Data Structure and Algorithm Lab Manual (Week-03)

Lab Instructor: Ms. Rabeeya Saleem

Session: 2024 (Fall 2025)

Example 1.1: Bubble Sort (A class of 5 students has marks `[45, 72, 39, 90, 66]`. Use **Bubble Sort** to rank them in ascending order)

```
void bubbleSort(int arr[], int n) {
    for(int i=0; i<n-1; i++) {
        for(int j=0; j<n-i-1; j++) {
            if(arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

Example 1.2:

Library shelf has books sorted by ID `[101, 125, 140, 160]`. A new book with ID `130` arrives. Use **Insertion Sort** to insert it in order. Updated array: {101, 125, 140, 160, 130}

```
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        // Shift larger elements to the right
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```

Example 1.3: Employees' sales numbers `[500, 320, 720, 610, 450]`. Use **Selection Sort** to arrange in descending order and find the **top performer**.

```
void selectionSort(int arr[], int n) {
    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;
        }
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}
```

Example 1.4: An e-commerce site has order amounts `[250, 1000, 450, 700, 300, 900]`. Sort them using **Quick Sort** to optimize order processing.

```
int partition(int arr[], int low, int high) {
   int pivot = arr[high];
   int i = low-1;
   for(int j=low; j<high; j++) {
      if(arr[j] <= pivot) {
         i++;
         int temp = arr[i]; arr[i]=arr[j]; arr[j]=temp;
      }
   }
   int temp = arr[i+1]; arr[i+1]=arr[high]; arr[high]=temp;
   return i+1;
}

void quickSort(int arr[], int low, int high) {
   if(low<high) {
      int pi = partition(arr, low, high);
      quickSort(arr, low, pi-1);
      quickSort(arr, pi+1, high);
   }
}

quickSort (arr,0,n-1);
```

Example 1.5: University exam scores `[38, 27, 43, 3, 9, 82, 10]` must be sorted efficiently for generating a merit list. Apply **Merge Sort**.

```
void merge(int arr[], int l, int m, int r) {
   int n1 = m-l+1, n2 = r-m;
   int L[50], R[50]; // assuming small input

   for(int i=0;i<n1;i++) L[i]=arr[l+i];
   for(int j=0;j<n2;j++) R[j]=arr[m+1+j];

   int i=0,j=0,k=l;
   while(i<n1 && j<n2) {
      if(L[i]<=R[j]) arr[k++] = L[i++];
      else arr[k++] = R[j++];
   }
   while(i<n1) arr[k++] = L[i++];
   while(j<n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int l, int r) {
   if(l<r) {
      int m = (l+r)/2;
      mergeSort(arr, l, m);
      mergeSort(arr, m+1, r);
      merge(arr, l, m, r);
   }
}
```

```
mergeSort (arr,0,n-1);
```

Example 1.6: Graphics system uses intensity values between `0–1`: `[0.78, 0.17, 0.39, 0.26, 0.72, 0.94]`. Use **Bucket Sort** to arrange them for rendering

```
void bucketSort(float arr[], int n) {
    // Step 1: Create buckets (1D array per bucket)
    float bucket[10][10];   // max 10 elements per bucket (for small input)
    int count[10] = {0};    // how many elements in each bucket

    // Step 2: Put elements into buckets
    for (int i = 0; i < n; i++) {
        int bi = arr[i] * 10;   // bucket index
        bucket[bi][count[bi]++] = arr[i];
    }

    // Step 3: Sort each bucket
    for (int i = 0; i < 10; i++) {
        if (count[i] > 0) {
            insertionSort(bucket[i], count[i]);
        }
    }

    // Step 4: Concatenate buckets back into original array
    int k = 0;
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < count[i]; j++) {
            arr[k++] = bucket[i][j];
        }
    }
}
```

Example 1.7: In an election, candidates are numbered 0–5. Votes `[1, 4, 2, 1, 3, 2, 4, 1]` are cast. Use **Counting Sort** to tally votes and display results in sorted order.

```
void countingSort(int arr[], int n, int maxVal) {
    int count[50] = {0};   // frequency array (assume max 50 elements)
    int output[50];        // sorted result

    // Step 1: Count frequency of each number
    for (int i = 0; i < n; i++) {
        count[arr[i]]++;
    }

    // Step 2: Convert count[] to cumulative sum
    for (int i = 1; i <= maxVal; i++) {
        count[i] = count[i] + count[i - 1];
    }

    // Step 3: Build output array (go right to left for stability)
    for (int i = n - 1; i >= 0; i--) {
        int val = arr[i];
        output[count[val] - 1] = val;
        count[val]--;
```

```
    }

    // Step 4: Copy back to original array
    for (int i = 0; i < n; i++) {
        arr[i] = output[i];
    }
}
```

Example 1.8: Customer IDs `[170, 45, 75, 90, 802, 24, 2, 66]` must be sorted. Since IDs are multi-digit integers, apply **Radix Sort**.

```
// Function to get the maximum value in array
int getMax(int arr[], int n) {
    int mx = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > mx)
            mx = arr[i];
    return mx;
}

// Counting sort for one digit (exp = 1, 10, 100, ...)
void countSort(int arr[], int n, int exp) {
    int output[50];   // output array (max size = 50 for demo)
    int count[10] = {0};

    // Count digits
    for (int i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;

    // Convert count[] to cumulative
    for (int i = 1; i < 10; i++)
        count[i] += count[i - 1];

    // Build output (stable, go right to left)
    for (int i = n - 1; i >= 0; i--) {
        int digit = (arr[i] / exp) % 10;
        output[count[digit] - 1] = arr[i];
        count[digit]--;
    }

    // Copy back to arr[]
    for (int i = 0; i < n; i++)
        arr[i] = output[i];
}

// Radix Sort main
void radixSort(int arr[], int n) {
    int m = getMax(arr, n);
    // Do counting sort for each digit
    for (int exp = 1; m / exp > 0; exp *= 10)
        countSort(arr, n, exp);
}
```

Example 1.9: Take two arrays from user/system and find the union from these two arrays. arr1[5] = {1, 2, 3, 4, 5}(n1, n2 is size of arr1 arr2) arr2[5] = {3, 4, 5, 6, 7};

```
int arr[20]; // combined array

// Step 1: Copy both arrays into arr[]
for (int i = 0; i < n1; i++) arr[i] = arr1[i];
for (int i = 0; i < n2; i++) arr[n1 + i] = arr2[i];
int n = n1 + n2;

// Step 2: Sort the combined array
sort(arr, arr + n);

// Step 3: Print unique elements (remove duplicates)
cout << "Union of arrays: ";
cout << arr[0] << " "; // first element always included
for (int i = 1; i < n; i++) {
    if (arr[i] != arr[i - 1]) {
        cout << arr[i] << " ";
    }
}

return 0;
```

Example 1.10: Multi-value sorting University Admissions Applicants are (ApplicantID, Marks, Age). Sort by Marks descending. If Marks are equal, sort by Age ascending (younger first). Print the final admission list. provide solution in c+

```
// Step 1: Input applicants
int id[]    = {101, 102, 103, 104, 105};
int marks[] = { 90,  85,  90,  75,  85};
int age[]   = { 19,  18,  17,  20,  19};
int n = 5;

// Step 2: Sort using Bubble Sort
for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
        // Condition: Higher marks first, if tie then younger first
        if (marks[j] < marks[j + 1] ||
           (marks[j] == marks[j + 1] && age[j] > age[j + 1])) {

            // Swap marks
            int temp = marks[j];
            marks[j] = marks[j + 1];
            marks[j + 1] = temp;

            // Swap age
            temp = age[j];
            age[j] = age[j + 1];
            age[j + 1] = temp;

            // Swap id
            temp = id[j];
            id[j] = id[j + 1];
            id[j + 1] = temp;
        }
```

```
        }
    }

    // Step 3: Print final admission list
    cout << "Final Admission List:\n";
    for (int i = 0; i < n; i++) {
        cout << "ApplicantID: " << id[i]
             << ", Marks: " << marks[i]
             << ", Age: " << age[i] << endl;
    }
```

Problems1-8:

Given an array, arr[0..n-1] of distinct elements and a range [low, high], find all numbers that are in a range, but not the array. The missing elements should be printed in sorted order.	**Input:** arr[] = {10, 12, 11, 15}, low = 10, high = 15 **Output:** 13, 14
Votes are stored as candidate IDs: [1, 3, 2, 2, 5, 1, 4, 3, 2]. Tasks: • Use **Counting Sort** to tally and sort votes. • Print each candidate's **total votes**. • Display the **winner candidate**.	Output: Candidate 2 with Vote count 3
Given an array of integers, the task is to arrange the elements such that all negative integers appear before all the positive integers in the array.	**Input:** arr[] = [12, 11, -13, -5, 6, -7, 5, -3, -6] **Output:** [-13, -5, -7, -3, -6, 12, 11, 6, 5]
Given two **sorted** arrays **a[]** and **b[],** the task is to return intersection. **Intersection** of two arrays is said to be elements that are **common** in both arrays.	**Input**: a[] = {1, 1, 2, 2, 2, 4}, b[] = {2, 2, 4, 4} **Output**: {2, 4} **Explanation**: 2 and 4 are only common elements in both the arrays. **Input**: a[] = {1, 2}, b[] = {3, 4} **Output**: {}
Find K-th Smallest/Largest Element In array [12, 3, 5, 7, 19], find the 3rd smallest using QuickSort partition logic.	Output: 3rd smallest element is 7
Sort Based on Multiple Keys Students (RollNo, Marks) → sort by Marks (descending), if tie then RollNo ascending.	Input: Students = [(1, 90), (2, 75), (3, 90), (4, 60), (5, 75)] [(1, 90), (3, 90), (2, 75), (5, 75), (4, 60)]
Airline Flight Scheduling Flights = [930, 745, 1230, 1100, 1545, 600] • Sort flights in ascending order. • Convert to **HH:MM format**.	Unsorted → [930, 745, 1230, 1100, 1545, 600] Sorted → [600, 745, 930, 1100, 1230, 1545] Sorted flight times: 06:00 07:45 09:30 11:00 12:30 15:45 Earliest flight: 06:00 Latest flight: 15:45

• Display the **earliest** and **latest** flights.	
5. Bank Transaction Alerts Transactions are stored as amounts [1200, 50, 330, 75, 8900, 600, 25]. • Sort transactions using Quick Sort. • Print all transactions **greater than 1000** as “Suspicious”. • Print all other transactions as “Normal”.	25 -> Normal 50 -> Normal 75 -> Normal 330 -> Normal 600 -> Normal 1200 -> Suspicious 8900 -> Suspicious