Folder SEM 3\Exp8

7 printable files

(file list disabled)

SEM 3\Exp8\Makefile

```makefile
# Compiler to use
CC = gcc

# Compiler flags
CFLAGS = -Wall -Wextra -g

# Object files to compile
OBJS = main.o bubble_Sort.o insertion_Sort.o Selection_sort.o quick_sort.o shell_Sort.o

# The final executable name
TARGET = Exp8_sorting_program

# Default target to build the executable
all: $(TARGET)

# Rule to link object files into the final executable
$(TARGET): $(OBJS)
    $(CC) -o $(TARGET) $(OBJS)

# Rule to compile each .c file into a .o file
%.o: %.c
    $(CC) $(CFLAGS) -c $<

# Clean target to remove object files and the executable
clean:
    rm -f $(OBJS) $(TARGET)

```

SEM 3\Exp8\Selection_sort.c

```c
// Function to perform selection sort
void selectionSort(int arr[], int size)
{
    for (int i = 0; i < size - 1; i++)
    {
        int minIndex = i;
        for (int j = i + 1; j < size; j++)
        {
            if (arr[j] < arr[minIndex])
            {
                minIndex = j; // Find the index of the minimum element
            }
        }
        // Swap the found minimum element with the first element
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}

```

SEM 3\Exp8\bubble_Sort.c

```c
#include <stdio.h>

// Function to perform bubble sort
void bubbleSort(int arr[], int size)
{
    for (int i = 0; i < size - 1; i++)
    {
        for (int j = 0; j < size - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                // Swap arr[j] and arr[j + 1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

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

```

SEM 3\Exp8\insertion_Sort.c

```c
// Function to perform insertion sort
void insertionSort(int arr[], int size)
{
    for (int i = 1; i < size; i++)
    {
        int key = arr[i];
        int j = i - 1;

        // Move elements greater than key to one position ahead
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

```

SEM 3\Exp8\main.c

```c
#include <stdio.h>

// Function declarations for sorting algorithms
void bubbleSort(int arr[], int size);
void insertionSort(int arr[], int size);
void selectionSort(int arr[], int size);
void quickSort(int arr[], int low, int high);
void shellSort(int arr[], int size);

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

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

    printf("Original array for Bubble Sort: ");
    display(arr1, size1);
    bubbleSort(arr1, size1);
    printf("Sorted array using Bubble Sort: ");
    display(arr1, size1);

    // Reset the array for next sorting
    int arr2[] = {64, 34, 25, 12, 22, 11, 90};
    int size2 = sizeof(arr2) / sizeof(arr2[0]);

    printf("\nOriginal array for Insertion Sort: ");
    display(arr2, size2);
    insertionSort(arr2, size2);
    printf("Sorted array using Insertion Sort: ");
    display(arr2, size2);

    // Reset the array for next sorting
    int arr3[] = {64, 34, 25, 12, 22, 11, 90};
    int size3 = sizeof(arr3) / sizeof(arr3[0]);

    printf("\nOriginal array for Selection Sort: ");
    display(arr3, size3);
    selectionSort(arr3, size3);
    printf("Sorted array using Selection Sort: ");
    display(arr3, size3);

    // Reset the array for next sorting
    int arr4[] = {64, 34, 25, 12, 22, 11, 90};
    int size4 = sizeof(arr4) / sizeof(arr4[0]);

    printf("\nOriginal array for Quick Sort: ");
    display(arr4, size4);
    quickSort(arr4, 0, size4 - 1);
    printf("Sorted array using Quick 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 Shell Sort: ");
    display(arr5, size5);
    shellSort(arr5, size5);
    printf("Sorted array using Shell Sort: ");
    display(arr5, size5);

    return 0;
}

```

SEM 3\Exp8\quick_sort.c

```c
// Function to perform quick sort
int partition(int arr[], int low, int high)
{
    int pivot = arr[high]; // Choosing the rightmost element as pivot
    int i = (low - 1);     // Index of smaller element

    for (int j = low; j < high; j++)
    {
        // If the current element is smaller than or equal to pivot
        if (arr[j] <= pivot)
        {
            i++; // Increment index of smaller element
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    // Swap the pivot element with the element at i + 1
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1; // Return the partitioning index
}

void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high); // Partitioning index
        quickSort(arr, low, pi - 1);        // Recursively sort elements before partition
        quickSort(arr, pi + 1, high);       // Recursively sort elements after partition
    }
}

```

SEM 3\Exp8\shell_Sort.c

```c
// Function to perform 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;

            // Shift earlier gap-sorted elements up until the correct location for arr[i] is found
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
            {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}

```