Insertion Sort

The algorithm iterates through each element, inserting it into the correct position in the sorted part of the array.

Time Complexity:

Best Case: O(n) — when the array is already sorted.
Average Case: O(n^2)
Worst Case: O(n^2) — when the array is sorted in reverse.

Space Complexity: O(1) (in-place sorting).

Implementation:

```
public static void sort(int[] arr) {
   if (arr == null) return;
   for (int i = 1; i < arr.length; i++) {
      int key = arr[i];
      int j = i - 1;
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }
}
```

Problems:

Not suitable for large arrays (will be slow for 10,000+ elements).

No check for almost sorted data.

Optimizations:

Add a check: if elements are already in place, skip the iteration.

Use binary search to find the insertion position.

Shell Sort

It starts with a large gap and reduces it until the array is sorted. The algorithm becomes Insertion Sort when the gap equals 1.

Time Complexity:

• Best Case: O(n log n)
• Average Case: O(n^(3/2)) depending on gap sequence
• Worst Case: O(n^2)

Space Complexity: O(1)

Implementation:

```
public static void sort(int[] arr) {
   if (arr == null) return;
   int n = arr.length;
   for (int gap = n/2; gap > 0; gap /= 2) {
      for (int i = gap; i < n; i++) {
         int temp = arr[i];
         int j = i;
         while (j >= gap && arr[j-gap] > temp) {
            arr[j] = arr[j-gap];
            j -= gap;
         }
         arr[j] = temp;
      }
   }
}
```

Min Heap

A Min Heap is a binary tree where each parent node is smaller than its children. The smallest element is always at the beginning.

Operations include:

insert(val): adds a new element while maintaining heap order.
extractMin(): removes and returns the smallest element.
heapify(i): ensures the heap property starting from index i.

Time Complexity:

Insert: O(log n)
Extract Min: O(log n)
Build Heap: O(n)

Space Complexity: O(n)

Java Implementation:

```
public void insert(int val) {
   if (size == heap.length) return;
   heap[size] = val;
   int i = size;
   size++;
   while (i > 0 && heap[parent(i)] > heap[i]) {
      int tmp = heap[i];
      heap[i] = heap[parent(i)];
      heap[parent(i)] = tmp;
      i = parent(i);
   }
}
```

Boyer-Moore Majority Vote

This algorithm finds the majority element in an array — the element that appears more than n/2 times.

Steps:

1. Select a candidate element and maintain a counter.
2. Increase the counter if the current element matches; otherwise, decrease it.
3. If the counter becomes zero, pick a new candidate.
4. Finally, verify if the candidate truly appears more than n/2 times.

Time Complexity: O(n)
Space Complexity: O(1)

Implementation:

```
public static int majorityElement(int[] arr) {
   int candidate = arr[0], count = 1;
   for (int i = 1; i < arr.length; i++) {
      if (count == 0) {
         candidate = arr[i];
         count = 1;
      } else if (arr[i] == candidate) {
         count++;
      } else {
         count--;
      }
   }
   int occur = 0;
```

```
    for (int x : arr) if (x == candidate) occur++;
    return (occur > arr.length / 2) ? candidate : Integer.MIN_VALUE;
}
```

Defect: Does not work if there is no element that occurs more than half the time

Benchmark Runner

It calls sorting algorithms, runs them on arrays, and prints the results to verify correctness and performance.

Implementation:

```
int[] arr = {5,2,9,1,5,6};
InsertionSort.sort(arr);
System.out.println("Insertion sorted:");
for (int x: arr) System.out.print(x + " ");
```

conclusion:
Insertion Sort is simple and efficient for small or nearly sorted datasets, but its quadratic complexity makes it impractical for large arrays.

Shell Sort demonstrates improved performance by reducing the number of comparisons and moves, approaching the efficiency of O(n log n) algorithms.

Min-Heap provides a foundation for priority queues and Heap Sort, offering logarithmic time for inserting and extracting elements.

Boyer-Moore Majority Vote is an example of an elegant and efficient linear algorithm that solves the problem in a single pass with constant memory usage.