I & C SCI 46 Winter 2023
Lecture 6: Analysis II

2

Logarithms

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

  $\log_{10} n = C \cdot \log_2 n$

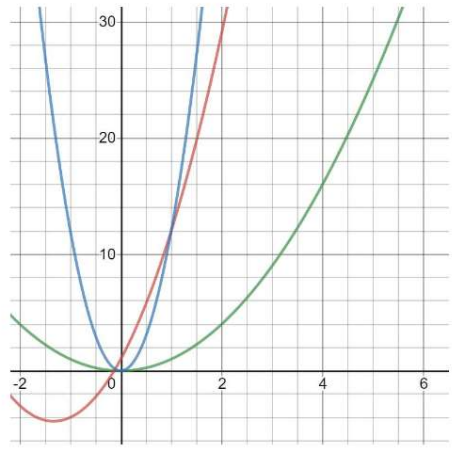
  $\log_{10} n$ is $O(\log_2 n)$

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

3

Ω and Θ

- $\mathcal{O}$: *upper bound on growth rate of function.*
- What are Ω and Θ used to describe?



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)

5 Running Time of DFS? $|V|=n$ $|E|=m$

```
DFS-iterative(s)
  ∀v discovered[v] = false
  S ← empty stack
  push s to S
  while S is not empty do
    u ← pop(S)
    if discovered[u] = false then
      discovered[u] = true
      for all edges (u, v) do
        push(S, v)
```

$\}\ O(n)$

$\}\ O(1)$

$\}\ O(m)$ iterations

$\}\ O(1)$ each

$\}\ O(1)$ each

$\}\ O(n)$ TOTAL

$\}\ O(\delta(u))$ each time

$O(\Sigma\, \delta(u))$ total

$O(M)$ total

$O(n)$

$O(m)$

$O(n+m)$

$O(n+m)$

6 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 $\mathcal{O}(n)$? ✓

$\Theta(n)$?

NO!

7

Step One: Gather Data

initial cap: 10

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

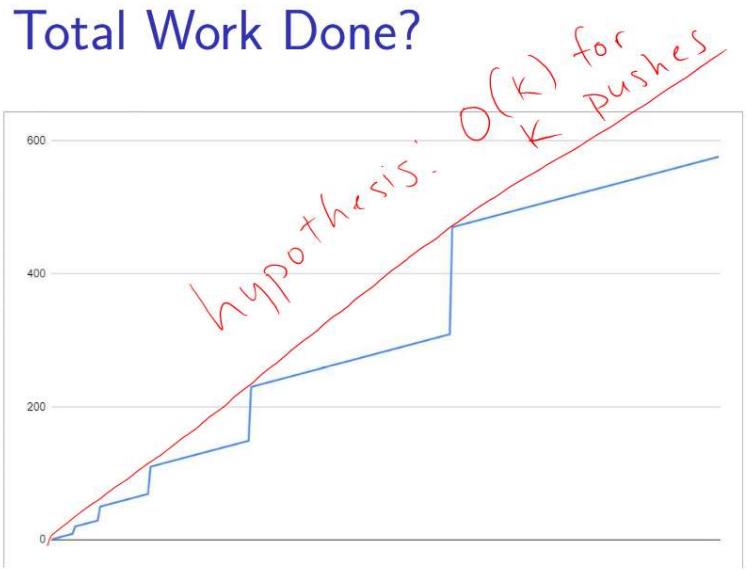
Goal at this step: develop a hypothesis.

Question: Total work after k pushes?

Operations	Cost	Total cost so far
1 – 10	1 each	10
11th	11	21
12 – 20	1 each	30
21	21	51
22 – 40	1 each	70
41	41	111

8

Total Work Done?



Running time of new procedure

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

$$\underbrace{\sum_{i=1}^{k} \Theta(i)}_{\text{pows of 2}} + \underbrace{\sum_{i=1}^{k} \Theta(1)}_{\text{not pow 2}} = \sum_{j=0}^{\lfloor \log k \rfloor} \left[\Theta(2^j) - \Theta(1)\right] + \sum_{i=1}^{k} \Theta(1)$$

$$= \Theta(k)$$

Running time of new procedure

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

amortizes to $O(1)$.

There's a better way: "credit argument"

Each push costs $\underline{1+2=3}$.

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

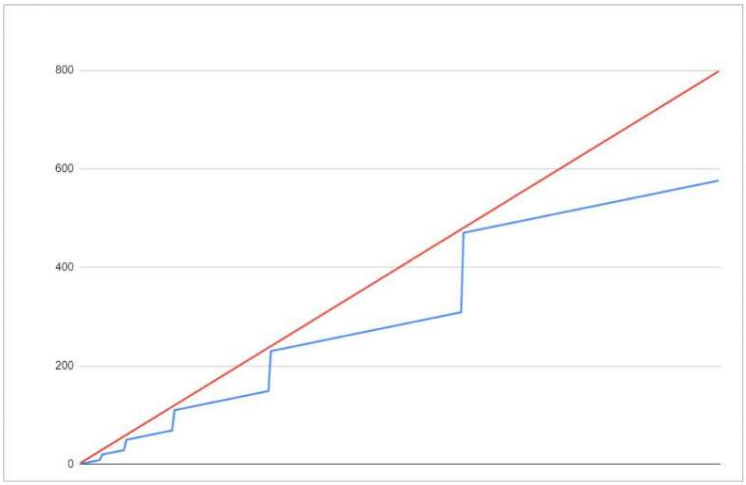
If $O(n)$ push: I pay $\underline{n}$ to double array.

I need \$n in savings.

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

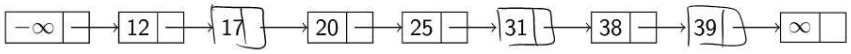
10

Another View



11

Warm Up for [redacted] Wednesday



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

`find(`k`)`