Folder SEM 3\Exp9

7 printable files

(file list disabled)

SEM 3\Exp9\Bucket_Sort.c

```c
// Function to perform bucket sort
void bucketSort(float arr[], int size)
{
    // Create buckets
    int bucketCount = 10;             // Number of buckets
    float buckets[bucketCount][size]; // 2D array to hold buckets
    int bucketSizes[bucketCount];     // Array to hold the size of each bucket

    // Initialize bucket sizes to 0
    for (int i = 0; i < bucketCount; i++)
        bucketSizes[i] = 0;

    // Insert elements into buckets
    for (int i = 0; i < size; i++)
    {
        int bucketIndex = (int)(bucketCount * arr[i]);             // Determine bucket index
        buckets[bucketIndex][bucketSizes[bucketIndex]++] = arr[i]; // Place element in bucket
    }

    // Sort individual buckets
    for (int i = 0; i < bucketCount; i++)
    {
        if (bucketSizes[i] > 0)
        {
            // Sort the bucket (using insertion sort here)
            insertionSort(buckets[i], bucketSizes[i]);
        }
    }

    // Concatenate all buckets into the original array
    int index = 0;
    for (int i = 0; i < bucketCount; i++)
    {
        for (int j = 0; j < bucketSizes[i]; j++)
        {
            arr[index++] = buckets[i][j];
        }
    }
}

```

SEM 3\Exp9\Heap_Sort.c

```c
// Function to heapify a subtree rooted at index i
```

```c
void heapify(int arr[], int size, int i)
{
    int largest = i;        // Initialize largest as root
    int left = 2 * i + 1;   // left = 2*i + 1
    int right = 2 * i + 2; // right = 2*i + 2

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

    // If right child is larger than largest so far
    if (right < size && 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 subtree
        heapify(arr, size, largest);
    }
}

// Function to perform heap sort
void heapSort(int arr[], int size)
{
    // Build heap (rearrange array)
    for (int i = size / 2 - 1; i >= 0; i--)
        heapify(arr, size, i);

    // One by one extract an element from heap
    for (int i = size - 1; i > 0; i--)
    {
        // Move current root to end
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

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

```

SEM 3\Exp9\Makefile

```
# Compiler to use
CC = gcc

# Compiler flags
CFLAGS = -Wall -Wextra -g
```

```
6
7  # Object files to compile
8  OBJS = main.o Bucket_Sort.o counting_sort.o Heap_Sort.o Merge_Sort.o Radix_sort.o
9
10 # The final executable name
11 TARGET = Exp9_sorting_program
12
13 # Default target to build the executable
14 all: $(TARGET)
15
16 # Rule to link object files into the final executable
17 $(TARGET): $(OBJS)
18     $(CC) -o $(TARGET) $(OBJS)
19
20 # Rule to compile each .c file into a .o file
21 %.o: %.c
22     $(CC) $(CFLAGS) -c $<
23
24 # Clean target to remove object files and the executable
25 clean:
26     rm -f $(OBJS) $(TARGET)
27
```

SEM 3\Exp9\Merge_Sort.c

```c
1  #include <stdio.h>
2
3  // Function to merge two subarrays
4  void merge(int arr[], int left, int mid, int right)
5  {
6      int i, j, k;
7      int n1 = mid - left + 1;
8      int n2 = right - mid;
9
10     // Create temporary arrays
11     int L[n1], R[n2];
12
13     // Copy data to temporary arrays
14     for (i = 0; i < n1; i++)
15         L[i] = arr[left + i];
16     for (j = 0; j < n2; j++)
17         R[j] = arr[mid + 1 + j];
18
19     // Merge the temporary arrays
20     i = 0;    // Initial index of first subarray
21     j = 0;    // Initial index of second subarray
22     k = left; // Initial index of merged subarray
23     while (i < n1 && j < n2)
24     {
25         if (L[i] ⩽ R[j])
26         {
27             arr[k] = L[i];
28             i++;
29         }
```

```
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

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

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

// Function to perform merge sort
void mergeSort(int arr[], int left, int right)
{
    if (left < right)
    {
        int mid = left + (right - left) / 2;

        // Sort first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

```

SEM 3\Exp9\Radix_sort.c

```
// Function to get the maximum value in an array
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;
}

// Function to perform counting sort based on a specific digit
void countingSort(int arr[], int size, int exp)
```

```
{
    int output[size];
    int count[10] = {0}; // Initialize count array

    // Store the count of occurrences in count[]
    for (int i = 0; i < size; i++)
        count[(arr[i] / exp) % 10]++;

    // Change count[i] so that count[i] contains actual position of this digit in output[]
    for (int i = 1; i < 10; i++)
        count[i] += count[i - 1];

    // Build the output array
    for (int i = size - 1; i >= 0; i--)
    {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    // Copy the output array to arr[], so that arr[] now contains sorted numbers
    for (int i = 0; i < size; i++)
        arr[i] = output[i];
}

// Function to perform radix sort
void radixSort(int arr[], int size)
{
    // Get the maximum number to know the number of digits
    int max = getMax(arr, size);

    // Apply counting sort to sort elements based on each digit
    for (int exp = 1; max / exp > 0; exp *= 10)
        countingSort(arr, size, exp);
}

```

SEM 3\Exp9\counting_sort.c

```
// Function to perform counting sort
void countingSort(int arr[], int size)
{
    int output[size];
    int count[100] = {0}; // Assuming the range of input numbers is known (0-99)

    // Store the count of occurrences
    for (int i = 0; i < size; i++)
        count[arr[i]]++;

    // Build the output array
    for (int i = 0, j = 0; i < 100; i++)
    {
        while (count[i] > 0)
        {
            output[j++] = i;
```

```
            count[i]--;
        }
    }

    // Copy the output array to arr[]
    for (int i = 0; i < size; i++)
        arr[i] = output[i];
}

```

SEM 3\Exp9\main.c

```c
#include <stdio.h>

// Function declarations (you can also include headers for better organization)
void mergeSort(int arr[], int left, int right);
void radixSort(int arr[], int size);
void countingSort(int arr[], int size);
void bucketSort(float arr[], int size);
void heapSort(int arr[], int size);

// Function to display the array (add this in main.c)
void display(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    // Array for testing sorting algorithms
    int arr1[] = {64, 34, 25, 12, 22, 11, 90};
    int size1 = sizeof(arr1) / sizeof(arr1[0]);

    // Merge Sort
    printf("Original array for Merge Sort: ");
    display(arr1, size1);
    mergeSort(arr1, 0, size1 - 1);
    printf("Sorted array using Merge Sort: ");
    display(arr1, size1);

    // Reset the array for next sorting
    int arr2[] = {64, 34, 25, 12, 22, 11, 90};
    printf("\nOriginal array for Radix Sort: ");
    display(arr2, size1);
    radixSort(arr2, size1);
    printf("Sorted array using Radix Sort: ");
    display(arr2, size1);

    // Reset the array for next sorting
    int arr3[] = {64, 34, 25, 12, 22, 11, 90};
    printf("\nOriginal array for Counting Sort: ");
    display(arr3, size1);
```

```
    countingSort(arr3, size1);
    printf("Sorted array using Counting Sort: ");
    display(arr3, size1);

    // Reset the array for next sorting
    float arr4[] = {0.78, 0.17, 0.39, 0.26, 0.72, 0.94, 0.21, 0.12, 0.23};
    int size4 = sizeof(arr4) / sizeof(arr4[0]);
    printf("\nOriginal array for Bucket Sort: ");
    display(arr4, size4);
    bucketSort(arr4, size4);
    printf("Sorted array using Bucket Sort: ");
    display(arr4, size4);

    // Reset the array for next sorting
    int arr5[] = {64, 34, 25, 12, 22, 11, 90};
    int size5 = sizeof(arr5) / sizeof(arr5[0]);
    printf("\nOriginal array for Heap Sort: ");
    display(arr5, size5);
    heapSort(arr5, size5);
    printf("Sorted array using Heap Sort: ");
    display(arr5, size5);

    return 0;
}

```