Algorithms and Data Notes

William Traub

Contents

1 January 9, Introduction **1**
- 1.1 What is CS3000: Algorithms and Data . . . 1
- 1.2 Egg Drop Contest . . . 2
- 1.3 The Chocolate Coin Problem . . . 2

2 January 13 **3**
- 2.1 Sorting Algorithms . . . 3
- 2.2 Describing Algorithms . . . 3
- 2.3 Divide and conquer . . . 4

3 January 16 **5**
- 3.1 Asymptotic Notation . . . 5

4 January 20, Divide and Conquer cont. **6**
- 4.1 Asymptotic Analysis cont. . . . 6

1 January 9, Introduction

1.1 What is CS3000: Algorithms and Data

The study of how to solve computational problems.

- How to design **efficient algorithms** - Resources: time, space, parallelism

Why do we care about this?

- Improve your problem solving
  - How to attack new problems
  - Which algorithmic tools apply
  - How to compare different solutions
  - How to know if a solution is the best possible
- etc

1.2 Egg Drop Contest

- You want to test your egg parachute (one egg, ladder with n steps)
- You can try dropping your egg odd different steps to see if it breaks

Linear scan

```
LinearScan(1,n):
H = 0
While H <= n and egg intact
  H = H + 1
  Drop egg from step H

Return H - 1
```

Worst case number of steps: n
Now with two eggs: start in the middle and if it breaks

```
BalancedScan(1,n):
  H = 0
  While H <= n and first egg intact
    H = H + n^1/2
    Drop egg from step H
  If first egg intact:
    Return n
  Else:
    LinearScan(H - n^1/2 + 1, H - 1)
```

Worst case time $2\sqrt{n}$
Suppose you have some $t \in \mathbb{N}$ eggs.
With t eggs you can get a worst case of $tn^{1/t}$
With ∞ eggs, use binary search for worst case $log_2(n)$

1.3 The Chocolate Coin Problem

You have a sack of n coins, one is chocolate and lighter. Use a scale to see which of the two is heavier

2 January 13

2.1 Sorting Algorithms

Take n elements and return them in ascending order. Selection Sort - Find the minimum, swap it

```
SelectionSort(A[1:n]):
  for j = 1,...,n-1:
    min_pos = j
    for k = j+1,...,n:
      if (A[k] < A[min_pos]):
        min_pos = k
    swap A[min_pos] and A[j]
```

Analysis:

- How to prove correctness?
- Analysis of run time

At any index j, the first $j-1$ elements are sorted and every element with index $i \geq j$ is larger than element j - 1.

- A[1: j - 1] contains the j - 1 smallest elements of A in order.
- A[j:n] contains the remaining elements of A.

Proof. Base case j=1: This is trivially true as A[1:0] is empty and A[1:n] is the entire array.
Suppose the hypothesis is true.
Induction step j+1: Between the start of iteration j and j+1, we have found a particular element and swapped □

2.2 Describing Algorithms

- Pseudocode: an easily readable, precise, unambiguous description of an algorithm
  - About clarity not format
  - More like comments than code
  - Often avoids the idiosycratic details of programming languages.

2.3 Divide and conquer

- Split your problem into smaller subproblems
- Solve the subproblems recursively
- Combine the solutions

Key tools

- Recursion
- proof by induction
- runtime analysis
- Θ notation

`https://oeis.org/`

3 January 16

3.1 Asymptotic Notation

Big Oh notation: f(n) = O(g(n)) if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ s.t. $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.

4 January 20, Divide and Conquer cont.

Quiz 1 1/21 - Basic iterative and recursive algorithms (study first 2 lectures). Practice quiz on canvas.
Homework 1 due on friday.

4.1 Asymptotic Analysis cont.

Question 1. *Rank the following functions in order of growth:*

1. $n \log_2 n$
2. n^2
3. $100n$
4. $3^{\log_2 n}$

Starting with the first 2. $\lim_{n\to\infty} \frac{\log_2 n}{n} = \lim_{n\to\infty} \frac{\ln n}{\ln 2 \cdot n}$.
Apply L'Hopital
$\lim_{n\to\infty} \frac{1/n}{\ln 2} = \lim_{n\to\infty} \frac{1}{n \cdot \ln 2} = 0$.
$\therefore \log_2 n = O(n)$
$3^{\log_2 n} = (2^{\log_2 3})^{\log_2 n} = (2^{\log_2 n})^{\log_2 3} = n^{\log_2 3} = n^{1.9}$
Answer:

1. $100n$
2. $n \log_2 n$
3. $3^{\log_2 n}$
4. n^2

h
Big Oh Rules:

- Constant factors can be ignored. $\forall C > 0, Cn = \mathcal{O}(n)$
- Lower order terms can be dropped. $n^2 + n^{3/2} + n = \mathcal{O}(n^2)$
- Smaller exponents are Big-Oh of larger exponents. $\forall a < b, n^a = \mathcal{O}(n^b)$
- Any logarithm is Big-Oh of any polynomial. $\forall a, b > 0, (log_2 n)^a = \mathcal{O}(n^b)$
- Any polynomial is Big-Oh of any exponential. $\forall a > 0, b > 1, n^a = \mathcal{O}(b^n)$
- Bases of logarithms can be ignored. $\forall a, b > 1, \log_a(n) = \mathcal{O}(\log_b(n))$
- Big-Omega