Estd. : 1999

TIT TECHNOCRATS

NBA ACCREDITED CSE (AIML, AIDS, AI, DS, CY) CSE, EX, EC, CE, ME, IT

nirf 2021-22 Ranking

Celebrating 26 YEARS of Excellence

Only Group of MP & CG with NBA in 3 Engg. Colleges

UNBEATABLE PLACEMENTS

Technocrats Group Campus, BHEL, Bhopal - 462021 MP, India
Phone : +91-755-2751679 | Mobile : +91-9826374295, +91-9893141968
e-mail : placements@titbhopal.net | website : http://www.technocratsgroup.edu.in

Campus Specific Training

By: Mr. Gautam Singh

Technocrats Group of Institutions
Technocrats Group Campus, BHEL, Bhopal - 462021 MP, India
Phone : +91-755-2751679
e-mail: info@titbhopal.net | website: www.technocratsgroup.edu.in

1. Bubble Sort - Step-by-Step Diagram

- Bubble Sort repeatedly swaps adjacent elements if they are in the wrong order.

Example: Sorting [5, 3, 8, 4, 2]

Pass 1: [5, 3, 8, 4, 2] → [3, 5, 8, 4, 2] → [3, 5, 8, 4, 2] → [3, 5, 4, 8, 2] → [3, 5, 4, 2, 8]

Pass 2: [3, 5, 4, 2, 8] → [3, 4, 5, 2, 8] → [3, 4, 2, 5, 8] → [3, 4, 2, 5, 8]

Pass 3: [3, 4, 2, 5, 8] → [3, 2, 4, 5, 8] → [3, 2, 4, 5, 8]

Pass 4: [2, 3, 4, 5, 8] (Sorted)

```java
import java.util.Arrays;

public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            boolean swapped = false;
            for (int j = 0; j < n ; j++) {
                if (arr[j] > arr[j + 1]) {
                    // Swap
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (!swapped) break; // Optimization: Stop if already sorted
        }
    }

    public static void main(String[] args) {
        int[] arr = {5, 2, 9, 1, 5, 6};
        bubbleSort(arr);
        System.out.println("Bubble Sorted: " + Arrays.toString(arr));
    }
}
```

2. Insertion Sort - Step-by-Step Diagram

- Insertion Sort builds the sorted array one element at a time.

Example: Sorting [5, 3, 8, 4, 2]

Step 1: [5, | 3, 8, 4, 2] → [3, 5, | 8, 4, 2]
Step 2: [3, 5, | 8, 4, 2] → [3, 5, 8, | 4, 2]
Step 3: [3, 5, 8, | 4, 2] → [3, 4, 5, 8, | 2]
Step 4: [3, 4, 5, 8, | 2] → [2, 3, 4, 5, 8]

```
import java.util.Arrays;

public class InsertionSort {
    public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int key = arr[i];
            int j = i - 1;

            // Shift elements of arr[0..i-1] that are greater than key
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }

    public static void main(String[] args) {
        int[] arr = {5, 2, 9, 1, 5, 6};
        insertionSort(arr);
        System.out.println("Insertion Sorted: " + Arrays.toString(arr));
    }
}
```

3. Selection Sort - Step-by-Step Diagram

- Selection Sort selects the smallest element and swaps it with the first element.

Step 1: [5, 3, 8, 4, 2] → [2, 3, 8, 4, 5] (2 is smallest, swap with 5)
Step 2: [2, 3, 8, 4, 5] → [2, 3, 8, 4, 5] (3 is already in correct position)
Step 3: [2, 3, 8, 4, 5] → [2, 3, 4, 8, 5] (4 is smallest in remaining, swap with 8)
Step 4: [2, 3, 4, 8, 5] → [2, 3, 4, 5, 8] (5 is smallest, swap with 8)

```
import java.util.Arrays;

public class SelectionSort {
   public static void selectionSort(int[] arr) {
      int n = arr.length;
      for (int i = 0; i < n - 1; i++) {
         int minIndex = i;
         for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
               minIndex = j;
            }
         }
         // Swap
         int temp = arr[minIndex];
         arr[minIndex] = arr[i];
         arr[i] = temp;
      }
   }

   public static void main(String[] args) {
      int[] arr = {5, 2, 9, 1, 5, 6};
      selectionSort(arr);
      System.out.println("Selection Sorted: " + Arrays.toString(arr));
   }
}
```

4. Quick Sort - Step-by-Step Diagram

- Quick Sort uses a pivot element and partitions the array.

Example: Sorting [5, 3, 8, 4, 2] (Pivot = Last Element 2)

Step 1: Partition around pivot 2 → [2, 3, 8, 4, 5]
Step 2: Recursively sort left [2] and right [3, 8, 4, 5] (Pivot = 5)
Step 3: Partition around 5 → [2, 3, 4, 5, 8]

[partition 1 | pivot | partition 2]
(value<pivot)   (value>pivot)

```java
import java.util.Arrays;

public class QuickSort {
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(arr, low, high);
            quickSort(arr, low, pivotIndex - 1); // Left partition
            quickSort(arr, pivotIndex + 1, high); // Right partition
        }
    }

    private static int partition(int[] arr, int low, int high) {
        int pivot = arr[high]; // Choosing last element as pivot
        int i = low - 1; // Index of smaller element

        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                // Swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // Swap pivot to correct position
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1; // Returning pivot index
    }

    public static void main(String[] args) {
        int[] arr = {5, 2, 9, 1, 5, 6};
        quickSort(arr, 0, arr.length - 1);
        System.out.println("Quick Sorted: " + Arrays.toString(arr));
    }
}
```

6. Merge Sort - Step-by-Step Diagram

- Merge Sort splits the array, sorts recursively, and merges them back.

Example: Sorting [5, 3, 8, 4, 2]

Step 1: Divide → [5, 3] and [8, 4, 2]

Step 2: Divide further → [5], [3], [8], [4, 2]

Step 3: Sort & Merge → [3, 5], [4, 2] → [2, 4, 8]

Step 4: Merge final → [2, 3, 4, 5, 8] (Sorted)

```
import java.util.Arrays;

public class MergeSort {
   public static void mergeSort(int[] arr, int left, int right) {
      if (left < right) {
         int mid = left + (right - left) / 2;

         // Recursively sort first and second halves
         mergeSort(arr, left, mid);
         mergeSort(arr, mid + 1, right);

         // Merge sorted halves
         merge(arr, left, mid, right);
      }
   }

   private static void merge(int[] arr, int left, int mid, int right) {
      int n1 = mid - left + 1;
      int n2 = right - mid;

      // Create temporary arrays
      int[] leftArray = new int[n1];
      int[] rightArray = new int[n2];

      // Copy data to temp arrays
      for (int i = 0; i < n1; i++) {
         leftArray[i] = arr[left + i];
      }
      for (int j = 0; j < n2; j++) {
         rightArray[j] = arr[mid + 1 + j];
      }
// Merge the temp arrays
      int i = 0, j = 0, k = left;
      while (i < n1 && j < n2) {
         if (leftArray[i] <= rightArray[j]) {
            arr[k] = leftArray[i];
            i++;
         } else {
            arr[k] = rightArray[j];
            j++;
         }
         k++;
      }

      // Copy remaining elements of leftArray
      while (i < n1) {
         arr[k] = leftArray[i];
         i++;
         k++;
      }

      // Copy remaining elements of rightArray
      while (j < n2) {
         arr[k] = rightArray[j];
         j++;
         k++;
      }
   }
   public static void main(String[] args) {
      int[] arr = {5, 2, 9, 1, 5, 6};
      mergeSort(arr, 0, arr.length - 1);
      System.out.println("Merge Sorted: " + Arrays.toString(arr));
   }
}
```

5. Heap Sort - Step-by-Step Diagram

- Heap Sort builds a **Max Heap** and extracts the largest element.

Example: Sorting [5, 3, 8, 4, 2]

Step 1: Convert to Max Heap → [8, 4, 5, 3, 2]

Step 2: Swap 8 with last → [2, 4, 5, 3, 8]

Step 3: Heapify → [5, 4, 2, 3, 8]

Step 4: Swap 5 with last → [3, 4, 2, 5, 8]

Step 5: Heapify → [4, 3, 2, 5, 8]

Step 6: Swap 4 with last → [2, 3, 4, 5, 8] (Sorted)

```
import java.util.Arrays;

public class HeapSort {
    public static void heapSort(int[] arr) {
        int n = arr.length;

        // Build max heap
        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(arr, n, i);
        }

        // Extract elements from heap one by one
        for (int i = n - 1; i > 0; i--) {
            // Swap root (largest) with the last element
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // Heapify the reduced heap
            heapify(arr, i, 0);
        }
    }
private static void heapify(int[] arr, int n, int i) {
      int largest = i; // Initialize largest as root
      int left = 2 * i + 1; // Left child
      int right = 2 * i + 2; // Right child

      // If left child is larger than root
      if (left < n && arr[left] > arr[largest]) {
         largest = left;
      }

      // If right child is larger than largest so far
      if (right < n && arr[right] > arr[largest]) {
         largest = right;
      }

      // If largest is not root
      if (largest != i) {
         int temp = arr[i];
         arr[i] = arr[largest];
         arr[largest] = temp;

         // Recursively heapify the affected sub-tree
         heapify(arr, n, largest);
      }
   }

   public static void main(String[] args) {
      int[] arr = {5, 2, 9, 1, 5, 6};
      heapSort(arr);
      System.out.println("Heap Sorted: " + Arrays.toString(arr));
   }
}
```

Sorting Algorithm	Best Case	Worst Case	Average Case	Space Complexity	Stable?
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	✅ Yes
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	✅ Yes
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	❌ No
Quick Sort	O(n log n)	$O(n^2)$	O(n log n)	O(log n)	❌ No
Heap Sort	O(n log n)	O(n log n)	O(n log n)	O(1)	❌ No
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	✅ Yes