Algorithms and Programming Techniques
COMP3121 UNSW

Hussain Nawaz
hussain.nwz000@gmail.com

2022T2

Contents

1 Introduction - Revision **2**
1.1 Rates of Growth 2
1.2 Assumed Data Structures 3

2 Algorithm Analysis **4**
2.1 Stable Matching Problem 4

1 Introduction - Revision

1.1 Rates of Growth

The Problem To analyse algorithms, we need a way to compare two functions representing the runtime of each algorithm. However, comparing values directly presents a few issues.

- Outliers may affect the results.
- One algorithm may be slower for a set period of time, but catch up after a while.
- The runtime of an algorithm may vary depending on the implementation or the architecture of the machine where, some instructions are faster than others.

Asymptotic Growth For algorithms, we often prefer to refer to them in terms of their asymptotic growth in runtime, with relation to the input size.

A function that quadruples with every extra input will always have a greater runtime than one that increases linearly with each new input, for some large enough input size.

Big O Notation We say $f(n) = O(g(n))$ if there exists a positive constants C, N such that

$$0 \leq f(n) \leq Cg(n) \quad \forall n \geq N.$$

We may refer to $g(n)$ to be the asymptotic upper bound for $f(n)$.

Big Omega Notation We say $f(n) = \Omega(g(n))$ if there exists positive constants c, N such that

$$0 \leq cg(n) \leq f(n) \quad \forall n \geq N.$$

Then, $g(n)$ is said to be an asymptotic lower bound for $f(n)$. It is useful to say that a problem is at least $\Omega(g(n))$.

Landau Notation $f(n) = \Omega(g(n))$ if and only if $g(n) = O(f(n))$.

There are strict version of Big O and Big $Omega$ notations; these are little o and little ω respectively.

We say $f(n) = \Theta(g(n))$ if

$$f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)).$$

That is, both f and g have the same asymptotic growth.

Logarithms Logarithms are defined so that for $a, b > 0$ where $n \neq 1$, let

$$n = \log_a b \Leftrightarrow a^n = b.$$

They have the following properties:

- $a^{\log_a n} = n$

- $\log_a(mn) = \log_a(m) + \log_a(n)$
- $\log_a(n^k) = k \log_a(n)$

By the change of base,

$$\log_a(x) = \frac{\log_b(x)}{\log_b(a)}.$$

As such, the denominator is constant in terms of x and so all log bases are equivalent under Big Theta notation.

1.2 Assumed Data Structures

Arrays We assume static arrays (though, it is possible to extend to dynamic arrays).

- We assume random-access in $O(1)$
- Insert / delete $O(n)$
- Search: $O(n)$ - $\log n$ if sorted

Linked Lists We assume the linked lists are doubly-linked since the)2× overhead is negligible.

- Accessing next / previous: $O(1)$
- Insert / delete to head or tail: $O(1)$
- Search: $O(1)$.

Stacks Last in, first out.

- Accessing top: $O(1)$
- Insert / delete from top: $O(1)$

Queue First in, first out.

- Access front: $O(1)$
- Insert front: $O(1)$
- Delete front: $O(1)$

Hash Tables Store values by their hashed keys. Ideally, no two keys will hash to the same value, however this may not be guaranteed.

- Search is expected to be $O(1)$ however, in the worst case, we expect to have to search through all the values in $O(n)$.
- Insertion is expected to be $O(1)$ however, in the worst case, we expect to have to search through all the values in $O(n)$.
- Deletion follows the same pattern of $O(1)$ expectation and $O(n)$ worst case.

2 Algorithm Analysis

2.1 Stable Matching Problem

Hospital's Stable Matching Problem Suppose there exists n doctors and hospital. Each hospital wants to hire exactly one new doctor and the hospital ranks the priority of the doctors they would prefer and each doctor also lists their preferences for the hospitals they'd like to go to.

For a stable match, we would like to create an allocation where no group would be happy to trades their allocations with each other.

Naive Solution The naive method computes all possible allocations and then picks one that is stable.

This runs in $O(n!)$ time which is not preferable.

Fun Fact about Factorials - Sterling's Approximation

$$n! \approx \left(\frac{n}{e}\right)^n$$

Gale - Shapely Algorithm - Assumptions

- Produces pairs in stages, with possible revisions.
- A hospital which has not been paired, will be called *free*.
- Hospitals will offer jobs to doctors, who will decide whether to accept a job or not. Doctors may renege.
- All hospitals start off as free.

Gale - Shapely - Solution While there exists a free hospital which has not offered jobs to all doctors, pick a free h and have it offer a job to the highest doctor d on its list to whom they have not offered a job yet.

If no one has offered d a job yet, they will always aspect the pair h, d.

Otherwise, if they're already on the pair h', d then if h is a higher preference than h' the doctor will renege h' for h to form (h, d). Otherwise, the hospital will seek another person to offer a job.

Proving Termination in n^2 In all rounds of the Gale-Shapely algorithm, the hospital will offer a job to one doctor. They may offer a job to each doctor only once. Thus, the hospital may only make n offers. There are n hospitals each making n offers so, the algorithm must terminate in n^2 rounds or less.

Proving Correctness of Gale-Shapely If the while loop has terminated with h still free, that implies that h has offered a job to everyone. Since the only way to get declined is if another, more appealing hospital has offered a job to other doctors.

If there are no more doctors available, then n other hospitals must have offered a job to n doctors. However this would imply there are $n+1$ hospitals in total, which is a contradiction.

Hospitals will be happy as they start with their highest preference first and, doctors will move up to accept their most preferred offer. Thus, there cannot exists a circumstance where both a hospital and doctor would like to swap.