Sort and Selection

1. With Memory

-- merge sort strategy
-- range sort startegy

2. In-Place Sort and Selection Using Partition Algorithm

The partition algorithm is fundamental for in-place sorting and selection problems. It divides the array into sections relative to a chosen pivot element. Below are the detailed steps and explanations of various partitioning strategies:

General Steps for Partitioning:

Step 1: Pick a Pivot
1. Select a pivot element from the array.
   - Common choices include the first element, the last element, a random element, or the median of the array.
2. The pivot is used to compare and partition the array into sections.

Step 2: Determine Pivot Partition Boundaries
1. Establish boundaries that differentiate elements smaller than, equal to, and greater than the pivot.
2. These boundaries guide the traversal and swapping during partitioning.

Step 3: Traverse and Adjust Boundaries Using Swaps
1. Traverse the array elements using a pointer or index.
2. Compare each element against the pivot and adjust the partition boundaries by swapping elements into their correct sections.

Partitioning Algorithms Based on Pivot Partition Boundaries

A. Lomuto Partition Scheme:
- Uses a single partition boundary to segregate elements smaller than the pivot.
- All elements less than the pivot are moved to its left, while others remain on its right.

1. Steps
- Initialize a pointer (`i`) to track the boundary of elements smaller than the pivot.
- Traverse the array with a pointer (`j`).
- If the current element at `j` is smaller than the pivot, increment `i` and swap the element at `j` with the element at `i`.
- Finally, place the pivot in its correct position by swapping it with the element at `i + 1`.

2. Boundary Condition: Elements less than the pivot are placed to its left.

B. Hoare Partition Scheme:
- Uses two partition boundaries: one at the low end and another at the high end.
- Scans from both ends of the array to adjust boundaries.

1. Steps:
- Initialize two pointers: low (start of the array) and high (end of the array).
- Move the low pointer rightward until an element greater than or equal to the pivot is found.
- Move the high pointer leftward until an element smaller than or equal to the pivot is found.
- If low < high, swap the elements at the low and high pointers and continue.
- Stop when the pointers cross each other.

2. Boundary Conditions:
- Low-end boundary: Elements less than or equal to the pivot.
- High-end boundary: Elements greater than or equal to the pivot.

3. Key Characteristic:
- Pivot is not necessarily placed in its final sorted position.

C. Three-Way Partitioning Scheme
- Divides the array into three sections:
  1. Elements less than the pivot.
  2. Elements equal to the pivot.
  3. Elements greater than the pivot.

1. Steps:
- Use three pointers: low, mid, and high.
- Initialize:
  - low at the start of the array,
  - mid at the start of the array,
  - high at the end of the array.
- Traverse the array using mid:
  - If element at mid < pivot: Swap with element at low, increment both low and mid.
  - If element at mid == pivot: Increment mid.
  - If element at mid > pivot: Swap with element at high, decrement high.

2. Boundary Conditions
- low: Elements less than the pivot.
- mid: Elements equal to the pivot.
- high: Elements greater than the pivot.

3. Key Characteristic:
- Ideal for arrays with many duplicate values.

Relationship Between Algorithms

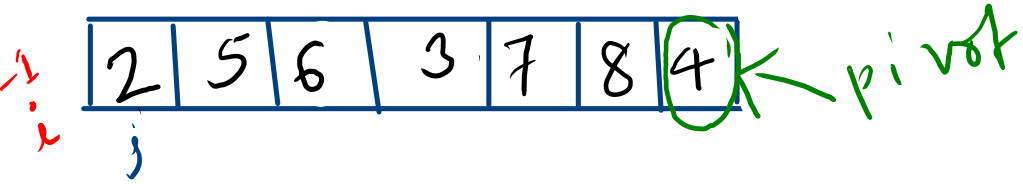
- **Hoare and Three-Way Partitioning** can be thought of as running two simultaneous Lomuto partitioning algorithms from opposite ends of the array.
- **Three-Way Partitioning** is particularly effective for scenarios where duplicate pivot values are frequent.

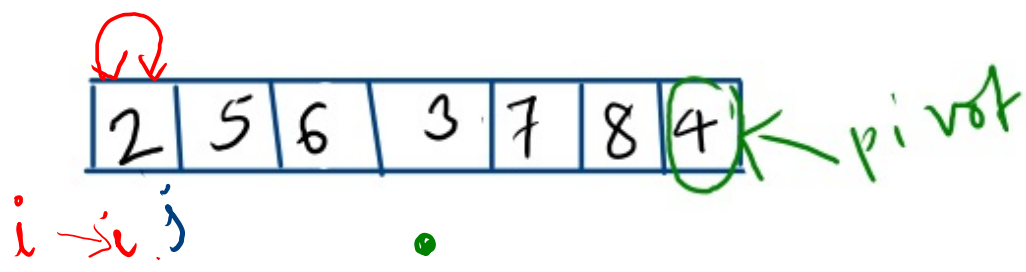
General Steps for Partitioning:

Step 1: Pick a Pivot
1. Select a pivot element from the array.
   - Common choices include the first element, the last element, a random element, or the median of the array.
2. The pivot is used to compare and partition the array into sections.

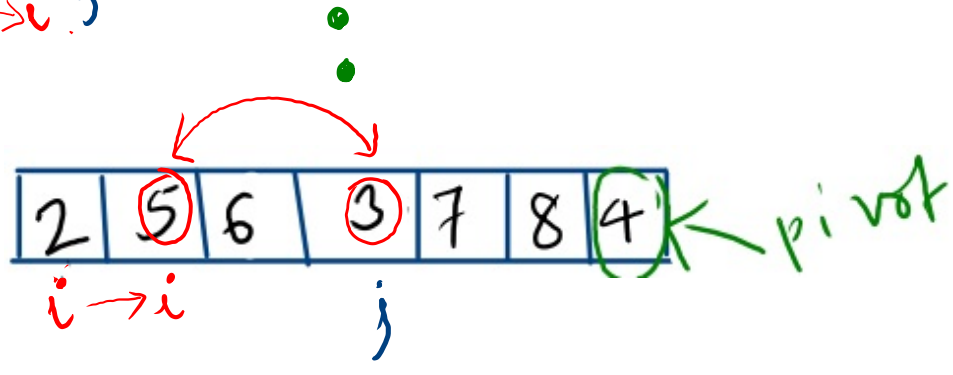
Step 2: Determine Pivot Partition Boundaries
1. Establish boundaries that differentiate elements smaller than, equal to, and greater than the pivot.
2. These boundaries guide the traversal and swapping during partitioning.

Step 3: Traverse and Adjust Boundaries Using Swaps
1. Traverse the array elements using a pointer or index.
2. Compare each element against the pivot and adjust the partition boundaries by swapping elements into their correct sections.





scan pointer 'j' encountered 2 which is less than pivot(4), so need to create space for 2. Hence, increment i to create the space for 2. and self-swap.



scan pointer 'j' encountered 3 which is less than pivot(4), so need to create space for 3. Hence, increment i to create the space for 3 and swap.