Project 2: Practicing Recursive Algorithms

Review the Fibonacci sequence and the implementation we discussed in class. Recursive functions are self-referential, which is to say the function calls *itself* until reaching a stopping condition. When the stopping condition is reached, the recursion stops, and the results from each function call are returned. While recursion can take some time to get used to, many problems like the problem of finding the nth Fibonacci number can be expressed elegantly using a recursive function. An understanding of recursion is also essential for understanding many common searching and sorting algorithms, such as the mergesort, quicksort, and binary search algorithms we reviewed in the last lecture.

Submission:
Because a large part of this project is focused on a conceptual understanding of recursion and the recursive algorithms we have covered, please submit your work in either a text file or a PDF document. Submit your work on iLearn.

Part 1:
Write a recursive function which prints every power of 2 from 1 to N, where N is a parameter of the function. Use the function signature below and do not define any other functions. This can be done with a single recursive function.

```
    int powers(int N);
```

Example usage:

```
    powers(128);
```

Example Output:

```
    2, 4, 8, 16, 32, 64, 128
```

Part 2:
Explain in your own words how the **insertion sort** algorithm works. The implementation we reviewed in class can be found on iLearn along with the other code covered in last week's lecture.

Part 3:
Given the following array of integers, walk through the **quicksort** algorithm by hand. The algorithm works by modifying the original array of numbers multiple times as the quicksort function is called recursively. Show the contents of the array at each stage of the sorting process, the same as we did on the board in class.

Array to be sorted: [5, 8, 9, 3, 5, 4, 1]

Example on a smaller array:
Consider the shorter array [4, 3, 2, 1]

The algorithm works by swapping the position of two numbers at a time. After the first swap, the array becomes:
[1, 3, 2, 4]

The program continues to selectively rearrange the numbers in the array until it is sorted. In this example the algorithm only needs a few iterations to fully sort the array:
[1, 2, 3, 4]