1. **Sorting Algorithms: Use a class of sort, that performs different sorting algorithms and determine the time taken for sorting with different values of n.**

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


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

    int temp = *a;

    *a = *b;

    *b = temp;

}


// Quick Sort Algorithm

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

    int pivot = arr[high];

    int i = (low - 1);

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

        if (arr[j] < pivot) {

            i++;

            swap(&arr[i], &arr[j]);

        }

    }

    swap(&arr[i + 1], &arr[high]);

    return (i + 1);

}


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

    if (low < high) {

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

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}
```

```
// Merge Sort Algorithm
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++) {
        L[i] = arr[l + i];
    }
    for (j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
    }
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

// Function to measure sorting time
void measureSortingTime(void (*sortFunc)(int[], int, int), int arr[], int n, const char *sortName) {
    int *arr_copy = (int*)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        arr_copy[i] = arr[i];
    }

    clock_t start, end;
    double cpu_time_used;

    start = clock();
    sortFunc(arr_copy, 0, n - 1);
    end = clock();

    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("%s: %f seconds\n", sortName, cpu_time_used);
    free(arr_copy);
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 10000;
    }

    // Measure sorting times
    measureSortingTime(quickSort, arr, n, "Quick Sort");
    measureSortingTime(mergeSort, arr, n, "Merge Sort");

    return 0;
}
```