data structures

Sorting Arrays

Sorting

- Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order.
- The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats.

sorting algorithms

Bubble Sort

Merge Sort

Insertion Sort

Shell Sort

Selection Sort

Sorting Applications

To prepare a list of student ID, names, and scores in a table (sorted by ID or name) for easy checking.

To prepare a list of scores before letter grade assignment.

To produce a list of horses after a race (sorted by the finishing times) for payoff calculation.

To prepare an originally unsorted array for ordered binary searching.

Selection Sort

- is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.

Selection Sort Algorithm

Step 1 – Set MIN to location 0

Step 2 – Search the minimum element in the list

Step 3 – Swap with value at location MIN

Step 4 – Increment MIN to point to next element

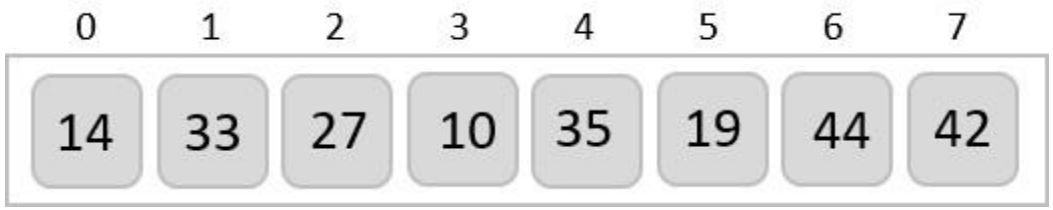
Step 5 – Repeat until list is sorted

Selection Sort Implementation

- The given program selects the minimum number of the array and swaps it with the element in the first index.
- The second minimum number is swapped with the element present in the second index.
- The process goes on until the end of the array is reached.

Selection Sort working

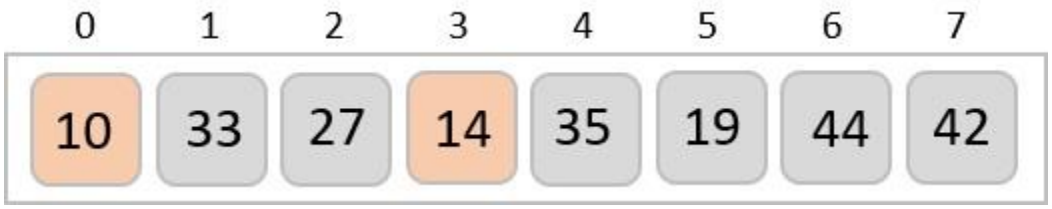
- Lets consider the following array



- First pass:
  - For the first position in the sorted list, the whole list is scanned sequentially.
  - The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

Selection Sort working

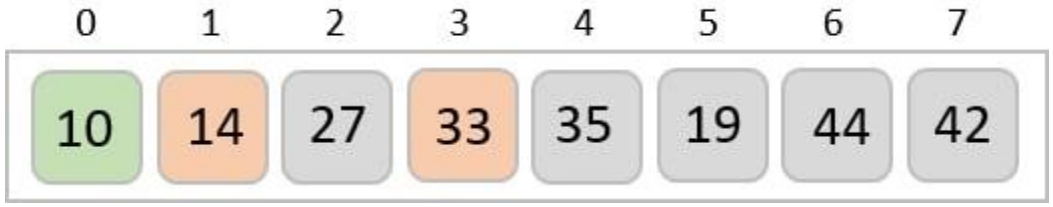
- So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.



- For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.
- We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

Selection Sort working

- After two iterations, two least values are positioned at the beginning in a sorted manner.

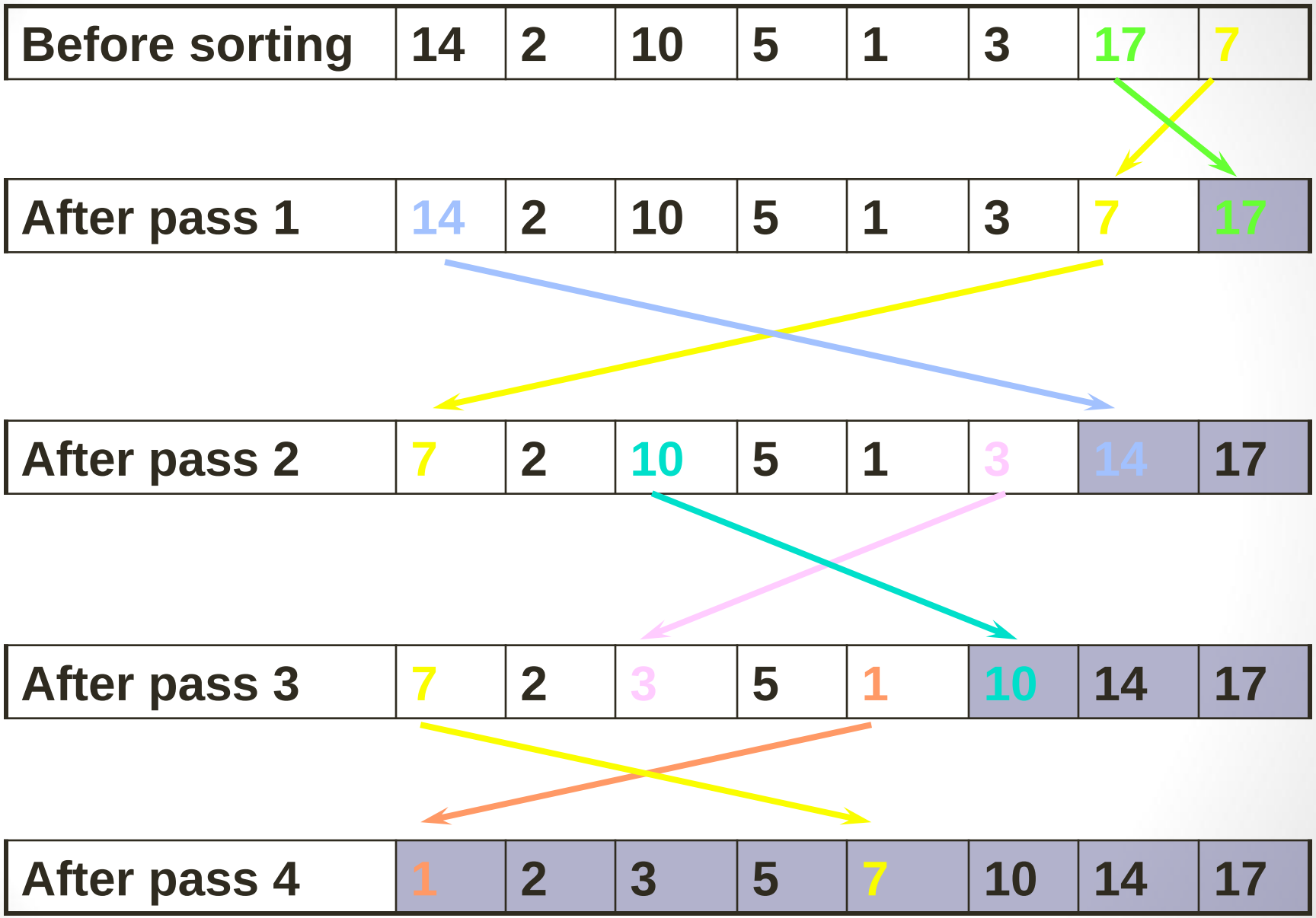


- The same process is applied to the rest of the items in the array

Selection Sort working

Selection Sort working

Selection Sort working


Before sorting 14 2 10 5 1 3 17 7
After pass 1 14 2 10 5 1 3 7 17
After pass 2 7 2 10 5 1 3 14 17
After pass 3 7 2 3 5 1 10 14 17
After pass 4 1 2 3 5 7 10 14 17

Selection Sort

```
def selection_sort(input_list):

    for idx in range(len(input_list)):

        min_idx = idx
        for j in range( idx +1, len(input_list)):
            if input_list[min_idx] > input_list[j]:
                min_idx = j
# Swap the minimum value with the compared value

        input_list[idx], input_list[min_idx] = input_list[min_idx],
input_list[idx]



l = [19,2,31,45,30,11,121,27]
selection_sort(l)
print(l)
```

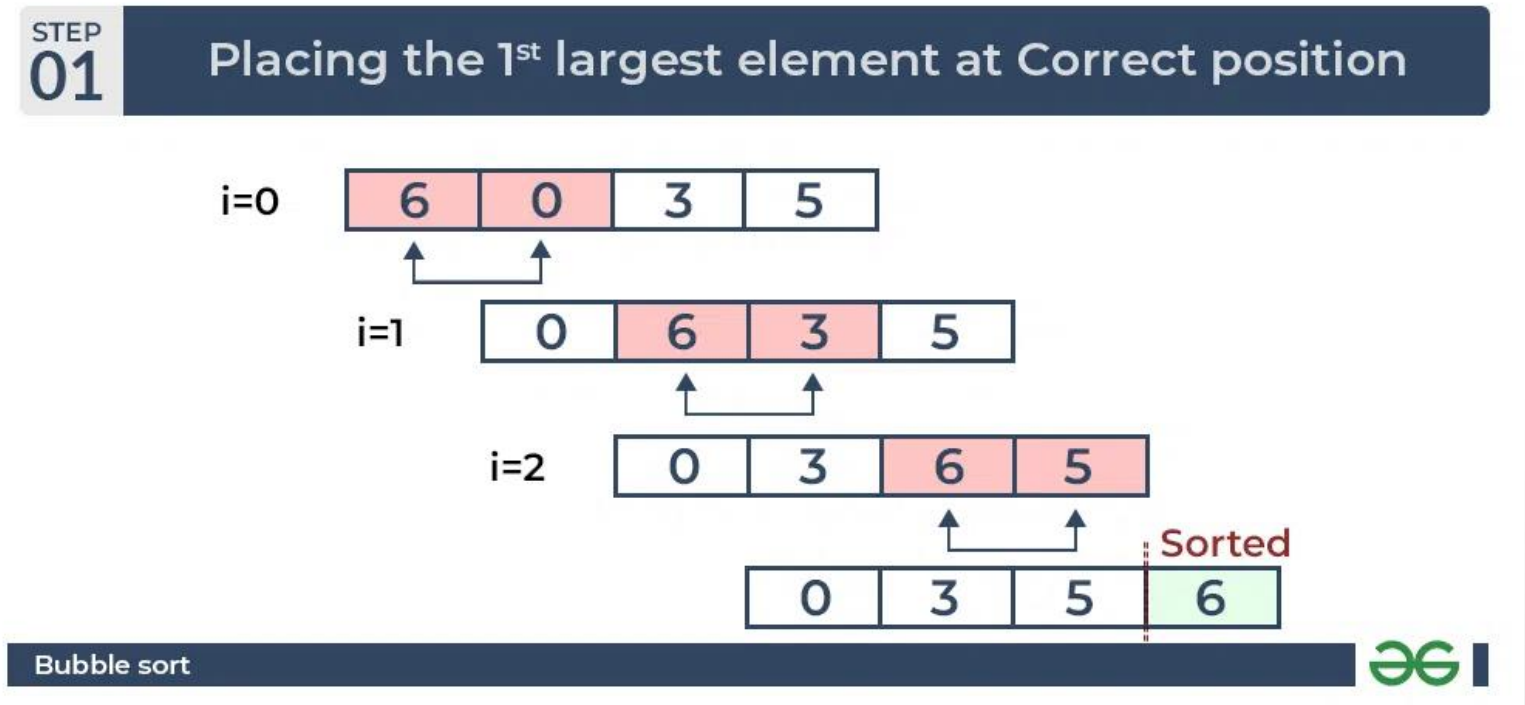
```
[2, 11, 19, 27, 30, 31, 45, 121]
```

Bubble Sort

- is the simplest sorting algorithm
- works by repeatedly swapping the adjacent elements if they are in the wrong order.
- This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high
- In **Bubble Sort algorithm**,
  - traverse from left and compare adjacent elements and the higher one is placed at right side.
  - In this way, the largest element is moved to the rightmost end at first.
  - This process is then continued to find the second largest and place it and so on until the data is sorted.

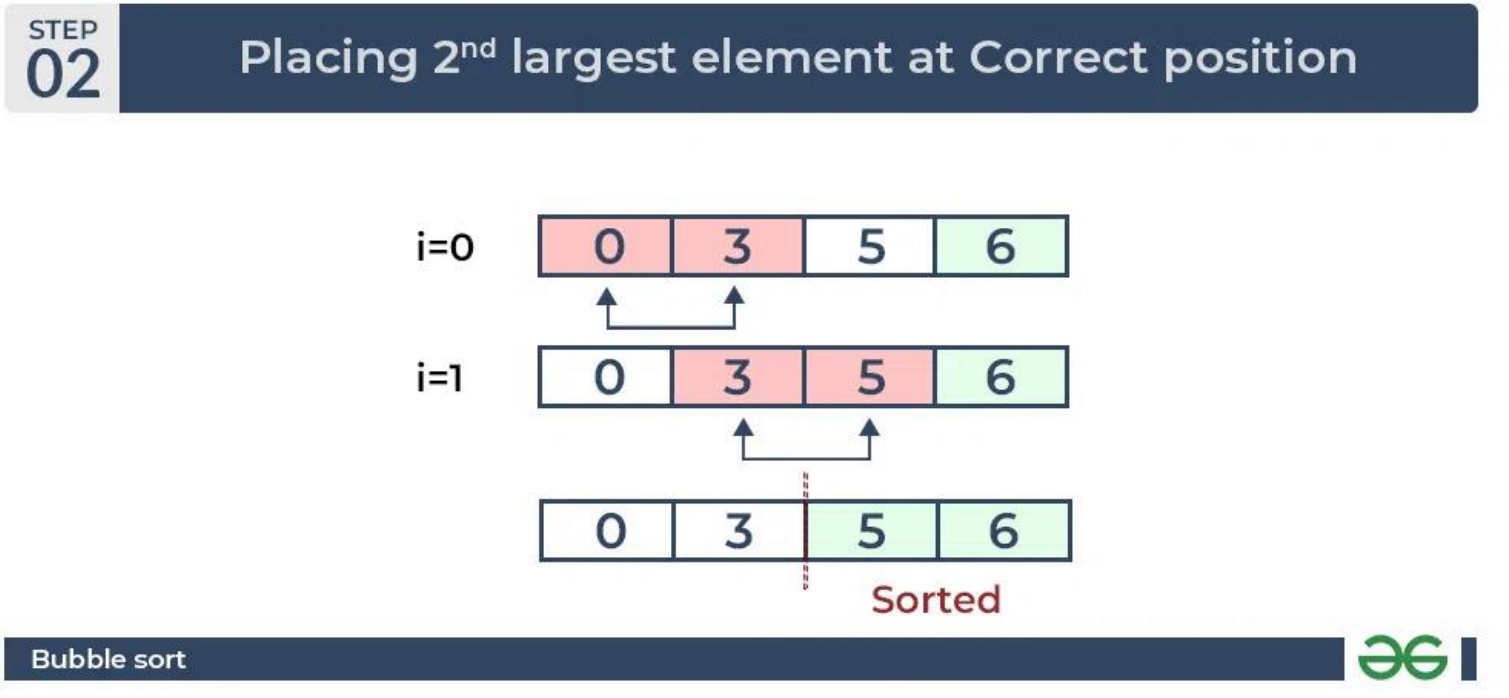
Bubble Sort Working

- First Pass:
  - The largest element is placed in its correct position, i.e., the end of the array.



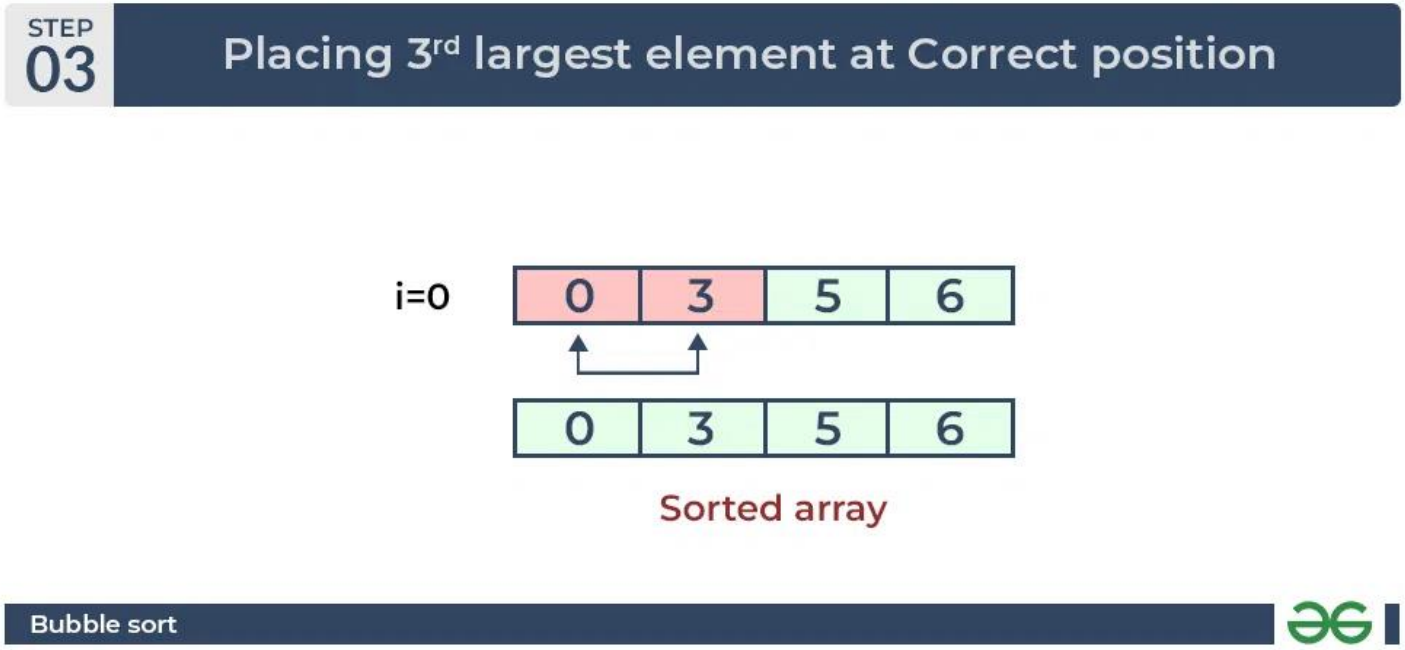
Bubble Sort Working

- Second Pass:
  - Place the second largest element at correct position



Bubble Sort Working

- Third Pass:
  - Place the remaining two elements at their correct positions.



Bubble Sort

```
def bubblesort(list):


# Swap the elements to arrange in order
    for iter_num in range(len(list)-1,0,-1):
        for idx in range(iter_num):
            if list[idx]>list[idx+1]:
                temp = list[idx]
                list[idx] = list[idx+1]
                list[idx+1] = temp



list = [19,2,31,45,6,11,121,27]
bubblesort(list)
print(list)
```

```
[2, 6, 11, 19, 27, 31, 45, 121]
```