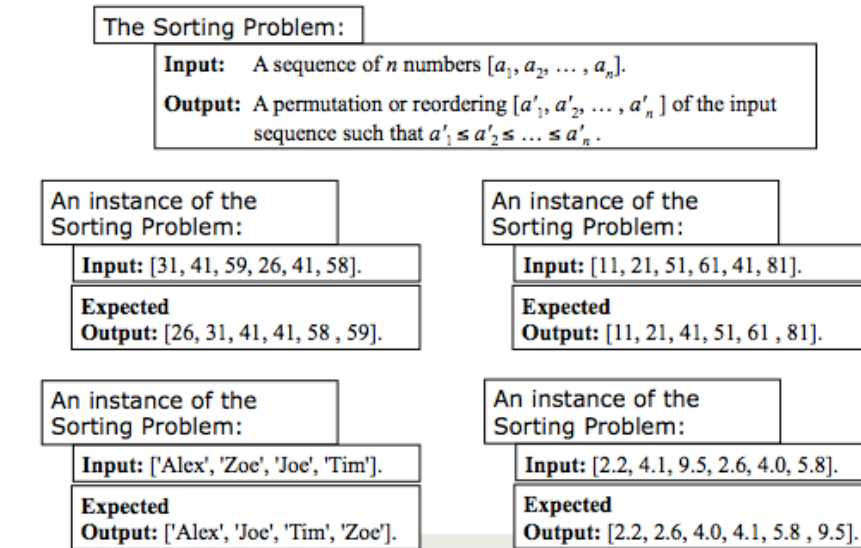


Sorting

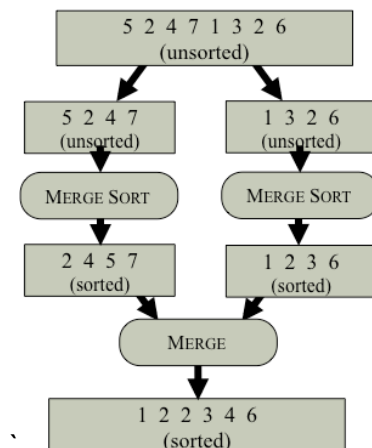
In this assignment you will implement a second solution to the sorting problem. The problem is formally specified in your textbook as follows:



Merge Sort

Merge Sort is an algorithm that solves the sorting problem. It closely follows the divide-and-conquer approach: break the problem into several subproblems that are similar to the original problem but smaller in size, solve the subproblems recursively, and then combine these solutions to create a solution to the original problem.

Merge Sort operates as follows:



Divide: Divide the n -element into two subsequences of $n/2$ elements each.

Conquer: Sort the two subsequences recursively using merge sort.

Combine: Merge the two sorted subsequences to produce the sorted answer.

EXERCISE 1. The figure above illustrates the operation of the Merge Sort algorithm on the input sequence [5 2 4 7 1 3 2 6]. Provide a similar figure for the operation of Merge Sort on each of the two subsequences in the figure.

The Algorithm

MERGE-SORT(A, p, r)

```
1  if  $p < r$ 
2       $q = \lfloor (p + r) / 2 \rfloor$ 
3      MERGE-SORT( $A, p, q$ )
4      MERGE-SORT( $A, q + 1, r$ )
5      MERGE( $A, p, q, r$ )
```

MERGE Procedure

MERGE(A, p, q, r)

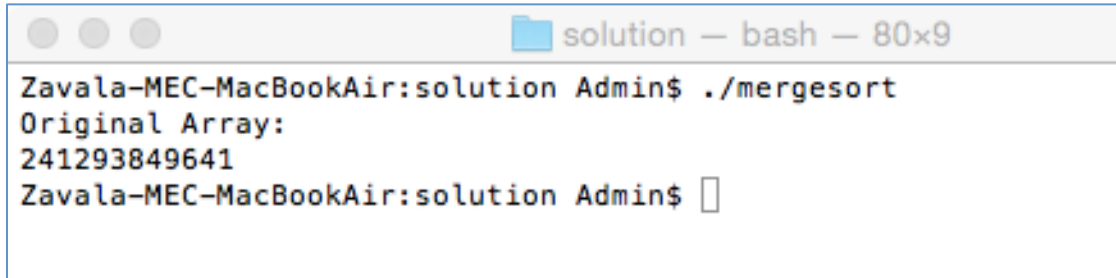
```
1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays
4  for  $i = 1$  to  $n_1$ 
5       $L[i] = A[p + i - 1]$ 
6  for  $j = 1$  to  $n_2$ 
7       $R[j] = A[q + j]$ 
8   $L[n_1 + 1] = \infty$ 
9   $R[n_2 + 1] = \infty$ 
10  $i = 1$ 
11  $j = 1$ 
12 for  $k = p$  to  $r$ 
13     if  $L[i] \leq R[j]$ 
14          $A[k] = L[i]$ 
15          $i = i + 1$ 
16     else  $A[k] = R[j]$ 
17          $j = j + 1$ 
```

The Starter Code

Download the starter code for this project (*mergesort.cpp*) and take a look at the code. You should notice that the `mergeSort` function is empty. You will write the code for it later on. First, you will complete other tasks.

TASK 1. Write the code for the `printArray` function. The function takes an array and the size of the array as arguments. The function should print to screen the contents of the array it receives as argument. The contents of the array should be printed on a single line.

Run the program. You should see the contents of array `A` (which is created in the first line of the `main` function) displayed to screen:

A screenshot of a terminal window titled "solution — bash — 80x9". The prompt is "Zavala-MEC-MacBookAir:solution Admin\$". The user has entered the command `./mergesort`. The output is "Original Array:" followed by the array elements "241293849641" on the next line. The prompt is now "Zavala-MEC-MacBookAir:solution Admin\$ " with a cursor.

```

Zavala-MEC-MacBookAir:solution Admin$ ./mergesort
Original Array:
241293849641
Zavala-MEC-MacBookAir:solution Admin$ 
```

Run the program. You should see the original array displayed to screen.

QUESTION

1. In a sentence describe what you think the tests in the first block of comments are supposed to do.

TASK 2. Write the code for the `isSorted` function. It should return *true* if the items in the array it receives as argument are sorted (in ascending order), and *false* otherwise.

Uncomment the first block of comments in the `main` function and run the program. The `isSorted` function should pass the test (you should not see any error message displayed).

QUESTIONS

Before you write the `mergeSort` function, you need to understand the *merge* function provided. The function implements the pseudocode of the MERGE procedure given above (page 31 of your textbook). Answer the following questions:

2. Other than differences in the names of the left and right arrays and some indexes (arrays start at 0 in C++ and the textbook starts them at 1), what significant differences do you notice between the MERGE pseudocode in the book and the `merge` function provided with the starter code?
3. In the pseudocode, what is the purpose of putting *sentinels* at the ends of the arrays `L` and `R` (lines 8-9)?
4. Why aren't lines 8-9 of the MERGE pseudocode implemented in the `merge` function?
5. Why is a while loop used in the `merge` function instead of the for loop in lines 12-17 of the MERGE pseudocode?
6. Why are there two extra while loops at the end of the `merge` function? And why aren't they in the MERGE pseudocode?

TASK 3. For task 3, you must make use of the `merge` function to sort array `A` manually. That is, you must make as many calls to the `merge` function as needed to have the array sorted. Make sure you write the calls before the second block of comments. Do not use any loops.

Uncomment the second block of comments in the `main` function and run the program. Your program should pass the tests (you should not see any error message displayed).

TASK 4. Write the body of the `mergeSort` function. The function must implement the *MERGE-SORT* algorithm given above (page 34 of your textbook). Make use of the `merge` function.

Uncomment the third block of comments in the `main` function and run the program. Your program should pass all tests (you should not see any error message displayed).

TASK 5. Create an array with 10 numbers of your choice in random order and call it *arrayLatName*, where you will substitute *LastName* for YOUR last name. For example, my array would be called `arrayZavala`. Then, call the `mergeSort` function to sort the items in your array. Next, use `assert` to test that the array is now sorted (using the `isSorted` function). Finally, you should print your sorted array (using the `printArray` function).