School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Complexity – Linked lists

Overview

- Today we will have
  - Review on the topic of last session
  - Formal definition of Big O
  - If we have time: Linked list!
  - If we have time: formal definition of Big Omega, Theta, little o

Review

- We’ve talked about complexity in general terms.

- Assumptions:
  - The complexity analysis focuses on algorithms
  - The input size is taken as argument.
  - The machine model is used to eliminate the influence of hardware

- The running time complexity of an algorithm matters in
  - the worst case
  - the average case

Review

- Is it always easy to find the complexity of an algorithm?
  - No!
- We provide a range for the complexity that we are after
  - Upper bound
  - Lower bound
- Usually used for the worst case or the average case.
- When you introduce an instance of the problem as the worst case, is it always possible to prove that it is the worst case?
  - No. (but for the simple search example it was obvious!)
  - Formally, it is usually called a hard instance.
  - It gives a lower bound on the worst case complexity, but for proving an upper bound we need to go generally for the proof

Questions! (with focus on worst case)

- As a customer
  - Is an upper bound on the worst case important?
  - Is a lower bound on the worst case important?
- As an analyst
  - Do you try to find a high upper bound or low?
  - Do you try to find a high lower bound or low?
  - How can you prove that a worst case upper bound is tight?
    - How can you prove that a worst case lower bound is tight?
- As the algorithm designer
  - Are you happy if an analyst finds a high upper bound or low?
  - Are you happy if an analyst finds a high lower bound or low?
  - Are you happy if an analyst finds high/low upper/lower bounds for your opponent's algorithm?

Example: A simple search function

```
Search(list, item)
{
    for(i=1 to n)
        If list[i]=itm
            return i

}
```

Required time:

- Set up time
- For loop

Example

- The for loop will take approximately n times the effort to make decision on one item.
- Set-up for this algorithm is constant.
- Mathematically, the execution time is equal to cn + s.
  - Where c and s are constants and represents the overhead per iteration, and the set-up costs, respectively.
- $f(n) = cn + s$ represents the actual execution time of this searching process.

Big O [O(g(n))]

- If f(n) represents the *actual* execution time of an algorithm, then, as we don't always know the details of f(n), we approximate it.

- $f(n) = O(g(n))$ if there exist positive constants c and n_o such that $f(n) <= c.g(n)$ when $n >= n_o$.

- We refer to this as Big-Oh(g(n)), O(g(n)) or "order g(n)"

- Let's see some examples

Example

- $n+1 = O(n)$?
- $n+2 = O(n)$?
- $2n+2 = O(n)$?
- $n^2 = O(n)$?
  - No!
- $\mathrm{sqrt}(n) = O(n)$?
- $1 = O(n)$?
- $\log n = O(n)$?
- $n \log n = O(n)$?
  - NO

Example

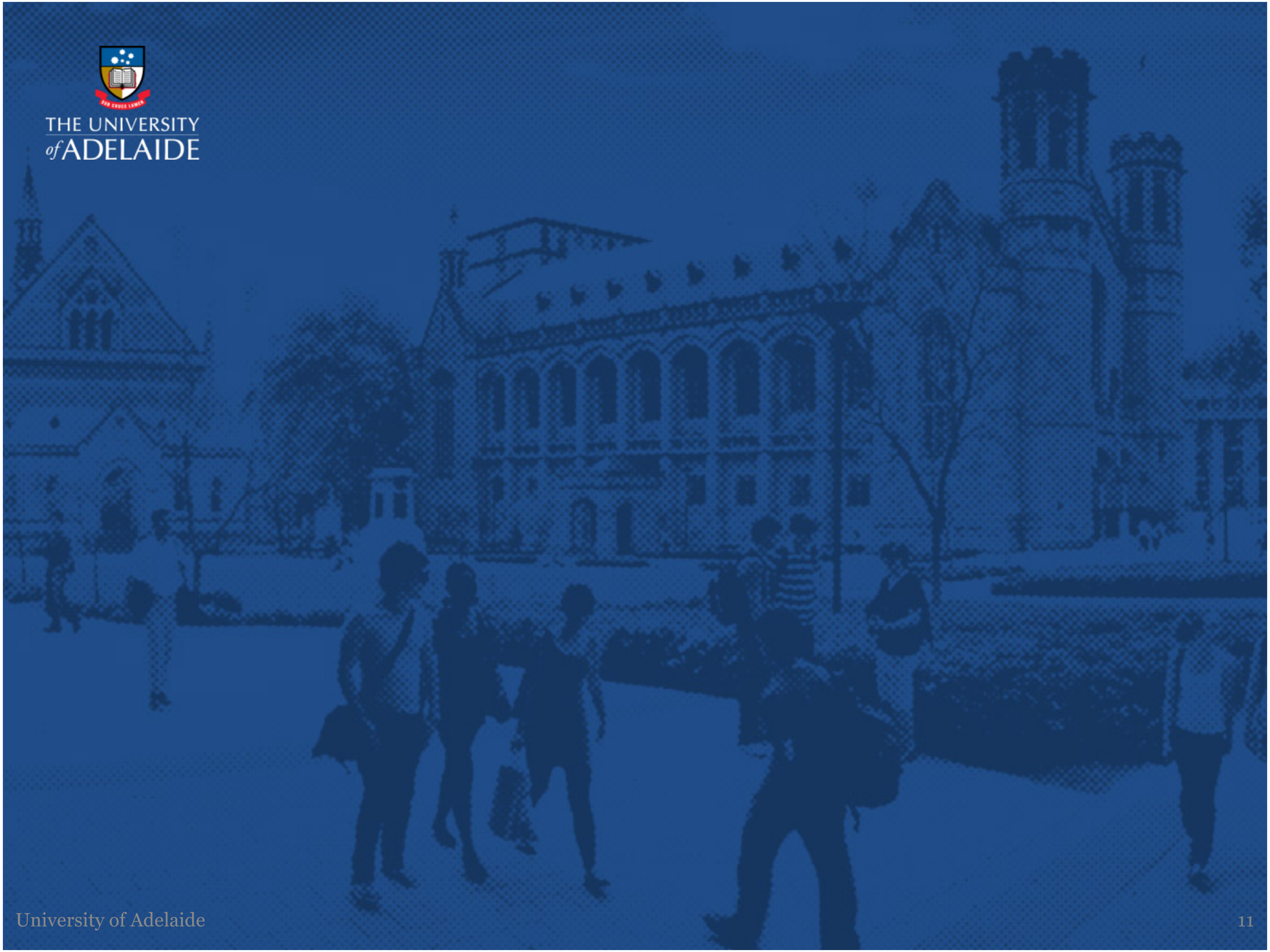
- $n = O(n^2)$?
  - Correct but not tight

- $n^2 = O(n^2)$?

- $n^3 = O(n^2)$?
  - No

- We want our Big Oh bounds to be:
  - Tight: We want $g(n)$ to be as close to $f(n)$ as it can be.
  - Simple: we can drop low order terms and constants. (why?)

- Growth rates are important


THE UNIVERSITY
of ADELAIDE