Marwadi University	**Marwadi University** **Faculty of Technology** **Department of Information and Communication Technology**
Subject: DSC (01CT0308)	**Aim:** Implementations of Shell sort, Radix sort, Insertion sort, Quick Sort, Merge sort, and Heap Sort menu-driven program.
Experiment No: 8	**Date:** 26- 10 - 2023	**Enrolment No:-** 92200133030

Experiment – 8

Objective: Implementations of Shell sort, Radix sort, Insertion sort, Quick Sort, Merge sort, and Heap Sort menu-driven program.

Code :-

```
#include <iostream>
using namespace std;

// Function to display an array
void displayArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

// Shell Sort
void shellSort(int arr[], int size) {
    for (int gap = size / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < size; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}


// Radix Sort
// A utility function to get the maximum value in arr[]
```

```
int getMax(int arr[], int size) {
    int max = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}

// A function to do counting sort based on significant place exp (1, 10, 100, etc.)
void countingSort(int arr[], int size, int exp) {
    const int RADIX = 10; // The base for counting sort
    int output[size];
    int count[RADIX] = {0};

    for (int i = 0; i < size; i++) {
        count[(arr[i] / exp) % RADIX]++;
    }

    for (int i = 1; i < RADIX; i++) {
        count[i] += count[i - 1];
    }

    for (int i = size - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % RADIX] - 1] = arr[i];
        count[(arr[i] / exp) % RADIX]--;
    }

    for (int i = 0; i < size; i++) {
        arr[i] = output[i];
    }
}

// The main function to implement Radix Sort
void radixSort(int arr[], int size) {
    int max = getMax(arr, size);

    for (int exp = 1; max / exp > 0; exp *= 10) {
        countingSort(arr, size, exp);
    }
}


// Insertion Sort
void insertionSort(int arr[], int size) {
    for (int i = 1; i < size; 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;
    }
}

// Quick Sort
// Partition function to find the correct position of the pivot
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Choose the rightmost element as the pivot
    int i = (low - 1);     // Initialize the index of the smaller element

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++; // Increment the index of the smaller element
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1); // Return the position of the pivot
}

// Recursive function to perform Quick Sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // Find pivot element such that element smaller than pivot
        // are on the left and elements greater are on the right
        int pi = partition(arr, low, high);

        // Recursively sort elements before and after the pivot
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}



// Merge Sort
// Merge two subarrays of arr[].
// First subarray is arr[left..middle]
// Second subarray is arr[middle+1..right]
void merge(int arr[], int left, int middle, int right) {
```

```
    int n1 = middle - left + 1;
    int n2 = right - middle;

    // Create temporary arrays
    int L[n1], R[n2];

    // Copy data to temp arrays L[] and R[]
    for (int i = 0; i < n1; i++) {
        L[i] = arr[left + i];
    }
    for (int i = 0; i < n2; i++) {
        R[i] = arr[middle + 1 + i];
    }

    // Merge the temp arrays back into arr[left..right]
    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of L[], if any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy the remaining elements of R[], if any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Main function to perform Merge Sort
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        // Same as (left+right)/2, but avoids overflow for large left and right
```

```
        int middle = left + (right - left) / 2;

        // Sort the first and second halves
        mergeSort(arr, left, middle);
        mergeSort(arr, middle + 1, right);

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


// Heap Sort
// To heapify a subtree rooted with node i which is an index in arr[].
// n is the size of the heap
void heapify(int arr[], int n, int i) {
    int largest = i; // Initialize largest as the 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 the largest is not the root
    if (largest != i) {
        swap(arr[i], arr[largest]);

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

// Main function to perform Heap Sort
void heapSort(int arr[], int n) {
    // Build a max heap
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }

    // Extract elements one by one from the heap
```

```
    for (int i = n - 1; i > 0; i--) {
        // Move the current root to the end
        swap(arr[0], arr[i]);

        // Call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}


int main() {
    int size;
    cout << "Enter the size of the array: ";
    cin >> size;

    int arr[size];

    cout << "Enter the elements of the array: ";
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }

    int choice;
    do {
        cout << "\nSorting Menu:\n";
        cout << "1. Shell Sort\n";
        cout << "2. Radix Sort\n";
        cout << "3. Insertion Sort\n";
        cout << "4. Quick Sort\n";
        cout << "5. Merge Sort\n";
        cout << "6. Heap Sort\n";
        cout << "7. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                shellSort(arr, size);
                cout << "Shell Sort Result: ";
                displayArray(arr, size);
                break;
            case 2:
                radixSort(arr, size);
                cout << "Radix Sort Result: ";
                displayArray(arr, size);
                break;
            case 3:
```

```
            insertionSort(arr, size);
            cout << "Insertion Sort Result: ";
            displayArray(arr, size);
            break;
        case 4:
            quickSort(arr, 0, size - 1);
            cout << "Quick Sort Result: ";
            displayArray(arr, size);
            break;
        case 5:
            mergeSort(arr, 0, size - 1);
            cout << "Merge Sort Result: ";
            displayArray(arr, size);
            break;
        case 6:
            heapSort(arr, size);
            cout << "Heap Sort Result: ";
            displayArray(arr, size);
            break;
        case 7:
            cout << "Exiting the program." << endl;
            break;
        default:
            cout << "Invalid choice. Please try again." << endl;
            break;
      }
   } while (choice != 7);

   return 0;
}
```

Output:

```
Enter the size of the array: 5
Enter the elements of the array: 45
22
89
54
78

Sorting Menu:
1. Shell Sort
2. Radix Sort
3. Insertion Sort
4. Quick Sort
5. Merge Sort
6. Heap Sort
7. Exit
Enter your choice: 1
Shell Sort Result: 22 45 54 78 89

Sorting Menu:
1. Shell Sort
2. Radix Sort
3. Insertion Sort
4. Quick Sort
5. Merge Sort
6. Heap Sort
7. Exit
Enter your choice: 2
Radix Sort Result: 22 45 54 78 89
```