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

#define LENGTH 10

  /*
    0) make sure you understand well the implementaions of the provided sorting algorithms
    1) compare times of different algorithms using gettimeofday. Change the input length by modifying LENGTH
    2) learn the syntax of qsort. Note that qsort uses function pointers we discussed in Lecture 7
    3) implement merge_sort we saw in class, and compare its running time to other algorithms
  */


// used for appying qsort on array of ints
int cmp_ints (const void * a, const void * b) {
   return ( *(int*)a - *(int*)b );
}

// helper function
void swap(int *x, int *y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

// selection sort
void selection_sort(int* arr, int n) {
    int i, j;
    int min_ind;

    // in the i'th iteration the minimal i elements are in the positions a[0..i-1]
    for (i=0; i<n-1; i++) {
        // find the index of the minimum element in a[i...n-1]
        min_ind=i;
        for (j=i+1; j<n; j++)
          if (arr[j] < arr[min_ind])
            min_ind = j;
        // swap the minimum with arr[i]
        swap(arr+min_ind, arr+i);
    }
}

void insertion_sort(int *arr, int n)
{
  int i, key, j;
  for (i = 1; i < n; i++)
  {
    key = arr[i];
    j = i-1;
```

```
    while (j >= 0 && arr[j] > key)
    {
      arr[j+1] = arr[j];
      j = j-1;
    }
    arr[j+1] = key;
  }
}


void merge(int *A, int *L, int leftcount, int *R, int rightcount)
{
  int i, j, k;
  i = 0;
  j = 0;
  k = 0;

  while(i<leftcount && j<rightcount)
  {
    if(L[i] < R[j])
    {
      A[k++] = L[i++];
    }
    else
    {
      A[k++] = R[j++];
    }
  }
  while (i < leftcount)
  {
    A[k++] = L[i++];
  }
  while (j < rightcount)
  {
    A[k++] = R[j++];
  }
}

// merge sort
void merge_sort(int* arr, int n) {
    // implement me
    int mid, *L, *R;
    if(n<2)
    {
      return;
    }
    mid = n/2;
    L = (int*)malloc(mid*sizeof(int));
    R = (int*)malloc((n-mid)*sizeof(int));

    for(int i = 0; i<mid; i++)
    {
      L[i] = arr[i];
    }

    for(int j = mid; j < n; j++)
    {
      R[j-mid] = arr[j];
```

```
    }

    merge_sort(L, mid);
    merge_sort(R, n-mid);
    merge(arr, L, mid, R, n-mid);

    free(L);
    free(R);
}



void check_sorted(int* ar, int n) {
  int i;

  // check that the array is sorted
  for (i=1; i<LENGTH; i++) {
    printf("%d ", ar[i]);
    if (ar[i-1]>ar[i])
      printf("WRONG");
  }
  printf("done checking...\n");
}


void set_rand(int* ar, int n) {
  srand(0);
  int i;
  for(i=0; i<n; i++)
    ar[i] = rand();
}


int main() {

  int ar1[LENGTH];
  int ar2[LENGTH];
  int ar3[LENGTH];
  int ar4[LENGTH];
  int ar5[LENGTH];

  // create a random array
  set_rand(ar1, LENGTH);
  // copy ar1 into ar2 and ar3 and ar4
  memcpy(ar2, ar1, LENGTH*sizeof(int));
  memcpy(ar3, ar1, LENGTH*sizeof(int));
  memcpy(ar4, ar1, LENGTH*sizeof(int));
  memcpy(ar5, ar1, LENGTH*sizeof(int));

  check_sorted(ar1, LENGTH);



  printf("insertion sort on array of length %d...\n", LENGTH);
  insertion_sort(ar5, LENGTH);
  printf("checking insertion sort...\n");
  check_sorted(ar5, LENGTH);
  printf("\n");
```

```
  printf("selection sort on array of length %d...\n", LENGTH);
  selection_sort(ar2, LENGTH);
  printf("checking selection sort...\n");
  check_sorted(ar2, LENGTH);
  printf("\n");


  printf("qsort on array of length %d...\n", LENGTH);
  qsort(ar3, LENGTH, sizeof(int), cmp_ints);  // the generic sorting algorithm
implemented in C -- requires comparison function
  printf("checking qsort...\n");
  check_sorted(ar3, LENGTH);
  printf("\n");

  printf("merge sort on array of length %d...\n", LENGTH);
  merge_sort(ar4, LENGTH);
  printf("checking merge sort...\n");
  check_sorted(ar4, LENGTH);
  printf("\n");



  return 0;
}
```