**Exercises - Introduction**

Answer the following questions by implementing the code samples and/or answering the questions in a word document. Upload all project files and word documents zipped to the exercise dropbox.

1. Write a program to solve the selection problem, for k = \frac{n}{2}n2, as described on page one of your textbook.   
*Note: This can be solved in as little as two lines of code.*

 See project code for solution 1

2a. The fibonacci sequence is a series of numbers with the following pattern: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... Notice that each term after the first two terms is the sum of the last two terms visited in the sequence. Each of the numbers in the sequence is given an index, starting at zero. So fib(0) = 0, fib(1) = 1, fib(2) = 1, ...

This sequence shows up nature and science in surprising areas, including data structure design! You can read more about the sequence here ([wiki article (Links to an external site.)](https://en.wikipedia.org/wiki/Fibonacci_number)). Write a function that returns an array of the first n fibonacci numbers. (ie. the input to the function should be an integer n, and the output should be an array of integers of size n that contains the first n numbers in the fibonacci sequence).

See project code for solution 2a

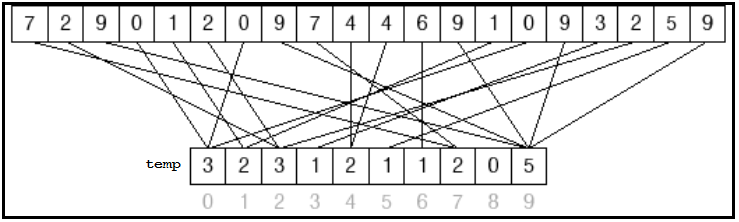
2b. Write a function that returns the nth fibonacci number (ie. fib(n)). Your function should use iteration and not recursion. We will discuss later how recursion can be a pitfall here. You should also use no more than a few temporary variables. The solution you provide should not use arrays of any other data structure.

See project code for solution 2b

3. Is it possible to use your solution to (2a) to solve (2b)? If this is not possible, make an argument why. If it is possible, then write the function and argue about why this is a good or bad design.

Yes, I easily combined 2a and 2b, but the solution is less efficient than directly accessing the elements of array (2a solution). However, 2b uses less memory as it finds larger numbers because its only storing the previous two Fibonacci numbers, not all. The downside to the 2b solution is that it must start from the beginning each time it looking for another Fibonacci number, so combining 2a and 2b solution means its looping through the same numbers every time. 2a only loops through number once. 2b is strictly designed to find one Fibonacci number. 2b is designed so that it can pick-up at a specific set of Fibonacci numbers, but an additional interface has not been added to support it.

4. Writing algorithms is all about finding strategies for solving a problem. Suppose you were given the problem of sorting an array of n random integers. One strategy for sorting is described below:



* Assume that each of the random integers is within the range 0 to k, where k is a positive integer. (ie. you could assume the array only has integers from 1-20)
* Create a temporary array of size k + 1. Set temp[index] = 0 for each element in the array.
* Loop over each element in your input array. Let **m** be the current element. Increment temp[**m**] by one. For example, if we see **m** three times in our input array during this loop then temp[**m**] will be 3. This process is illustrated below:
* Lastly, loop over your temporary array to print your results in sorted order. For each **m** = temp[*index*], print *index* **m** times. (ie. if temp[2] = 4, then print 2, 2, 2, 2 as part of your sorted output).

Write a function that follows the steps above. Test your function with a few unsorted input arrays.

See project code for solution 4