Week 2 - Friday

COMP 4500

Last time

- What did we talk about last time?
- Stable marriage
- Representative problems

Questions?

Logical warmup

- There are two lengths of rope
- Each one takes exactly one hour to burn completely
- The ropes are not the same lengths as each other
- Neither rope burns at a consistent speed (10% of a rope could take 90% of the burn time, etc.)
- How can you burn the ropes to measure out exactly 45 minutes of time?

Three-Sentence Summary of Survey of Common Running Times

Survey of Common Running Times

Common running times

- It's a good idea to know the lay of the land when it comes to running times
- Some running times (O(n), O(log n), O(n^2)) come up all the time
- There are often common characteristics between algorithms with the same running times

Linear time

- O(***n***) is linear time
- Linear algorithms only need to scan through the input once (or perhaps a constant number of times)
- **Online algorithms** and **data stream algorithms** take this approach of processing an item as it arrives (and are often linear)

Find the maximum

- The following straightforward algorithm finds the maximum in linear time
  - Note that this book (and many others) indexes from 1 rather than 0

- $\boldsymbol{max} = \boldsymbol{a}_1$
- For $\boldsymbol{i}$ = 2 to $\boldsymbol{n}$
  - If $\boldsymbol{a}_i > \boldsymbol{max}$ then
    - $\boldsymbol{max} = \boldsymbol{a}_i$

Merge two sorted lists

- List $\boldsymbol{A} = \boldsymbol{a}_1, \boldsymbol{a}_2, \ldots \boldsymbol{a}_n$ and $\boldsymbol{B} = \boldsymbol{b}_1, \boldsymbol{b}_2, \ldots \boldsymbol{b}_n$
- Keep $\boldsymbol{current}_a$ and $\boldsymbol{current}_b$ pointers into each list, initialized to point to $\boldsymbol{a}_1$ and $\boldsymbol{b}_1$, respectively
- While both lists are non-empty
  - Let $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$ be the elements that $\boldsymbol{current}_a$ and $\boldsymbol{current}_b$ point to
  - Append the smaller to the output list
  - Update the appropriate pointer
- Once a list is empty, append the remainder of the other list to the output

Analysis of merge

- You can't say that you do constant work per element, since some elements might be compared many, many times
- However, you can account for each item by "charging" it whenever it is added to the final list
- There are only as many iterations as there are charges
- Each element can only be charged once, and there are $2n$ elements
- Thus, the total time is $\Theta(n)$

Linearithmic time

- O(***n*** log ***n***) time is a common running time
  - Sometimes called linearithmic
  - In practice only slightly worse than linear
- This running time is usually associated with divide and conquer algorithms
  - Any algorithm that recursively divides its input into ***k*** equal pieces and then combines the solutions for those pieces in linear time will be O(***n*** log ***n***)
- O(***n*** log ***n***) is the best possible time for a comparison-based sort, including merge sort
- Any algorithm that sorts elements and then does a linear scan will be O(***n*** log ***n***)

Quadratic time

- Comparing all pairs of n things will lead to an $O(n^2)$ algorithm (since there are $n(n-1)/2$ pairs)
  - This is a naïve approach for finding the two closest points in a plane
  - It turns out there is a cleverer way to do it in $O(n \log n)$ time
- Quadratic time also commonly arises when there are two nested loops

Cubic time

- Cubic time is considered to be close to the slowest practical running time for many problems
- If testing to see if an element is in a set can be done in constant time, testing to see which of $\boldsymbol{n}$ sets are disjoint can be done in cubic time
- For each set $\boldsymbol{S_i}$
  - For each other set $\boldsymbol{S_j}$
    - For each element $\boldsymbol{p}$ of $\boldsymbol{S_i}$
      - Determine whether $\boldsymbol{p}$ belongs to $\boldsymbol{S_j}$
    - If no element of $\boldsymbol{S_i}$ belongs to $\boldsymbol{S_j}$
      - Report that $\boldsymbol{S_i}$ and $\boldsymbol{S_j}$ are disjoint

Matrix multiplication

- Matrix multiplication often comes up in the discussion of cubic time
- Multiplying two n x n matrices A and B takes cubic time
- For i = 1 to n
  - For j = 1 to n
    - C[i, j] = 0
      - For k = 1 to n
        - $C[i, j] = C[i, j] + A[i, k] \cdot B[k, j]$
- Matrix chain multiplication optimization also has an $O(n^3)$ dynamic programming algorithm
- Ironically, both matrix multiplication and matrix chain multiplication optimization both have better algorithms

$O(n^k)$ time

- If you want to search over all pairs, you get $O(n^2)$ algorithms
- If you want to search over all subsets of size k, you'll get $O(n^k)$ algorithms
- For example, the following algorithm will find independent sets of size k
- For each subset S of k nodes
  - Check whether S is an independent set
  - If S is independent
    - Print "Success!" and exit
- Print "Failure!"

Analysis of size *k* independent set algorithm

- How many ***k***-element subsets are there?

$$\binom{n}{k} = \frac{n(n-1)(n-2)\ldots(n\ -k+1)}{k(k-1)(k-2)\ldots(2)(1)} \leq \frac{n^k}{k!}$$

- Since ***k*** is a constant, ***k***! is a constant too
- For each set of ***k*** things, we have to check O($\boldsymbol{k}^2$) pairs to see if they have an edge between them
  - Since ***k*** is a constant, this is a constant too
- The total work is thus O($\boldsymbol{n}^k$)

Beyond polynomial time

- The full maximum independent set algorithm:
- For each subset ***S*** of nodes
  - Check whether ***S*** is an independent set
  - If ***S*** is a larger independent set than the largest seen yet
    - Record the size of ***S*** as the new maximum

Running time for maximum independent set

- There are 2^n subsets of an $\boldsymbol{n}$-element set
- Since each subset could be as large as $\boldsymbol{n}$, testing whether it is independent could mean testing $O(\boldsymbol{n}^2)$ pairs
- The total running time is thus $O(\boldsymbol{n}^2 2^n)$

Factorial

- If you have ***n*** items that you want to match up with ***n*** other items, there are ***n***! possibilities
- Recall that $n! = n(n - 1)(n - 2) \ldots (2)(1)$
- ***n***! grows even **faster** than 2^n
- Even though there are ***n***! ways to match up ***n*** men with ***n*** women, our stable marriage algorithm worked in $O(n^2)$ time
- O(***n***!) time also comes up when you're trying to order ***n*** items

Sublinear time

- Is it possible to do better than linear time?
- Yes, both O(log ***n***) time and $O(\sqrt{n})$ are better than linear time
- A great example of logarithmic running time is binary search on sorted array ***A***, looking for value ***k***
- ***start*** = 1
- ***end*** = ***n***
- While ***start*** < ***end***
  - ***middle*** = (***start*** + ***end***) / 2
  - If ***A***[***middle***] < ***k***
    - ***end*** = ***middle*** – 1
  - Else if ***A***[***middle***] > ***k***
    - ***start*** = ***middle*** + 1
  - Else
    - Print "Value found at location " + ***middle***

Running time for binary search

- We cut the search space in half every time
- At worst, we keep cutting $\boldsymbol{n}$ in half until we get 1
- Let's say $\boldsymbol{x}$ is the number of times we look:

$$\left(\frac{1}{2}\right)^{\boldsymbol{x}} \boldsymbol{n} = 1$$

$$\boldsymbol{n} = 2^{\boldsymbol{x}}$$

$$\log_2 \boldsymbol{n} = \boldsymbol{x}$$

- The running time is $\boldsymbol{O}(\log \boldsymbol{n})$

Worked Exercises

Exercise 1

- Put the following functions in ascending order of growth rate
  - $f_1(n) = 10^n$
  - $f_2(n) = n^{\frac{1}{3}}$
  - $f_3(n) = n^n$
  - $f_4(n) = \log_2 n$
  - $f_5(n) = 2^{\sqrt{\log_2 n}}$

Exercise 2

- Let $f(n)$ and $g(n)$ be two functions that take nonnegative values, and suppose that $f(n)$ is $O(g(n))$.
- Prove that $g(n)$ is $\Omega(f(n))$.

Quiz

Upcoming

Next time…

- Proofs by mathematical induction
- Definitions and applications for graphs

Reminders

- **Assignment 1 is due tonight at midnight**
- Start on Assignment 2 when it's assigned
- Read section 3.1