CS251 Midterm 1 - Fall 2022

schari

September 2022

1 Summations and Logarithm Rules

- Summations
  - Given c is a constant, $\sum_{i=m}^{n} c = c(n - m + 1)$
  - $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$
  - $\sum_{i=1}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1)$
  - Given a function $f(i)$, $\sum_{i=m}^{n} f(i) = \sum_{i=1}^{n} f(i) - \sum_{i=1}^{m-1} f(i)$
- Log Rules
  - In CS 251, if you are just given a $\log(n)$ without a base, they probably mean $\log_2(n)$
  - $\log(ab) = \log(a) + \log(b)$
  - $\log(\frac{a}{b}) = \log(a) - \log(b)$
  - Given 2 numbers a and b, $\log_a(n) = \frac{\log_b(n)}{\log_b(a)}$
  - $\log(n^a) = a\log(n)$
  - $a^{\log_a(n)} = n$
  - $a^{b\log_a(n)} = n^b$

2 Experimental Analysis

- Limitations
  - Different machines can vary the run time
  - other processes/noise
  - May not be precise all the time

3 Recursive Functions

- Functions that call themselves in order to solve simpler problems
- Recursive functions don't call themselves infinitely; eventually stop when they reach a base case
-

4 Runtime Analysis

- Represents the efficiency of an algorithm
- Three types of asymptotic runtime analysis: $O(n)$, $\Omega(n)$, and $\Theta(n)$
- $O(n)$
  - The asymptotic upper bound
  - Definition: Given functions $f(n)$ and $g(n)$, then $f(n) \in O(g(n))$ if there exists constants c and n_0 where $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$
  - In other words, $f(n) \in O(g(n))$ if $f(n)$ doesn't grow faster than $g(n)$.
  - Growth order:
    - $O(1) < O(\log(n)) < O(n) < O(n \log(n)) < O(n^2) < O(n^3) < O(2^n) < O(n!)$
  - Going from the definition above, multiple functions can be big-O of another function.
    - Ex: $n \in O(n)$, and $n \in O(n^2)$
  - Generally, if they're asking for big-O of a function, you want to give the tightest bound of the function.
  - When giving the $O(n)$ of a function, you take the fastest-growing term and remove the constants from it
    - Ex: $4n^2 + 2n \log(n) + 3n + 123456 \in O(n^2)$
- $\Omega(n)$
  - Asymptotic lower bound
  - Definition: Given functions $f(n)$ and $g(n)$, then $f(n) \in \Omega(g(n))$ if there exists constants c and n_0 where $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$
  - In other words, $f(n) \in \Omega(g(n))$ if $f(n)$ doesn't grow slower than $g(n)$.
  - Like how multiple functions can be $O(n)$ of a function, multiple functions can also be $\Omega(n)$ of a function
    - Ex: $n \in \Omega(n)$, and $n \in \Omega(\log(n))$
  - Again, generally you should give the tightest bound

  - Process for getting big-Ω of a function is same as getting big-O

- $\Theta(n)$
  - Asymptotic tightest bound of a function
  - $f(n) \in \Theta(g(n))$ if $f(n)$ doesn't grow faster or slower than $g(n)$

- Asymptotic Growth Properties
  - If $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$, then $f(n) \in \Omega(g(n))$ and vice versa
  - If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$
  - If $f(n) \in \Omega(g(n))$ and $g(n) \in \Omega(h(n))$, then $f(n) \in \Omega(h(n))$
  - If $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n))$, then $f(n) \in \Theta(h(n))$
  - If $f(n) \in O(h(n))$ and $g(n) \in O(h(n))$, then $f(n) + g(n) \in O(h(n))$
  - If $f(n) \in \Theta(g(n))$, then $f(n) + g(n) \in \Theta(g(n))$
  - If $f(n) \in O(g(n))$, then $g(n) \in \Omega(f(n))$

5 Arrays and LinkedLists

6 Stacks

- Data structure to store and remove data
- Last data pushed into the stack would be the first data popped off (LIFO)
  - Think of it like a stack of plates; the last plate placed on top is the first plate taken from the stack
- Standard methods for stacks:
  - `push()` - Add an element to the top of the stack
  - `pop()` - Remove the element from the top of the stack
  - `isEmpty()` - Whether or not there are elements on the stock
  - `size()` - Number of elements on the stack
  - `peek()` - View the element at the top of the stack without removing it
- Implementation using Arrays vs LinkedLists
  - Arrays: Lower memory overhead; unable to resize to accomodate more elements
  - LinkedLists: Pointers require more memory; can expand to increase number of elements in the stack

7 Queues

- Data structure to store and remove data
- First data enqueued would be the first data dequeued (FIFO)
  - Think of it as a queue of people; first person to enter is also the first person who gets served
- Standard methods for queues:
  - `enqueue()` - Add an element to the end of the queue
  - `dequeue()` - Remove the element from the front of the queue
  - `isEmpty()` - Whether or not there are elements in the queue
  - `size()` - Number of elements in the queue
  - `peek()` - View the element at the front of the queue without removing it
- Implementation using Arrays vs LinkedLists
  - Arrays: In addition to lower memory overhead, a queue implementation will loop from the end of array to the front, and will keep track of the index of the front and back of the queue (see slides for example).
  - LinkedLists: Because LinkedLists are relatively resizable, you don't need to keep track of indices, but you will have to keep track of the front and back of the queue nodes.

8 Trees