7. Quick Sort and Merge Sort (with basic timing hooks)

Description: Implement both sorts; user can use `time` command externally for timing or include `clock()`.

```c
// sorts.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void merge(int arr[], int l, int m, int r){
    int n1=m-l+1, n2=r-m;
    int L[n1], R[n2]; for(int i=0;i<n1;i++) L[i]=arr[l+i]; for(int
j=0;j<n2;j++) R[j]=arr[m+1+j];
    int i=0,j=0,k=l;
    while(i<n1 && j<n2) arr[k++] = (L[i]<=R[j])? L[i++]: R[j++];
    while(i<n1) arr[k++]=L[i++]; while(j<n2) arr[k++]=R[j++];
}
void mergeSort(int arr[], int l, int r){ if(l<r){ int m=(l+r)/2;
mergeSort(arr,l,m); mergeSort(arr,m+1,r); merge(arr,l,m,r);} }

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++; int t=arr[i];
arr[i]=arr[j]; arr[j]=t; }} int t=arr[i+1]; arr[i+1]=arr[high]; arr[high]=t;
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);}
}

int main(){ int n; scanf("%d", &n); int a[n], b[n]; for(int i=0;i<n;i++){
scanf("%d", &a[i]); b[i]=a[i]; }
    clock_t s = clock(); quickSort(a,0,n-1); clock_t e = clock();
printf("QuickSort done. Time: %f seconds\n", (double)(e-s)/CLOCKS_PER_SEC);
    s = clock(); mergeSort(b,0,n-1); e = clock(); printf("MergeSort done.
Time: %f seconds\n", (double)(e-s)/CLOCKS_PER_SEC);
    return 0;
}
```
