Module 2

Searching and Algorithm Analysis

- ▼ What kind of array does a binary search algorithm need?

  sorted array

- ▼ What is the complexity of a binary search algorithm?
  - logarithmic O(log n)
  - faster than linear O(n)
- ▼ How does binary search work?
  - The first iteration of this algorithm tests the middle element in the array. If this matches the search key, the algorithm ends.
  - Assuming the array is sorted in ascending order, and then if the search key is less than the middle element, it cannot match any element in the second half of the array and the algorithm continues with only the first half of the array.
  - If the search key is greater than the middle element, it cannot match any element in the first half of the array and the algorithm continues with only the second half.
  - Each iteration tests the middle value of the remaining portion of the array. If the search key does not match the element, the algorithm eliminates half of the remaining elements.
  - The algorithm ends by either finding an element that matches the search key or reducing the subarray to zero size.
- ▼ How does linear search work?

- The algorithm searches each element in an array sequentially.
- If the search key does not match an element in the array, the algorithm informs the user that the search key is not present.
- If the search key is in the array, the algorithm tests each element until it finds one that matches the search key and returns the index of that element.

▼ What is Big O notation?

it indicates how hard an algorithm may have to work to solve a problem

▼ What is O(1)?

- constant run time
- “order one”
- means the number of comparisons is constant—it does not grow as the size of the array increases

▼ What is O(n)?

- linear run time
- “order n”
- an algorithm that requires a total of *n* – 1 comparisons
- If the size of the array is *doubled*, the number of comparisons that the algorithm must perform is also *doubled.*

▼ What is O(n^2)?

- quadratic run time
- “order n-squared”
- When *n* is small, quadratic algorithms will not noticeably affect performance, but as *n*
  grows, you'll start to notice performance degradation.

▼ What are recursive algorithms?

- Recursive algorithms are the ones that make calls to themselves until a certain condition is satisfied.

- **A recursive algorithm breaks down a problem into smaller subproblems and applies the algorithm itself to solve the smaller subproblems.**
- Types: single and multiple recursion, indirect recursion, anonymous recursion, and generative recursion

▼ Recursive example: Fiboannaci

- Sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, . . .
- Notice after the second number, each number in the series is the sum of the two previous numbers.
- A recursive function to calculate the nth number in the Fibonacci series:

```
if n == 0: # base case
  return 0
elif n == 1: # base case
  return 1
 else:
   return fib(n − 1) + fib(n − 2)
```

▼ What are some reasons not to use recursion?

- Recursive function calls are certainly less efficient than loops. Each time a function is called, the system incurs overhead that is not necessary with a loop.
- In many cases, a solution using a loop is more evident than a recursive solution.