**LAB SESSION 12: SORTING**

**AIM:** To implement various sorting algorithms.

**PROBLEM DEFINITION:**

Develop a C program to implement the following sorting algorithms:

1. Insertion Sort using a character array
2. Merge Sort using a linked list
3. Quick Sort
4. Heap Sort

**THEORY:**

**Insertion sort:** Insertion sort works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.

The same approach is applied in insertion sort. The idea behind the insertion sort is that first take one element, iterate it through the sorted array. Although it is simple to use, it is not appropriate for large data sets as the time complexity of insertion sort in the average case and worst case is O(n2), where n is the number of items. Insertion sort is less efficient than the other sorting algorithms like heap sort, quick sort, merge sort, etc.

**Merge sort:** Merge sort uses the divide and conquer approach to sort the elements. It is one of the most popular and efficient sorting algorithms. It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the merge() function to perform the merging.

The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process. The sorted two-element pairs is merged into the four-element lists, and so on until we get the sorted list.

**Quick sort:** Quick sort picks an element as pivot, and then it partitions the given array around the picked pivot element. In quick sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot. After that, left and right sub-arrays are also partitioned using the same approach. It will continue until the single element remains in the sub-array.

**Heap sort:** Heap sort processes the elements by creating the min-heap or max-heap using the elements of the given array. Min-heap or max-heap represents the ordering of array in which the root element represents the minimum or maximum element of the array. Heap sort basically recursively performs two main operations :Build a heap H, using the elements of array and

Repeatedly delete the root element of the heap formed in 1st phase.

**ALGORITHMS**

1. Insertion Sort

1. Start with the second element (index 1) and compare it with the elements before it.

2. Move the element to the left until a smaller element is encountered or until the beginning of the array is reached.

3. Repeat the process for the remaining unsorted elements, moving each element to its correct position.

4. Continue until the entire array is sorted.

2. Merge Sort

1. Divide the unsorted list into n sub-lists, each containing one element (base case).

2. Repeatedly merge sub-lists to produce new sorted sub-lists until there is only one sub-list remaining.

3. The final sub-list is the sorted list.

3. Quick Sort

1. Choose a pivot element from the array (typically the last element).

2. Partition the array into two sub-arrays: elements less than the pivot and elements greater than the pivot.

3. Recursively apply the quick sort algorithm to the two sub-arrays.

4. Combine the sorted sub-arrays and the pivot to get the final sorted array.

**PROGRAM**

#include <stdio.h>

#include <stdlib.h>

struct Node{

    int data;

    struct Node \*next;

};

void swap(int \*a, int \*b){

    int t = \*a;

    \*a = \*b;

    \*b = t;

}

void printList(struct Node \*node){

    while (node){

        printf("%d ", node->data);

        node = node->next;

    }

    printf("\n");

}

void printArray(int arr[], int n){

    for (int i = 0; i < n; i++)

        printf("%d ", arr[i]);

    printf("\n");

}

int partition(int arr[], int low, int high, int order){

    int pivot = arr[high];

    int i = low - 1;

    for (int j = low; j <= high - 1; j++){

        if (order \* arr[j] < order \* pivot){

            i++;

            swap(&arr[i], &arr[j]);

        }

    }

    swap(&arr[i + 1], &arr[high]);

    return i + 1;

}

void quickSort(int arr[], int low, int high, int order){

    if (low < high){

        int pi = partition(arr, low, high, order);

        printf("QuickSort Iteration %d: ", pi);

        printArray(arr + low, high - low + 1);

        quickSort(arr, low, pi - 1, order);

        quickSort(arr, pi + 1, high, order);

    }

}

void heapify(int arr[], int n, int i, int order){

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    if (left < n && order \* arr[left] > order \* arr[largest])

        largest = left;

    if (right < n && order \* arr[right] > order \* arr[largest])

        largest = right;

    if (largest != i)

        swap(&arr[i], &arr[largest]);

        heapify(arr, n, largest, order);

}

void insertionSort(char arr[], int n, int order){

    int i, key, j;

    for (i = 1; i < n; i++){

        key = arr[i];

        j = i - 1;

        while (j >= 0 && order \* arr[j] > order \* key){

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

        printf("Insertion Sort Iteration %d: ", i);

        for (int k = 0; k < n; k++)

            printf("%c ", arr[k]);

        printf("\n");

    }

}

struct Node \*merge(struct Node \*left, struct Node \*right, int order){

    if (!left)

        return right;

    if (!right)

        return left;

    struct Node \*result = NULL;

    if ((order == 1 && left->data <= right->data) || (order == -1 && left->data >= right->data)){

        result = left;

        result->next = merge(left->next, right, order);

    }

    else{

        result = right;

        result->next = merge(left, right->next, order);

    }

    return result;

}

void split(struct Node \*source, struct Node \*\*left, struct Node \*\*right){

    struct Node \*fast;

    struct Node \*slow;

    if (!source || !source->next){

        \*left = source;

        \*right = NULL;

    }

    else{

        slow = source;

        fast = source->next;

        while (fast){

            fast = fast->next;

            if (fast){

                slow = slow->next;

                fast = fast->next;

            }

        }

        \*left = source;

        \*right = slow->next;

        slow->next = NULL;

    }

}

struct Node \*mergeSort(struct Node \*head, int order, int iteration){

    if (!head || !head->next)

        return head;

    struct Node \*left;

    struct Node \*right;

    split(head, &left, &right);

    left = mergeSort(left, order, iteration);

    right = mergeSort(right, order, iteration);

    struct Node \*merged = merge(left, right, order);

    printf("MergeSort Iteration %d: ", iteration);

    printList(merged);

    return merged;

}

void push(struct Node \*\*head, int newData){

    struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

    newNode->data = newData;

    newNode->next = \*head;

    \*head = newNode;

}

void heapSort(int arr[], int n, int order){

    for (int i = n / 2 - 1; i >= 0; i--){

        heapify(arr, n, i, order);

        printf("Heap Sort Iteration %d: ", n / 2 - i);

        printArray(arr, n);

    }

    for (int i = n - 1; i > 0; i--){

        swap(&arr[0], &arr[i]);

        heapify(arr, i, 0, order);

        printf("Heap Sort Iteration %d: ", n - i);

        printArray(arr, i);

    }

}

int main(){

    int choice;

    int order;

    do{

        printf("\nMenu:\n");

        printf("1. Insertion Sort (Character Array)\n");

        printf("2. Merge Sort (Linked List)\n");

        printf("3. Quick Sort\n");

        printf("4. Heap Sort\n");

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice){

        case 1:{

            char arr[100];

            int n;

            printf("Enter the size of the character array: ");

            scanf("%d", &n);

            printf("Enter the character array elements: ");

            for (int i = 0; i < n; i++)

                scanf(" %c", &arr[i]);

            printf("Unsorted array: ");

            for (int i = 0; i < n; i++)

                printf("%c ", arr[i]);

            printf("\n");

            insertionSort(arr, n, 1);

            printf("Sorted array (Ascending): ");

            for (int i = 0; i < n; i++)

                printf("%c ", arr[i]);

            printf("\n\n");

            insertionSort(arr, n, -1);

            printf("Sorted array (Descending): ");

            for (int i = 0; i < n; i++)

                printf("%c ", arr[i]);

            printf("\n");

            break;

        }

        case 2:{

            struct Node \*head = NULL;

            int num, data;

            printf("Enter the number of elements in the linked list: ");

            scanf("%d", &num);

            printf("Enter the linked list elements: ");

            for (int i = 0; i < num; i++){

                scanf("%d", &data);

                push(&head, data);

            }

            printf("Linked List before sorting \n");

            printList(head);

            head = mergeSort(head, 1, 1);

            printf("Linked List after sorting (Ascending): ");

            printList(head);

            printf("\n");

            head = mergeSort(head, -1, 2);

            printf("Linked List after sorting (Descending): ");

            printList(head);

            break;

        }

        case 3:{

            int num;

            printf("Enter the number of elements in the array: ");

            scanf("%d", &num);

            int arr[num];

            printf("Enter the array elements: ");

            for (int i = 0; i < num; i++)

                scanf("%d", &arr[i]);

            printf("Unsorted array: \n");

            printArray(arr, num);

            quickSort(arr, 0, num - 1, 1);

            printf("Sorted array (Ascending): ");

            printArray(arr, num);

            printf("\n");

            quickSort(arr, 0, num - 1, -1);

            printf("Sorted array (Descending):") ;

            printArray(arr, num);

            break;

        }

        case 4:{

            int num;

            printf("Enter the number of elements in the array: ");

            scanf("%d", &num);

            int arr[num];

            printf("Enter the array elements: ");

            for (int i = 0; i < num; i++)

                scanf("%d", &arr[i]);

            printf("Unsorted array: \n");

            printArray(arr, num);

            heapSort(arr, num, 1);

            printf("Sorted array (Ascending): \n");

            printArray(arr, num);

            printf("\n");

            heapSort(arr, num, -1);

            printf("Sorted array (Descending): \n");

            printArray(arr, num);

            break;

        }

        case 5: break;

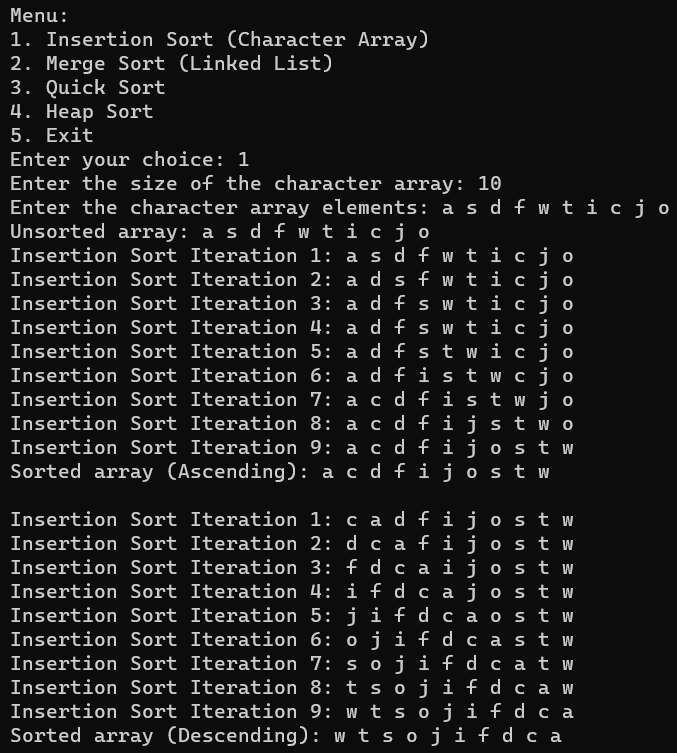
        default: printf("Invalid choice\n");

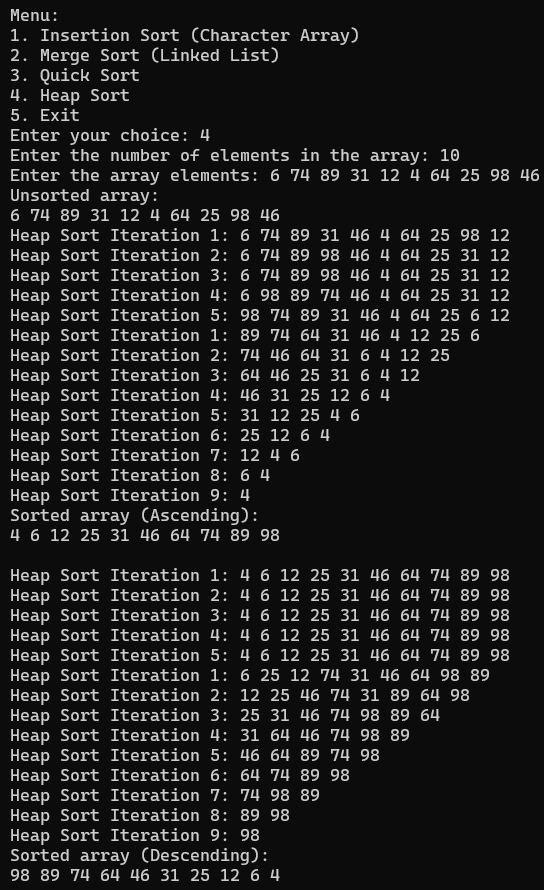
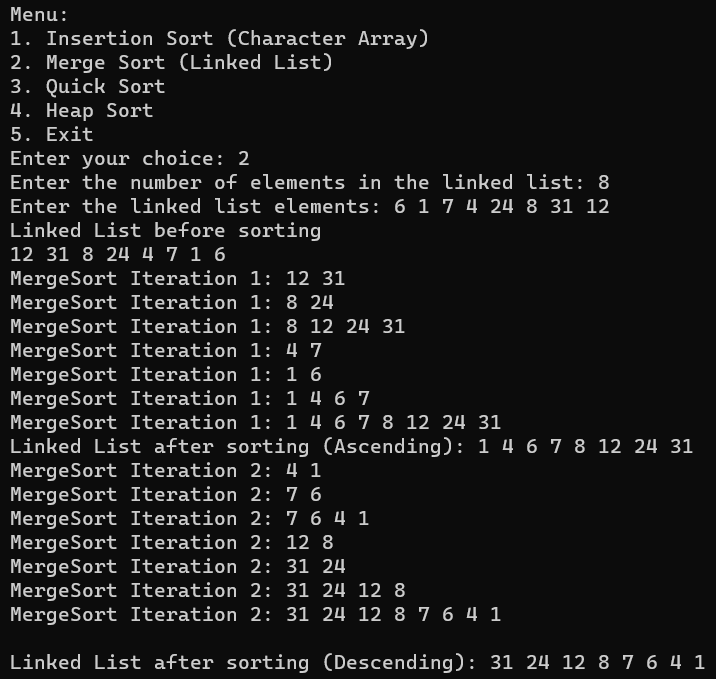
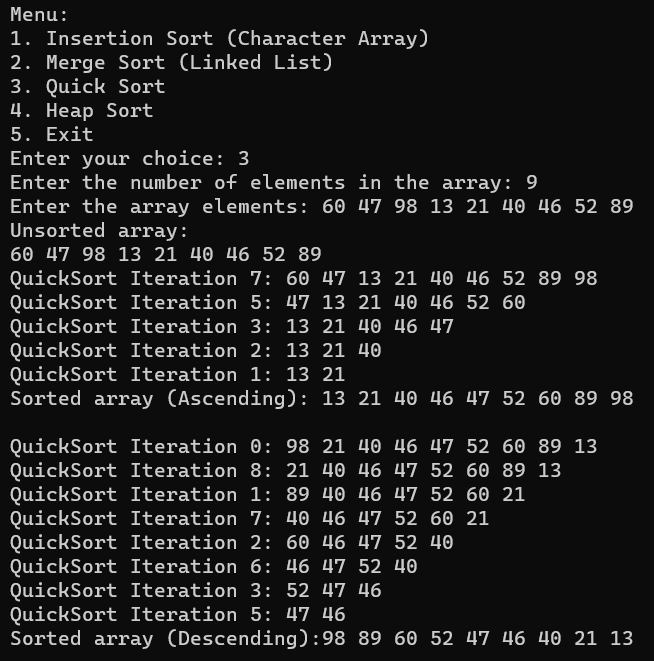
        }

    } while (choice != 5);

    return 0;

}

****

****

**CONCLUSION:**

A program implementing

* Insertion Sort using a character array
* Merge Sort using a linked list
* Quick Sort
* Heap Sort

Was written and outputs successfully obtained.