2 Logarithms

- \( f(n) = \log_{10} n \) and \( g(n) = \log_2 n \).
  How do they relate?

\[
\log_{10} n = \frac{C}{\log_2 n} \quad \text{where} \quad \log_{10} n \in \Theta(\log_2 n)
\]

- \( f(n) = \log n \). What base do I mean?
3. $\Omega$ and $\Theta$

- $\Theta$: upper bound on growth rate of function.
- What are $\Omega$ and $\Theta$ used to describe?

![Graph showing functions]

4. A reminder

There are two steps for analysis:
1. What am I analyzing?
   (worst-case? best-case? etc)
2. What bound am I providing?
   (upper, lower, etc)
Running Time of DFS? \( |V| = n \quad |E| = m \)

**DFS-iterative(s)**
- \( \forall v \text{ discovered}[v] = \text{false} \) \( \big\{ O(n) \big\) 
- \( S \leftarrow \text{empty stack} \) \( \big\{ O(1) \big\) 
- \( \text{push } s \text{ to } S \)
- \( \text{while } S \text{ is not empty do} \)
  - \( u \leftarrow \text{pop}(S) \) \( \big\{ O(1) \text{ each} \big\) 
  - \( \text{if discovered}[u] = \text{false then} \)
    - \( \text{discovered}[u] = \text{true} \) \( \big\{ O(m) \text{ total} \big\) 
    - \( \text{for all edges } (u, v) \text{ do} \)
      - \( \text{push}(S, v) \) \( \big\{ O(\delta(u)) \text{ each time} \big\) 
      - \( O(\sum \delta(u)) \text{ total} \)
      - \( O(m) \text{ total} \)

\( \big\{ O(n + m) \big\) 

\( \big\{ O(n + m) \big\) 

---

A new procedure for Stacks/Queues

When push to a "full" array-based stack or queue:
- Create new Object *, twice size of current
- Copy everything from elements to new array.
- Delete the old elements
- Set the array pointer to point to this new one.

- Is this \( O(n) \)?

\( \Theta(n) \)

\( \text{NO!} \)
Step One: Gather Data

- Some operations cost more than others
- Many cheap operations, then an expensive
- What about a sequence of operations?

**Goal at this step:** develop a hypothesis.

**Question:** Total work after $k$ pushes?

<table>
<thead>
<tr>
<th>Operations</th>
<th>Cost</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11-20</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>21-40</td>
<td>21</td>
<td>30</td>
</tr>
<tr>
<td>41</td>
<td>41</td>
<td>70</td>
</tr>
</tbody>
</table>

Total Work Done?

**Hypothesis:** $O(k)$ for $k$ pushes
> Running time of new procedure

- Assume elements began at small size.
- A power of two
- Only double in size when needed.
- A push takes $O(1)$ unless we are doubling.
- The doubling takes $O(n)$, but happens rarely.
- I want to show: $k$ pushes takes $O(k)$

$$
\sum_{i=1}^{k} \theta(i) + \sum_{i=1}^{k} \theta(1) = \sum_{j=0}^{\log_2 k} \left[ \theta(2^j) - \theta(1) \right] + \sum_{i=1}^{k} \theta(1)
\quad = \Theta(k)
$$

> Running time of new procedure

- Assume elements began at small size.
- Only double in size when needed.
- A push takes $O(1)$ unless we are doubling.
- The doubling takes $O(n)$, but happens rarely.
- I want to show: $k$ pushes takes $O(k)$

There’s a better way: “credit argument”

Each push costs $1 + 2 = 3$.

If $O(1)$ push: I pay $1 + n$, save rest.

If $O(n)$ push: I pay $n$ to double array.

I need $\frac{1}{n}$ in savings.

$\frac{n}{2}$ pushes since last big expense
Another View

Warm Up for **Wednesday**

- Linked list is sorted
- Linked list has $-\infty$ and $\infty$
- Want largest key whose value is *at most* $k$.  

find($k$)