```
// ***
// *** You must modify this file
// ***
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "hw09.h"

// DO NOT MODIFY --->>>> From Here
// Do not modify this function. It is used for grading.
void printInput(char * msg, int * arr, int l, int m, int r)
{
  printf("%s(%6d, %6d, %6d)", msg, l, m, r);
  int ind;
  for (ind = l; ind <= r; ind ++)
    {
      printf("%d\n", arr[ind]);
    }
}
// DO NOT MODIFY <<<<--- Until Here

#ifdef TEST_READDATA
// the input file is binary, storing integers
//
// arr stores the address of a pointer for storing the
// address of the allocated memory
//
// size stores the address keeping the size of the array
bool readData(char * filename, int * * arr, int * size)
{
  FILE *fptr; //file pointer
  int sz; //size returned by ftell
  //int numint; //number of integers
  //int val; //placeholder for fread
  int count = 0;

  fptr = fopen(filename, "r");
  if(fptr == NULL)
  {
   return false;
  }

  if(fseek(fptr, 0, SEEK_END) != 0) //check whether fseek fails
  {
   fclose(fptr);
   return false;
  }

  sz = ftell(fptr); //*size ?
```

```
  if(fseek(fptr, 0, SEEK_SET) != 0)
  {
   fclose(fptr);
   return false;
  }

  *size = sz / sizeof(int);

  //ASK ABOUT THIS PART
  int * newArr = malloc(sizeof(int) * (* size)); //allocates memory
for the array

  if (newArr == NULL)
  {
   //free(newArr);
   fclose(fptr);
   return false;
  }

  count = fread(newArr, sizeof(int), *size, fptr);
  if(count != *size)
  {
        free(newArr);
        fclose(fptr);
        return false;
  }

  for (int i = 0; i < *size; i++)
  {
   printf("%d \n", newArr[i]);

  }

  //if(count != *size) //correct? previously had numint which was
equal to sz/sizeof(int)
  //{
   //free(newArr);
   //fclose(fptr);
   //return false;
  //}

  fclose(fptr);

  * arr = newArr; //updates array address

  // use fopen to open the file for read
  // return false if fopen fails

  // use fseek to go to the end of the file
```

```
  // check whether fseek fails
  // if fseek fails, fclose and return false

  // use ftell to determine the size of the file

  // use fseek to go back to the beginning of the file
  // check whether fseek fails

  // if fseek fails, fclose and return false

  // the number of integers is the file's size divided by
  // size of int

  // allocate memory for the array
  // if malloc fails, fclose and return false

  // use fread to read the number of integers in the file

  // if fread does not read the correct number
  // release allocated memory
  // fclose
  // return false

  // if fread succeeds
  // close the file

  // update the argument for the array address

  // update the size of the array
  return true;
}
#endif

#ifdef TEST_WRITEDATA
// the output file is binary, storing sorted integers
// write the array of integers to a file
// must use fwrite. must not use fprintf
bool writeData(char * filename, const int * arr, int size)
{
  // fopen for write
  FILE *fptr;
  fptr = fopen(filename, "w");
  if(fptr == NULL)
  {
   return false;
  }

  int count = fwrite(arr, sizeof(int), size, fptr);

  if (count != size)
```

```
  {
   fclose(fptr);
   return false;
  }

  fclose(fptr);

  // if fopen fails, return false

  // use fwrite to write the entire array to a file

  // check whether all elements of the array have been written

  // fclose

  // if not all elements have been written, return false

  // if all elements have been written, return true

return true;
}
#endif


#ifdef TEST_MERGE
// input: arr is an array and its two parts arr[l..m] and arr[m+1..r]
// are already sorted
//
// output: arr is an array and all elements in arr[l..r] are sorted
//
// l, m, r mean left, middle, and right respectively
//
// You can assume that l <= m <= r.
//
// Do not worry about the elements in arr[0..l-1] or arr[r+1..]

static void merge(int * arr, int l, int m, int r)
// a static function can be called within this file only
// a static function is invisible to other files
{
  // at the beginning of the function
#ifdef DEBUG
  // Do not modify this part between #ifdef DEBUG and #endif
  // This part is used for grading.
  printInput("Merge in", arr, l, m, r);
#endif

  int i; //index reference of first subarray
  int j; //index reference of second subarray
  int k; //index reference of merged subarray
```

```
int leftElem = m - l + 1; //number of elements in left array
int rightElem = r - m; //number of elements in right array
// if one or both of the arrays are empty, do nothing

i = 0;
j = 0;
k = l;

//if(leftElem != 0 && rightElem != 0)
//{
 int * leftArray = malloc(sizeof(* leftArray) * leftElem);
 int * rightArray = malloc(sizeof(* rightArray) * rightElem);

 for(i = 0; i < leftElem; i++)
 {
  leftArray[i] = arr[l + i];
 }
 for(j = 0; j < rightElem; j++)
 {
  rightArray[j] = arr[m + 1 + j];
 }

 i = 0;
 j = 0;
 while( i < leftElem && j < rightElem)
 {
  if(leftArray[i] <= rightArray[j])
  {
   arr[k] = leftArray[i];
   i++;
  } //if
  else
  {
   arr[k] = rightArray[j];
   j++;
  } //else
  k++;
 } //while

 while(i < leftElem)
 {
  arr[k] = leftArray[i];
  i++;
  k++;
 }
 while(j < rightElem)
 {
  arr[k] = rightArray[j];
  j++;
```

```
    k++;
   }
  //} //if
  free(leftArray);
  free(rightArray);

  // Hint: you may consider to allocate memory here.
  // Allocating additiional memory makes this function easier to write

  // merge the two parts (each part is already sorted) of the array
  // into one sorted array

  // the following should be at the bottom of the function
#ifdef DEBUG
  // Do not modify this part between #ifdef DEBUG and #endif
  // This part is used for grading.
  printInput("Merge out", arr, l, m, r);
#endif
}
#endif

// merge sort has the following steps:

// 1. if the input array has one or no element, it is already sorted
// 2. break the input array into two arrays. Their sizes are the same,
//    or differ by one
// 3. send each array to the mergeSort function until the input array
//    is small enough (one or no element)
// 4. sort the two arrays
#ifdef TEST_MERGESSORT
void mergeSort(int arr[], int l, int r)
{
  // at the beginning of the function
#ifdef DEBUG
  // Do not modify this part between #ifdef DEBUG and #endif
  // This part is used for grading.
  printInput("mergeSort", arr, l, r, -1);
#endif

  // if the array has no or one element, do nothing
  if (l < r)
  {
   int middle = (l+ r) / 2;
   mergeSort(arr, l, middle);
   mergeSort(arr, middle + 1, r);

   merge(arr, l, middle, r);
  }

  // divide the array into two arrays
```

```
  // call mergeSort with each array
  // merge the two arrays into one

}
#endif
```