Tutorial 9

- Efficiency
- Efficiency in recursion
- Monster Hunting
Efficiency

- When looking at a function, it is often useful to understand its running time.
- To do this, we compute the running time as a function of the input size, and use Big-O notation to simplify the computation.
Understanding O()

- Big-O is useful because it gives a broad idea of how fast an algorithm runs.

- Three specific runtimes we will review in this tutorial are linear ($O(n)$), constant ($O(1)$), and logarithmic ($O(\log n)$).

- It is important to remember that we analyze each function in terms of its own parameters.

- Many array functions are analyzed in terms of the array’s length, but there can be other parameters to the runtime.
Linear Time

- We say an algorithm runs in linear time when its runtime is $O(n)$

- These algorithms use a number of operations which can be bounded by $c \cdot n$ for some constant $c$

- Doubling the input size $n$ also doubles the runtime of the algorithm

- e.g. An algorithm that squares and prints every element in an array
Constant Time

- Constant time algorithms ($O(1)$) perform a constant number of operations regardless of the input size
- Doubling the input size $n$ does not change the runtime of the algorithm
- e.g. A function at prints and squares the first 10 elements in an array
Logarithmic Time

- The last type of algorithm we’ll discuss in this tutorial is logarithmic ($O(\log n)$)

- Usually, these are algorithms that divide the input at every step

- Doubling the input size $n$ increases the runtime of the algorithm by $\log 2$

- Squaring the input size $n$ doubles the runtime of the algorithm
Efficiency in Recursion

Steps for recursive functions:

1. Identify the runtime of the function excluding any recursion

2. Determine the size of the data for the next recursive call(s)

3. Write the full recurrence relation (combine step 1 & 2)

4. Look up the closed-form solution in a table (Section 8, Slide 37)
Monster Hunting

• We will be completing the Monster Hunter module on Seashell.