```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>

int bubble_sort(int *A, int n)
{
    int i,newn;
    int temp;

    while(1){
        newn = -1;
        for(i=0;i<n-1;i++){
            if(A[i]>A[i+1]){
                temp = A[i]; A[i] = A[i+1]; A[i+1] = temp;
                newn = i;
            }
        }
        if(newn==-1) break;
    }
    return 0;
}

int insertion_sort(int *A, int n)
{
    int i,j;
    int key;

    for(i=1;i<n;i++){
        key = A[i];
        j = i - 1;
        while ((j>=0) && (A[j]>key)){
            A[j+1] = A[j];
            j = j-1;
        }
        A[j+1] = key;
    }
    return 0;
}

int selection_sort(int *A, int n)
{
    int i,j,mini;
    int temp;

    for(i=0;i<n-1;i++){
        mini = i;
        for(j=i+1;j<n;j++){
            if(A[j]<A[mini]){
                mini=j;
            }
        }
        if(mini!=i){
            temp = A[i];
            A[i] = A[mini];
            A[mini] = temp;
        }
    }
    return 0;
}

static int * L;
static int * R;

int merge(int *A, int p, int q, int r)
```

```c
{
    int i,j,k;
//    for(k=p;k<=q;k++){
//        L[k-p] = A[k];
//    }
    memcpy(L,&A[p],(q-p+1)*sizeof(int));
    L[q-p+1] = INT_MAX;
//    for(k=q+1;k<=r;k++){
//        R[k-q-1] = A[k];
//    }
    memcpy(R,&A[q+1],(r-q)*sizeof(int));
    R[r-q] = INT_MAX;
    i = 0;
    j = 0;
    for(k=p;k<=r;k++){
        if(L[i]<=R[j]){
            A[k] = L[i];
            i++;
        }
        else{
            A[k] = R[j];
            j++;
        }
    }

    return 0;
}

int m_sort(int *A, int p, int r)
{
    int q;

    if(p<r){
        q = (p+r)/2;
        m_sort(A,p,q);
        m_sort(A,q+1,r);
        merge(A,p,q,r);
    }
    return 0;
}

int merge_sort(int *A, int n)
{
    L = malloc((n/2+n%2+1)*sizeof(int));
    R = malloc((n/2+1)*sizeof(int));

    m_sort(A,0,n-1);

    free(L);
    free(R);
    return 0;
}

void exchange(int *A, int i, int j)
{
    int temp;

    if(i!=j){
        temp = A[i];
        A[i] = A[j];
        A[j] = temp;
    }
}

int partition(int *A, int p, int r)
{
    int x;
    int i,j;

    x = A[r];
    i = p - 1;
    for(j=p;j<r;j++){
        if(A[j]<=x){
            i++;
            exchange(A,i,j);
        }
    }
    exchange(A,i+1,r);
    return(i+1);
}

int sort(int *A, int p, int r)
{
    int q;
    if(p<r){
        q = partition(A,p,r);
        sort(A,p,q-1);
        sort(A,q+1,r);
    }
    return 0;
}

int quick_sort(int *A, int n)
{

    sort(A,0,n-1);

    return 0;
}

int main()
{
    int i,j,n;

//    int maxn = 6;
//    int A[maxn];
//    A[0] = 5; A[1] = 2; A[2] = 4;
//    A[3] = 6; A[4] = 1; A[5] = 3;
//    clock_t start = clock(); merge_sort(A,maxn); clock_t end = clock();
//    float M_seconds = (float)(end - start) / CLOCKS_PER_SEC;
//    printf("\n T(%7d) = %8.3f\n",maxn,M_seconds);
//    for(i=0;i<maxn;i++) printf("%d ",A[i]);
//    return 0;


    int maxn=160000;
    int A[maxn];
    int B[maxn];

    srand(time(NULL));

    for (i=0;i<maxn;i++){
        A[i] = rand();
        B[i] = A[i];
    }

    n = 10000;
    printf("\n         |     B    |     I    |     S    |     M    |    Q\n");
    printf("---------+----------+----------+----------+----------+-----------\n");
    for(j=1;j<5;j++){
        n = n*2;
        // Bubble sort
        clock_t start = clock(); bubble_sort(A,n); clock_t end = clock();
        float B_seconds = (float)(end - start) / CLOCKS_PER_SEC;
        memcpy(A,B,n*sizeof(int));
        // Insertion sort
        start = clock(); insertion_sort(A,n); end = clock();
        float I_seconds = (float)(end - start) / CLOCKS_PER_SEC;
        memcpy(A,B,n*sizeof(int));
        // Selection sort
        start = clock(); selection_sort(A,n); end = clock();
        float S_seconds = (float)(end - start) / CLOCKS_PER_SEC;
        memcpy(A,B,n*sizeof(int));
        // Merge sort
        start = clock(); merge_sort(A,n); end = clock();
        float M_seconds = (float)(end - start) / CLOCKS_PER_SEC;
        memcpy(A,B,n*sizeof(int));
        // Quick sort
        start = clock(); quick_sort(A,n); end = clock();
        float Q_seconds = (float)(end - start) / CLOCKS_PER_SEC;
        // Scores
        printf(" %7d | %8.3f | %8.3f | %8.3f | %8.3f | %8.3f  \n",n,B_seconds,I_seconds,S_seconds,M_seconds,Q_seconds);
    }

//    for(i=0;i<maxn;i++) printf("%d ",A[i]);
//    printf("\nZoran Ovcin\nmaxn = %d, sortiranje je trajalo %8.5f sekundi.\n",maxn,seconds);

    return 0;
}
```