Wesleyan University, Fall 2025, COMP 211

Homework 6: Arrays and Sorting

Due by 11:59pm on October 28, 2025

---

1. Written problems (5 points)

Problem 1. *For each of the the three coding problems, explain (1) the cost of your implementation and (2) the amount of extra memory used (in addition to the array given to the function). Simply giving the big-Oh notation for the algorithms you implemented is not sufficient although necessary to do: you must also explain how you arrived at whatever cost you did.*

Solution:

Palindrome:

- $O(n)$ **time:** we loop for as long as $left < right$. Since $left$ is initially 0 and $right$ is the string length $n-1$, and each iteration we increment $left$ by 1 and decrement $right$ by 1, then we will loop $n/2 \in O(n)$ times.
- $O(1)$ **space:** since only two integer variables are used, the extra memory used is $O(1)$.

Insertion sort:

- $O(n^2)$ **time:** the outer loop loops n times, while the inner loop may have to shift up to n elements. Hence, the total number of times the operations in the inner loop may need to be executed is $O(n^2)$.
- $O(1)$ **space:** since only a constant number of extra integer variables is used, the extra memory used is $O(1)$.

3-way mergesort:

- $O(n \log n)$ **time:** Roughly, the first recursion will divide the array into 3 subarrays each of size n/3. The second recursion will divide the array into 9 subarrays each of size n/9, and so on. The number of times `merge_sort3` will recurse is $\log_3 n$. Now for each recursion, n amount work will need to be done, to merge back together the subarrays. Since the logarithm base is just a constant, we don't need to write the base in the big-Oh notation.
- $O(n)$ **space:** since in order to do the merge an array of up to length n must be used to temporarily hold the merged subarrays.

2. Coding problems (15 points)

Problem 2. *Write a function* `is_palindrome` *that satisfies the following specification:*

- **Function header.** `bool is_palindrome(char[])`
- **Pre-condition.** *s is null-terminated.*

- **Function body.** *This should satisfy the following:*

$$\texttt{is\_palindrome}(s) = \begin{cases} \texttt{true} & \text{if } s[i] = s[n-1-i] \text{ for all } i < (n-1-i), \text{ where } n = \texttt{strlen}(s) \\ \texttt{false} & \text{otherwise.} \end{cases}$$

In other words, the function you will implement is an algorithm to check whether the string s is a palindrome. A palindrome is a string that is the same whether written forwards or backwards. For instance, racecar is a palindrome. Don't forget about edge cases: the empty string, and all length-1 strings, are palindromes.

You will get substantial credit for a reasonable and correct implementation. But for full credit, your implementation must have $O(n)$ cost, and any extra space required should be independent of the size of the string argument.

PROBLEM 3. *For this problem you will implement the insertion sort algorithm. Conceptually, insertion sort starts with an empty sorted list of length 0, and then successively inserts values into their correct place, growing the sorted list length by 1 with each insertion. In more detail:*

> **Algorithms in C, by Sedgewick:** *Insertion sort "consider[s] the elements one at a time, inserting each in its proper place among those already considered (keeping them sorted). In a computer implementation we need to make space for the element being inserted by moving larger elements one position to the right, and then inserting the element into the vacated position. As in selection sort, the elements to the left of the current index are in sorted order during the sort, but they are not in their final position, as they may have to be moved to make room for smaller elements encountered later."*

Write a function `insertion_sort` *with the following signature.*

```
void insertion_sort(int A[], int n)
```

See Section 6.3 in `Algorithms in C` *for more details on insertion sort and ideas about simplifying your code. However, you should develop your implementation without referring back to the insertion sort code given in the book.*

PROBLEM 4. *In class, we saw a recursive implementation of the mergesort algorithm that divides the input array into halves, recursively sorts each half, and then merges the results. In this problem you will write a three-way recursive mergesort algorithm that divides the input array into thirds, recursively sorts each third, and then merges the results. Specifically, write a function* `merge_sort3` *with the following signature:*

```
void merge_sort3(int A[], int lo, int hi)
```

The stubs for this function as well as the `merge` *function are given to you in* `hw6.c`.

3. Code distribution

This assignment comes with a *code distribution*, comprising files that you will need to complete the programming problems for this assignment:

- `hw6.h`: header file for the code you will write. This file declares the functions that you must implement. Do not change the contents of this file; if it appears to be causing problems with compilation, the problem is with your solution.
- `hw6.c`: Function stubs matching hw6.h have been implemented for you to get you started.
- `comp211.h`: a header file that defines `dotest`, which is used in `tests.c`.
- `tests.c`: a small testing program. This program provides just a few tests. You should certainly add more. To compile the testing program, use the command

  ```
  gcc --std=c99 -o tests tests.c hw6.c
  ```

- `driver.c`: a small driver program. This program provides a simple interactive program that uses your `unimodal_search_r` function. You may modify it however you like. To compile the driver program, use the command

  ```
  gcc --std=c99 -o driver driver.c hw6.c
  ```

- `Makefile`: a Makefile for this assignment. Instead of using the above commands for compiling your code, you can use the commands `make tests` and `make driver`.

4. Submission

Submit your written work as `hw6.pdf` and your code as `hw6.c` to the Google Drive directory I have created for you named `comp211-f25-USERNAME/hw6/`. You should replace `USERNAME` with your Wesleyan username.

Do not forget that your written work must be submitted as a PDF! And make sure that at the top of each file you have put your name! Do not, however, change the names of the files.

You do *not* submit `hw6.h` or any test or driver programs. When we test your code, we will add in our copy of `hw6.h` and our own testing program. In particular, if you change `hw6.h` in order to make your code compile, then your code will probably fail to compile with our `hw6.h`, and hence you will receive little to no credit for the coding portion of this assignment.