Big O is by far the most common in the real world. There are other variants that I have not mentioned. They come up from time to time, but they're outside the scope of this module.

<u>Exercise.</u> Prove or disprove the following.

1. $n! = O((n+1)!)$
2. $(n+1)! = O(n!)$
3. $n! = \Theta((n+1)!)$

<u>Exercise.</u> Show that

$$\sum_{i=1}^{n} i^6 = \Theta(n^7).$$

Back to Search.

We said that SimpleSearch is $O(n)$ and BinarySearch is $O(\log n)$. I hope this is now clarified after our detour learning big O notation.

Since $\log n = O(n)$ and $n \neq O(\log n)$, we say that BinarySearch is more efficient.

<u>Be careful:</u> Efficiency does not always mean faster. *Eventually* it does, but this kind of analysis is blind to how large inputs need to grow to see a tangible difference in runtimes.

For example, an algorithm might use exactly

$$2^{2^{2^{2^{2}}}} n$$ ~~[illegible]~~ operations

and another $2n^2$ operations. The former is $O(n)$ and the latter $O(n^2)$. For small-sized inputs the second likely is faster, but eventually (as the size of the input increases) the first will out-perform.

<u>Real-world example:</u>

Matrix multiplication is such an important computational problem. Assuming matrices are $n \times n$, the naive alg. (what we do ~~[illegible]~~ by hand) uses $O(n^3)$ operations. The current best complexity is

$$O(n^{2.371552}).$$

A feature of some of these algorithms is that they are effectively impractical — one only sees the gains on <u>massive</u> inputs, far bigger than typical input.

It's conjectured that there is an algorithm that uses only $O(n^2)$ operations, which if correct would be insane.

Graphics Processing Units (GPUs) specialize in doing matrix multiplication very fast.

Queues, Stacks, and Deques

These all provide ~~an ADT~~ specific data structure to lists. ~~some~~ ~~may argue that~~

- Queues: they are first-in-first out (FIFO), like a queue at Dunnes.

  Functions include:

  - add(x): adds the value x to the queue.
  - remove(): removes the value, y, "longest" in the queue.

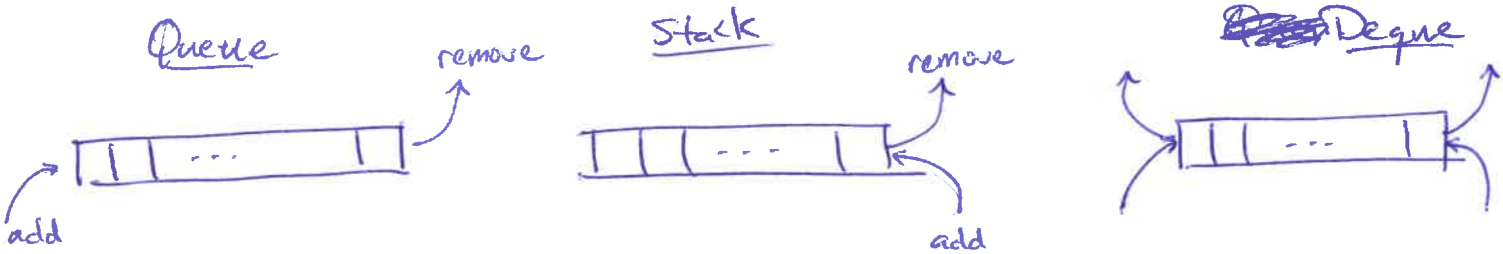
- Stacks: they are last-in-first-out (LIFO), like a stack of dirty plates needing to be cleaned.

  They have the same functions as Queues but it works differently.

- Deques: generalize both ~~the~~ Queues and Stacks. They can do both, like a stack of cards.

  Functions:

  - add_first(x)
  - remove_first(x)
  - add_last(x)
  - remove_last(x).



These ~~ADTs~~ are quite bare-bones when it comes to lists.

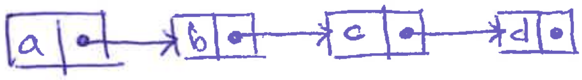
We'll discuss the Array and Linked-List soon, but it will be useful to discuss memory. Memory might be the most important resource to manage when programming.

(memory slides.) ← Online!

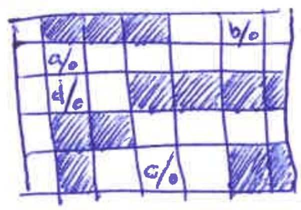
Linked lists

With linked lists objects ~~are~~ of a list are stored in arbitrary open locations of memory.

Visualization:



In Memory:



Each slot in memory takes the object data and the address for the next entry in the list. Additional data "overhead" for a linked list is (usually)

- length : number of entries
- head : address for the first entry
- tail : address for the last entry.

Linked lists can ~~effectively~~ perform both Stack and Queue operations in constant time.