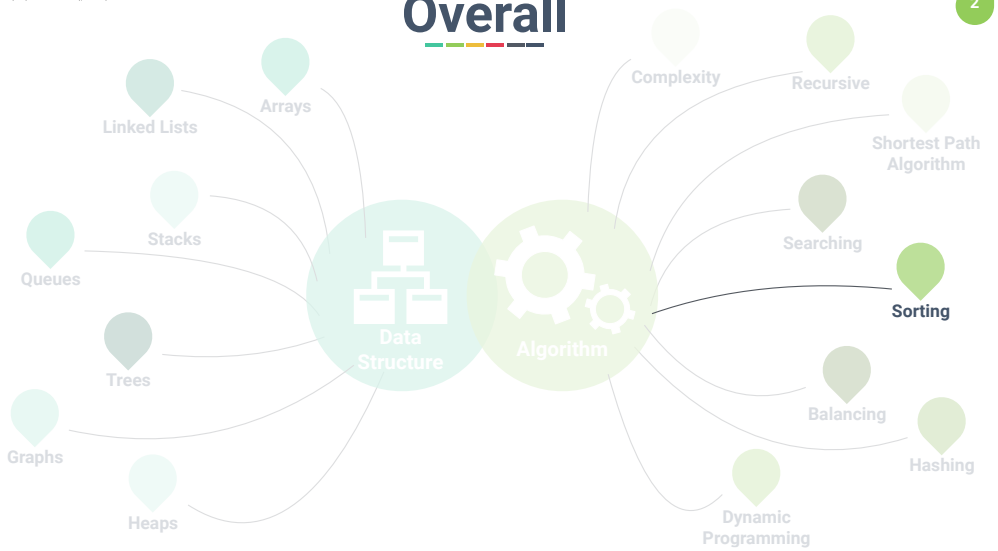
Chapter 10: Sorting

Dr. Sirasit Lochanachit

Overall

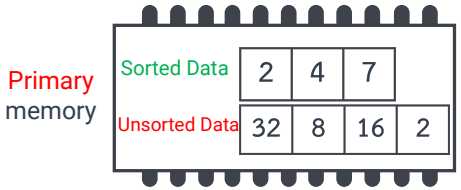


Outline

Sorting

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

Sorting

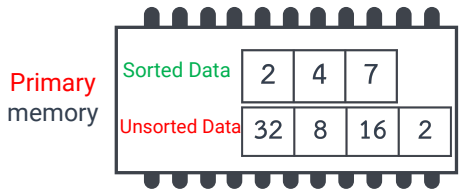


Sorting is the process of placing elements from a collection in some kind of order.

For instance, a list of words could be sorted by alphabet from a-z or z-a.

- Having a sorted data benefits many algorithms such as binary search.
- However, sorting a large number of items can take a huge amount of computing resources.

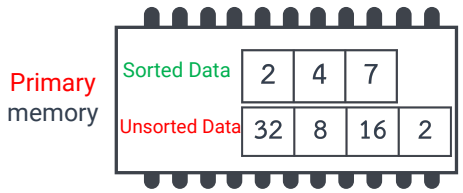
Sorting Operations



Generally, sorting has two operations:

1. **Compare** between two values to determine which is smaller (or larger).
   - The **total number of comparisons** is crucial to measure sorting efficiency.

Sorting Operations



Generally, sorting has two operations:

2. **Exchange** two values.
   - Exchange is a costly operation.
   - The **total number of exchanges** is also important for evaluating efficiency of the algorithm.

Typically, swapping two elements in a Python list requires a temporary storage location (an additional memory location).

```
temp = alist[i]
alist[i] = alist[j]
alist[j] = temp
```

Bubble Sort

Bubble sort is a sorting algorithm which makes multiple iterations through a given list of unsorted elements.

- It compares each pair of adjacent elements and exchanges those that are out of order.
- In any of the adjacent pairs, if the first element is greater/larger than the second element, then it swaps the elements and if not, then it moves on to the next pair of elements.
- Each round/iteration through the list places the smallest/largest value in its proper place.

Ref : https://www.faceprep.in/bubble-sort-in-c/

Bubble Sort

78	56	32	45	8	23	19

- If there are *n* items in the list, then there are *n* - 1 pairs that needs to be compared on the first round.
- At the start of the 2nd round, there are *n* - 1 items left to sort which means there will be *n* - 2 pairs.
- Therefore, the total number of rounds is *n* - 1.

https://youtu.be/lyZQPjUT5B4?t=54

Bubble Sort

Round 1

Right-to-left

n - 1 pairs

78	56	32	45	8	23	19	Exchange
78	56	32	45	8	19	23	No Exchange
78	56	32	45	8	19	23	Exchange
78	56	32	8	45	19	23	Exchange
78	56	8	32	45	19	23	Exchange
78	8	56	32	45	19	23	Exchange
8	78	56	32	45	19	23	

Bubble Sort

Round 2

Right-to-left

n - 2 pairs

8	78	56	32	45	19	23	No Exchange
8	78	56	32	45	19	23	Exchange
8	78	56	32	19	45	23	Exchange
8	78	56	19	32	45	23	Exchange
8	78	19	56	32	45	23	Exchange
8	19	78	56	32	45	23	

Bubble Sort

Round 3

Right-to-left

n - 3 pairs

8	19	78	56	32	45	23	Exchange
8	19	78	56	32	23	45	Exchange
8	19	78	56	23	32	45	Exchange
8	19	78	23	56	32	45	Exchange
8	19	23	78	56	32	45	

Bubble Sort

Round 4

Right-to-left

n - 4 pairs

8	19	23	78	56	32	45	No Exchange
8	19	23	78	56	32	45	Exchange
8	19	23	78	32	56	45	Exchange
8	19	23	32	78	56	45	

Bubble Sort

Round 5

Right-to-left

n - 5 pairs

8	19	23	32	78	56	45	Exchange
8	19	23	32	78	45	56	Exchange
8	19	23	32	45	78	56	

Bubble Sort

Round 6

Right-to-left

n - 6 pairs

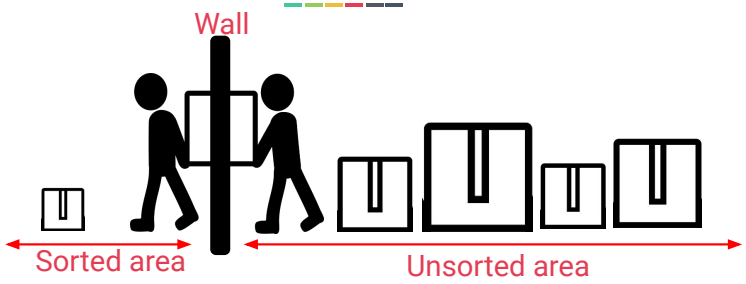
8	19	23	32	45	78	56	Exchange
8	19	23	32	45	56	78	

Bubble Sort

Round	Number of Comparison
1	*n* - 1
2	*n* - 2
3	*n* - 3
...	...
n - 1	1

- Regardless of how the items are arranged in the initial list, *n* −1 rounds will be made to sort a list of size *n*.
- A bubble sort is often considered the most inefficient sorting method since it must exchange items before the final location is known.
- These “wasted” exchange operations are very costly.

Selection Sort

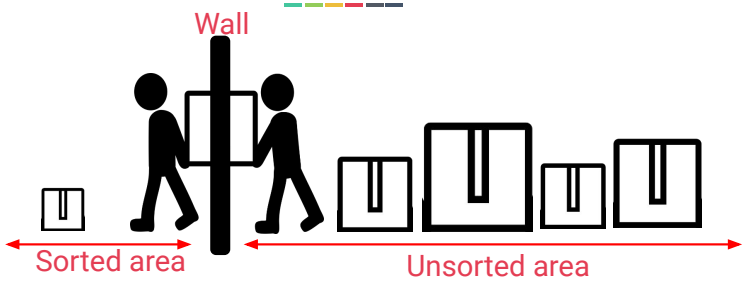


How to selection sort:

Given a list of unsorted data, it selects the smallest value and place it in a sorted list.

These steps are then repeated until all of the data are sorted.

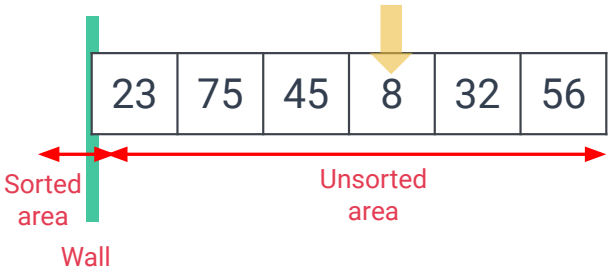
Selection Sort



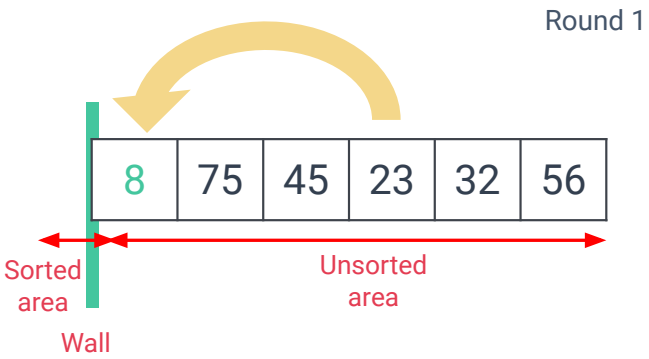
In other words, the list is divided into two sub-lists, sorted and unsorted, which are divided by an imaginary wall.

The smallest element from the unsorted sub-list are selected and exchange it with the element at the beginning of the unsorted data.

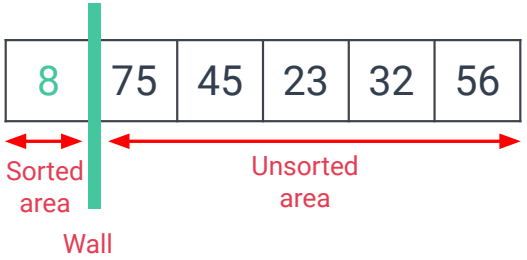
Selection Sort



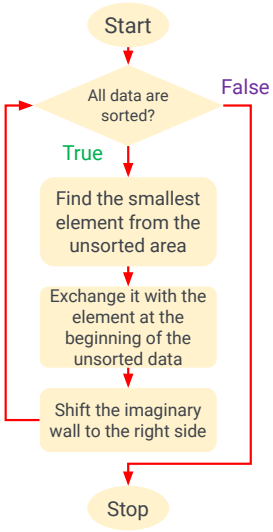
Selection Sort

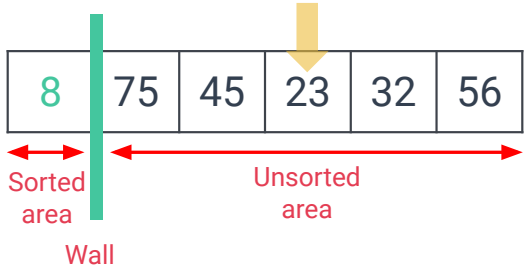


Selection Sort



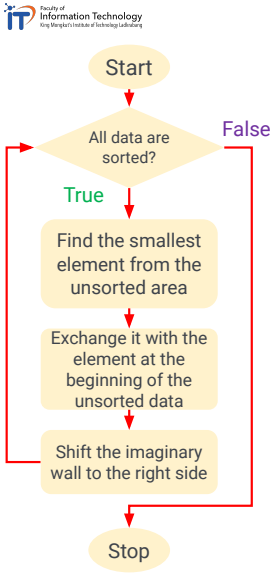
Selection Sort

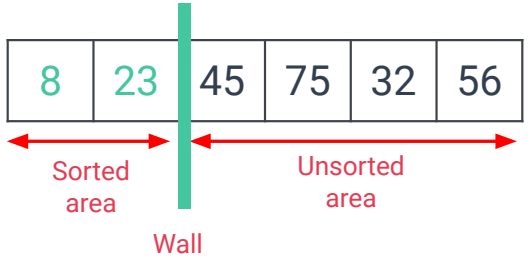




Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

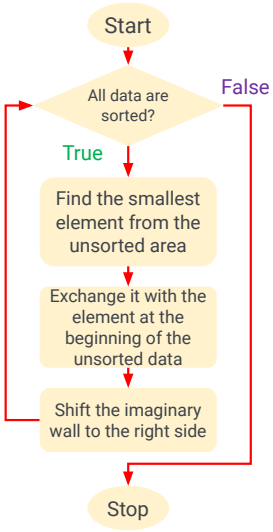
Selection Sort

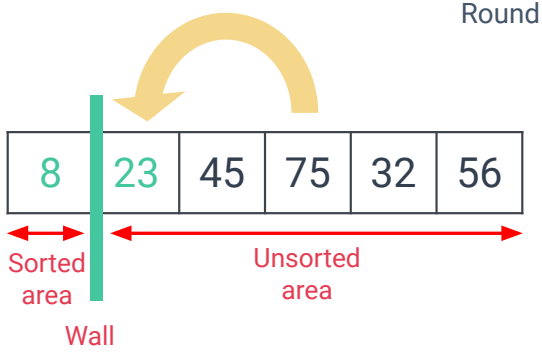




Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

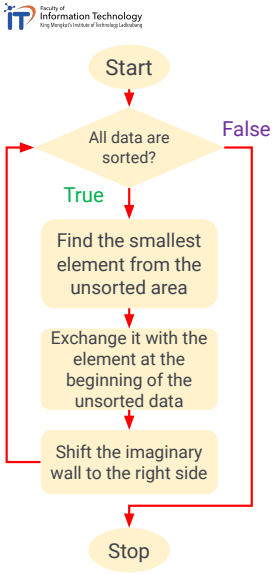
Selection Sort

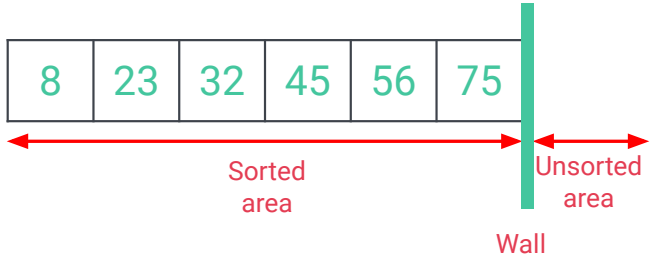




Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Selection Sort





Selection Sort

Round	Number of Comparison
1	*n* - 1
2	*n* - 2
3	*n* - 3
...	...
n - 1	1

- Similar to bubble sort, regardless of how the items are arranged in the initial list, *n* −1 rounds will be made to sort a list of size *n*.
- A selection sort makes the same number of comparisons as the bubble sort.
- However, selection sort reduces the number of exchanges, which makes it faster than bubble sort.

Insertion Sort

Similar to selection sort, a list is divided to two parts: sorted and unsorted.

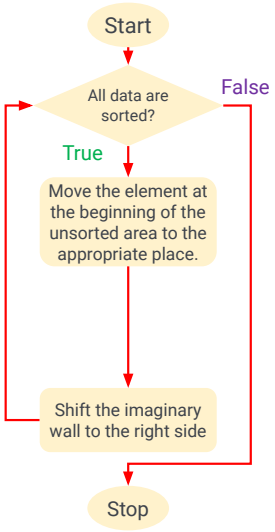
In each round, the first element of the unsorted sublist is transferred to the sorted sublist by inserting it at the appropriate place.

A real example is sorting cards by card players. As they pick up each card, they insert it into the proper sequence in their hand.

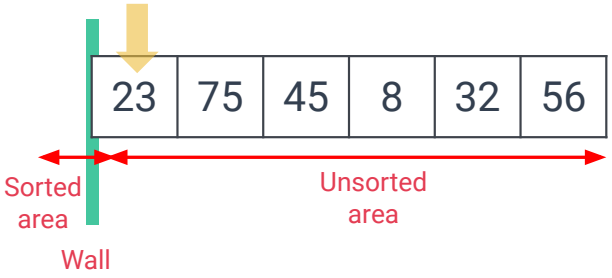
Insertion Sort

Where to insert?

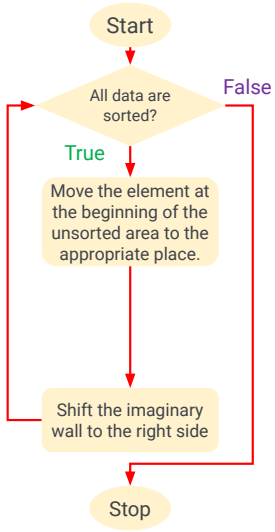
- The selected item is checked against items in the sorted sublist.
- The items in the sorted sublist that have greater value are shifted to the right.
- When reach a smaller item or the start of the sublist, the selected item can be inserted.



Insertion Sort

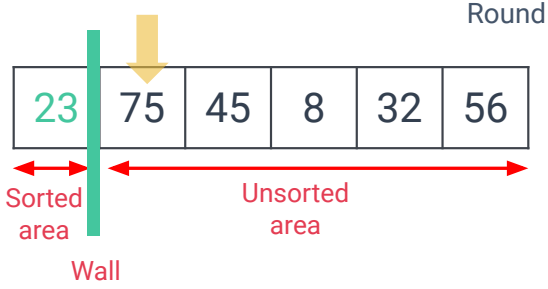


Insertion Sort

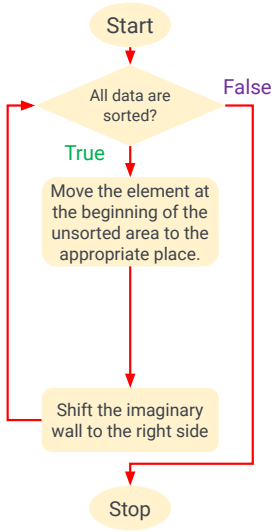


Inserted 23

Round 1

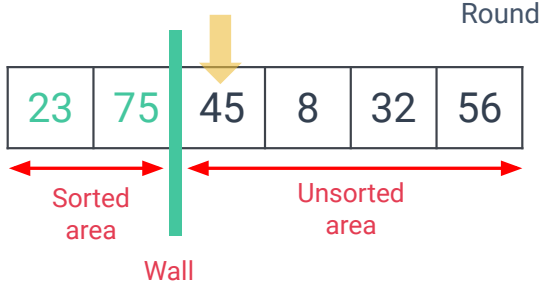


Insertion Sort

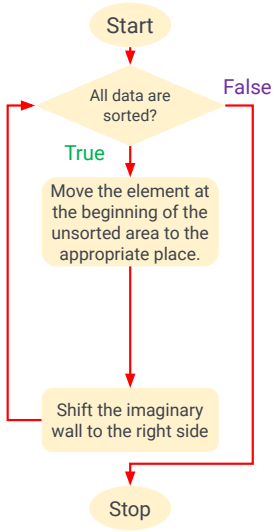


Inserted 75

Round 2

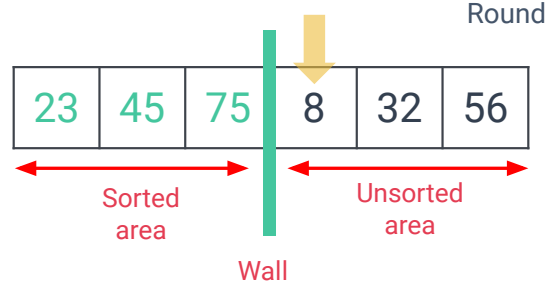


Insertion Sort

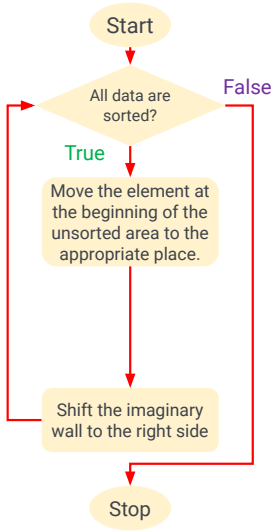


Inserted 45

Round 3

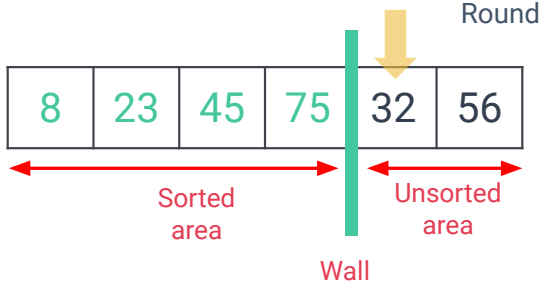


Insertion Sort

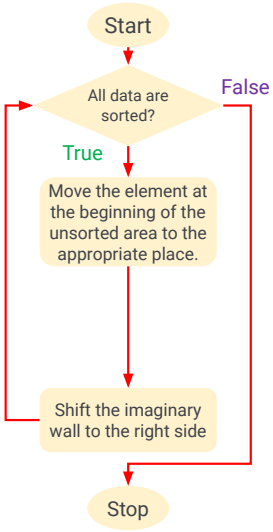


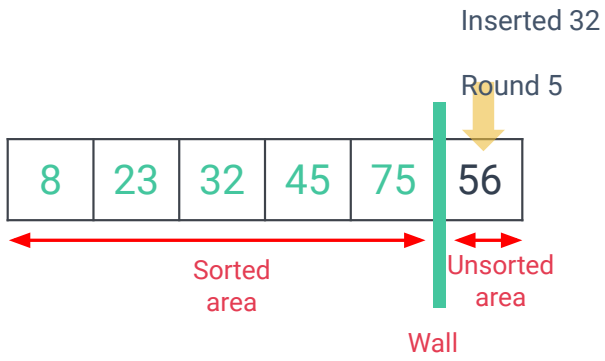
Inserted 8

Round 4

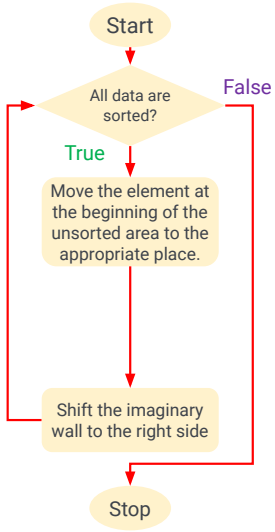


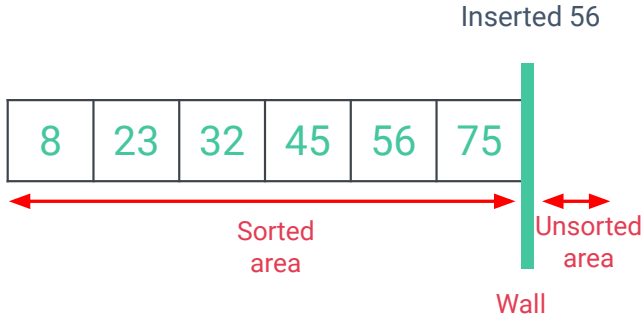
Insertion Sort





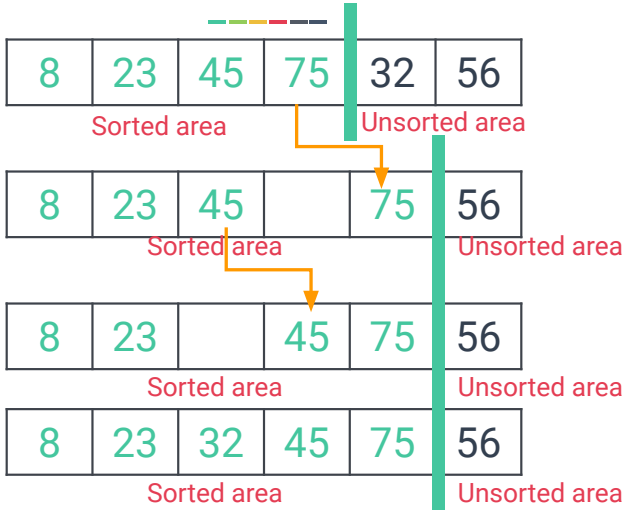
Insertion Sort





Insertion Sort: Behind the Scene

Need to insert 32



Insertion Sort

Round	Number of Comparison
1	1
2	2
3	3
...	...
n - 1	*n* - 1

- Similar to selection and bubble sort, regardless of how the items are arranged in the initial list, *n* −1 rounds will be made to sort a list of size *n*.
- The **maximum** number of comparisons for an insertion sort is the same number of comparisons as the selection sort and bubble sort.

Divide and Conquer (D&C)

A algorithm's strategy consisting of the following three steps:

- Divide: Divide the input data into two or more disjoint subsets.
- Recur: Recursively solve the subproblems associated with the subsets.
- Conquer: Take the solutions to the subproblems and "merge" them into a solution to the original problem.

Merge Sort

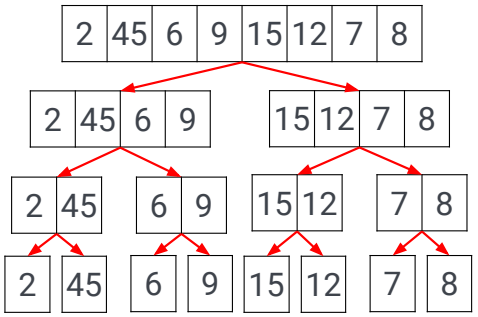
Merge sort is a recursive sorting technique based on divide and conquer technique by continually splits a list into equal halves.

Base case: If the list is empty or has only one item, it is sorted.

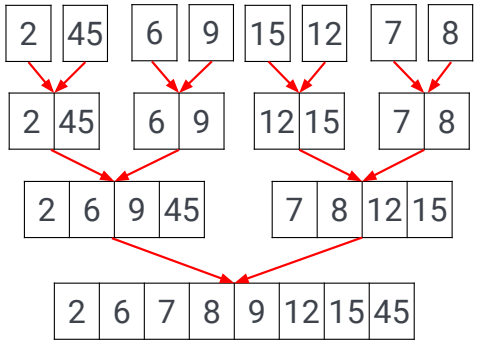
Recursive case: If the list has > 1 item, split the list and recursively merge sort on both halves.

After the two halves are sorted, then a merge is invoked, combining them together into a single sorted new list.

Merge Sort

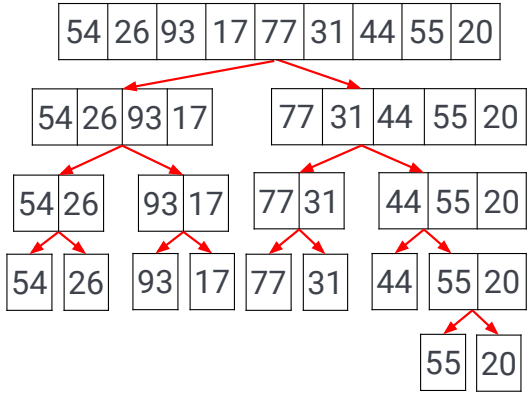


Merge Sort



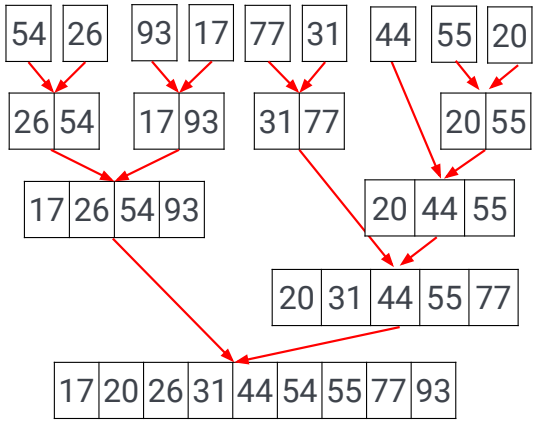
Merge Sort

Even elements



Merge Sort

Even elements

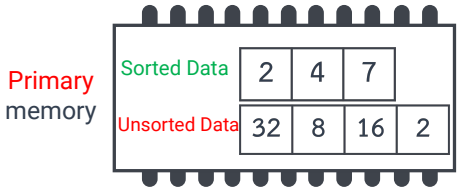


Merge Sort

Merge sort requires extra memory space to hold two halves as they are continually splitting.

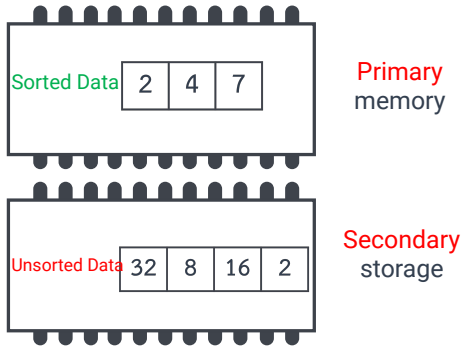
- This can be a critical factor if the list is large.

What are the internal and external sorts?



Internal Sort

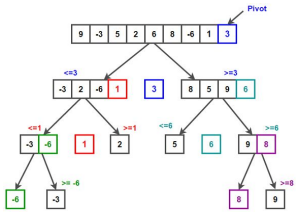
Sort the data while all the data are stored in the main memory in the computer.



External Sort

Some of the data might be stored in secondary memory.

Quick Sort



Quick Sort is one of the most efficient sorting algorithms and is based on the splitting of a list into smaller ones without using additional storage.

Like merge sort, the quick sort also falls into the category of divide and conquer approach.

However, it is possible that the list may not be divided in half.

Ref: https://www.interviewbit.com/tutorial/quicksort-algorithm/

Quick Sort

Example: Sort the papers containing the names of the students, by name from A-Z.

1. Select any splitting character such as 'L'.
   - The splitting value is also known as **pivot value**.
2. Divide the stack of papers into two. A-L and M-Z.
   - It is not necessary that the piles should be equal.
3. Repeat the above two steps with the A-L pile, splitting it into its significant two halves. And M-Z pile, split into its halves. The process is repeated until the piles are small enough to be sorted easily.
4. Ultimately, the smaller piles can be placed one on top of the other to produce a fully sorted and ordered set of papers.

Quick Sort

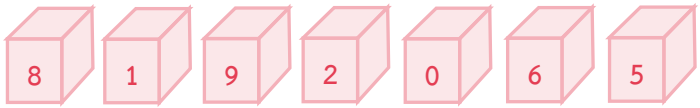
Technically, quick sort follows the below steps:

1. Make any element as **pivot**.
2. Partition the list on the basis of pivot.
3. Apply quick sort on left partition recursively.
4. Apply quick sort on right partition recursively.

Quick Sort

To determine the pivot element, there are 3 general approaches:

1) Choosing the first or last element of the unsorted list.
2) Random method.
3) Median-of-three method. It considers three elements including the first, last, and middle element in the list.



Quick Sort

Example: Sort this array/list with divide and conquer strategy without using extra place.

50	23	9	18	61	32

Step 1: Make any element as **pivot**.

- For convenience, the rightmost index is selected as pivot.
- 'Low' and 'High' pointers corresponds to the first index and last index respectively.
- In the example, the low is 0 and high is 5.
- The value at pivot is 32.

Quick Sort

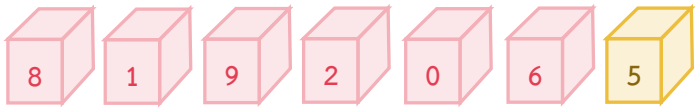
Example: Sort this array/list with divide and conquer strategy without using extra place.

50	23	9	18	61	32

Step 2: Partition the list on the basis of pivot.

- Rearranges the list in such a way that pivot(32) is at its proper position.
  - To the left of the pivot, the value of all elements is less than or equal to it.
  - To the right of the pivot, the value of all elements is higher than it.

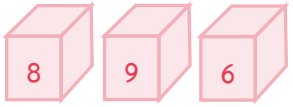
Quick Sort



- In the first sublist, the value of all elements is less than or equal to the pivot element.

Pivot

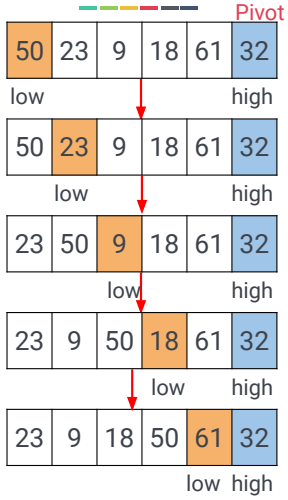
- In the second sublist, the value of all elements is higher than the pivot element.



- In the third sub-list, there is only the pivot element.

Quick Sort Example

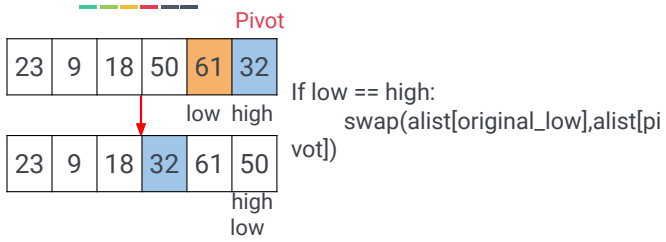
Example




```
If alist[low] < alist[pivot]:
    swap(alist[low], alist[low-1])
    low += 1
else:
    continue
```

Quick Sort Example (cont.)

Example



Quick Sort

Example: Sort this array/list with divide and conquer strategy without using extra place.

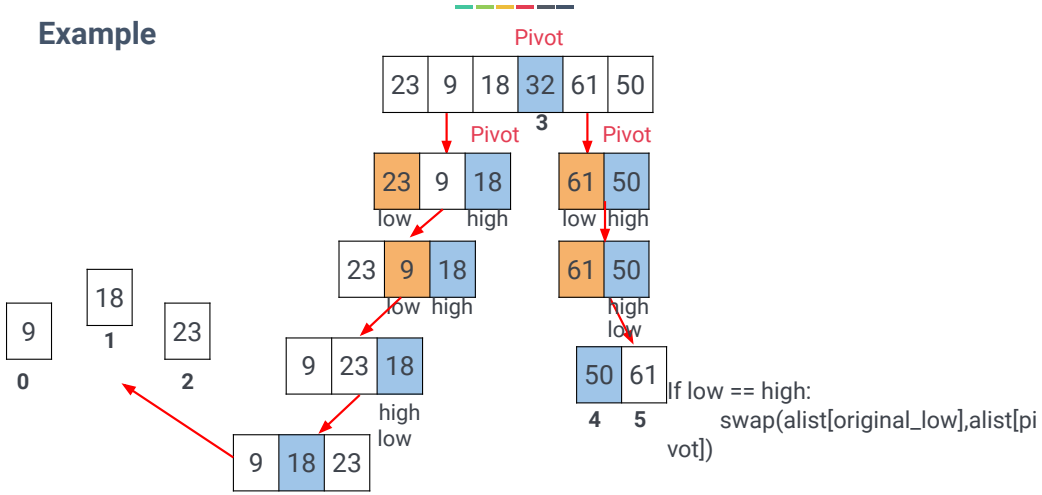
50	23	9	18	61	32

Step 3: Divide the list into two parts - left and right sublists.

Step 4: Repeat the steps for the left and right sublists recursively.

Quick Sort Example (cont.)

Example



Quick Sort

Merge sort requires extra memory space to hold two halves as they are continually splitting.

- This can be a critical factor if the list is large.

For quick sort, there is no need for additional memory as in the merge sort process.

However, the split points may not be in the middle and can be very skewed to the left or the right, leaving a very uneven division.

For example, sorting a list of *n* items divides into sorting a list of 0 items and a list of n−1 items.