11 Sorting

For COMSC 132

Sam Bowne

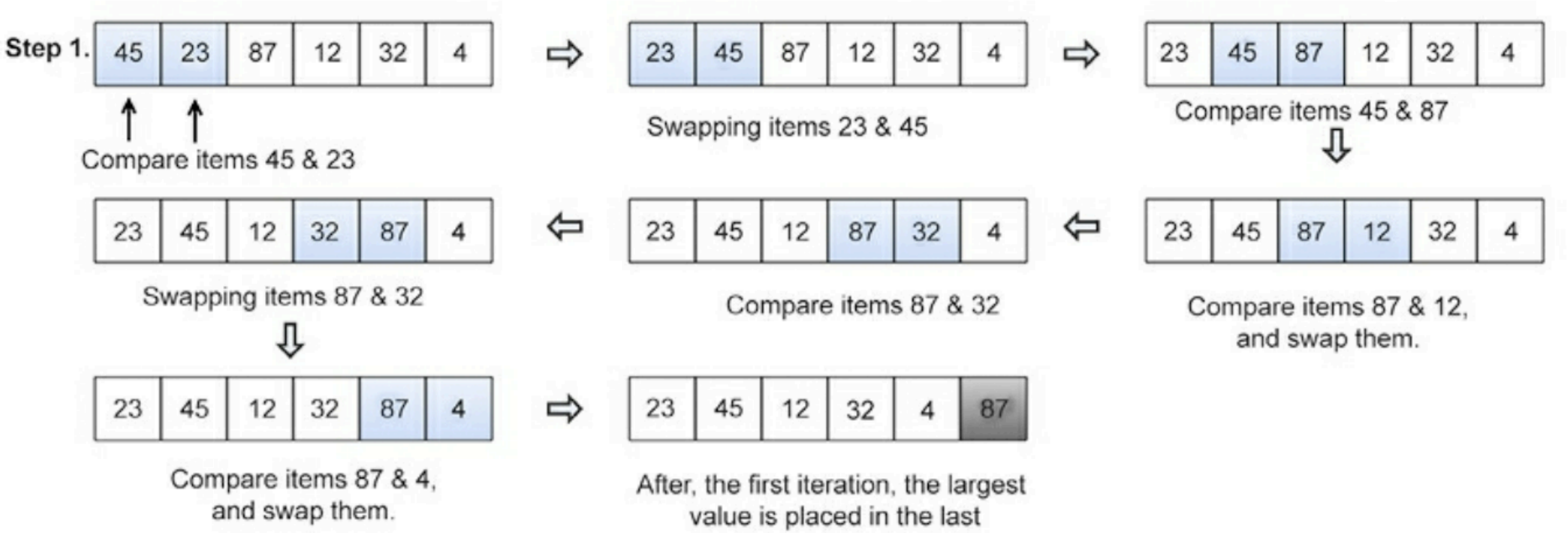
Nov 10, 2024

Topics

- Bubble sort
- Insertion sort
- Selection sort
- Quicksort
- Timsort

Bubble sort

- Compare adjacent elements
- Swap if not in order



Step 1.

45	23	87	12	32	4

Compare items 45 & 23

23	45	87	12	32	4

Swapping items 23 & 45

23	45	87	12	32	4

Compare items 45 & 87

23	45	87	12	32	4

Compare items 87 & 12, and swap them.

23	45	12	87	32	4

Compare items 87 & 32

23	45	12	32	87	4

Swapping items 87 & 32

23	45	12	32	87	4

Compare items 87 & 4, and swap them.

23	45	12	32	4	87

After, the first iteration, the largest value is placed in the last

Bubble sort

- Takes $\boldsymbol{n}$-1 operations to move the largest element to the top
- Next cycle takes $\boldsymbol{n}$-2 operations, etc.

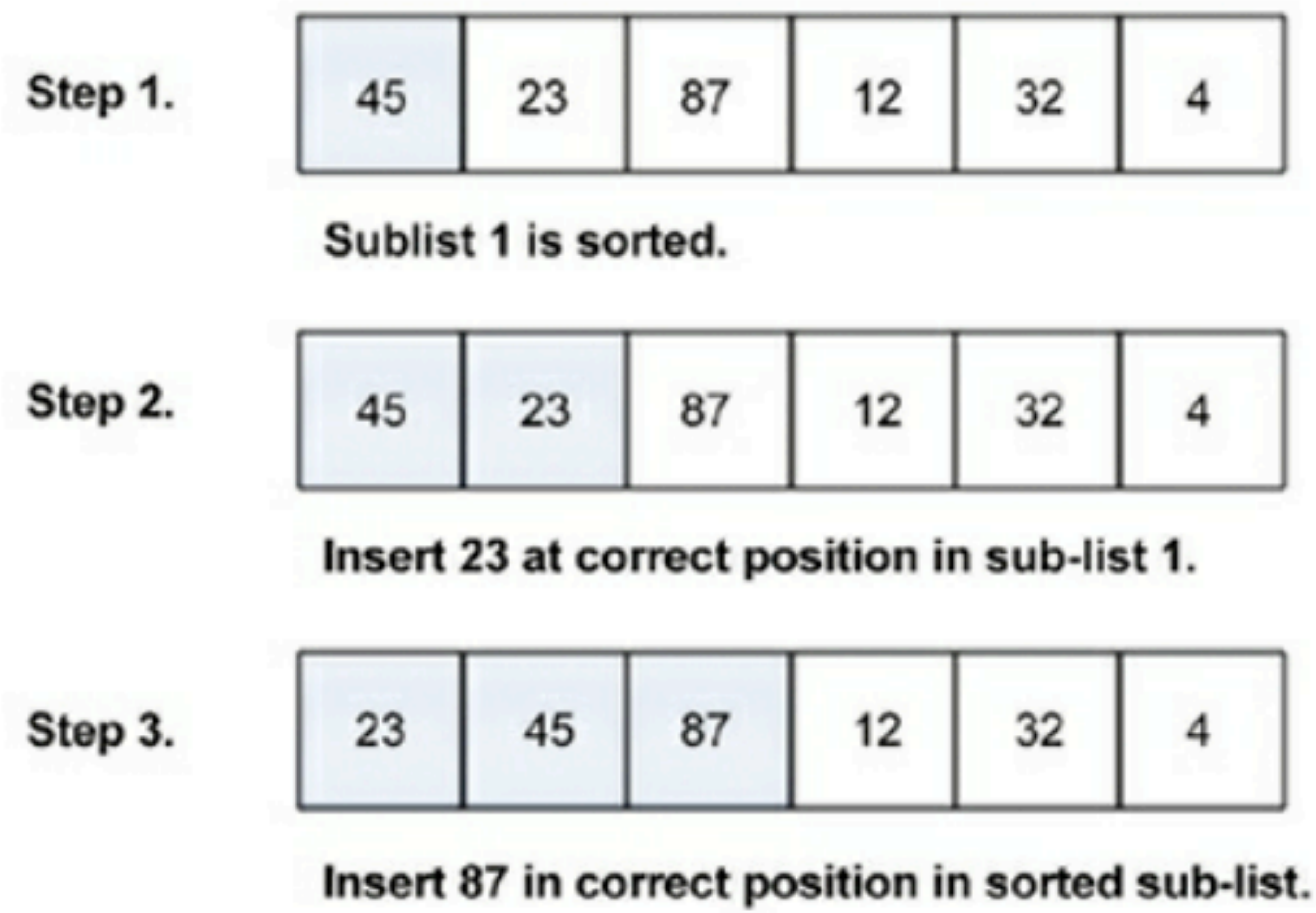
$$
\begin{array}{ccccccccccc}
(\boldsymbol{n}-1) & + & (\boldsymbol{n}-2) & + & (\boldsymbol{n}-3) & + \ldots & + & 2 & + & 1 \\
1 & + & 2 & + & 3 & + \ldots & + & (\boldsymbol{n}-2) & + & (\boldsymbol{n}-1) \\
\hline
\boldsymbol{n} & + & \boldsymbol{n} & + & \boldsymbol{n} & + \ldots & + & \boldsymbol{n} & + & \boldsymbol{n}
\end{array}
$$

# operations = $\boldsymbol{n}(\boldsymbol{n}-1)/2$

Bubble sort

- Complexity O($\boldsymbol{n}^2$)
- Too slow for use on large lists

Insertion sort



Step 1.

45	23	87	12	32	4

Sublist 1 is sorted.

Step 2.

45	23	87	12	32	4

Insert 23 at correct position in sub-list 1.

Step 3.

23	45	87	12	32	4

Insert 87 in correct position in sorted sub-list.

Insertion sort

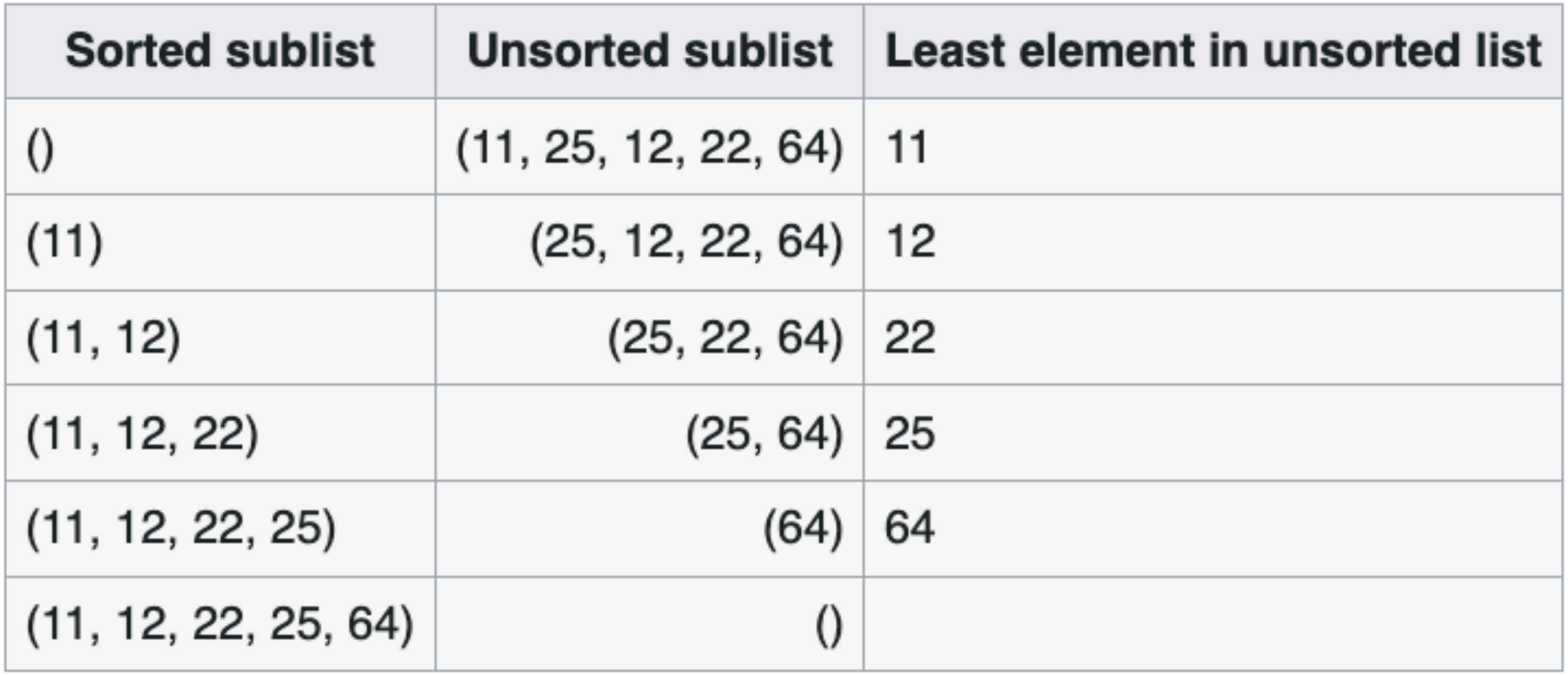
- The first insertion takes one comparision
- The next one takes two, etc.

$$1 + 2 + 3 + \ldots + (\boldsymbol{n}-2) + (\boldsymbol{n}-1)$$

- Complexity O($\boldsymbol{n}^2$)
- Good to use when the list has a small number of elements
- And the data arrives one by one

Selection sort

- Move smallest element to position 1
- Move next-smallest to position 2
- etc.



Sorted sublist	Unsorted sublist	Least element in unsorted list
()	(11, 25, 12, 22, 64)	11
(11)	(25, 12, 22, 64)	12
(11, 12)	(25, 22, 64)	22
(11, 12, 22)	(25, 64)	25
(11, 12, 22, 25)	(64)	64
(11, 12, 22, 25, 64)	()	

Selection sort

- First element takes ($\boldsymbol{n}$-1) comparisons

  $(\boldsymbol{n}-1) + (\boldsymbol{n}-2) + (\boldsymbol{n}-3) + \ldots + 2 + 1$

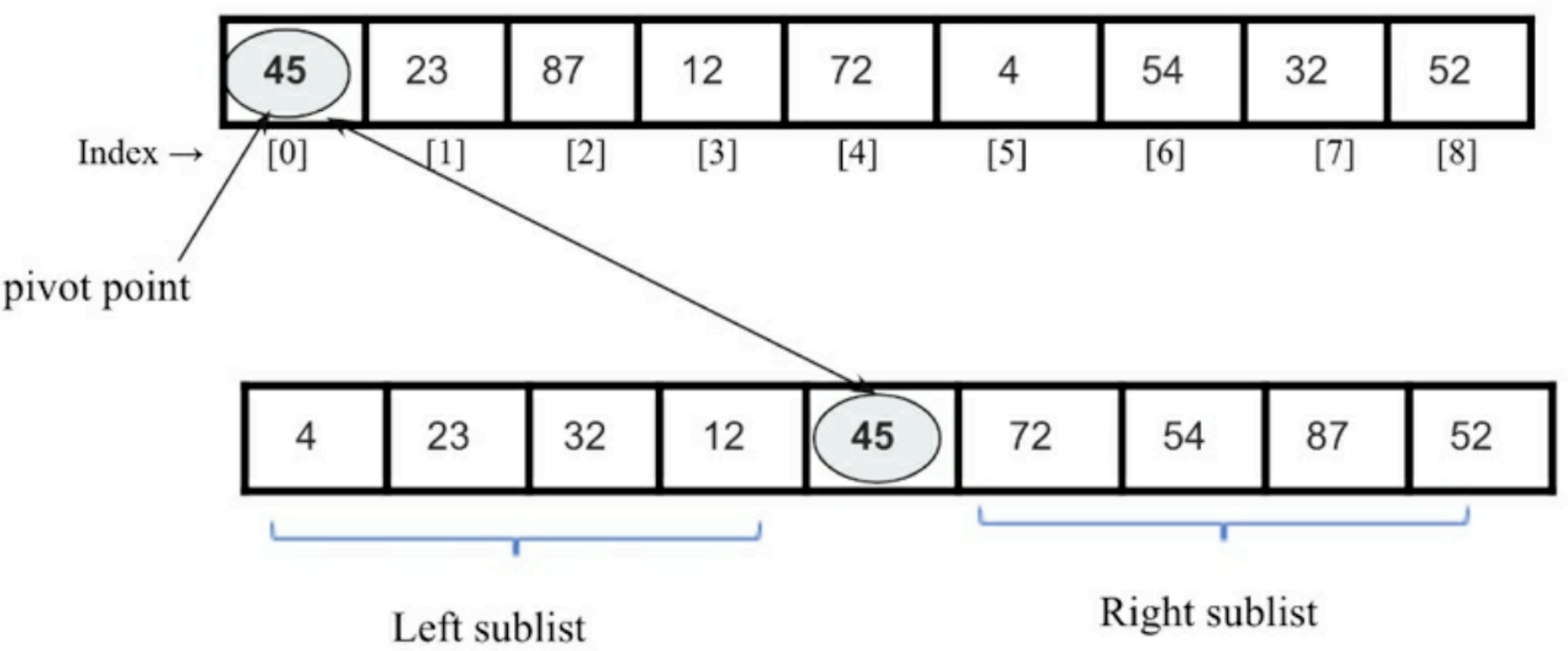
- Complexity $O(\boldsymbol{n}^2)$
- Too slow for use on large lists

- Note: diagram in the textbook is wrong
- The image on the previous slide is from Wikipedia

Quicksort

- Divide and conquer
- Choose a ***pivot element***
  - Such as the first element
- Place all smaller elements to its left
  - All larger elements to its right
- Repeat for the two sublists

Quicksort



Index →	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	45	23	87	12	72	4	54	32	52

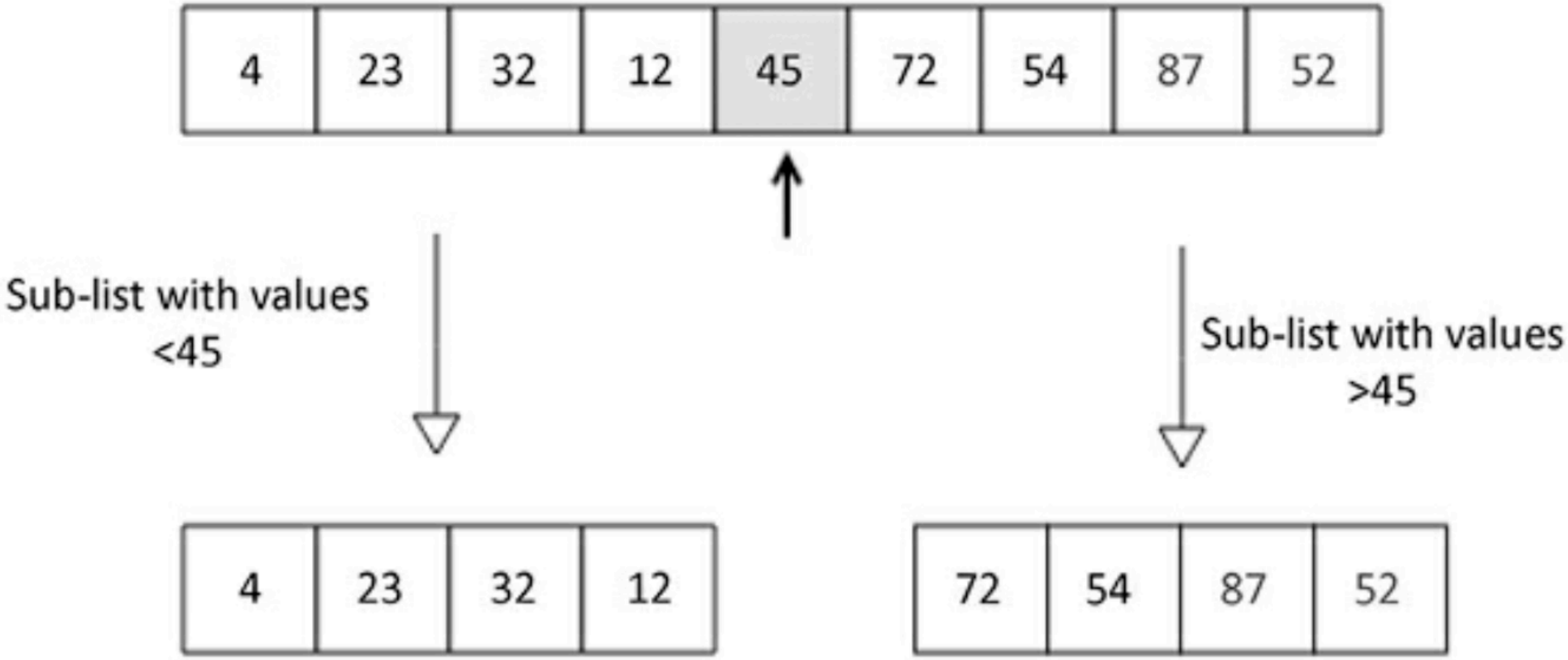
pivot point

4	23	32	12	45	72	54	87	52

Left sublist

Right sublist

Quicksort


4
23
32
12
45
72
54
87
52
Sub-list with values
<45
Sub-list with values
>45
4
23
32
12
72
54
87
52

Quicksort

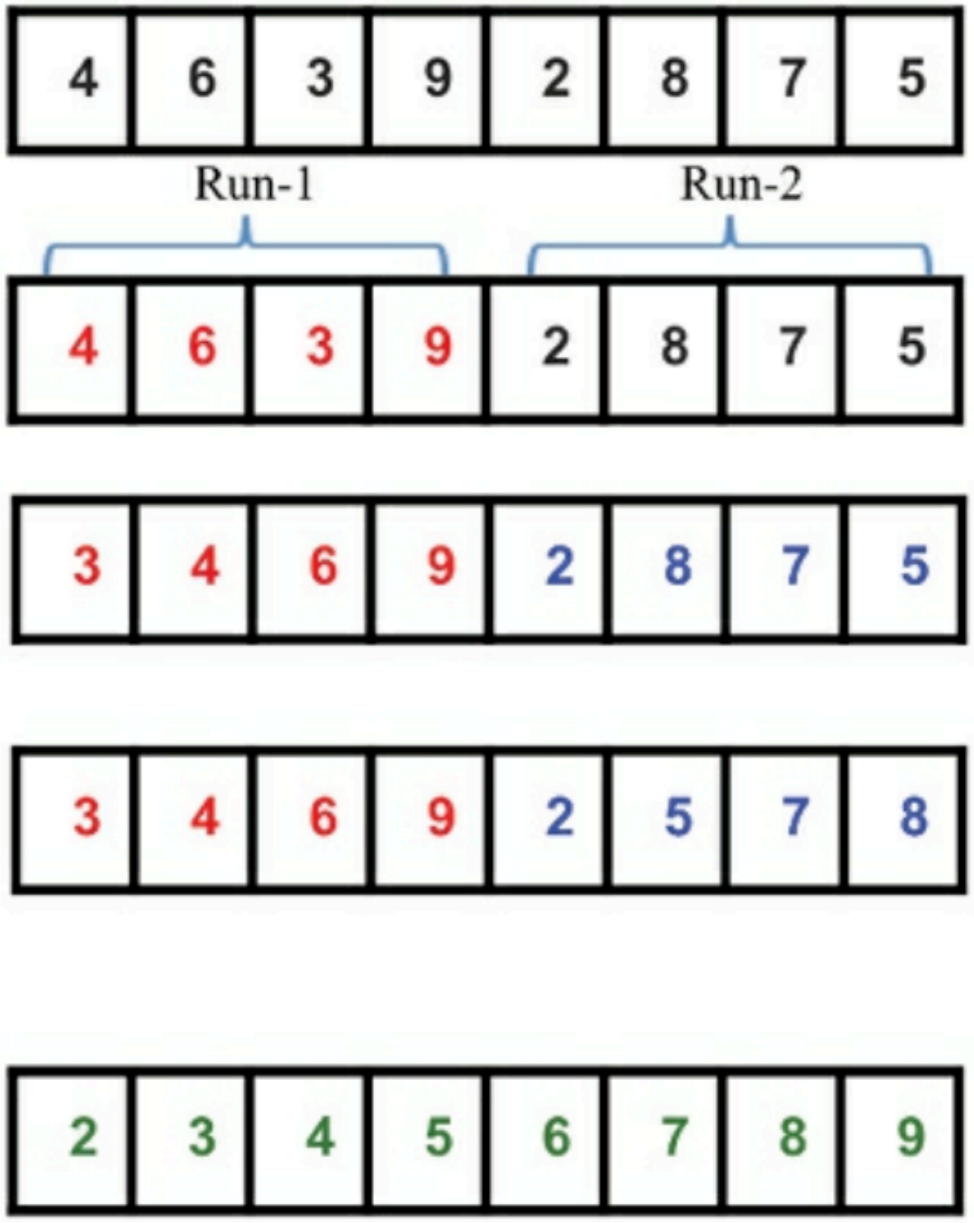
- Partition takes O($\boldsymbol{n}$) time
- Must repeat O(log $\boldsymbol{n}$) times
- Complexity O($\boldsymbol{n}$ log $\boldsymbol{n}$) for average case
- Worst case is O($\boldsymbol{n}^2$)
- Efficient for large lists

Timsort

- Default sorting algorithm for Python

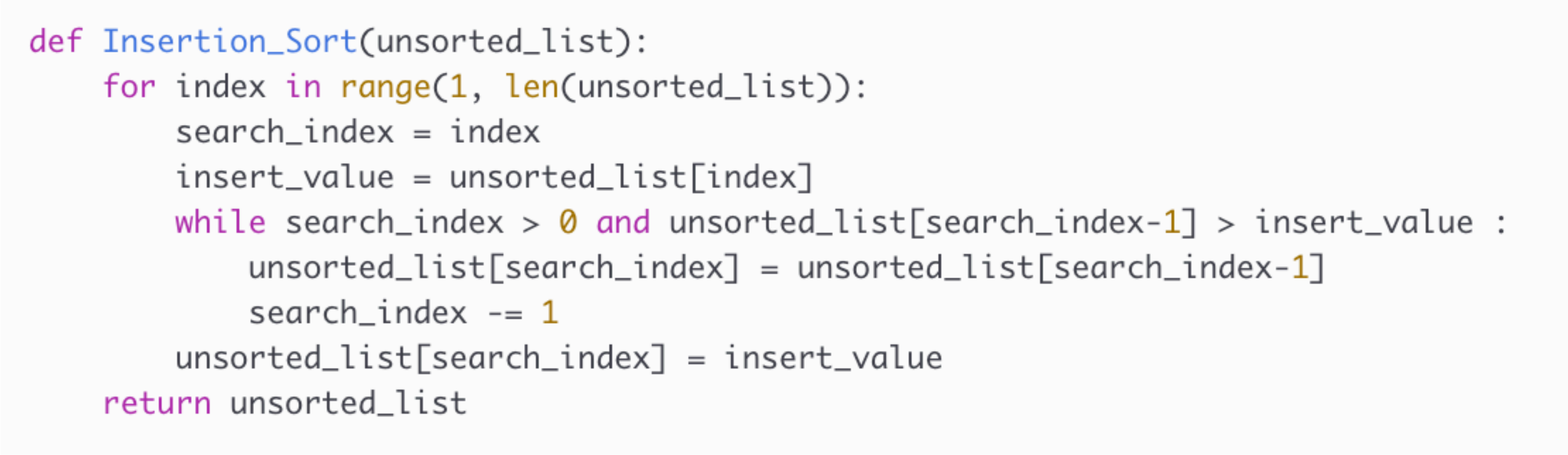
- Divide list into blocks (or ***runs***) of 32 or 64
- Use insertion sort on each block
- Merge blocks with merge sort

Timsort



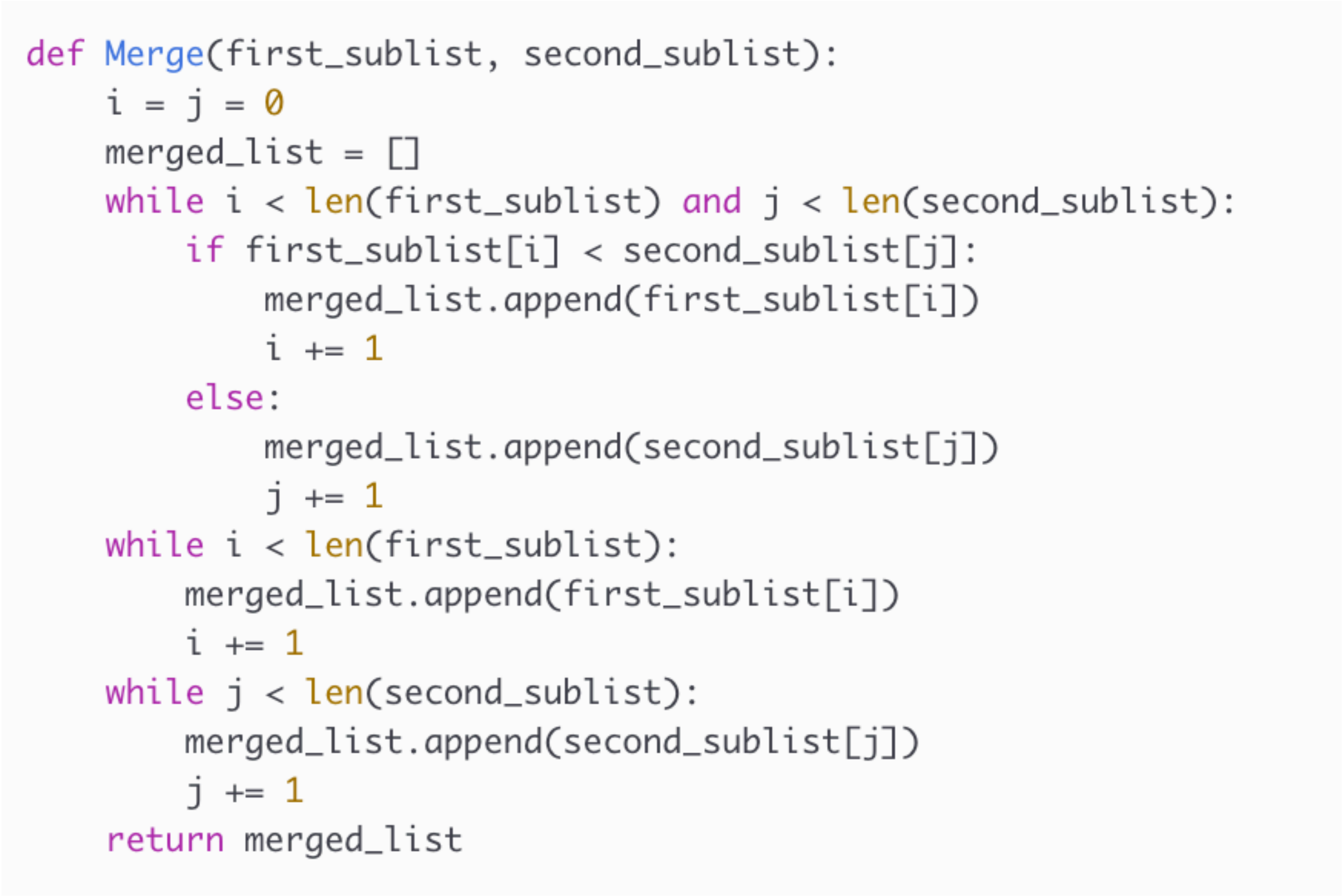
4	6	3	9	2	8	7	5	
4	6	3	9	2	8	7	5	We apply insertion sort to sort this run 1.
3	4	6	9	2	8	7	5	Run 1 is sorted, and we apply insertion sort on run 2.
3	4	6	9	2	5	7	8	Both run 1 and run 2 are sorted. We apply merge method to sort the complete list.
2	3	4	5	6	7	8	9	After merging run 1 and run 2 we get the sorted array.

Insertion sort



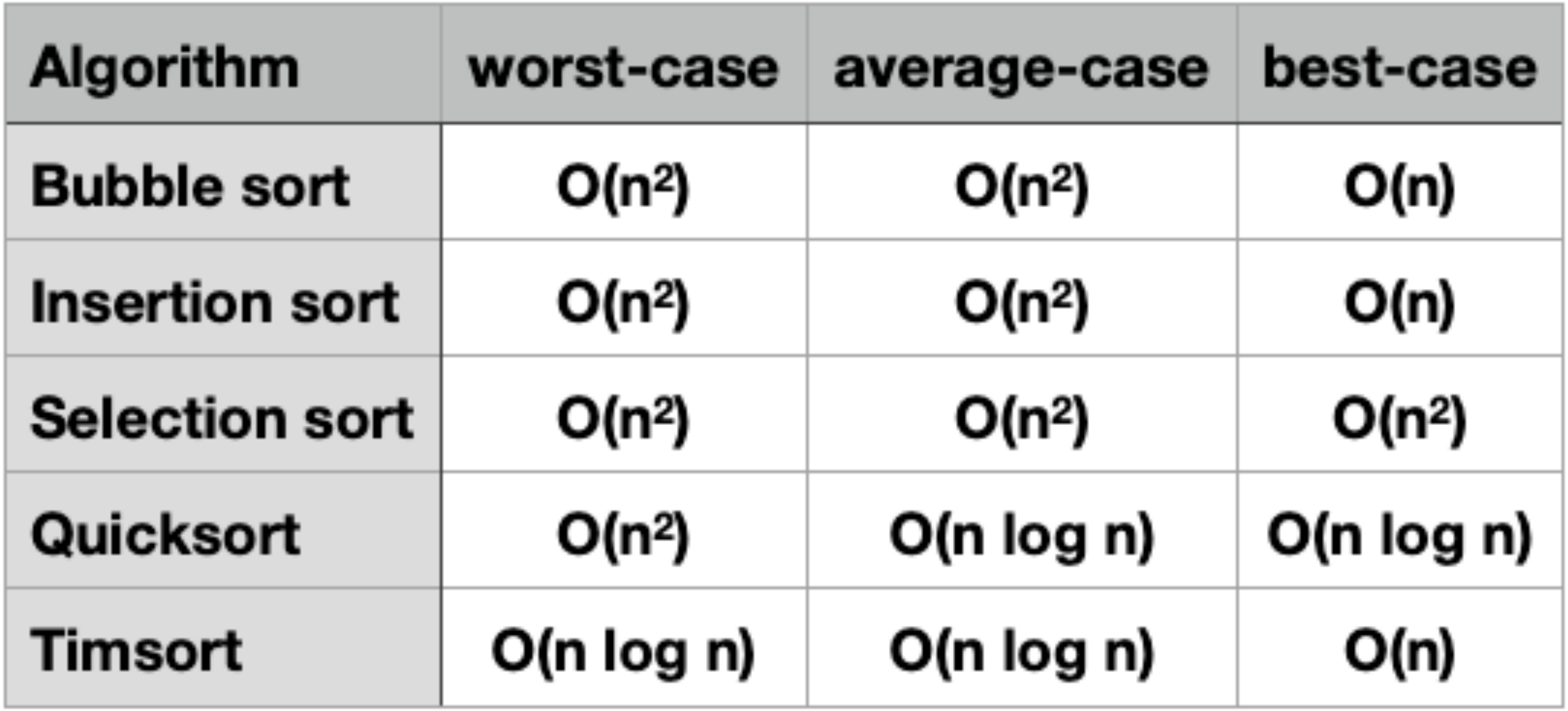
```python
def Insertion_Sort(unsorted_list):
    for index in range(1, len(unsorted_list)):
        search_index = index
        insert_value = unsorted_list[index]
        while search_index > 0 and unsorted_list[search_index-1] > insert_value :
            unsorted_list[search_index] = unsorted_list[search_index-1]
            search_index -= 1
        unsorted_list[search_index] = insert_value
    return unsorted_list
```

Merge sort



```
def Merge(first_sublist, second_sublist):
    i = j = 0
    merged_list = []
    while i < len(first_sublist) and j < len(second_sublist):
        if first_sublist[i] < second_sublist[j]:
            merged_list.append(first_sublist[i])
            i += 1
        else:
            merged_list.append(second_sublist[j])
            j += 1
    while i < len(first_sublist):
        merged_list.append(first_sublist[i])
        i += 1
    while j < len(second_sublist):
        merged_list.append(second_sublist[j])
        j += 1
    return merged_list
```

Comparing Sorting Algorithms



Algorithm	worst-case	average-case	best-case
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n)$
Insertion sort	$O(n^2)$	$O(n^2)$	$O(n)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quicksort	$O(n^2)$	$O(n \log n)$	$O(n \log n)$
Timsort	$O(n \log n)$	$O(n \log n)$	$O(n)$

Kahoot!

Ch 11