COSC 2306
Data Programming

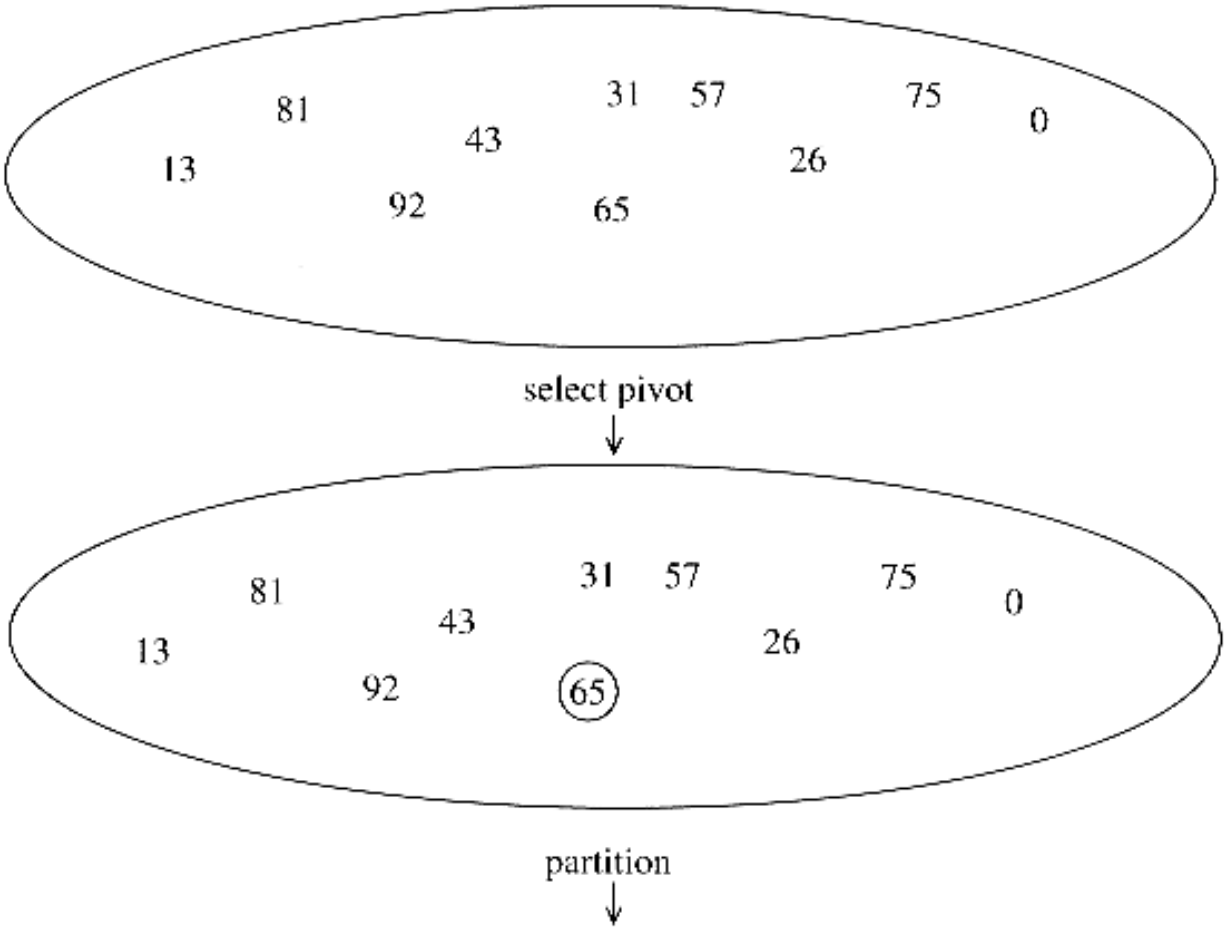
Sort

Quick Sort

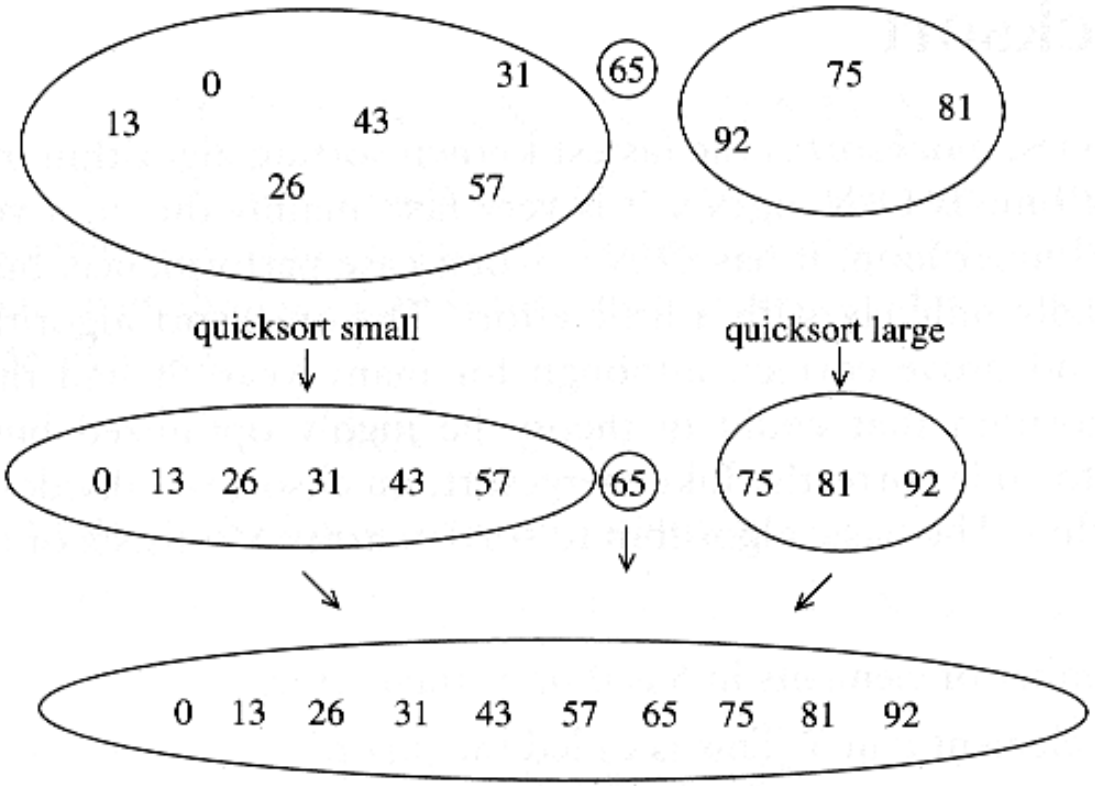
Quick Sort: Main Idea

1. A smarter divide without using extra space
2. Pick any element v in the array S (called the pivot)?
3. Partition the elements in S except v into two disjoint groups (or v is in its final position):
   1. $S_1 = \{x \in S - \{v\} \mid x \leq v\}$ #all elements in $S_1 \leq v$
   2. $S_2 = \{x \in S - \{v\} \mid x \geq v\}$ #all elements in $S_2 \geq v$
4. Return {QuickSort(S_1) + v + QuickSort(S_2)}
5. If the number of elements in S is 0 or 1, then return (base case)

Quick Sort: Example



Example of Quick Sort...

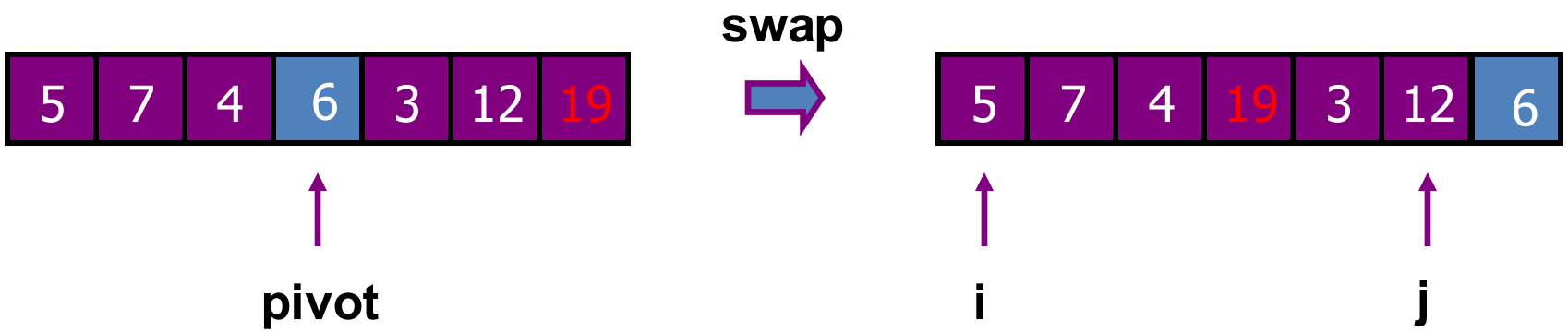


Issues to Consider

- How to pick the pivot?
  - Different methods (e.g., first, middle, last)
- How to partition?
  - Several methods exist
  - We introduce an easy and efficient partition strategy

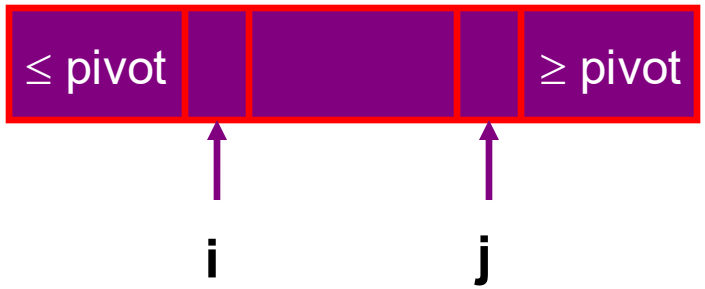
Partitioning Strategy

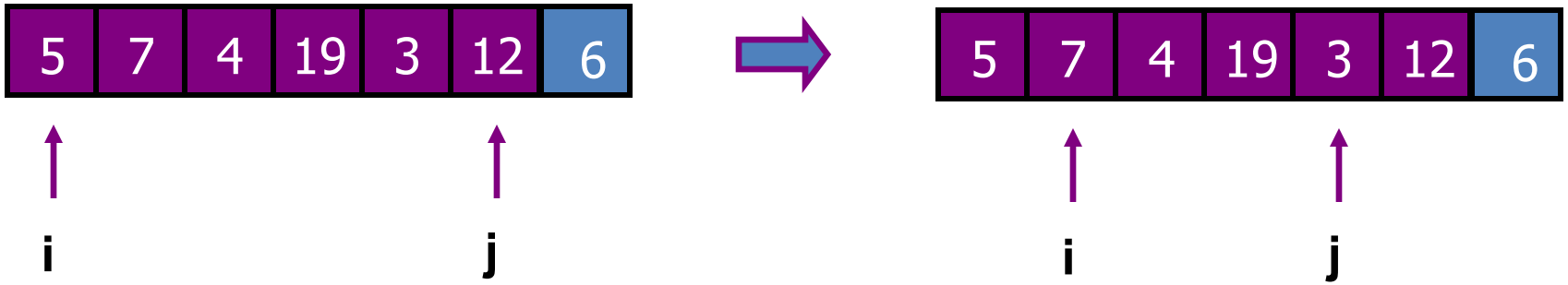
- Mid element as pivot = A[(left+right)/2].
- We want to partition array A[left .. right].
- First, get the pivot element out of the way by swapping it with the last element (swap pivot and A[right]).
- Let i start at the first element and j start at the next-to-last element (i = left, j = right – 1)



Partitioning Strategy

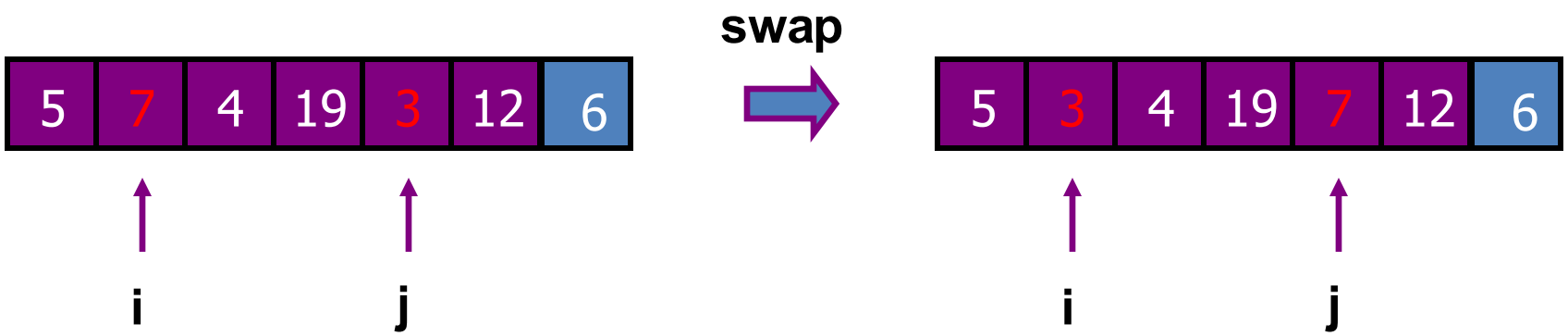
- Want to have
  - A[k] ≤ pivot, for k < i
  - A[k] ≥ pivot, for k > j
- When i < j
  - Move i right, skipping over elements smaller than the pivot
  - Move j left, skipping over elements greater than the pivot
  - When both i and j have stopped
    - A[i] ≥ pivot
    - A[j] ≤ pivot ⇒ A[i] and A[j] should now be swapped





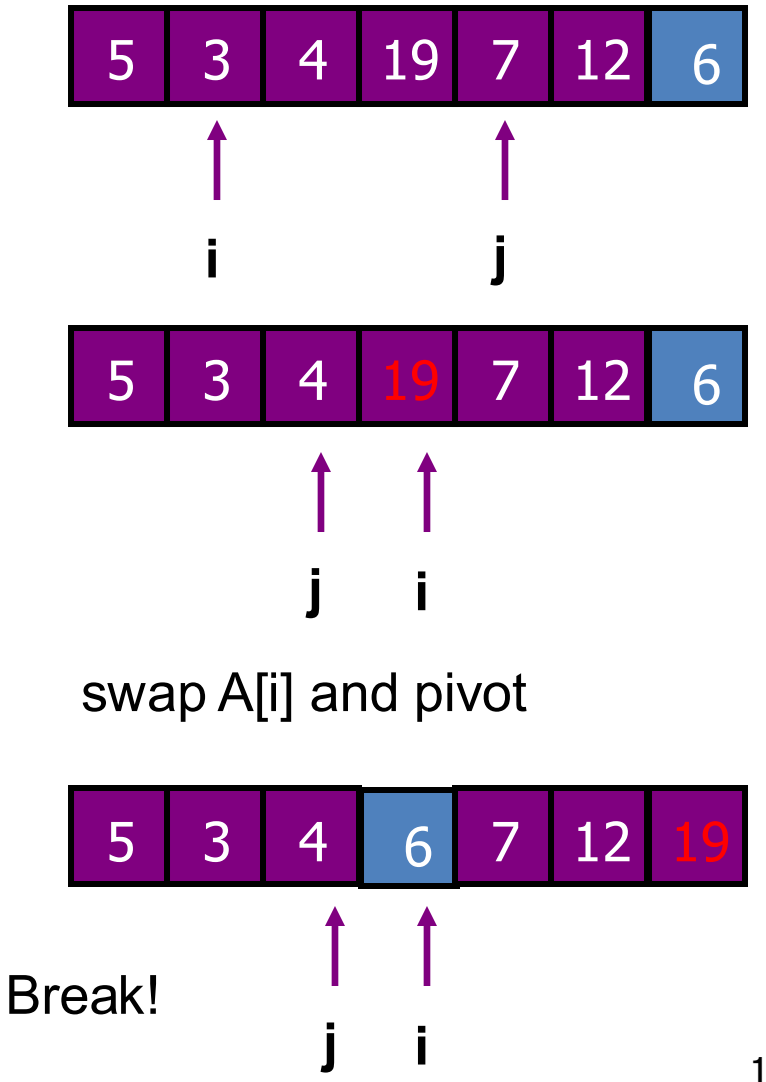
Partitioning Strategy

- When i and j have stopped and i is to the left of j (not crossed yet)
  - Swap A[i] and A[j]
    - The large element is pushed to the right and the small element is pushed to the left
  - After swapping
    - A[i] ≤ pivot
    - A[j] ≥ pivot
  - Repeat the process until i and j cross

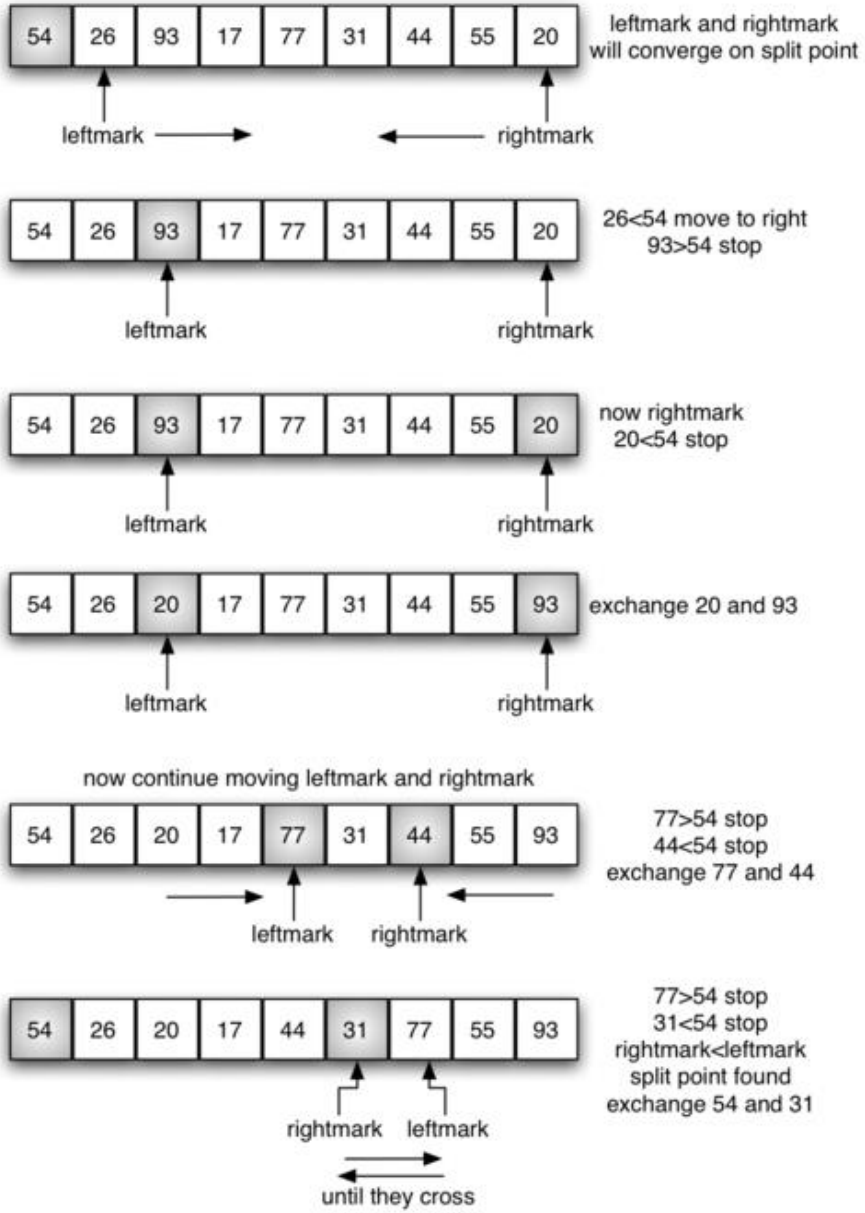


Partitioning Strategy

- When i and j have crossed
  - swap A[i] and pivot
- Result:
  - A[k] ≤ pivot, for k < i
  - A[k] ≥ pivot, for k > i



First element as the Pivot



Visualization

https://www.hackerearth.com/practice/algorithms/sorting/quick-sort/visualize/

Quick Sort Coding

- Write a recursive function quick_sort(a_list) to do quick sort.

Base case: if the sublist has 0 or 1 element, it is already sorted.

General case: length of array > 1;
- select pivot (assume first element as pivot),
- partition array into left and right,
- recursively do this for left and right subarray.

In each partition:
- move left/right marks towards each other until crossed,
- both marks stopped yet not crossed: found a pair to swap
- marks crossed: finished and swap pivot with right mark (pivot is in its final position).

Quick Sort Coding

```
def quick_sort(a_list):
    quick_sort_helper(a_list, 0, len(a_list) - 1)
def quick_sort_helper(a_list, first, last):
    if first < last:   #recursive base case
        split_point = partition(a_list, first, last) #partition
        quick_sort_helper(a_list, first, split_point - 1)  #recursive on left
        quick_sort_helper(a_list, split_point + 1, last) #recursive on right
def partition(a_list, first, last):
    pivot_value = a_list[first] #use the first element as pivot
    left_mark = first + 1
    right_mark = last
    done = False
    while not done:
        while left_mark <= right_mark and a_list[left_mark] <= pivot_value:
            left_mark = left_mark + 1 #advance left mark to right
        while a_list[right_mark] >= pivot_value and right_mark >= left_mark:
            right_mark = right_mark – 1 #advance right mark to left
        if right_mark < left_mark: #check if left_mark and right_mark crossed
            done = True
        else:  #swap left mark and right mark
            a_list[left_mark], a_list[right_mark] = a_list[right_mark], a_list[left_mark]
    a_list[first], a_list[right_mark] = a_list[right_mark], a_list[first] #swap pivot and right_mark
    return right_mark  #return position -- next step knows where to partition
```

```
a_list = [54, 26, 93, 17, 77, 31, 44, 55, 20]
quick_sort(a_list)
print(a_list)
```

Analysis of Quick Sort

- Best case: O(nlogn)
  - Partition always in the middle, total logn splits
  - To find the split point, each n items compare with pivot value, so overall nlogn
- Worst case: $O(n^2)$
  - Split point is highly skewed to the left/right
  - For extreme uneven division: a list of 0 items and n-1 items
  - Sorting a list of n-1, which splits again to 0 items and n-2 items
  - Basically no real split and thus a total of n splits
  - thus $O(n^2)$
  - The worst case is unlikely in practice