CRICOS PROVIDER 00123M

School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Bucket sort, stack and Queue

Bucket sort- Scattering and sorting

- The choice of the number of buckets and the way that we scatter makes a big difference.
  - Works best if items are evenly distributed across the buckets
- Also it it important what sort of sorting algorithm is used for sorting the elements inside each bucket.
- If we use one bucket, and insertion sort, what happens?
  - O(n^2)
- For n values, uniformly distributed over a range, what happens if we use k buckets and then insertion sort?
  - Scatter the items: O(n)
  - Sort buckets: k. O((n/k)^2) = O(n(n/k))
    - = O(n) if k=cn
  - Gather items: O(k+n) = O(n) if k=cn

Bucket Sort

- Distribution of data matters!
  - Take one bucket for a wide range that does not have many items.
  - Assign smaller ranges to buckets where the items are concentrated
- Worst-case performance for Bucket sort is $O(n^2)$
  - If number of buckets is not unreasonably huge!
- Average case complexity $O(n+k)$
  - which is $O(n)$ if $k=O(n)$

Summary on sorting

- Comparison sort:
  - Insertion sort
  - Selection sort
  - Bubble sort
  - Merge sort
  - Quicksort
- Comparison sorts have a lower bound of $\Omega(n \log n)$.
- Distribution sorts do better if we can come up with a proper way of scattering that isn't high complexity
- Distribution sort:
  - Counting sort
  - Bucket sort
- There are other sorting algorithms which you may see in later courses.

Review

- Recall the definition of ADT.
  - A data type consists of a collection of values together with a set of basic operations defined on those values.
  - A data type is called an abstract data type if the programmers who use the type do not have access to the implementation details.
- You should not need to know anything about the implementation in order to use that type
- ADT in C++
  - Built-in
  - User defined

List; as an ADT

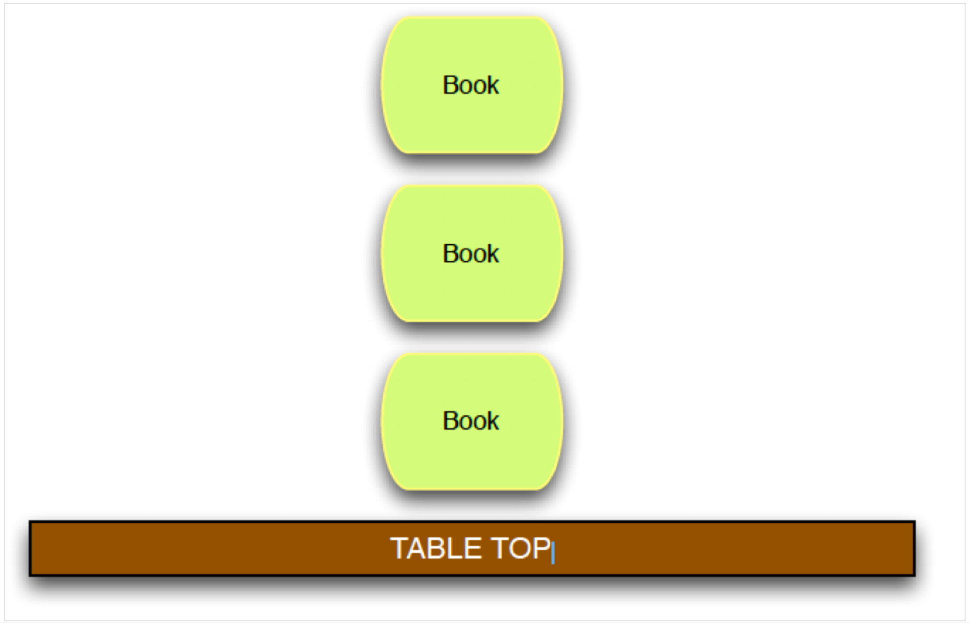
- A general list of form A_0, A_1, …, A_{n-1}
- For any list except the empty list, we say that A_i follows A_{i-1} $(i<n)$ and A_{i-1} precedes A_i
- Operations:
  - toString
  - isEmpty
  - search
  - insert
  - remove
  - getValue

Review

- We're familiar with
  - Arrays, dynamic arrays and vectors
    - What is the difference between a vector and an array?
    - Efficient for accessing the elements by index (random access)
  - Linked lists
    - Dynamically grows and shrinks
    - By the way! When you delete an item from the linked list, don't forget to delete the node object from the heap.
- Both can be used for implementing a list.
  - How to implement each of the operations? What is the complexity?
- Two common applications of lists:
  - stacks
  - queues
- All of these discussions when we need a linked list, we will use singly-linked lists.

Stack

- A stack is a data structure that retrieves data in the opposite order to which it was stored.
- You only have access to the top of the stack.
- This is called Last-In/First-Out (LIFO).
- Backseat of a taxi!



Stack operations

- The operations associated with a stack are:
  - push - we put an element on top of the stack
  - pop - we take the top element off the stack and return it
  - empty - we return true if the stack is empty, false otherwise
- What is the precondition for pop?

Stacks, using linked lists or arrays

- Stacks are very easy to implement with linked lists
  - How do you suggest it should be done?
  - What do you think about the complexity of the operations?

  - Push adds a node to the **top** of the list.
  - Pop removes the node at the **top**, returns the value and destroys the old node, updating **top pointer** to point to the new top.
  - Empty checks to see if **top** points to NULL.

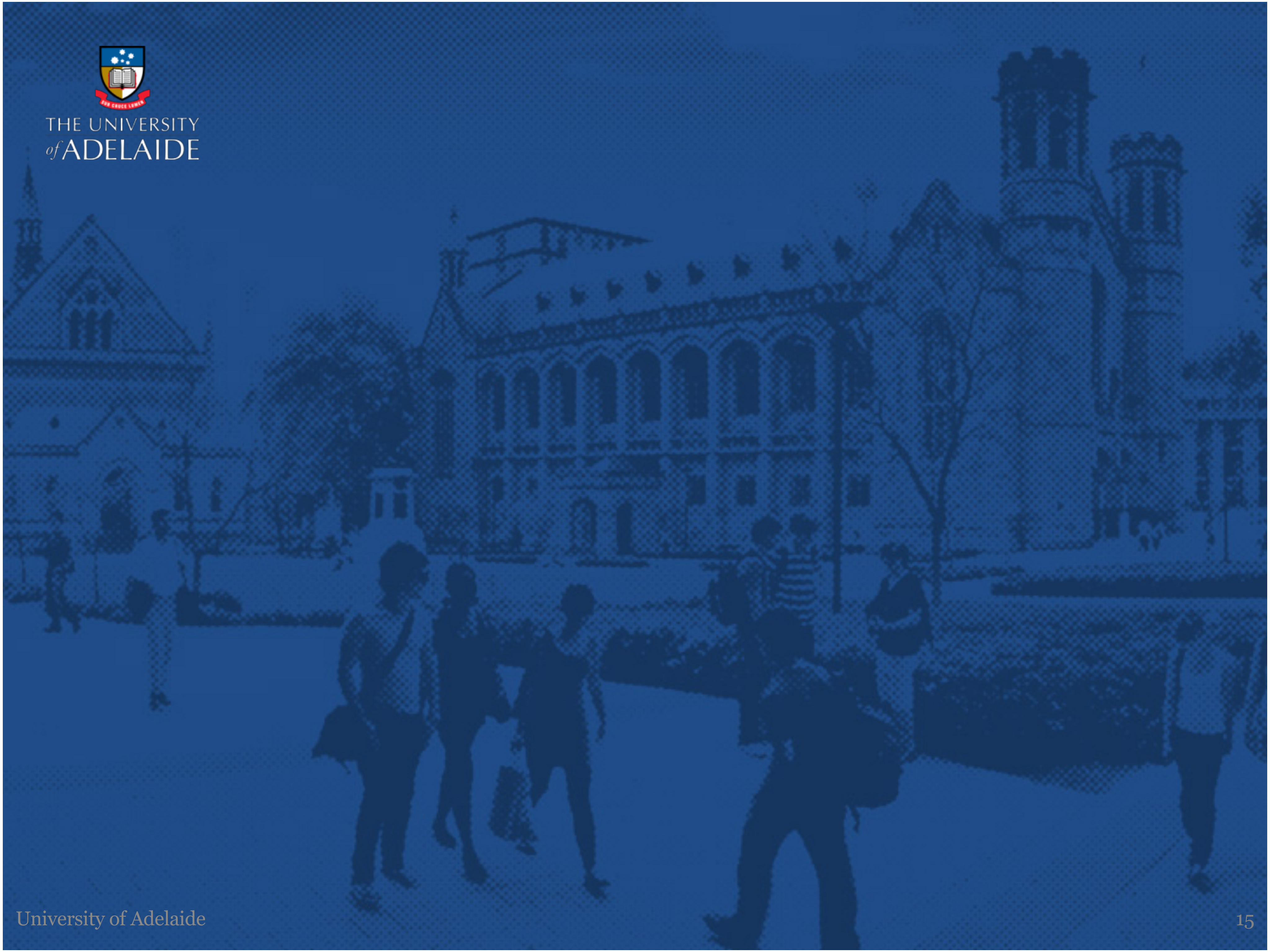
Stacks, using linked lists or arrays

- Different implementations
  - Linked list implementation
    - Push
      - O(1)
    - Pop
      - O(1)
    - IsEmpty
      - O(1)
  - Array implementation

- Both implementations can guarantee O(1) complexity for the basic operations.
  - When we know that the size of stack will not be too large, it is easier and more efficient to use arrays

Notes for stacks

- Black box
  - Inside, we may have a linked list or an array
  - While you have access to the whole chain, the functions that you use in this data structure **restrict** you to only accessing certain elements.
- This enforces the **LIFO** semantics of the data structure and this allows you to write your code knowing that this will be enforced.
- What would happen if your stack allowed random access?
  - Higher chance of ruining the LIFO property



THE UNIVERSITY
of ADELAIDE