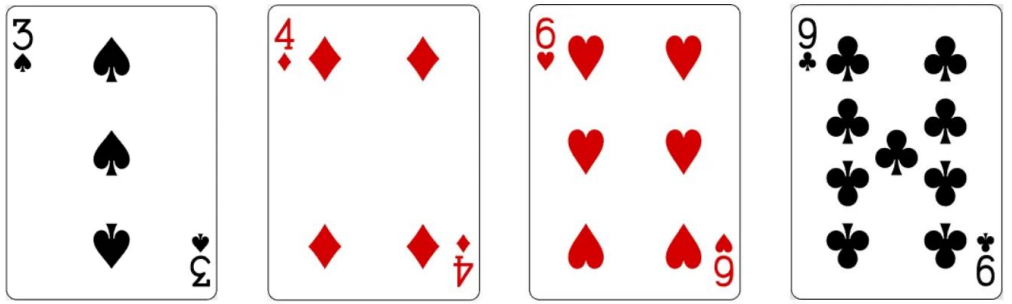


Sorted

Selection Sort

Left Hand

Unsorted



Right Hand

Sorted

Selection Sort

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

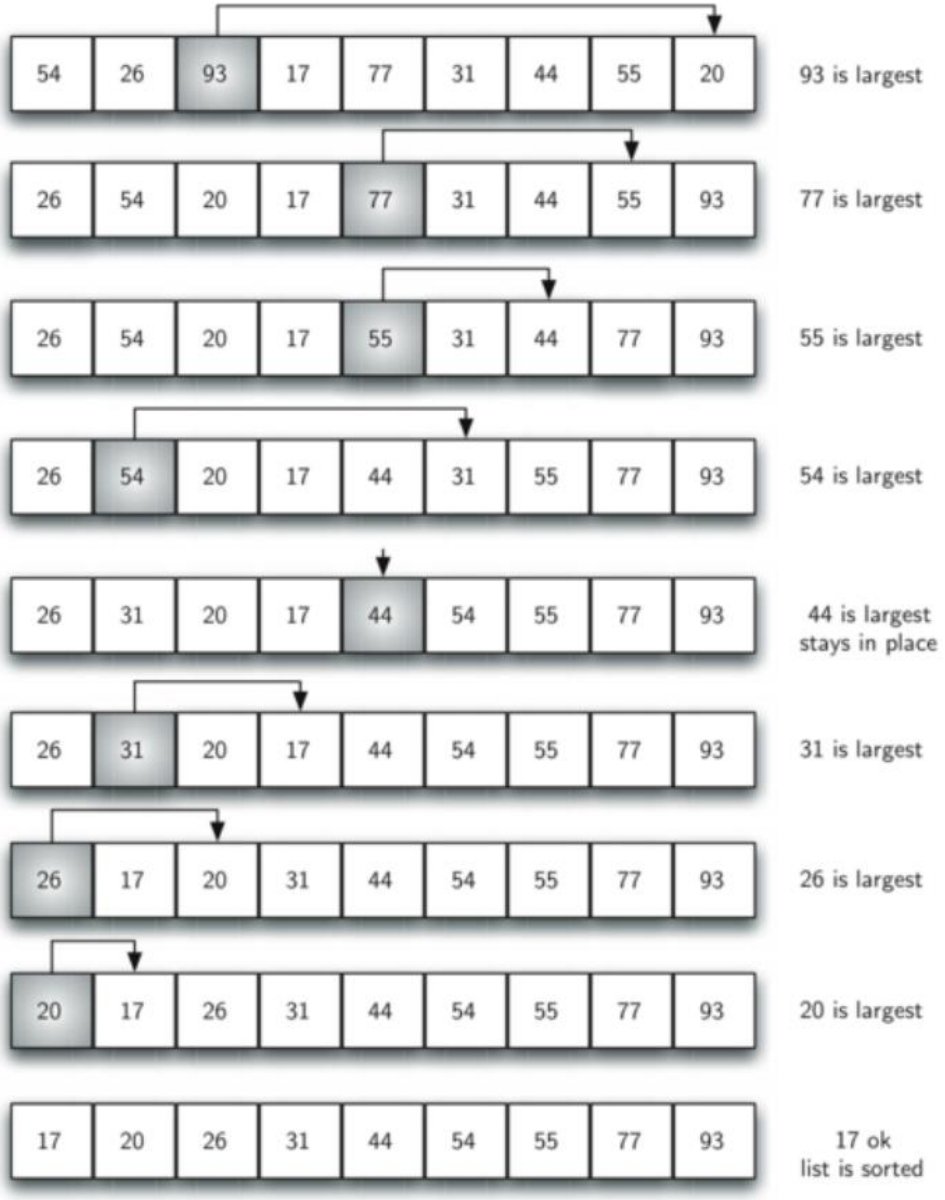
2	7	4	1	5	3

1	7	4	2	5	3

1	2	4	7	5	3

1	2	3	7	5	4

1	2	3	4	5	7



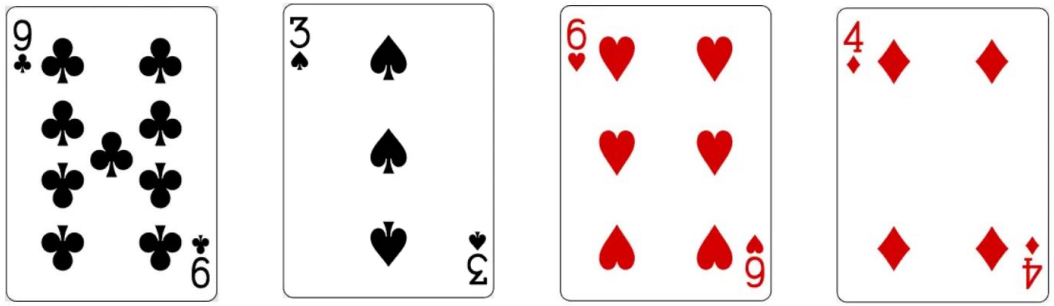
Selection Sort - Code

```
1
2   def SelectionSort(a_list):
3       n = len(a_list)
4
5       for i in range(0, n-1):
6           iMin = i
7
8           for j in range(i+1, n):
9               if a_list[j] < a_list[iMin]:
10                  iMin = j
11
12          temp = a_list[i]
13          a_list[i] = a_list[iMin]
14          a_list[iMin] = temp
15
16      return a_list       # Return or Not
17
```

Exercise

ทำ Selection Sort โดยนำค่าที่มากที่สุดไปไว้ทางขวามือของ Array แทน และทำการสั่ง Print เพื่อแสดงผลลัพธ์การ Sort ในแต่ละ Loop ของโปรแกรม

Insertion Sort



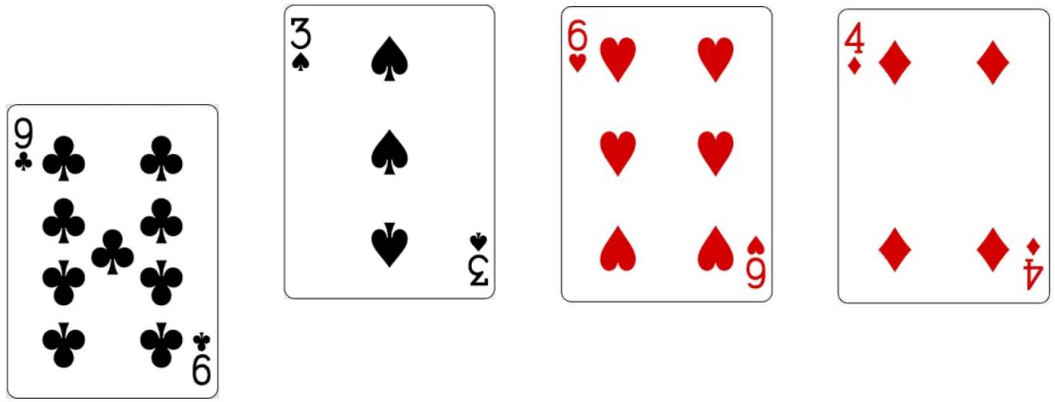
Left Hand

Unsorted

Right Hand

Sorted

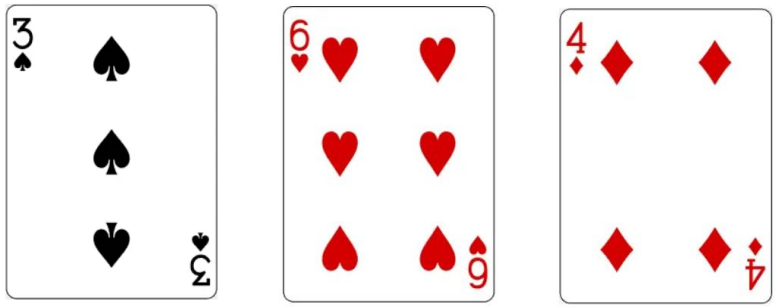
Insertion Sort



Unsorted

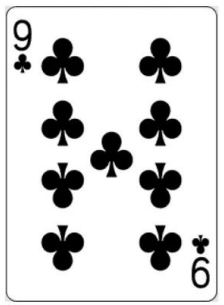
Sorted

Insertion Sort



Left Hand

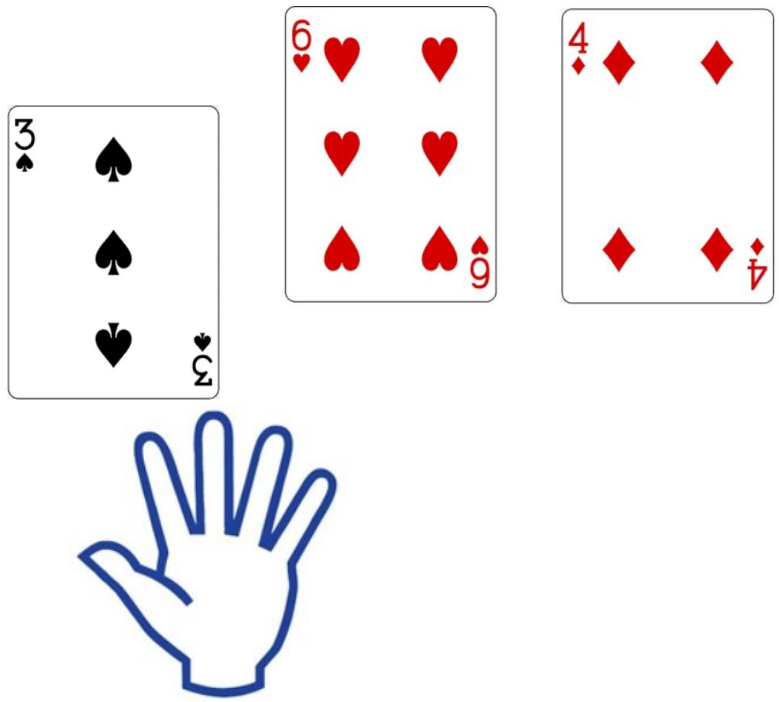
Unsorted



Right Hand

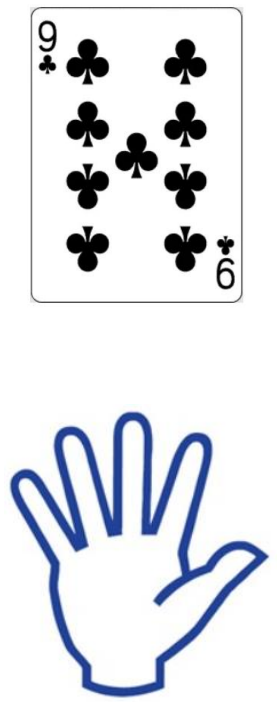
Sorted

Insertion Sort



Left Hand

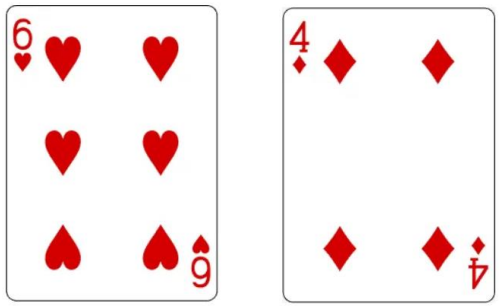
Unsorted



Right Hand

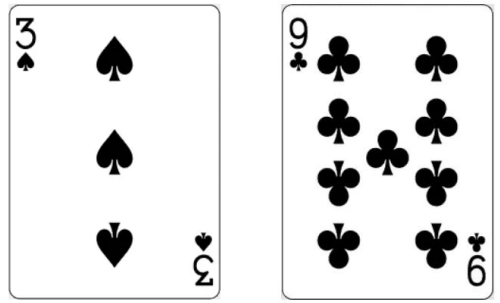
Sorted

Insertion Sort



Left Hand

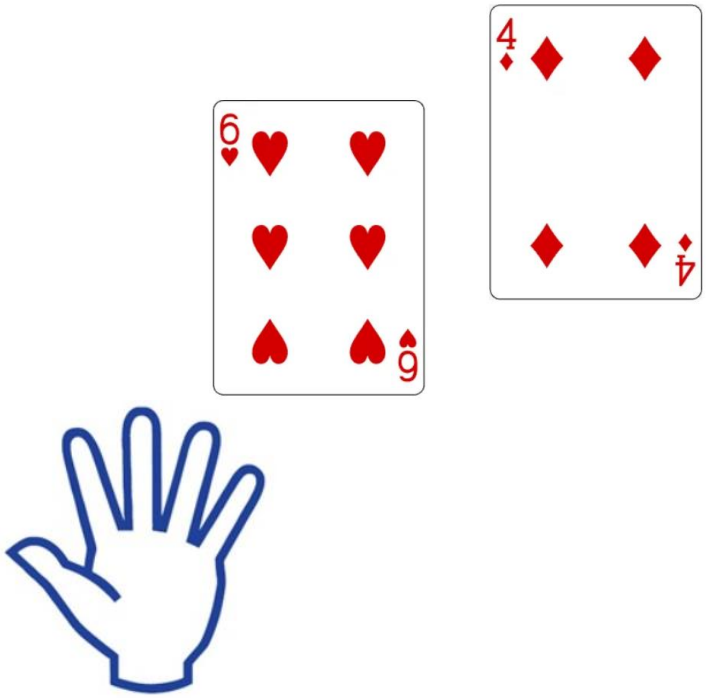
Unsorted



Right Hand

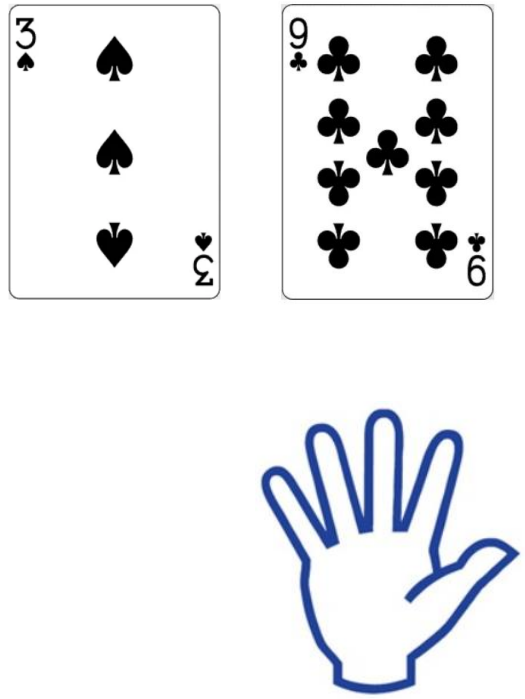
Sorted

Insertion Sort



Left Hand

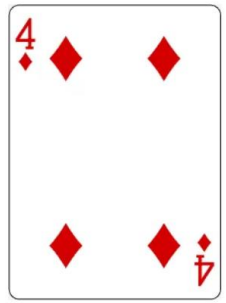
Unsorted



Right Hand

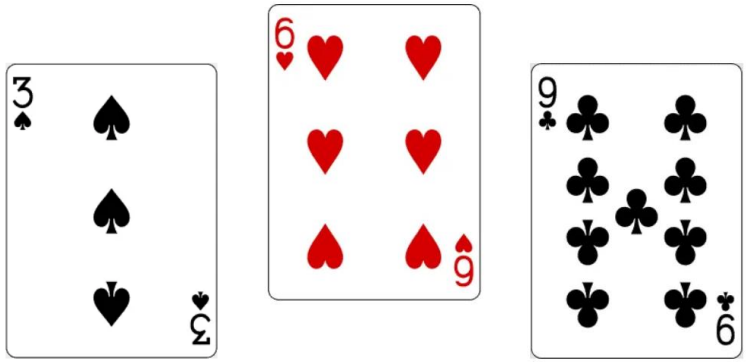
Sorted

Insertion Sort



Left Hand

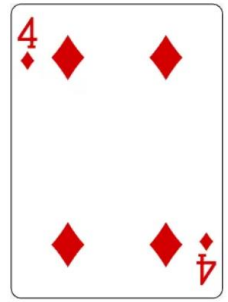
Unsorted



Right Hand

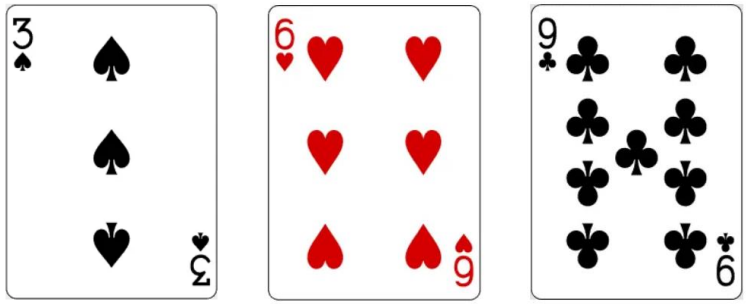
Sorted

Insertion Sort



Left Hand

Unsorted



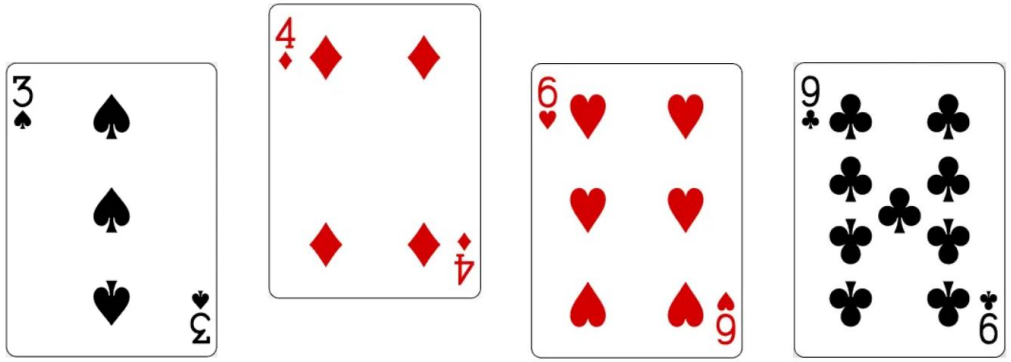
Right Hand

Sorted

Insertion Sort

Left Hand

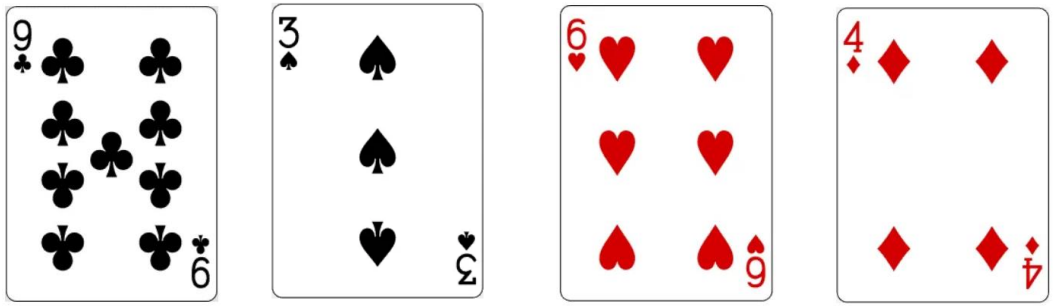
Unsorted



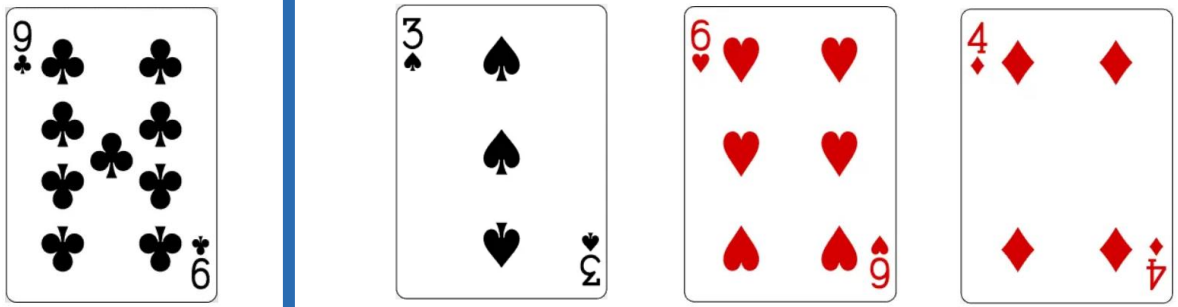
Right Hand

Sorted

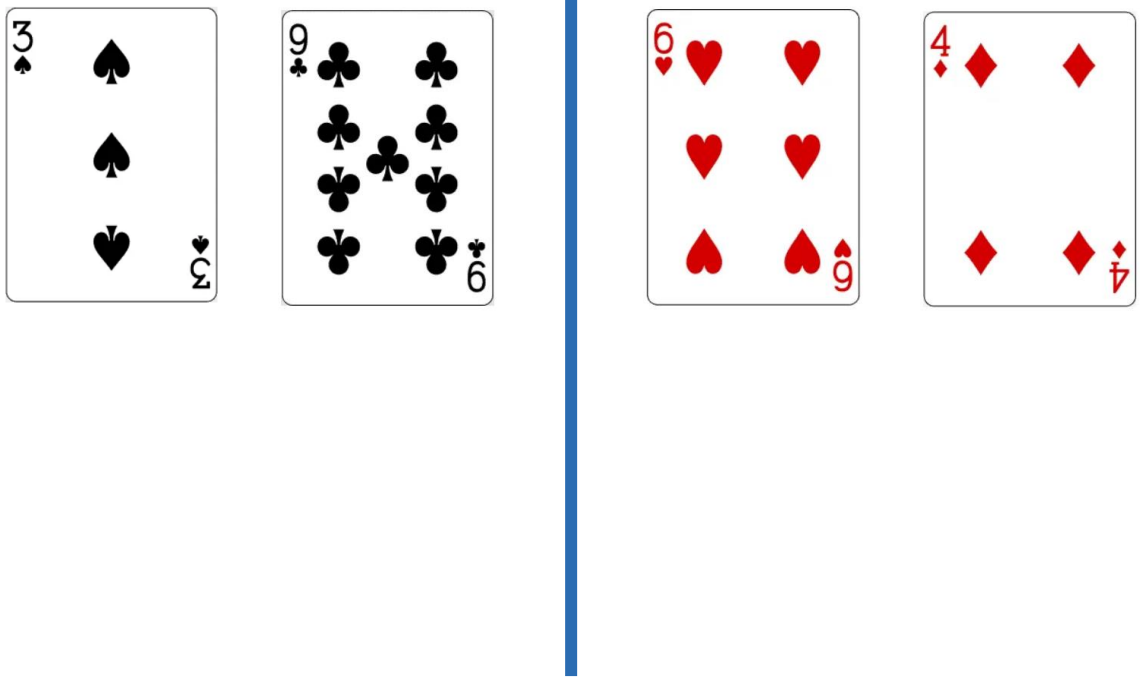
Insertion Sort



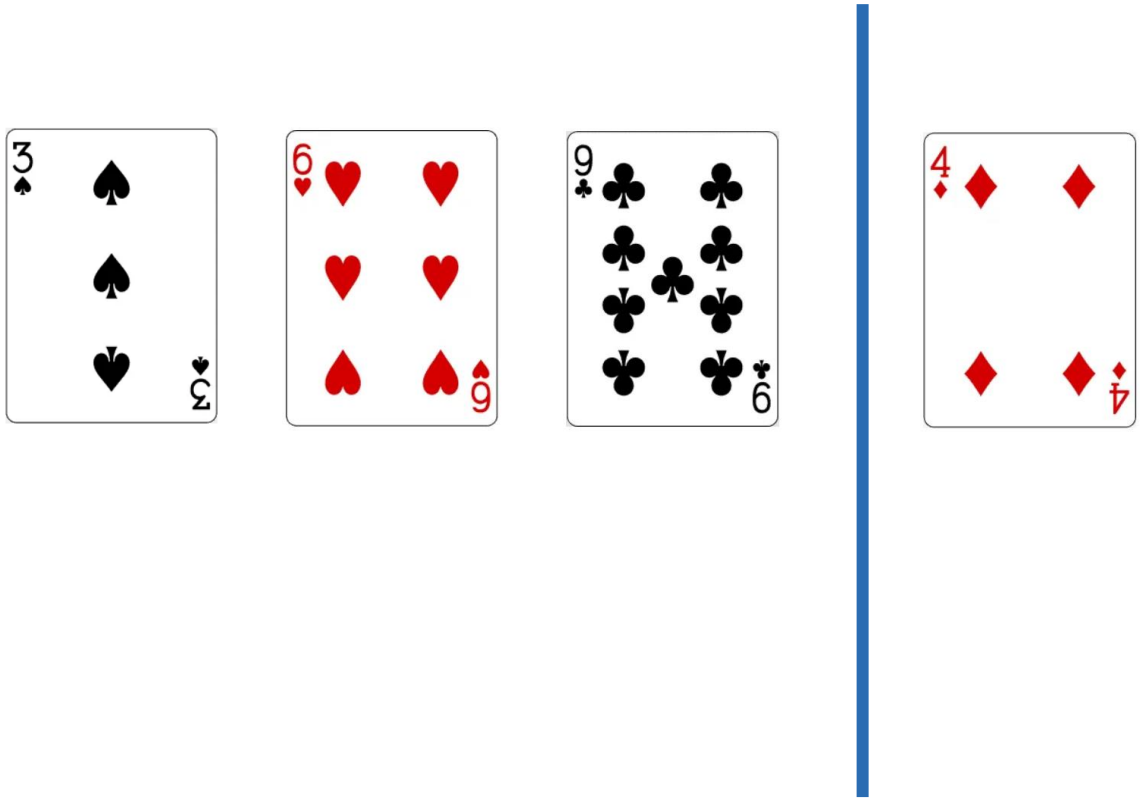
Insertion Sort



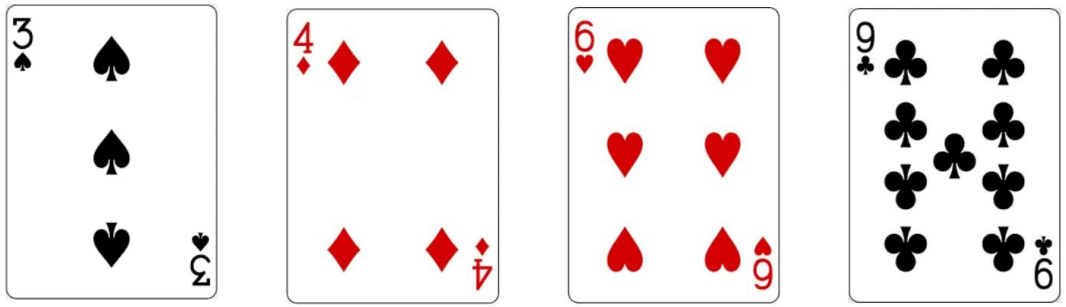
Insertion Sort



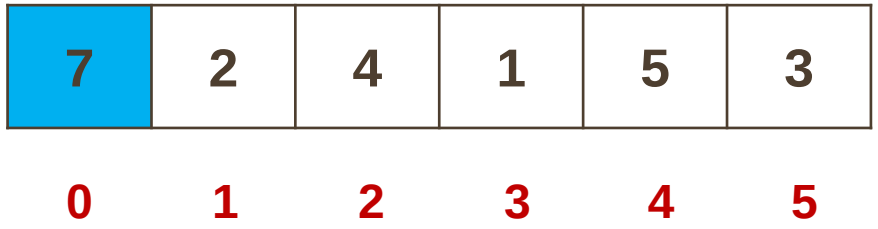
Insertion Sort



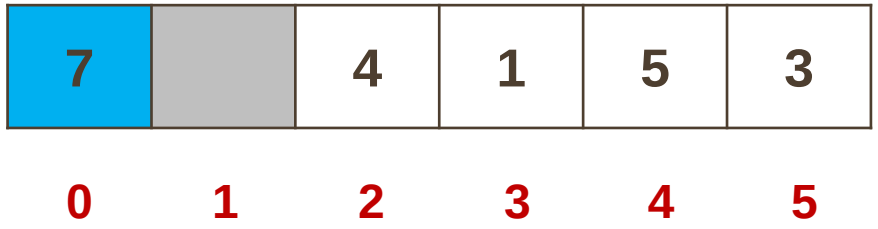
Insertion Sort



Insertion Sort

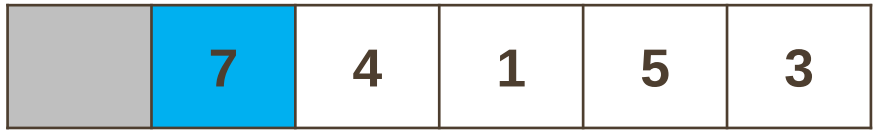


Insertion Sort



Temp | 2

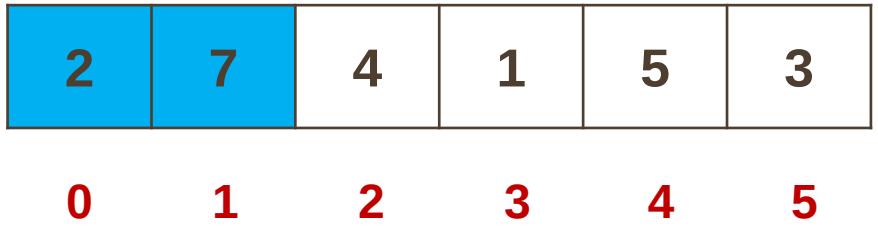
Insertion Sort



Temp **2**

Shift to the Right (If [Temp] less than it)

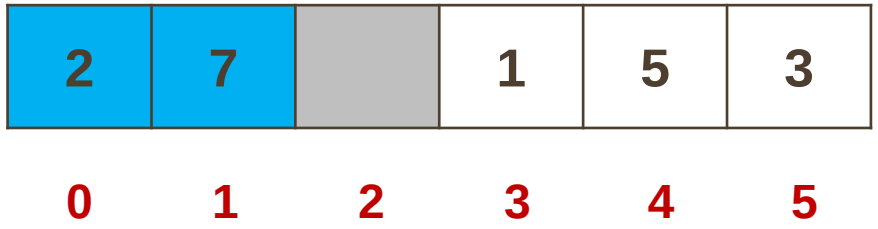
Insertion Sort



Temp

Shift to the Right (If [Temp] less than it)

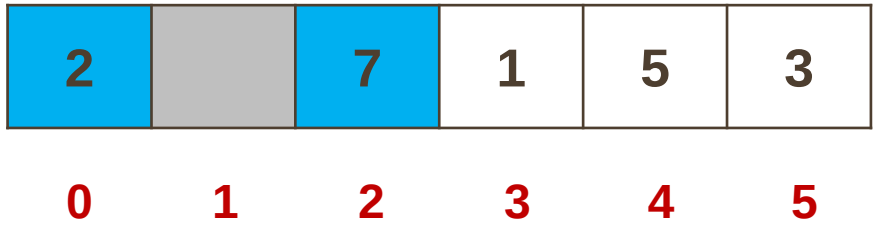
Insertion Sort



Temp **4**

Shift to the Right (If [Temp] less than it)

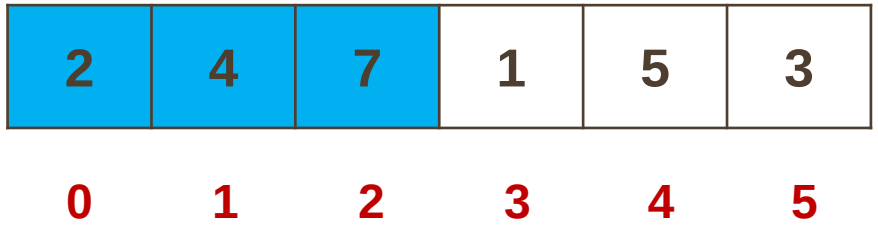
Insertion Sort



Temp **4**

Shift to the Right (If [Temp] less than it)

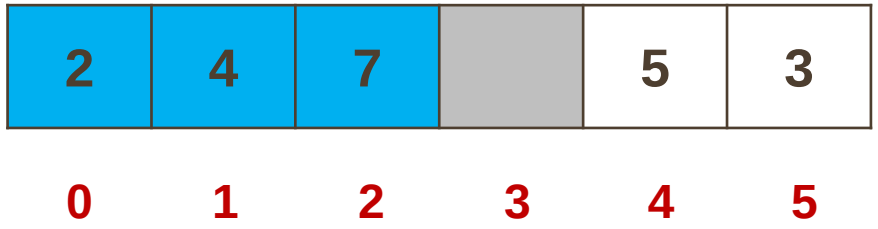
Insertion Sort



Temp

Shift to the Right (If [Temp] less than it)

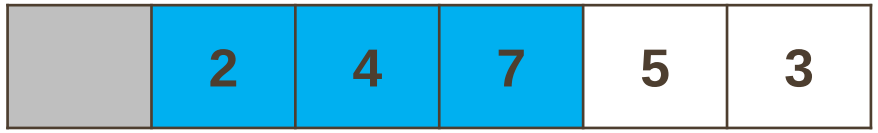
Insertion Sort



Temp | 1

Shift to the Right (If [Temp] less than it)

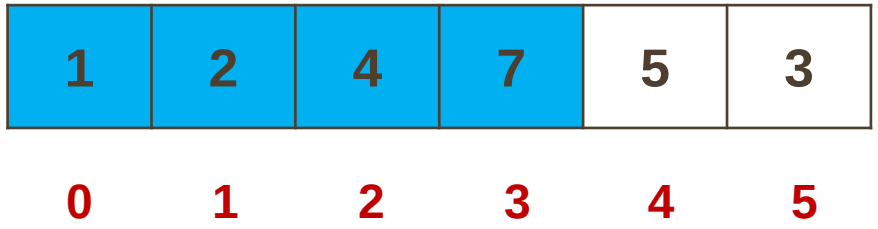
Insertion Sort



Temp | 1

Shift to the Right (If [Temp] less than it)

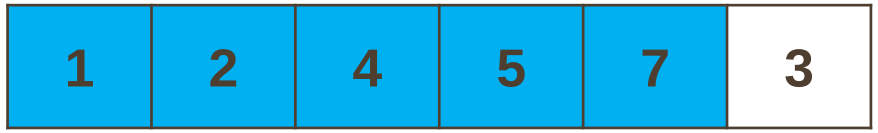
Insertion Sort



Temp

Shift to the Right (If [Temp] less than it)

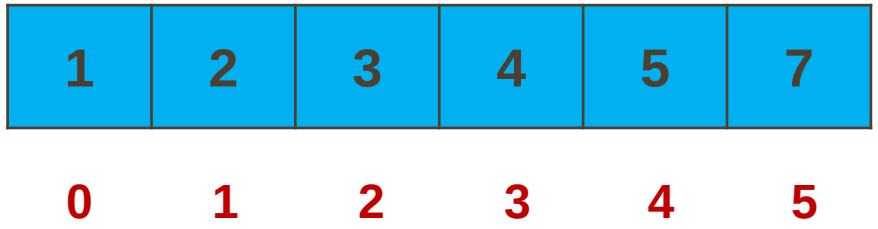
Insertion Sort



Temp

Shift to the Right (If [Temp] less than it)

Insertion Sort



Temp

Shift to the Right (If [Temp] less than it)

Insertion Sort - Code

```
1
2   def InsertionSort(a_list):
3       n = len(a_list)
4
5       for i in range(1, n):
6
7           temp = a_list[i]
8           hole = i
9
10          while hole>0 and a_list[hole-1]>temp:
11              a_list[hole] = a_list[hole-1]
12              hole = hole-1
13
14          a_list[hole] = temp
15
16      return a_list       # Return or Not
17
```

Best – case:

Sorted

1, 2, 3, 4, 5

O(n)

Worst – case:

Reverse sorted

5, 4, 3, 2, 1

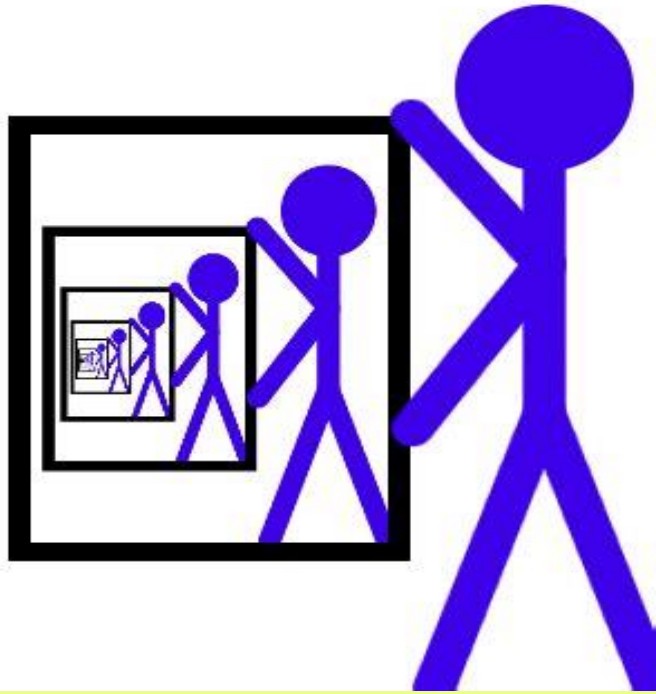
$O(n^2)$

Average – case:

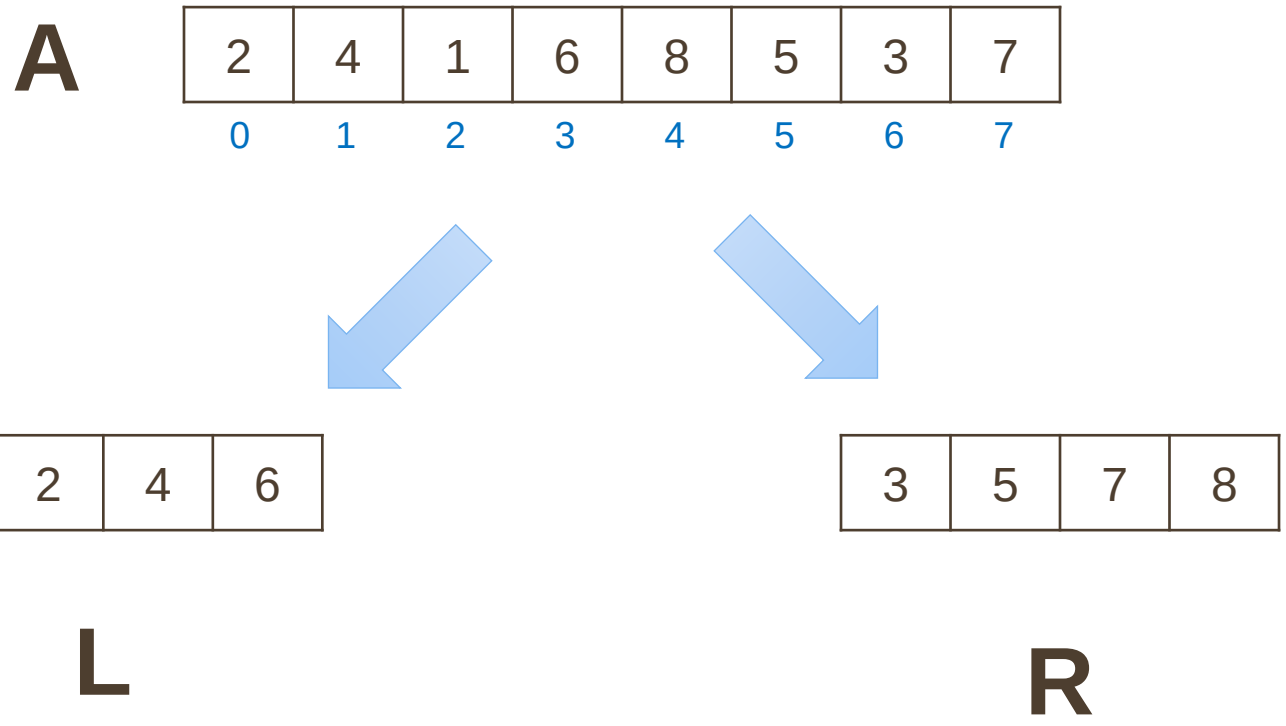
$O(n^2)$

Merge Sort

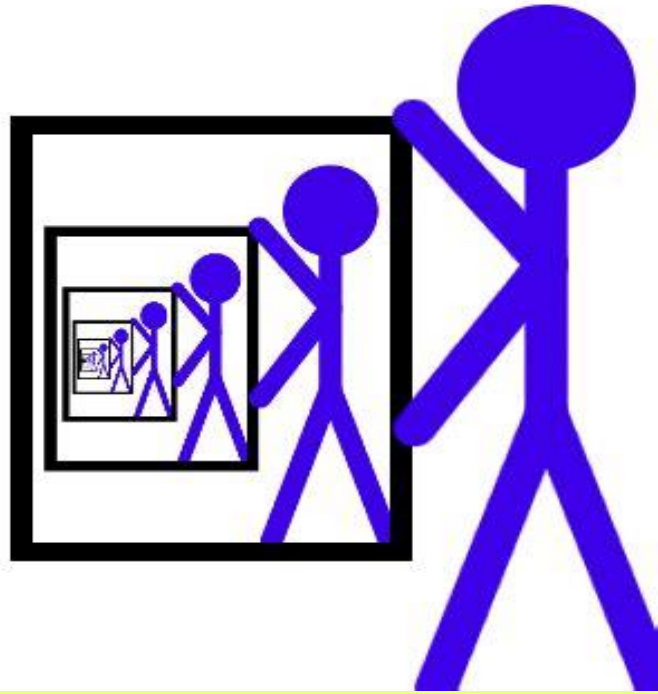
2	4	1	6	8	5	3	7
0	1	2	3	4	5	6	7



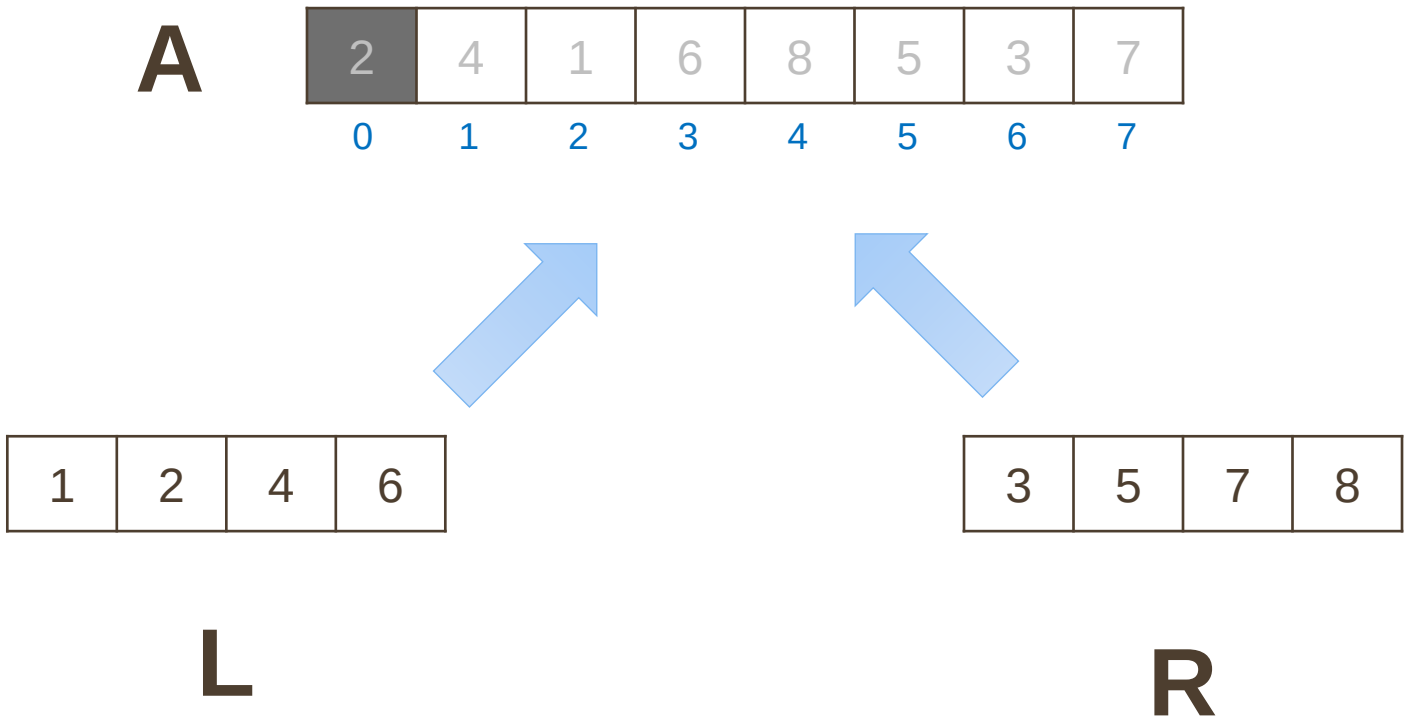
Merge Sort

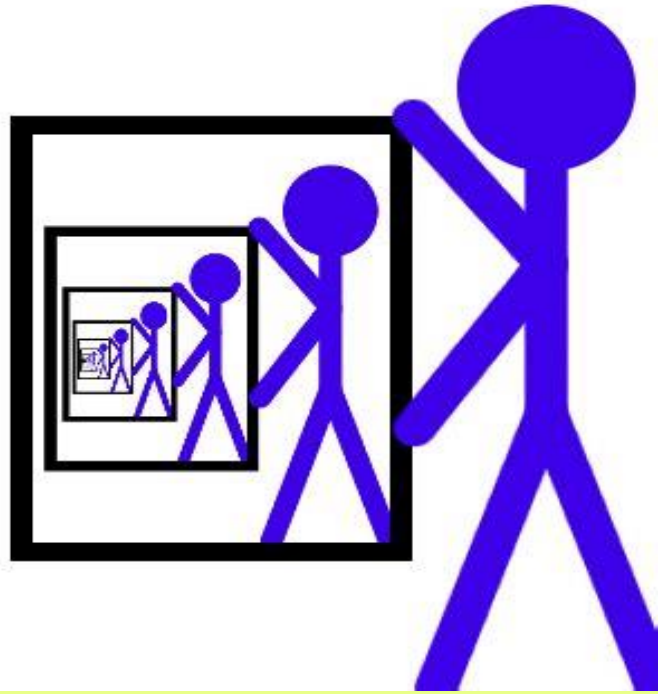


Assume, we somehow get these sorted sub-lists

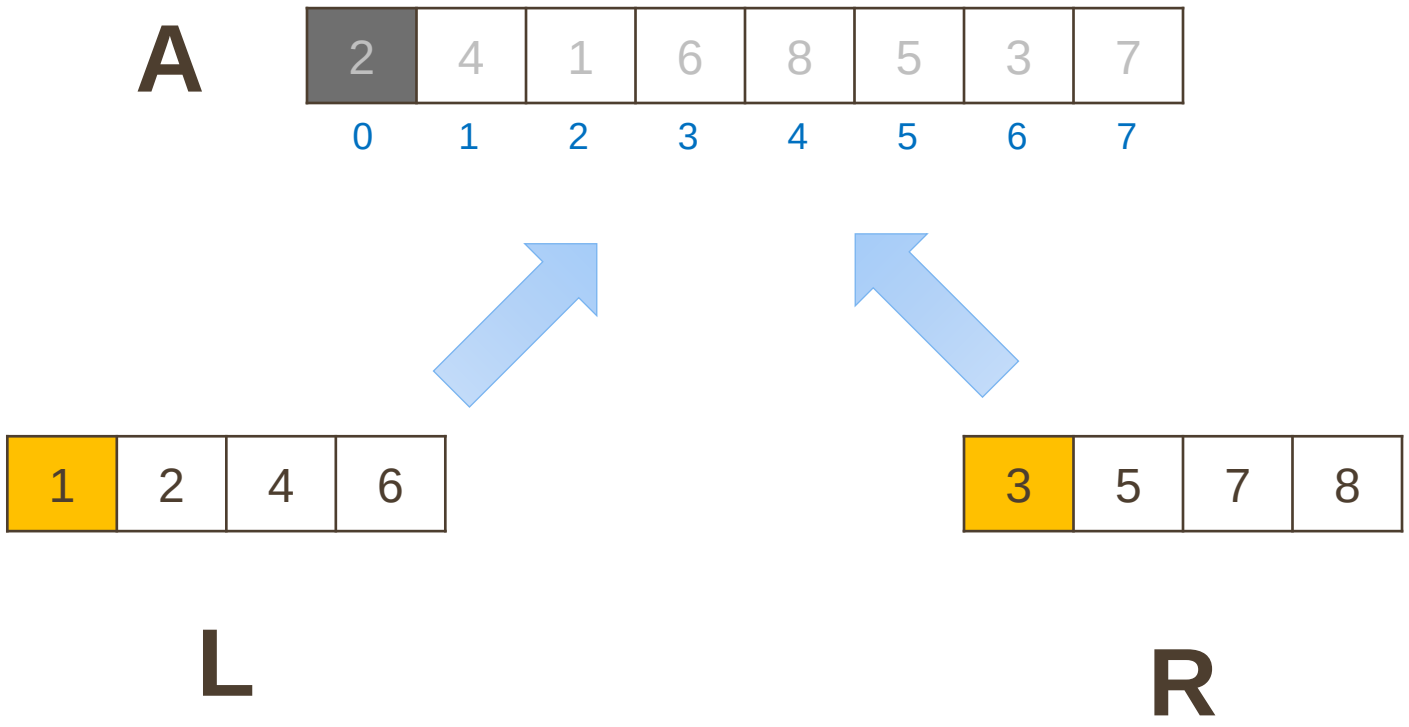


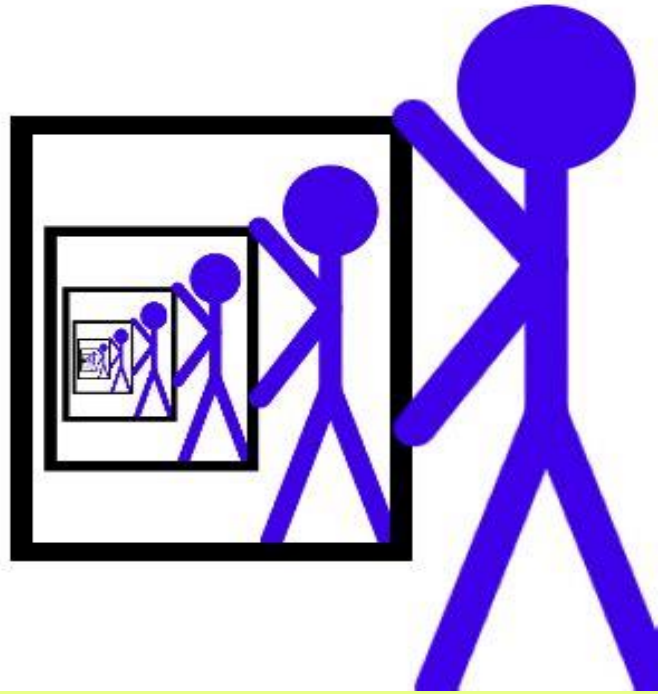
Merge Sort



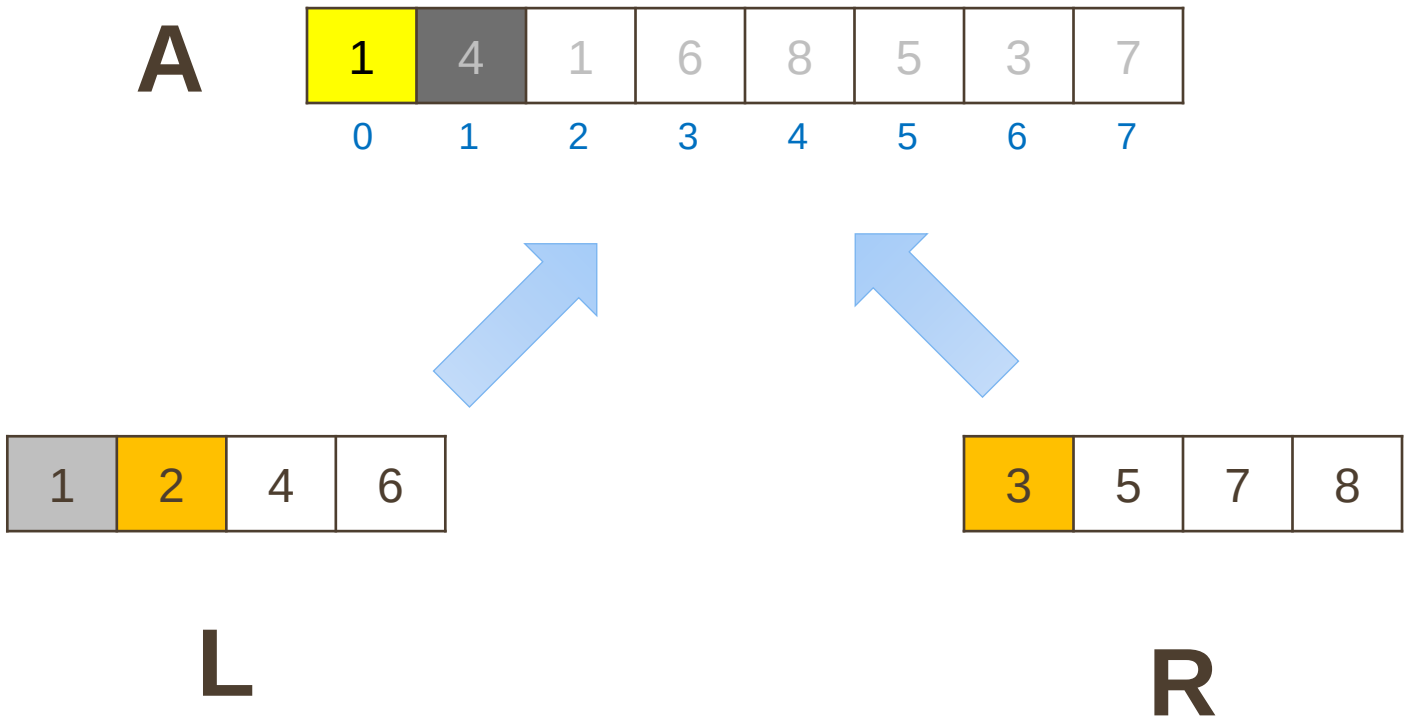


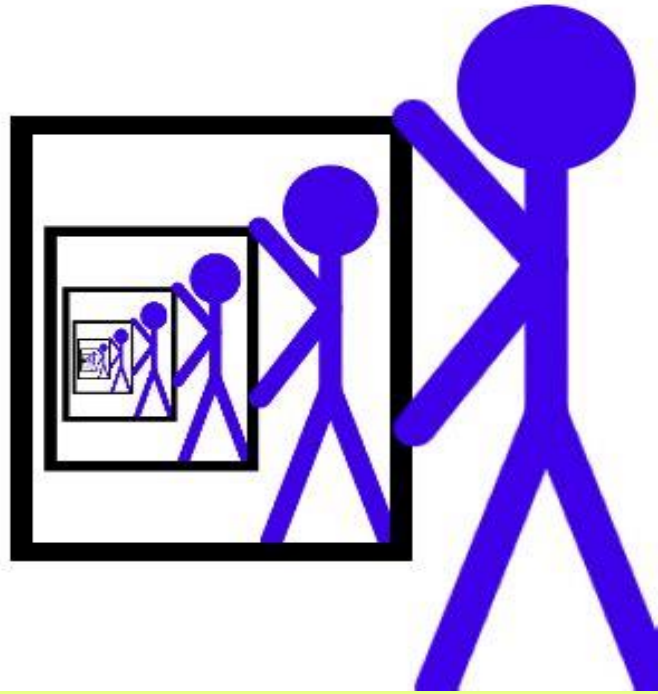
Merge Sort



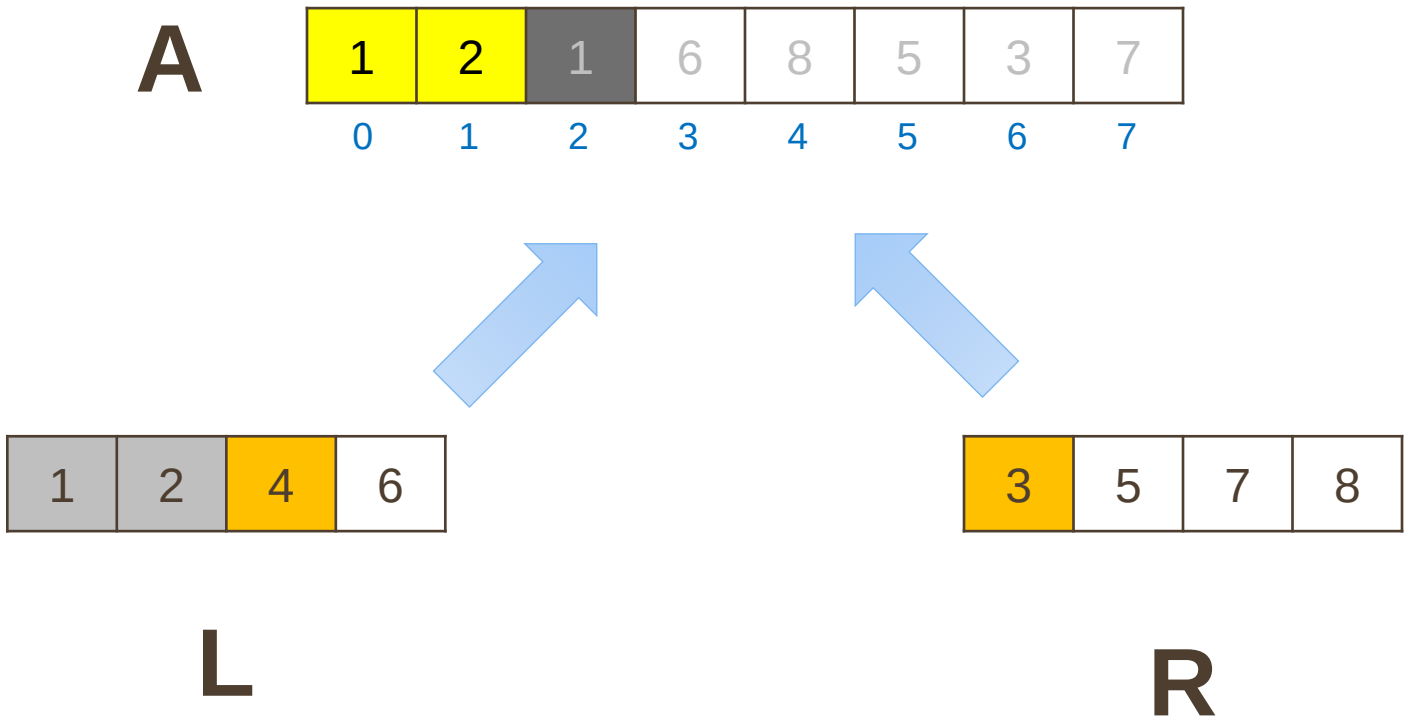


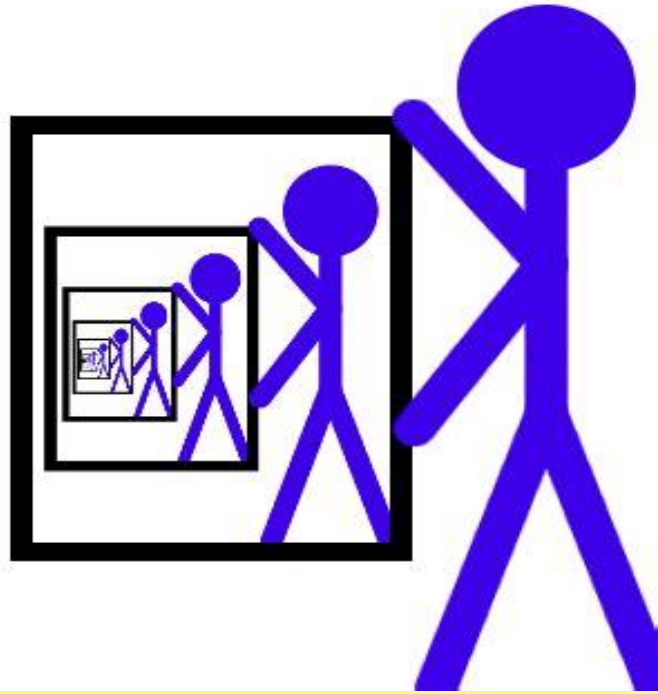
Merge Sort



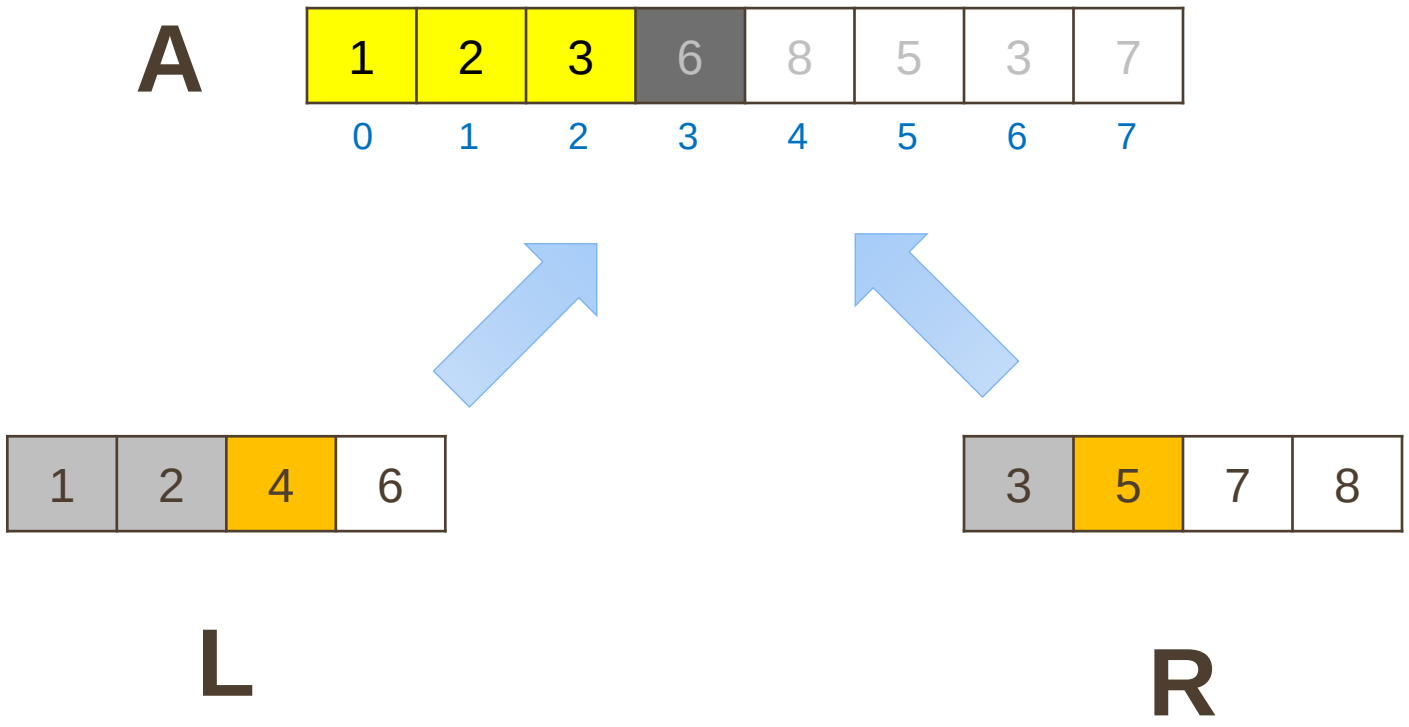


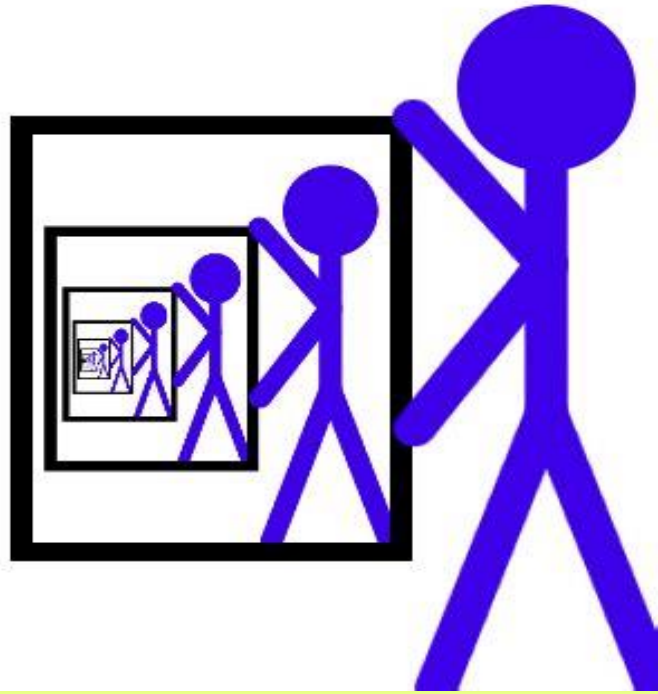
Merge Sort



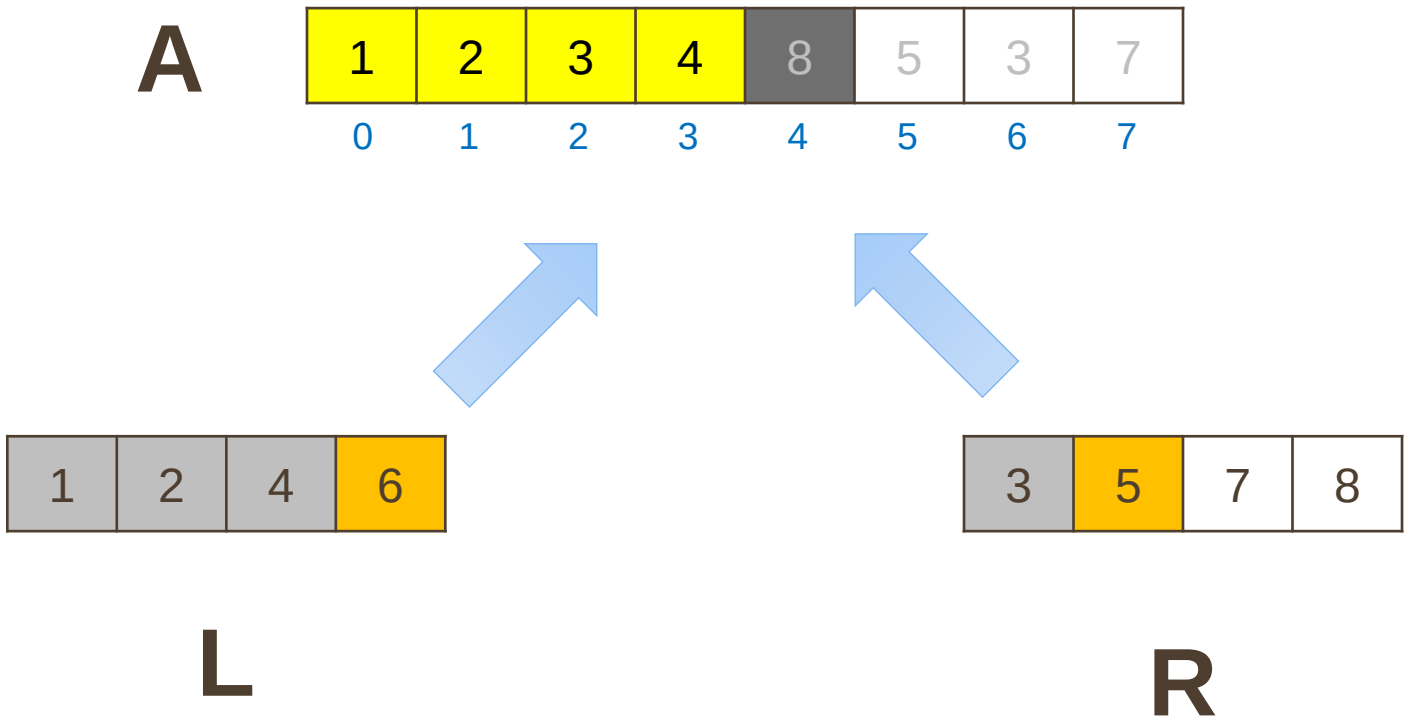


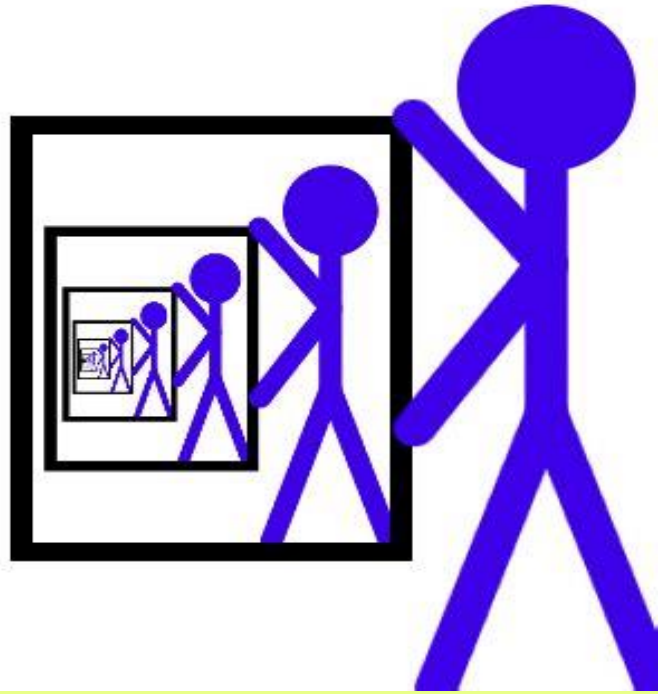
Merge Sort



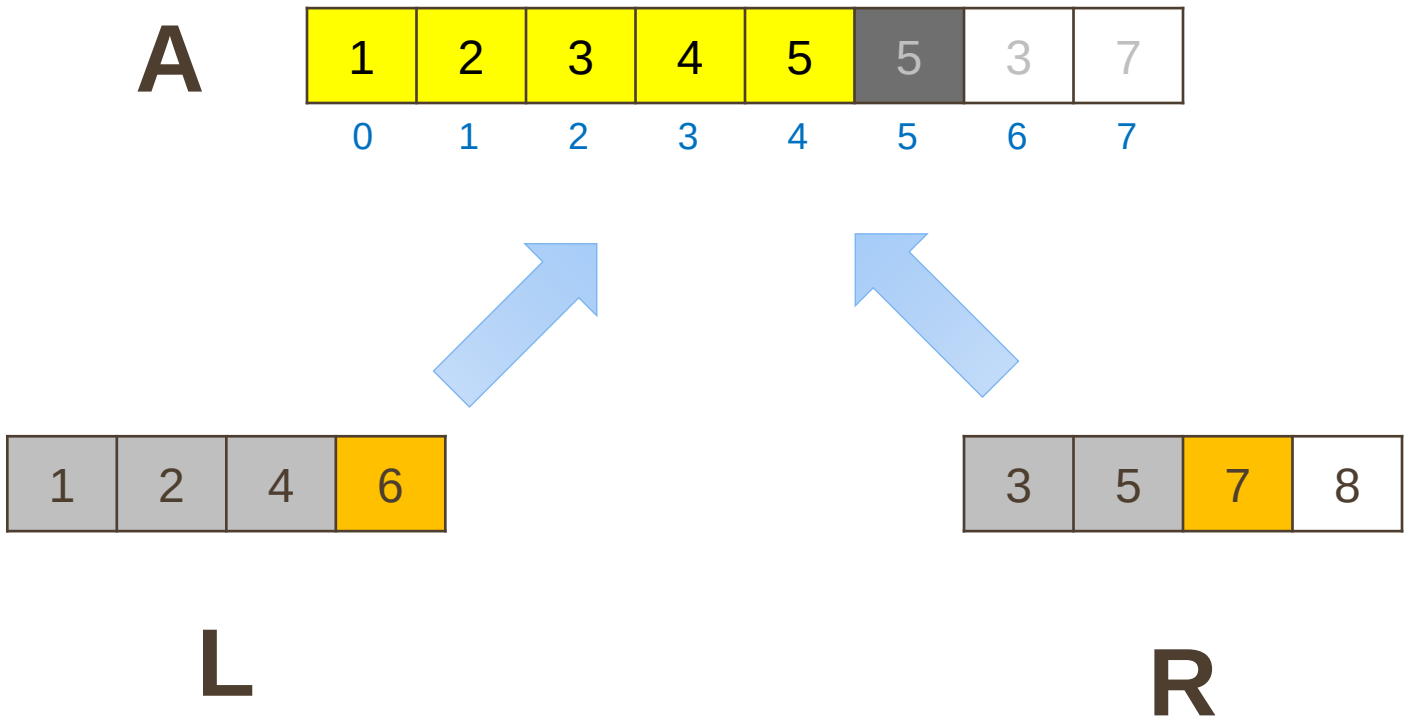


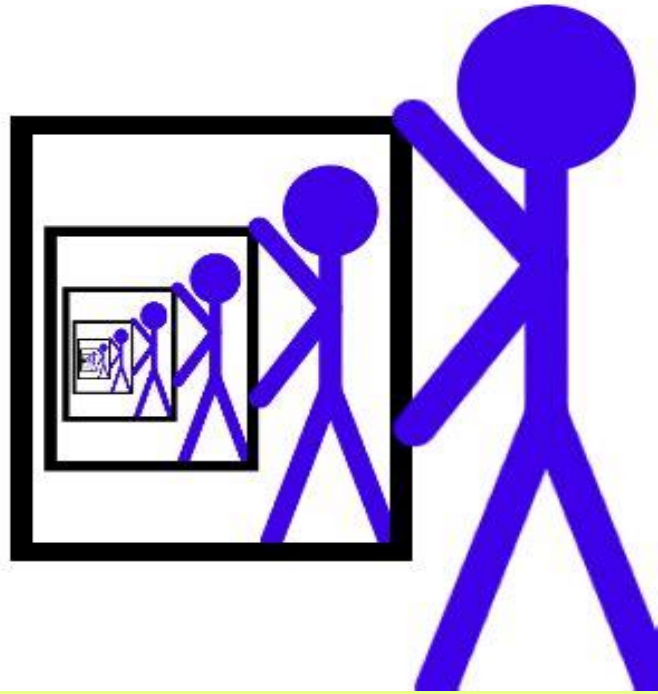
Merge Sort



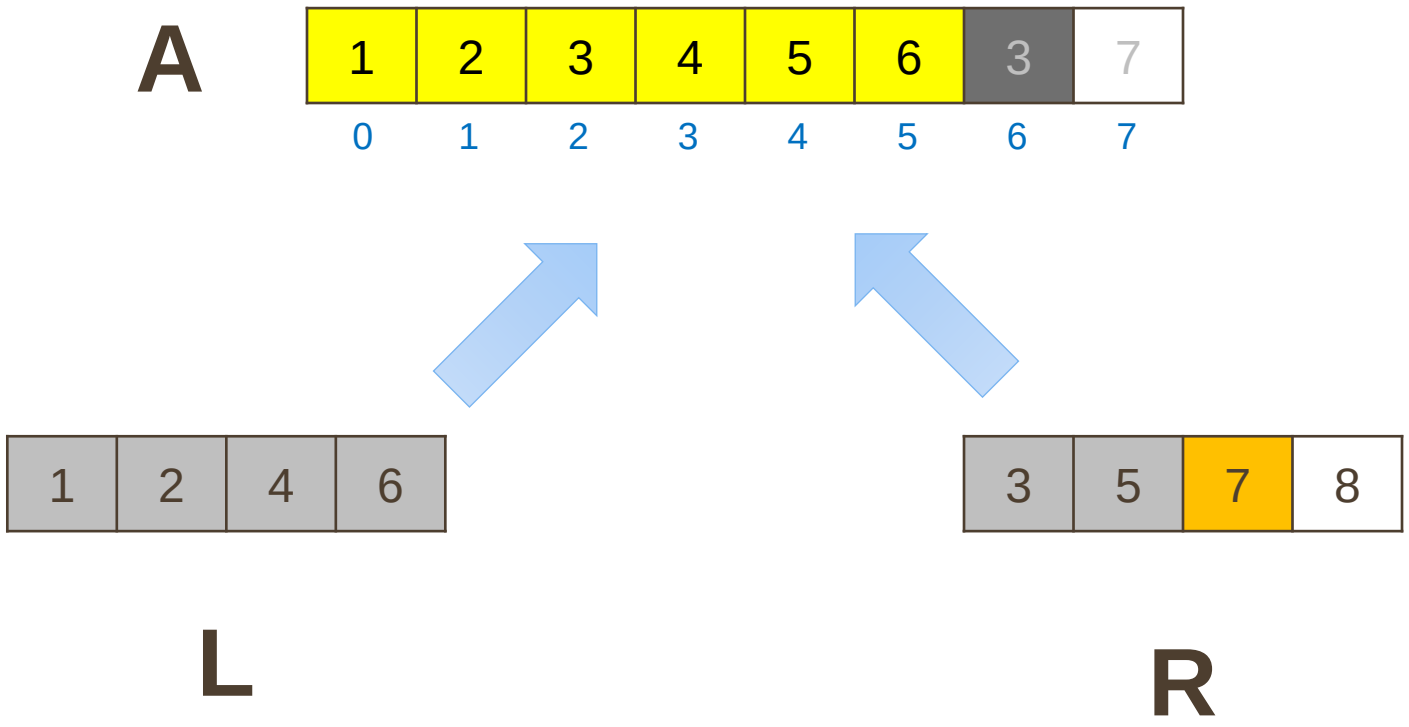


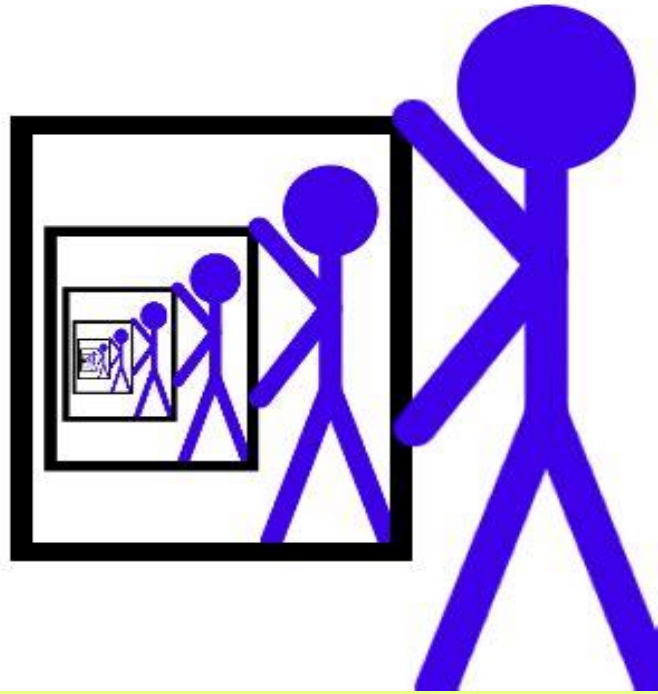
Merge Sort



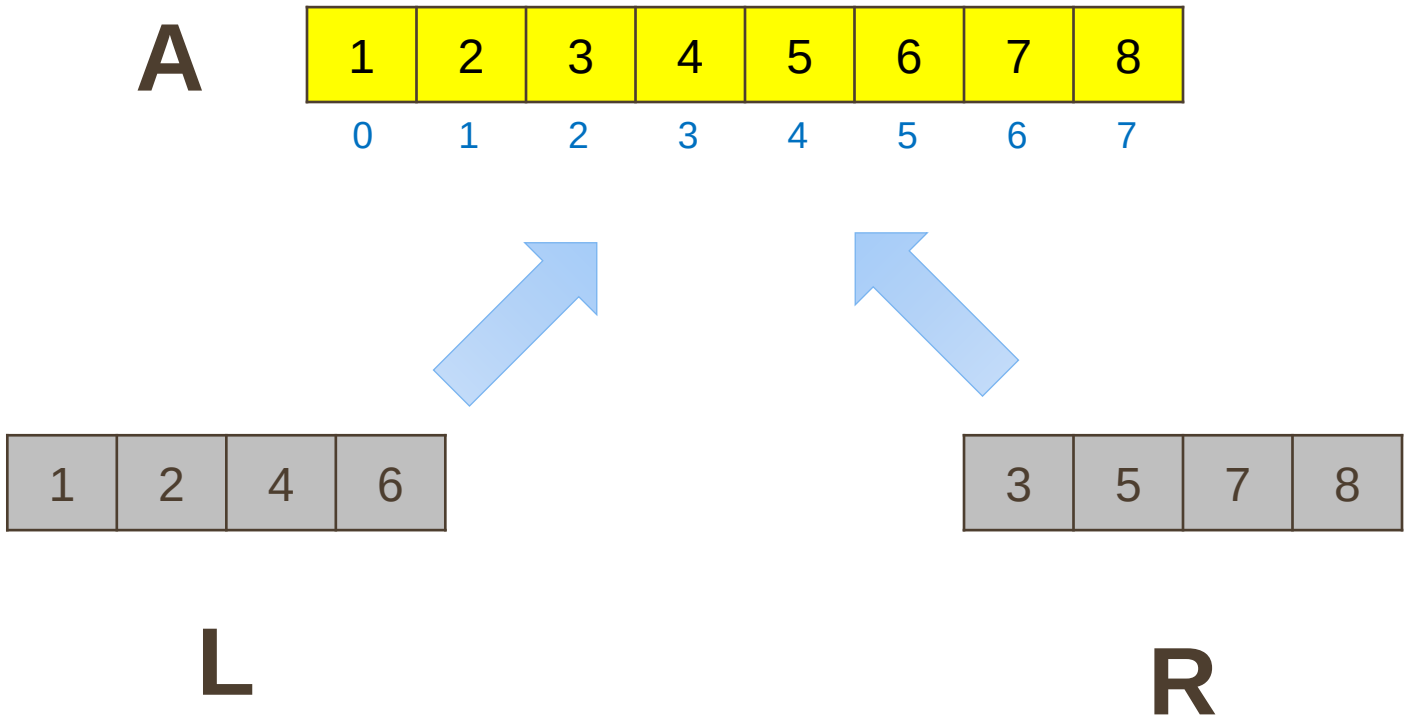


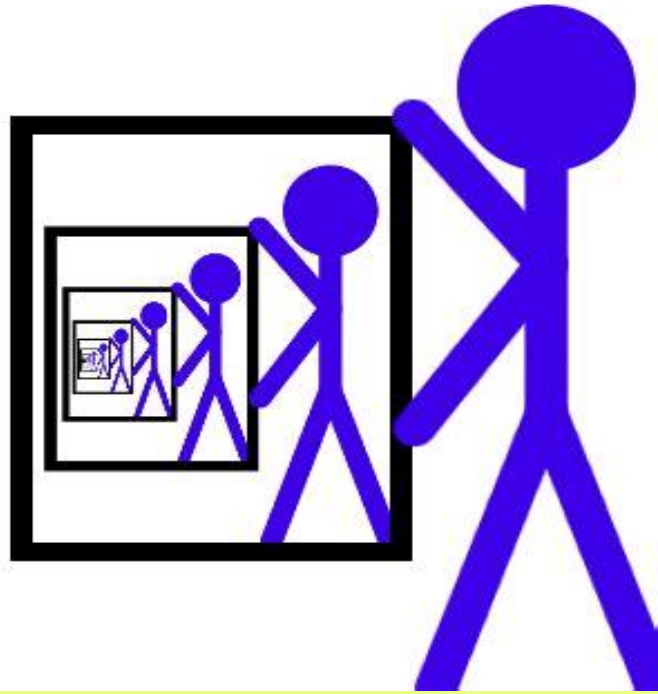
Merge Sort



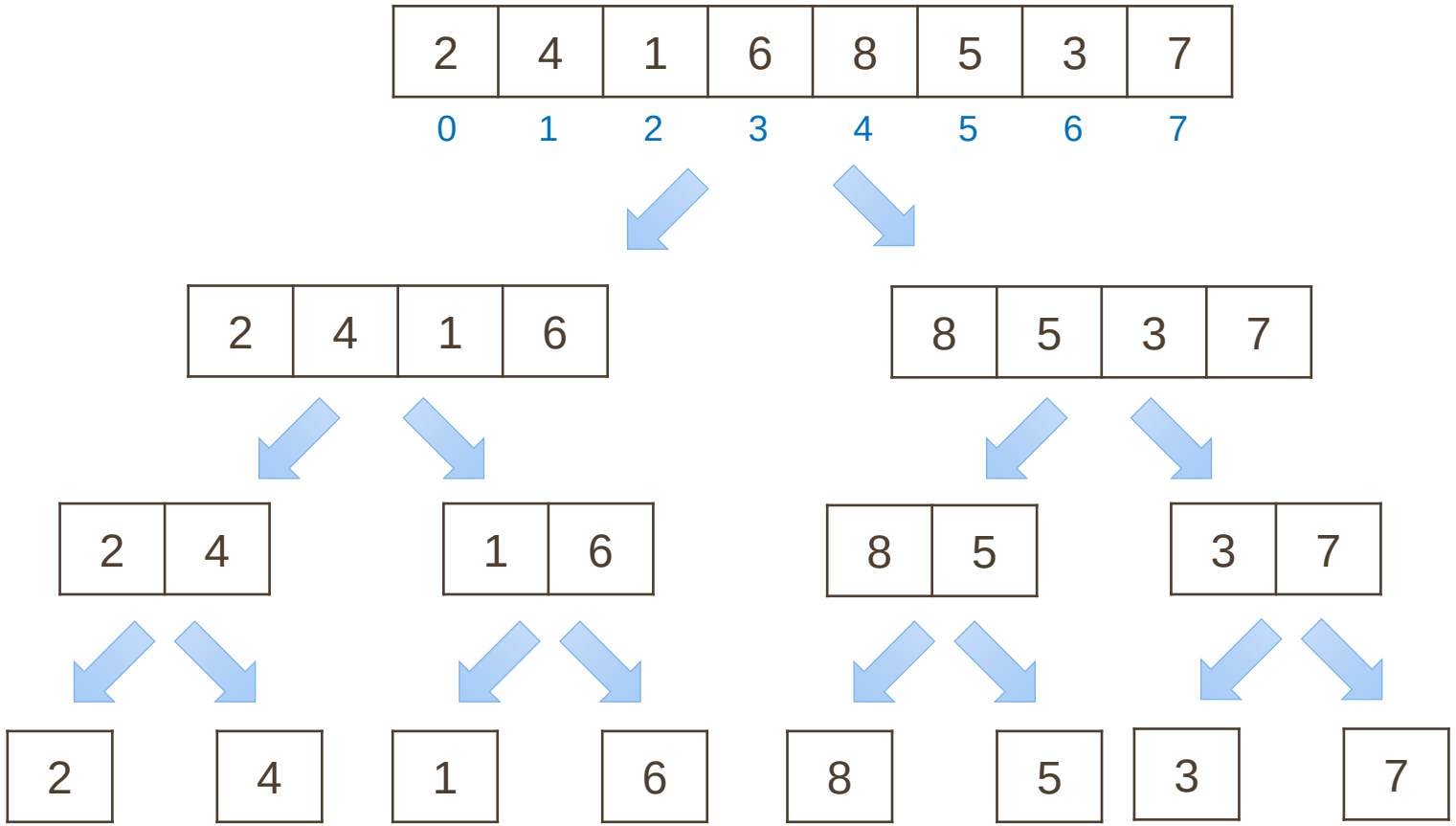


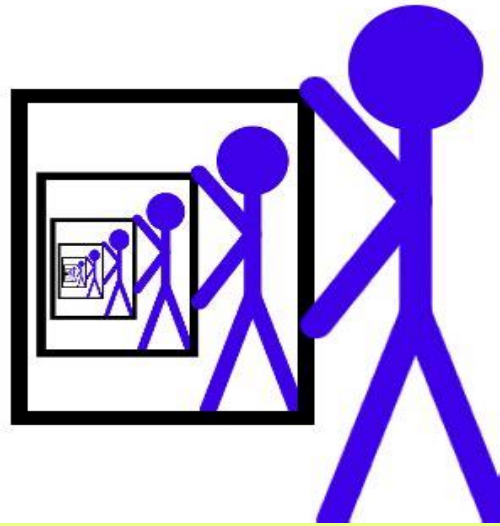
Merge Sort



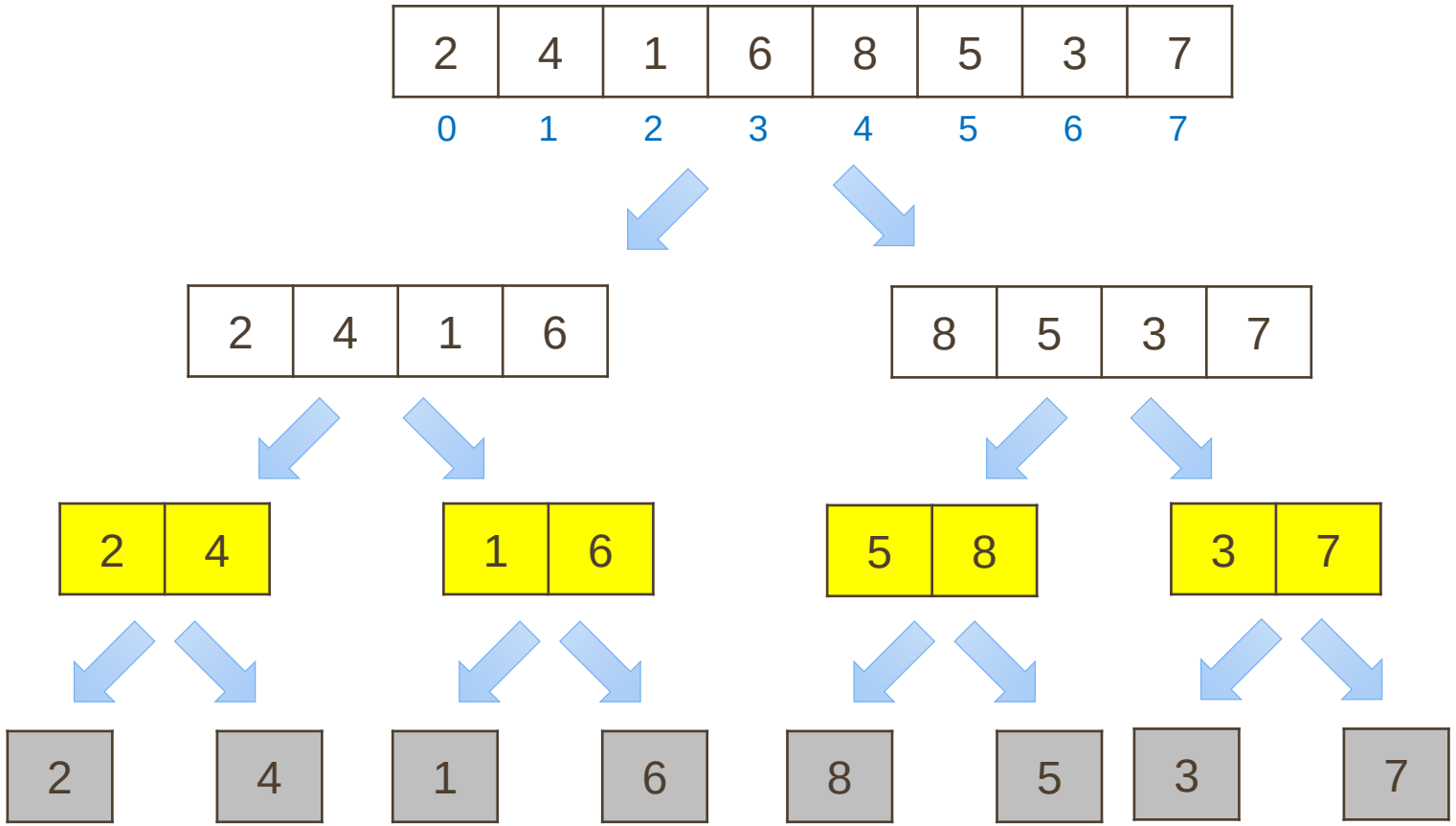


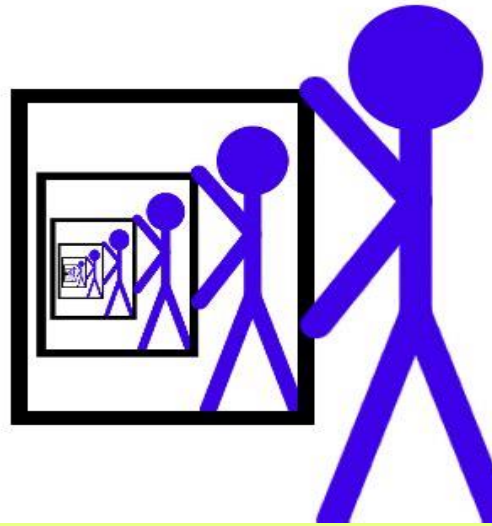
Merge Sort



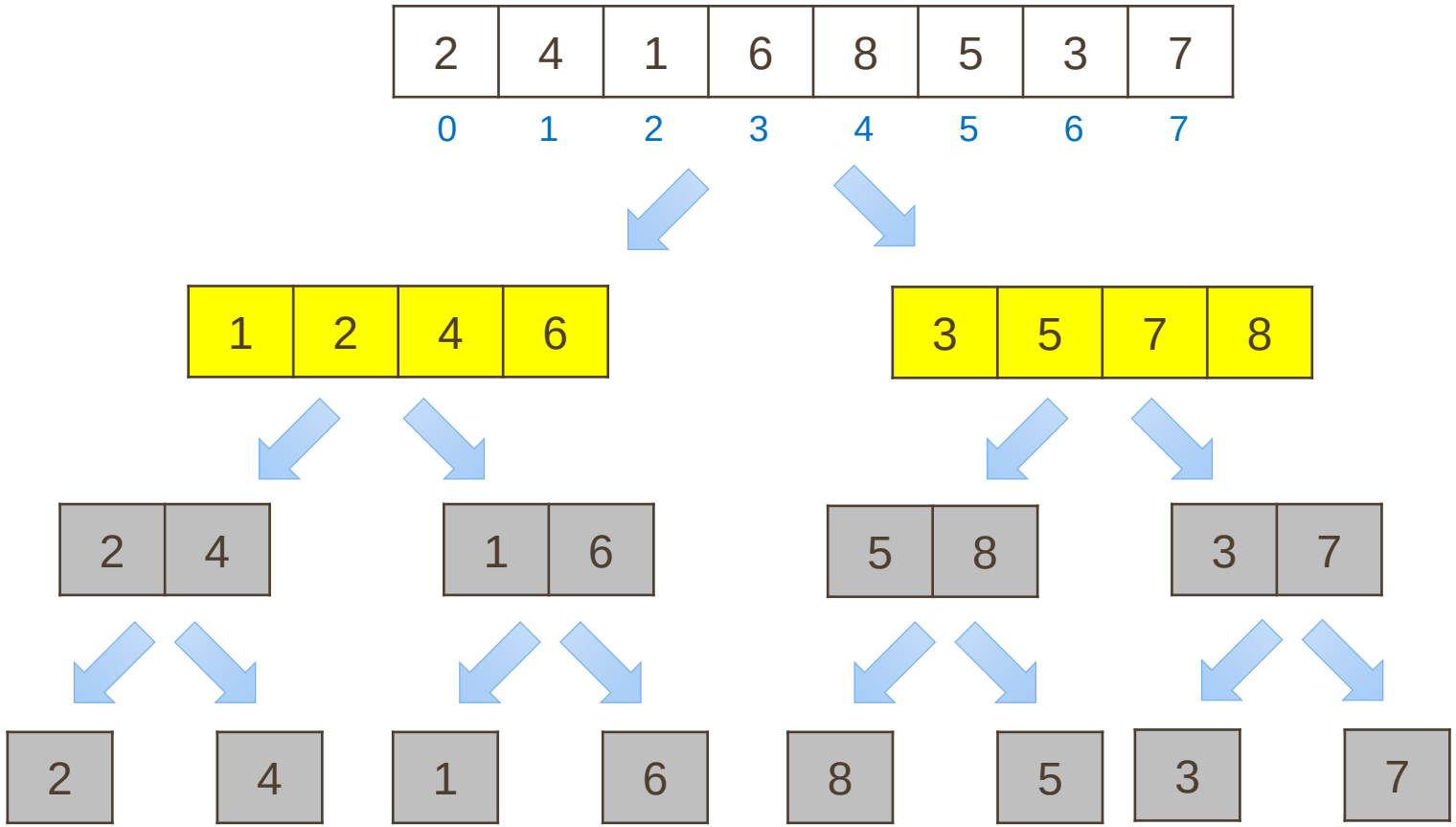


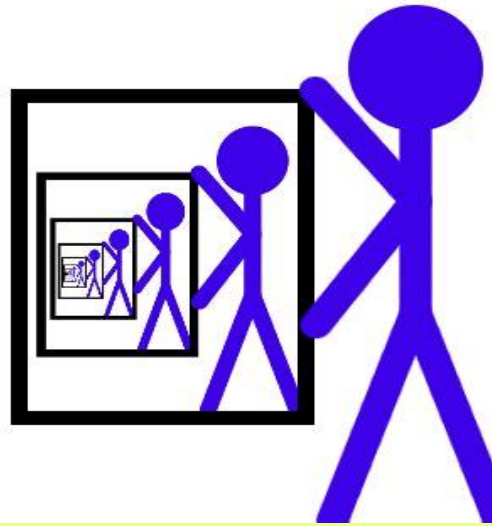
Merge Sort



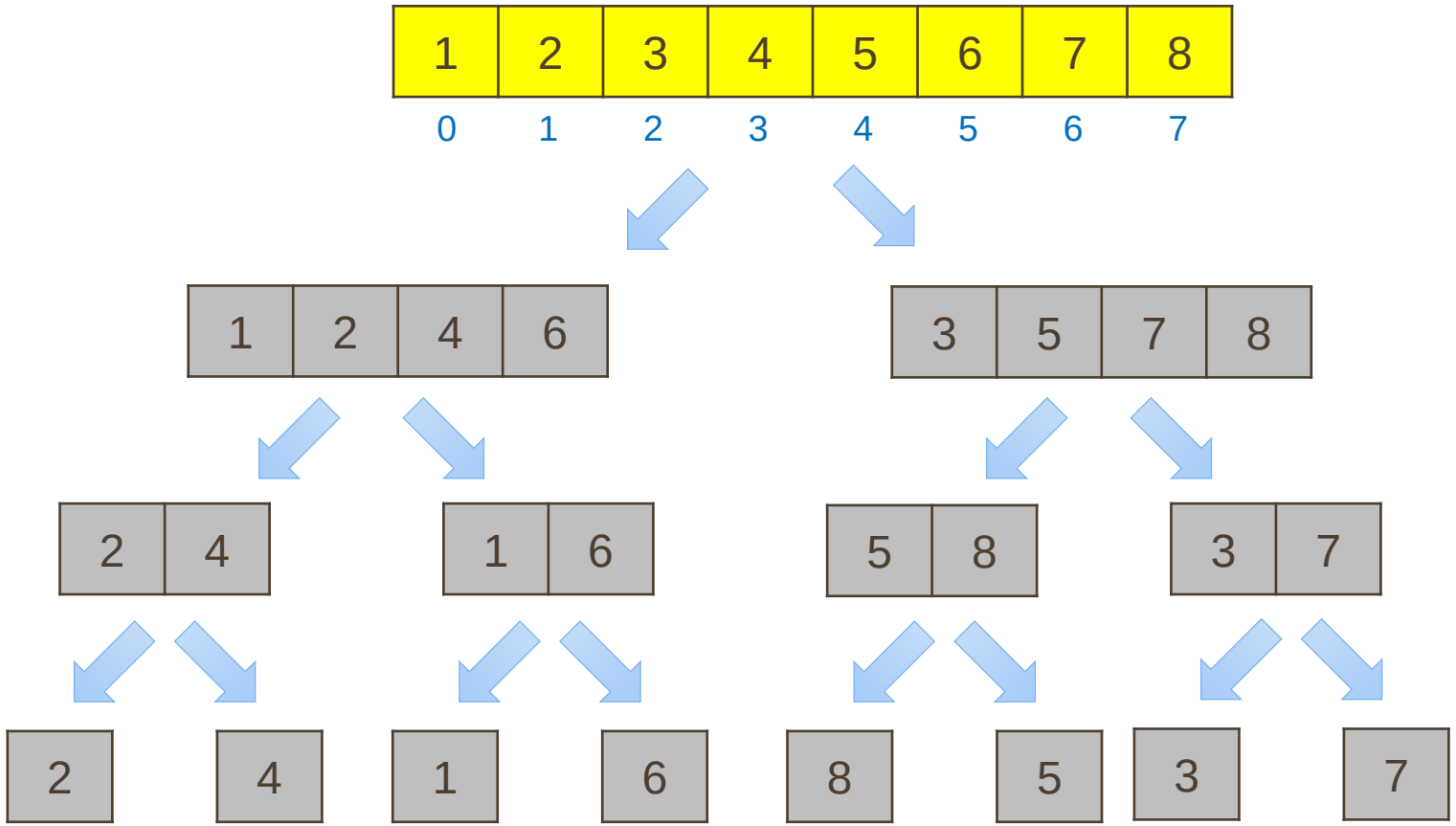


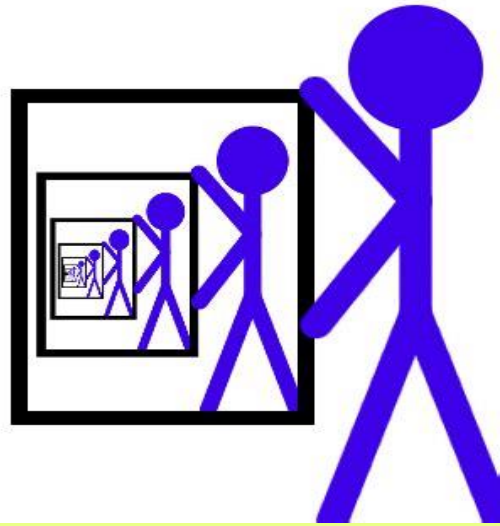
Merge Sort





Merge Sort

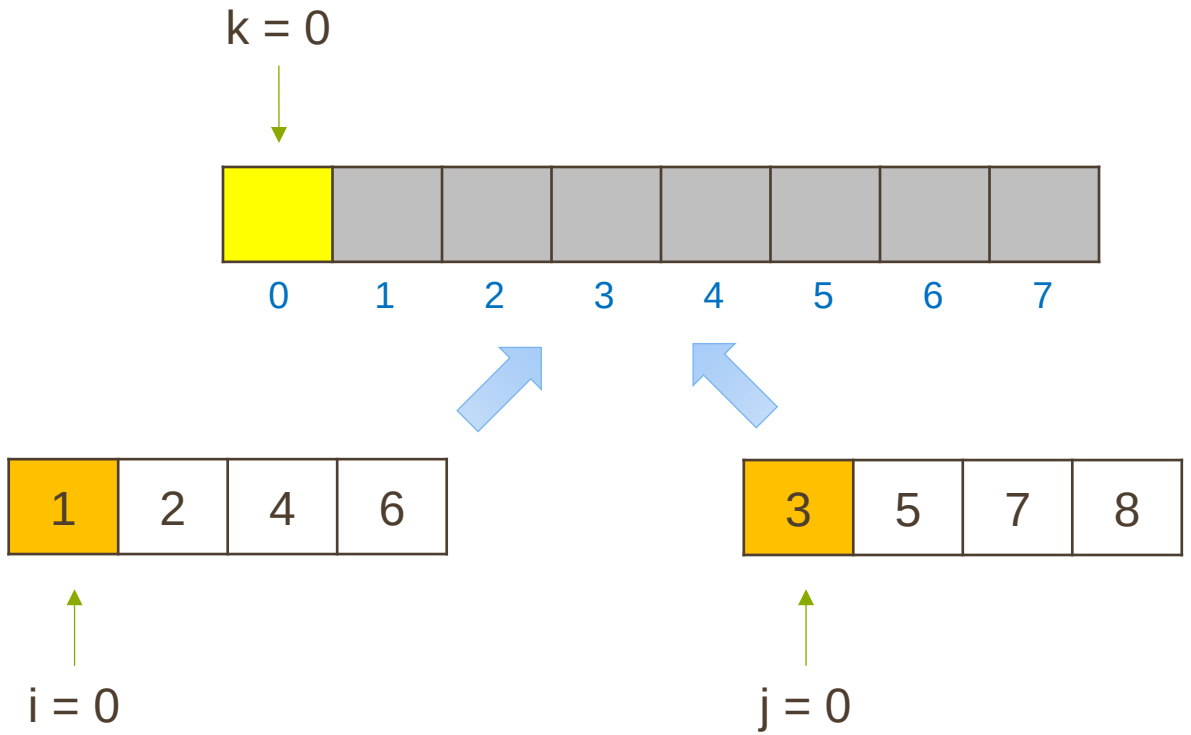




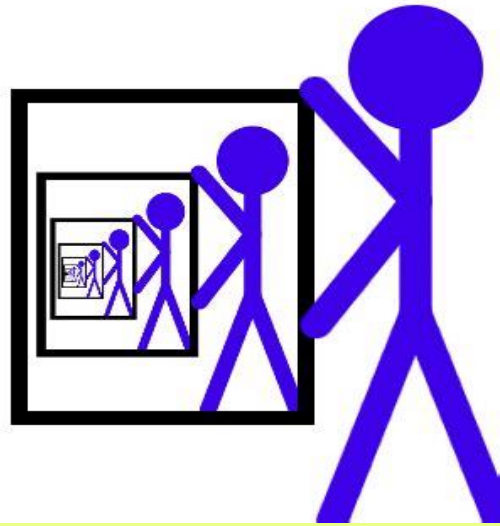
Merge Sort - Code

Divide and Conquer

Not In-Place (Need to allocate memory)



Recursive Function



Merge Sort - Code

```
1
2   def MergeSort(a_list):
3       n = len(a_list)
4
5       if n < 2:
6           return a_list
7
8       mid = n // 2
9       left = a_list[:mid]
10      right = a_list[mid:]
11
12      MergeSort(left)
13      MergeSort(right)
14
```

```
15      i = 0;    j = 0;    k = 0
16
17      while i < len(left) and j < len(right):
18          if left[i] < right[j]:
19              a_list[k] = left[i]
20              i = i + 1
21          else:
22              a_list[k] = right[j]
23              j = j + 1
24          k = k + 1
25
26      while i < len(left):
27          a_list[k] = left[i]
28          i = i + 1
29          k = k + 1
30
31      while j < len(right):
32          a_list[k] = right[j]
33          j = j + 1
34          k = k + 1
35
36      return a_list
37
```

Quick Sort

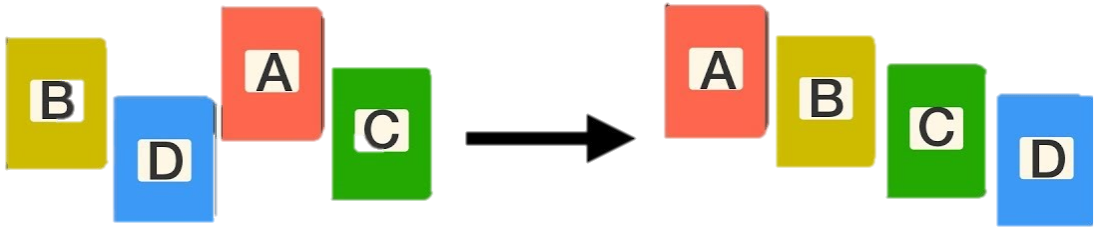
In-Place (No need to allocate memory)
Most practical sort, found in a lot of library

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

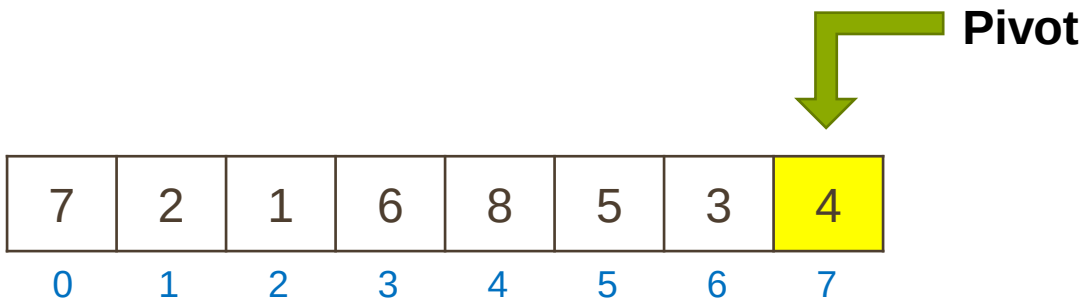
1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7

Unsorted

Sorted



Quick Sort



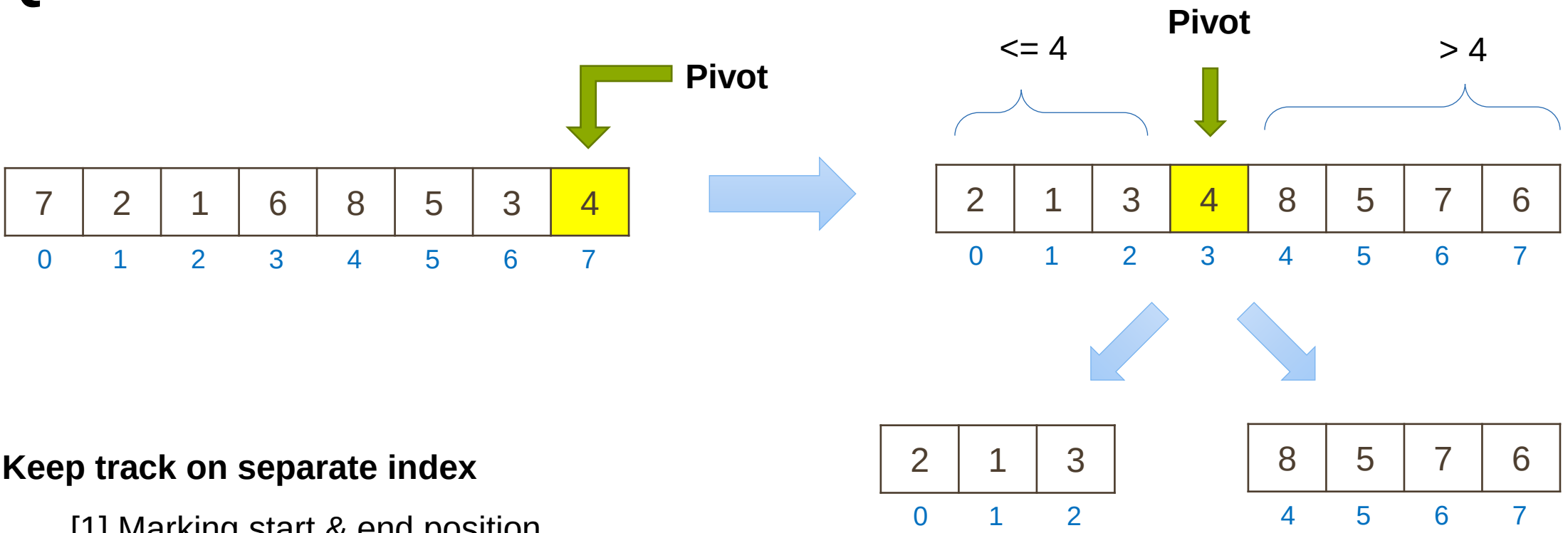
Portioning the list

[1] select Pivot (can select any point)

[2] put less value to the left side

[3] put more value to the right side

Quick Sort

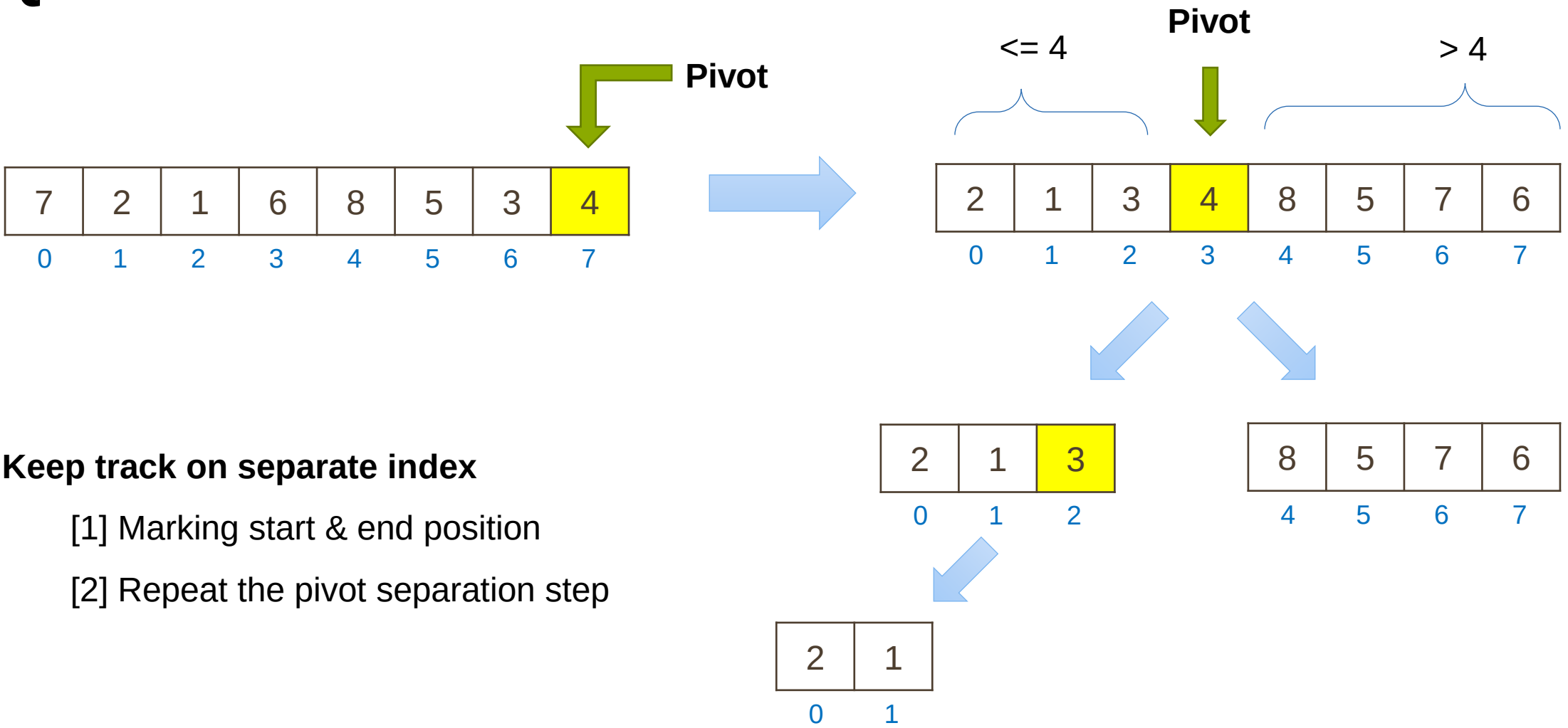


Keep track on separate index

[1] Marking start & end position

[2] Repeat the pivot separation step

Quick Sort

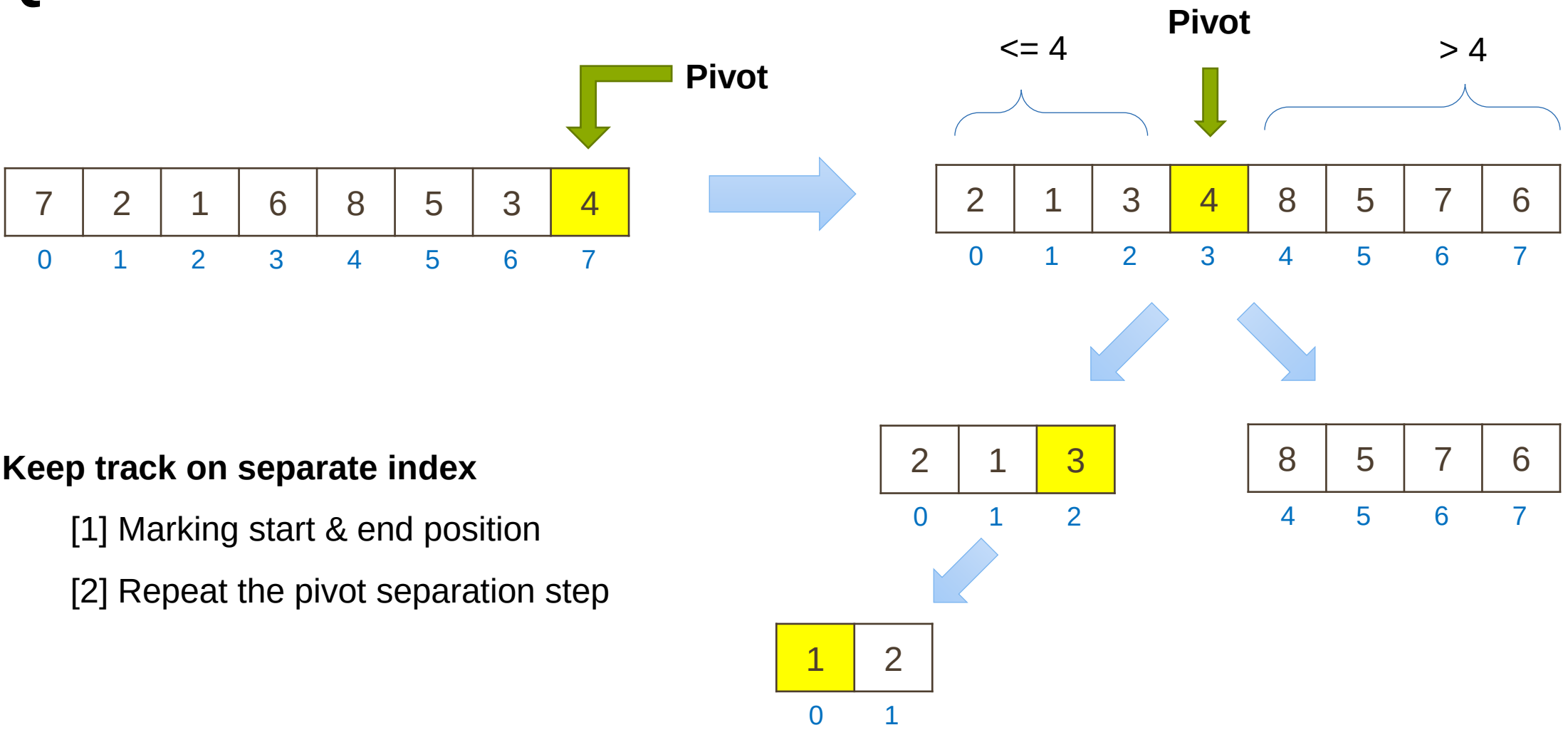


Keep track on separate index

[1] Marking start & end position

[2] Repeat the pivot separation step

Quick Sort

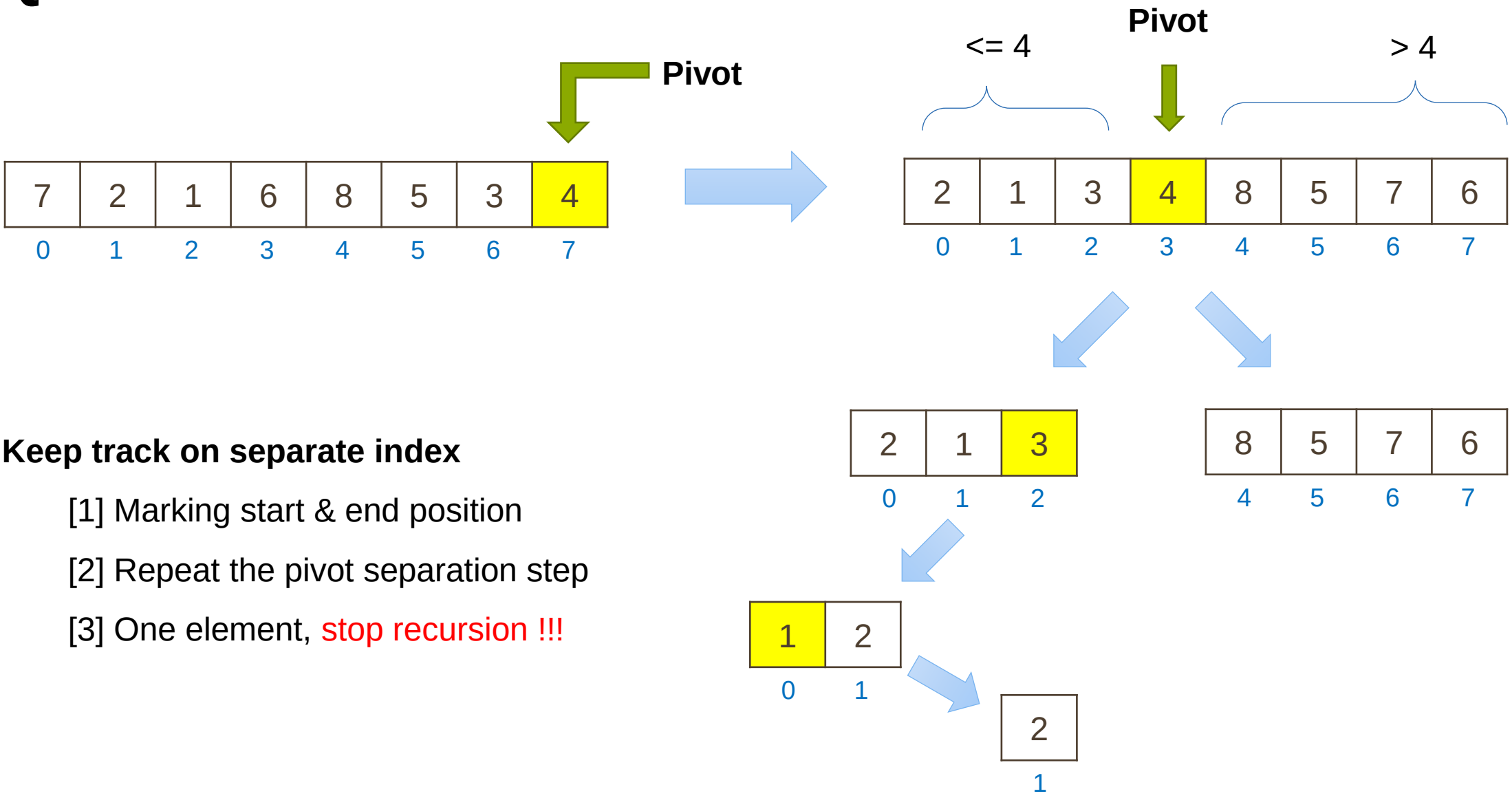


Keep track on separate index

[1] Marking start & end position

[2] Repeat the pivot separation step

Quick Sort



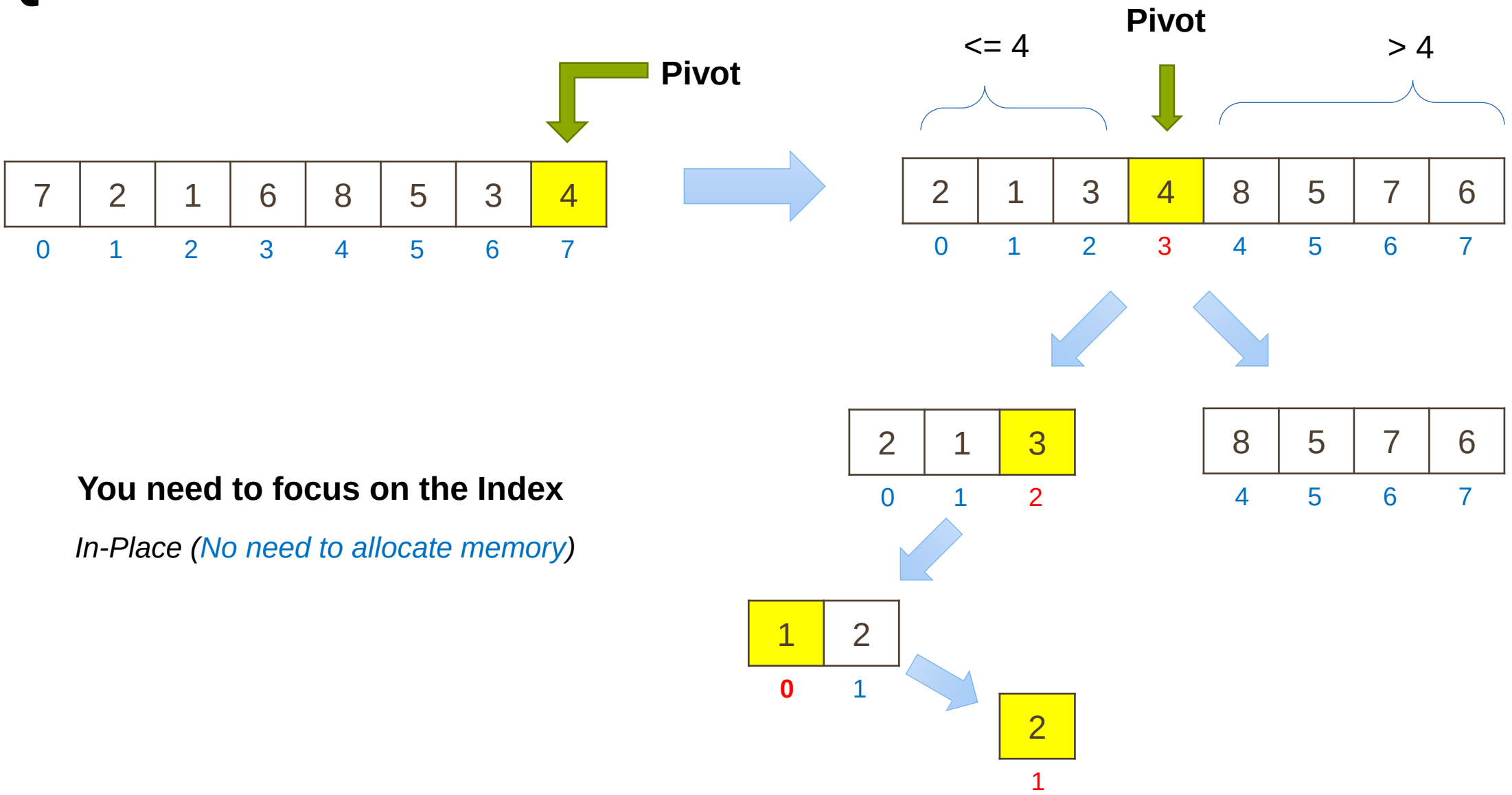
Keep track on separate index

[1] Marking start & end position

[2] Repeat the pivot separation step

[3] One element, stop recursion !!!

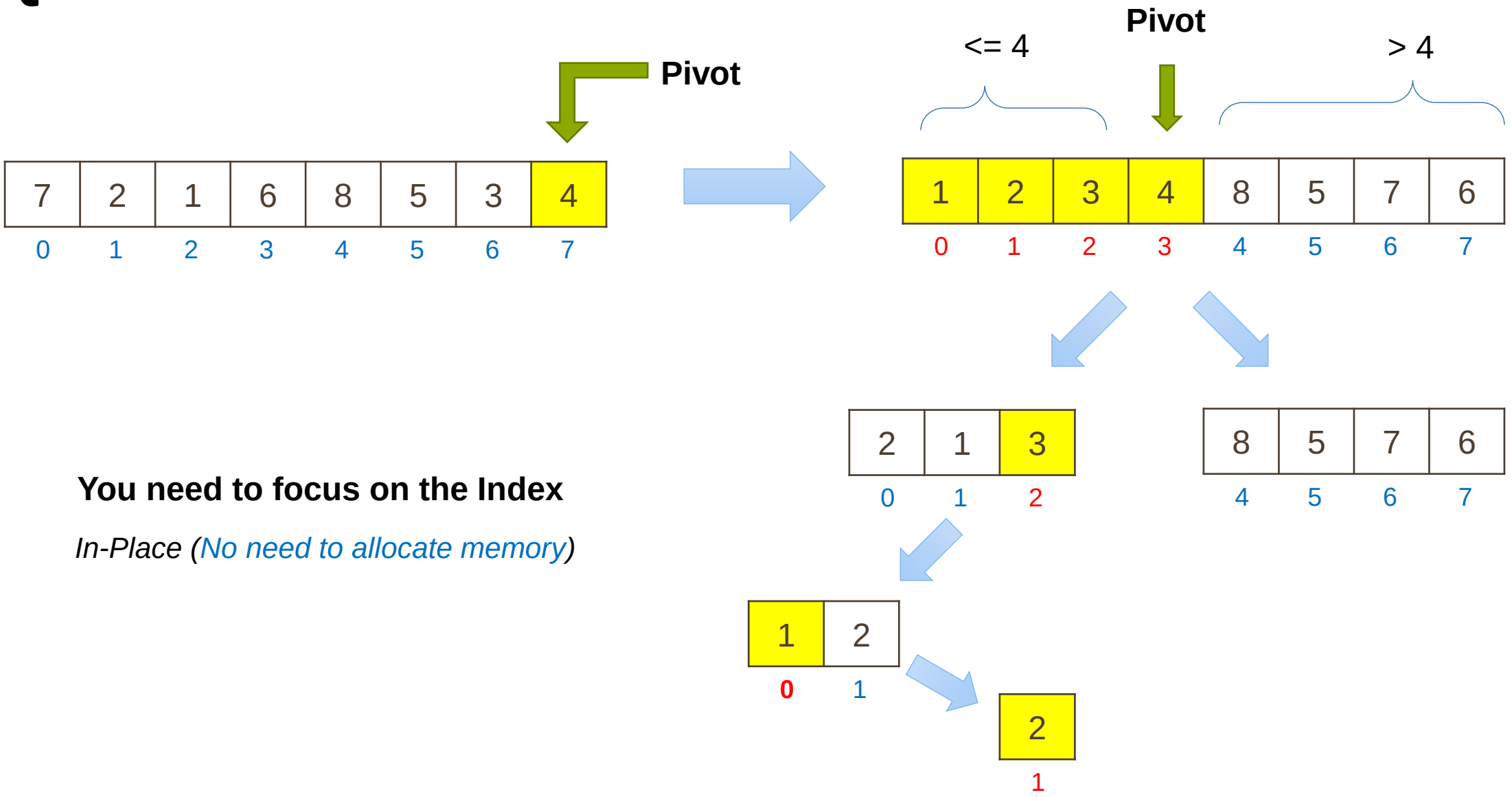
Quick Sort



You need to focus on the Index

In-Place (No need to allocate memory)

Quick Sort



You need to focus on the Index

In-Place (No need to allocate memory)

Quick Sort - Code

```
1
2   def QuickSort(a_list, start, end):
3       n = len(a_list)
4
5       if start < end:
6           pIndex = QuickSort_partition(a_list, start, end)
7
8           QuickSort(a_list, start, pIndex-1)
9           QuickSort(a_list, pIndex+1, end)
10
11      return a_list
12
```

Quick Sort - Code

```
13  def QuickSort_partition(a_list, start, end):
14      pIndex = start
15      # select last pos as index
16      pivot = a_list[end]
17
18      # push less value to the left
19      for i in range(start, end):
20          if a_list[i] <= pivot:
21              temp = a_list[i]
22              a_list[i] = a_list[pIndex]
23              a_list[pIndex] = temp
24              pIndex = pIndex + 1
25      temp = a_list[pIndex]
26      a_list[pIndex] = a_list[end]
27      a_list[end] = temp
28
29      return pIndex
30
```

Quick Sort - Code

```
30
31  if __name__ == '__main__':
32
33      A = [7, 2, 1, 6, 8, 5, 3, 4]
34      print(A)
35
36      QuickSort(A, 0, len(A)-1)
37      print(A)
```

Assignment #2

ให้นักศึกษาเปรียบเทียบประสิทธิภาพของ Sorting Algorithm ที่เรียนมา (Bubble, Insertion, Selection, Merge, Quick Sort) โดยทดลองกับฐานข้อมูล Thai Address

- หลังจากตัด String มาแล้วให้เก็บข้อมูล 2 Field คือ ID และ Name
- เขียนฟังก์ชันเพื่อสลับลำดับของข้อมูลแบบ Random
- ทดลองใช้ Sorting Algorithm 5 รูปแบบที่เรียนมาเพื่อเรียงลำดับข้อมูลตามเลข ID จาก น้อย ไป มาก โดยให้เขียนฟังก์ชันจับเวลาการทำงาน
- แสดงผลระยะเวลาการทำงานที่ใช้ในการจัดเรียงข้อมูลของแต่ละ Algorithm
- ส่งงานโดยการนำ Source Code ขึ้น GitHub และเปิดเป็น Public
- เขียนชื่อและรหัสนักศึกษา ใน readme ให้ชัดเจน (1-2 คน ต่อ Repository)