CS 32 – Discussion 1B

Week 7

Algorithm Analysis / "Big O" notation

Runtime analysis: Quantify efficiency

Deterministic algorithm A: Given input X, A runs for some fixed, predetermined # steps on X

↙

basic operations (read/write/comparisons)

Idea: measure intrinsic efficiency of the algorithm.

Q: How many "steps" does an alg. perform w.r.t given input.

Typically: analyze # steps compared to size

e.g: Container class (list/array): size is # elements

Example:

```
Contains (A, x):
    for i = 1, ..., n:
        if A[i] == x: return true;
    return false;
```

Best case: 1 step

Worst case: n steps

always $\leq n$

Worst case analysis:

Given alg. A & $f : \mathbb{N}^+ \to \mathbb{R}^+$

"A runs in time $f(n)$" if

for every input x of size n (for all $n \in \mathbb{N}^+$)

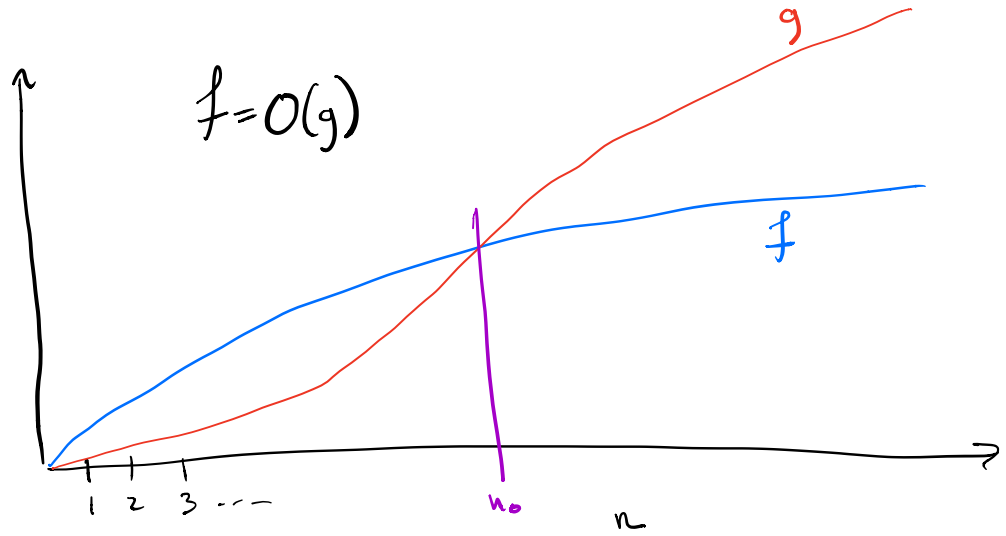
A runs in at most $f(n)$ steps.

Big O notation: compare asymptotic (as $n \to \infty$) growth rate of functions.

Defn: Given $f, g : \mathbb{N}^+ \to \mathbb{R}^+$, $f = O(g)$ if

there exists constants $c, n_0 > 0$ such that (n_0: threshold)

for every $n \geq n_0$, $f(n) \leq c \cdot g(n)$



In calculus terms:

$$f = O(g) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} \text{ converges.}$$

Example: $(\log n)^2$ vs. $\sqrt{n}$. $(\log n)^2 = O(\sqrt{n})$

$$\lim_{n\to\infty} \frac{(\log n)^2}{n^{1/2}} = \lim_{n\to\infty} \frac{2\log n \cdot \frac{1}{n}}{\frac{1}{2}\cdot n^{-1/2}}$$

$$= \lim_{n\to\infty} \frac{4\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{4}{n\cdot \frac{1}{2} n^{-1/2}}$$

$$= \lim_{n\to\infty} \frac{8}{\sqrt{n}} = 0$$

Sorting: $O(n^2)$

Bubble sort, insertion sort, selection sort

$O(n \log n)$

Merge sort, quicksort

Selection-Sort (A)

for $i = 1, \dots, n$: // $A[1], \dots, A[i-1]$ is sorted as expected.

n iterations

find i^{th} min ← $n - i$ steps

swap with $A[i]$ ← 3 steps

Selection-Sort analysis:

$$\sum_{i=1}^{n}((n-i)+3) = 3n + \sum_{i=1}^{n}(n-i)$$

$$= 3n + \binom{n}{2}$$

$$= 3n + \frac{n(n-1)}{2}$$

$$= \underbrace{\frac{1}{2}\cdot n^2} + \underbrace{\left(3n - \frac{1}{2}n\right)}$$

$$= O(n^2)$$

Merge-Sort (A)

if $n \leq 1$: return A // base

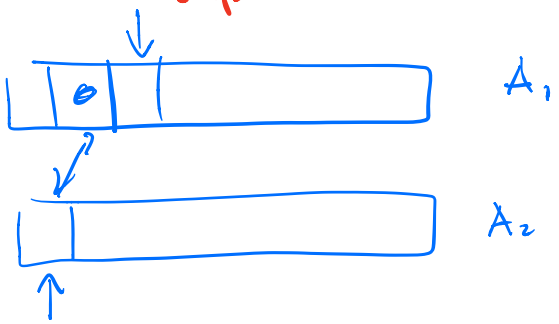
→ A_1 = merge-sort $(A[1], \ldots, A[\frac{n}{2}])$

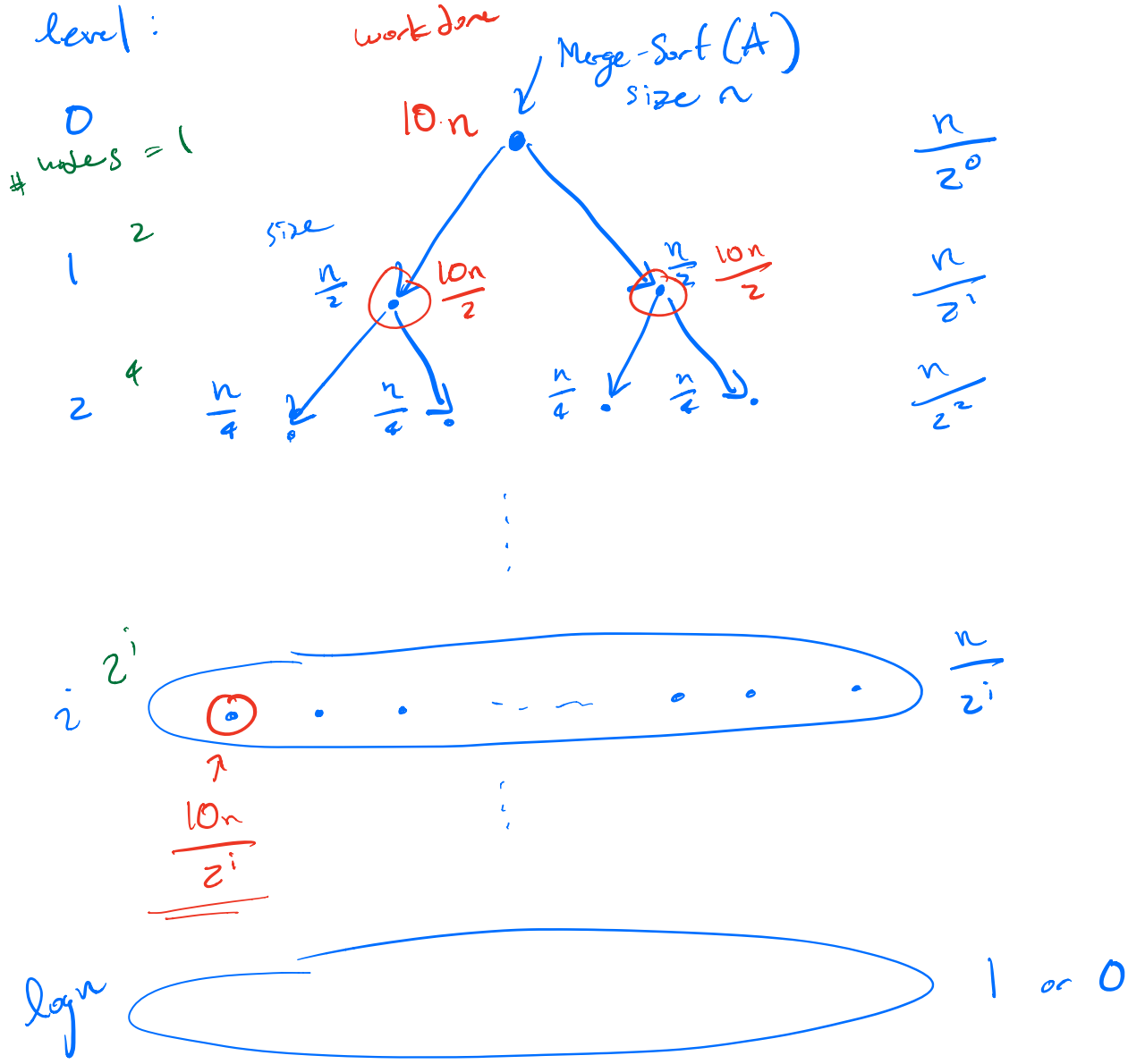
→ A_2 = merge-sort $(A[\frac{n}{2}+1], \ldots, A[n])$

} recursive calls

→ return merge (A_1, A_2) // merge step

steps 10n





$$\text{runtime} = \sum_{i=0}^{\log n} 2^i \cdot \frac{10n}{2^i} = \sum_{i=0}^{\log n} 10n = 10n \cdot (\log n + 1)$$

$$= O(n \log n)$$

2^i: # nodes in level i

$\frac{10n}{2^i}$: work done by a node in level i