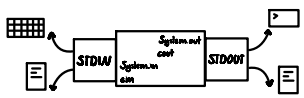
Announcements

Assignment #1 ⇒ Due 02/08/23

- Beware to test your program, Steve's test cases are given after the assignment
  - **Test edge cases** ⇒ No input should be tested
  - **Test a few lines of input** ⇒ Just type in the terminal
  - **Test with a million lines of input** ⇒ Generate millions of lines and output to STDOUT with redirection



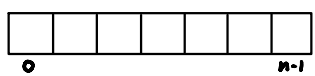
`./Foo > file.txt` or `java Foo > file.txt`. Then redirect the created file to be the input `./Bar < file.txt` or `java Bar < file.txt`. Use `tac` to display the input file in reverse, then `diff` to check for differences

- Extra functionality is okay (using a template, iterable, more functions, etc)

Algorithm (program) Complexity

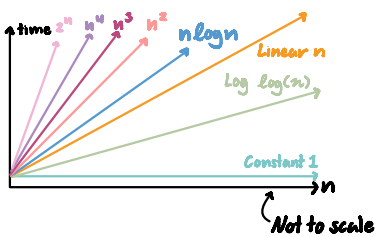
Space ⇒ How much memory do we need to run the algorithm (excluding the input array)

- Imagine an array-like input for a algorithm of **size n**
- **Constant/inplace algorithms** ⇒ Do not use (really) any extra memory
- **Not in place** ⇒ Uses extra space, an example type is linear



Time ⇒ How much work does algorythm do in relation to n

- Imagine an array-like input for an algorithm of size n
- Pick the key operation of the algorithm, "what is inside the loop"
- Time complexity is important for when the data is large
- We ignore coefficents and smaller powers, $2n$ and n^2+n become n^2



Big O notation ⇒ Denotes time complexity as the upper bound (at the time stated or less)

- There are other ways of quantifying time complexity, but big O is most commonly used
- O(time complexity) ⇒ O(1), O(n), O(n lgn)...
- Big O is independant of the speed of the computer the program is being ran on
- O(1) ⇒ Insert into a set
- O(n) ⇒ Linear search, find max
- O(lgn) ⇒ Binary search
- O(n lgn) ⇒ Merge sort
- $O(n^2)$ ⇒ Selection sort, bubble sort, incertion sort *

*Very efficent for small collections

- Use the `time` command to time how long it takes for something to execute `time ./Foo` or `time java Foo`. Test the program against itself with larger and larger data sets and see how the time changes

Priority Queue

- In a queue elements are added to one end and elements are removed from the other
- In a priority queue, elements are organized by priority
- Elements can be promoted to higher priority, but never demoted
- A linked list or array are no longer efficent for this. Instead we will use a heap (which is built on an array)
- Elements can be added anywhere, but are only removed from the front

Heap

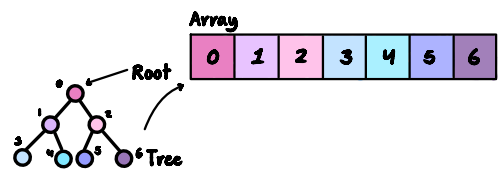
- A heap is a nearly complete binary tree that respects the heap property

Heap property $\Rightarrow A[\text{parent}(i)] \geq A[i]$

- The value of a node is, at most, the value of its parent

Binary tree

- Complete $\Rightarrow$ Every level is full
- Nearly complete $\Rightarrow$ Complete on every level except the last, where the last level is contiguous



- Turn a binary tree into an array
- We can represent the heap as a binary tree because it is a nearly complete binary tree
- In a binary tree which is collapsed into an array, for node i, the parent of i is at $(i-1)/2$, the left node is at $2i+1$, and the right node is at $2i+2$

Height of a node in a tree is the number of edges in the longest path from a leaf to the root

↳ Node with no children