Description:

C program that implements Merge Sort using dynamic arrays, prints the array before/after sorting, and measures execution time with clock().

Source Code:

```
#include <stdio.h>
#include <stdlib.h> // Required for malloc and free
#include <time.h>
void merge(int arr[], int left, int mid, int right) {
    int i, j, k;

    // Calculate the size of the two subarrays
    int n1 = mid - left + 1;
    int n2 = right - mid; // The number of elements in the right subarray

    // Create temporary arrays L and R
    // Using malloc to dynamically allocate memory
    int *L = (int *)malloc(n1 * sizeof(int));
    int *R = (int *)malloc(n2 * sizeof(int));

    // Copy data to temp arrays L[] and R[]
    for (i = 0; i < n1; i++) {
    L[i] = arr[left + i];
    }
    for (j = 0; j < n2; j++) {
    R[j] = arr[mid + 1 + j];
```

```
        }

        // Merge the temp arrays back into arr[left..right]
        i = 0; // Initial index of first subarray
        j = 0; // Initial index of second subarray
k = left; // Initial index of merged subarray

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

        // Copy the remaining elements of L[], if any
        while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
        }
// Copy the remaining elements of R[], if any
        while (j < n2) {
```

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

        // Free the dynamically allocated memory
        free(L);
        free(R);
}
void mergeSort(int arr[], int left, int right) {
if (left < right) {
        // Find the middle point
        int mid = left + (right - left) / 2;

        // Recursively sort the first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Merge the sorted halves
        merge(arr, left, mid, right);
        }
}
void printArray(int arr[], int size) {
        int i;
        for (i = 0; i < size; i++) {
        printf("%d ", arr[i]);
```

```
        }
        printf("\n");
}
int main() {
        int arr[] = {12, 11, 13, 5, 6, 7};

        clock_t start, end;
        double cpu_time_used;

        // Calculate the number of elements in the array
        int n = sizeof(arr) / sizeof(arr[0]);

        printf("Array is:\n");
        printArray(arr, n);

        start = clock();

        // Call mergeSort with the initial range (0 to n-1)
        mergeSort(arr, 0, n - 1);

        end = clock();

        printf("\nSorted:\n");
        printArray(arr, n);

        // Calculate time
```

```
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;


    // Print execution time
    printf("\nExecution time: %.6f seconds\n", cpu_time_used);
return 0;
}
```

Output:

```
Array is:
12 11 13 5 6 7

Sorted:
5 6 7 11 12 13

Execution time: 0.000005 seconds
```