ADA LAB 2

Quick Sort:

Using Iteration

```
#include <iostream>

// Function to swap two elements
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

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

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

// Iterative QuickSort function
void quickSortIterative(int arr[], int l, int h) {
    // Create an auxiliary stack
    int stack[h - l + 1];

    // Initialize top of stack
    int top = -1;

    // Push initial values of l and h to stack
    stack[++top] = l;
    stack[++top] = h;

    // Keep popping from stack while it is not empty
    while (top >= 0) {
```

```
        // Pop h and l
        h = stack[top--];
        l = stack[top--];

        // Set pivot element at its correct position
        int p = partition(arr, l, h);

        // If there are elements on left side of pivot, push left side to stack
        if (p - 1 > l) {
            stack[++top] = l;
            stack[++top] = p - 1;
        }

        // If there are elements on right side of pivot, push right side to stack
        if (p + 1 < h) {
            stack[++top] = p + 1;
            stack[++top] = h;
        }
    }
}

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

int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    std::cout << "Unsorted array: ";
    printArray(arr, n);

    quickSortIterative(arr, 0, n - 1);

    std::cout << "Sorted array: ";
    printArray(arr, n);
    return 0;
}
```

Using Recursion

```
#include <iostream>

// Function to swap two elements
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

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

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

// Recursive QuickSort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

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

// Function to print an array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;
}
```

```
int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    std::cout << "Unsorted array: ";
    printArray(arr, n);

    quickSort(arr, 0, n - 1);

    std::cout << "Sorted array: ";
    printArray(arr, n);
    return 0;
}
```

Merge Sort:

Using Iteration

```
#include <iostream>

// Function to merge two subarrays of arr[]
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 temporary arrays leftArray[] and rightArray[]
    for (int i = 0; i < n1; i++)
        leftArray[i] = arr[left + i];
    for (int i = 0; i < n2; i++)
        rightArray[i] = arr[mid + 1 + i];

    // Merge the temporary arrays back into arr[left..right]

    int i = 0;    // Initial index of the first subarray
    int j = 0;    // Initial index of the second subarray
    int k = left; // Initial index of merged subarray

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

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

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

    // Free the temporary arrays
    delete[] leftArray;
    delete[] rightArray;
}

// Iterative Merge Sort function to sort arr[0...n-1]
void mergeSort(int arr[], int n) {
    // For current size of subarrays to be merged curr_size varies from 1 to n/2
    int curr_size;

    // For picking starting index of left subarray to be merged
    int left_start;

    // Merge subarrays in bottom up manner. First merge subarrays of size 1 to create sorted
subarrays of size 2, then merge subarrays of size 2 to create sorted subarrays of size 4, and so
on.
    for (curr_size = 1; curr_size <= n-1; curr_size = 2*curr_size) {
        // Pick starting point of different subarrays of current size
        for (left_start = 0; left_start < n-1; left_start += 2*curr_size) {
            // Find ending point of left subarray. mid+1 is starting point of right
            int mid = std::min(left_start + curr_size - 1, n-1);

            int right_end = std::min(left_start + 2*curr_size - 1, n-1);

            // Merge Subarrays arr[left_start...mid] & arr[mid+1...right_end]
            merge(arr, left_start, mid, right_end);
        }
    }
}

// Function to print an array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;
}
```

```
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Unsorted array: ";
    printArray(arr, arr_size);

    mergeSort(arr, arr_size);

    std::cout << "Sorted array: ";
    printArray(arr, arr_size);
    return 0;
}
```

Using Recursion

```
#include <iostream>

// Function to merge two subarrays of arr[]
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 temporary arrays leftArray[] and rightArray[]
    for (int i = 0; i < n1; i++)
        leftArray[i] = arr[left + i];
    for (int i = 0; i < n2; i++)
        rightArray[i] = arr[mid + 1 + i];

    // Merge the temporary arrays back into arr[left..right]

    int i = 0;    // Initial index of the first subarray
    int j = 0;    // Initial index of the second subarray
    int k = left; // Initial index of merged subarray

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

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

    // Copy the remaining elements of rightArray[], if there are any
    while (j < n2) {
```

```
        arr[k] = rightArray[j];
        j++;
        k++;
    }

    // Free the temporary arrays
    delete[] leftArray;
    delete[] rightArray;
}

// Recursive Merge Sort function to sort arr[0...n-1]
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 the sorted halves
        merge(arr, left, mid, right);
    }
}

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

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Unsorted array: ";
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    std::cout << "Sorted array: ";
    printArray(arr, arr_size);
    return 0;
}
```

Selection Sort:

Using Iteration

```
#include <iostream>

// Function to swap two elements
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

// Iterative Selection Sort function to sort arr[0...n-1]
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        // Find the minimum element in the unsorted array
        int min_idx = i;
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[min_idx])
                min_idx = j;
        }

        // Swap the found minimum element with the first element
        swap(arr[min_idx], arr[i]);
    }
}

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

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
    std::cout << "Unsorted array: ";
    printArray(arr, n);
    selectionSort(arr, n);
    std::cout << "Sorted array: ";
    printArray(arr, n);
    return 0;
}
```

Using Recursion

```
#include <iostream>

// Function to swap two elements
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

// Function to find the index of the minimum element
int findMinIndex(int arr[], int start, int end) {
    int minIndex = start;
    for (int i = start + 1; i <= end; i++) {
        if (arr[i] < arr[minIndex]) {
            minIndex = i;
        }
    }
    return minIndex;
}

// Recursive Selection Sort function to sort arr[start...n-1]
void selectionSortRecursive(int arr[], int start, int n) {
    // Base case: when start index reaches n-1
    if (start >= n - 1) {
        return;
    }

    // Find the index of the minimum element in the remaining unsorted array
    int minIndex = findMinIndex(arr, start, n - 1);

    // Swap the found minimum element with the first element
    swap(arr[start], arr[minIndex]);

    // Recursively call selection sort on the remaining unsorted array
    selectionSortRecursive(arr, start + 1, n);
}

// Function to print an array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
```

```
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
    std::cout << "Unsorted array: ";
    printArray(arr, n);

    selectionSortRecursive(arr, 0, n);

    std::cout << "Sorted array: ";
    printArray(arr, n);
    return 0;
}
```