School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Algorithmic Strategies - Heap

Review: Algorithmic Strategies

- Brute force (exhaustive search)
- Backtracking
- Branch and bound
- Divide-and-Conquer
- Transform-and-Conquer
- Dynamic Programming
- Greedy Algorithms
- Heuristic Algorithms

Greedy Algorithms

- Remember our counting coins algorithm?
- The change-making problem is generally defined as "How do I give the right amount of change using the smallest number of coins?"
- In our dynamic programming approach, we just made change, with no extra rules. We found the number of possible solutions.
- Now we would like an optimal solution (minimum number of coins).

Greedy Algorithms

- We can obtain local optimal solutions to optimisation problems by constructing a solution through a set of steps, where:
  - Each choice is feasible and satisfies our requirements
  - Each choice is the best choice to make from all possible choices FOR THAT STEP
  - We cannot change the choice we make here, as we make more choices further on.

Heuristic Algorithms

- Rule of thumb!
- No guarantee on finding the best solution
- Can reduce complexity of finding an acceptable solution

- Example:
  – Visit a number of cities in different states of Australia
  – You feel that it is more rational to visit all cities of one state and then move to another state…

Heap

and

Heap Sort

Review - Binary Search Tree

- A binary search tree (BST) is a binary tree with the following properties:
  - Node values are distinct and comparable
  - The left subtree of every node contains only values that are *less than* the node's own value.
  - The right subtree of every node contains only values that are *greater than* the node's own value.
- Basic Operations:
  - Search
  - Min and Max
  - Insert
  - Remove
- A BST does not have to be balanced. Worst case O(n)

Review - Self-balancing BSTs

- A self-balancing BST automatically keeps its structure balanced.
- Example: **AVL tree**
- An AVL tree is a BST with a balance condition
  - For every node, the heights of two child subtrees can only differ by at most 1.
  - After insertion / deletion, if the above property is violated, then some housekeeping is needed to restore the property, which takes O(log n) extra time. (single or double rotations)
  - Since the tree is always fairly balanced, searching, insertion, and deletion all take logarithmic time in the worst case.

- Today, we will see a binary tree with minimized height
  - Not a binary search tree

Review: Priority Queue

- We saw how to implement this with an array or linked list
- Basic operations:
  - deleteMin
  - insert

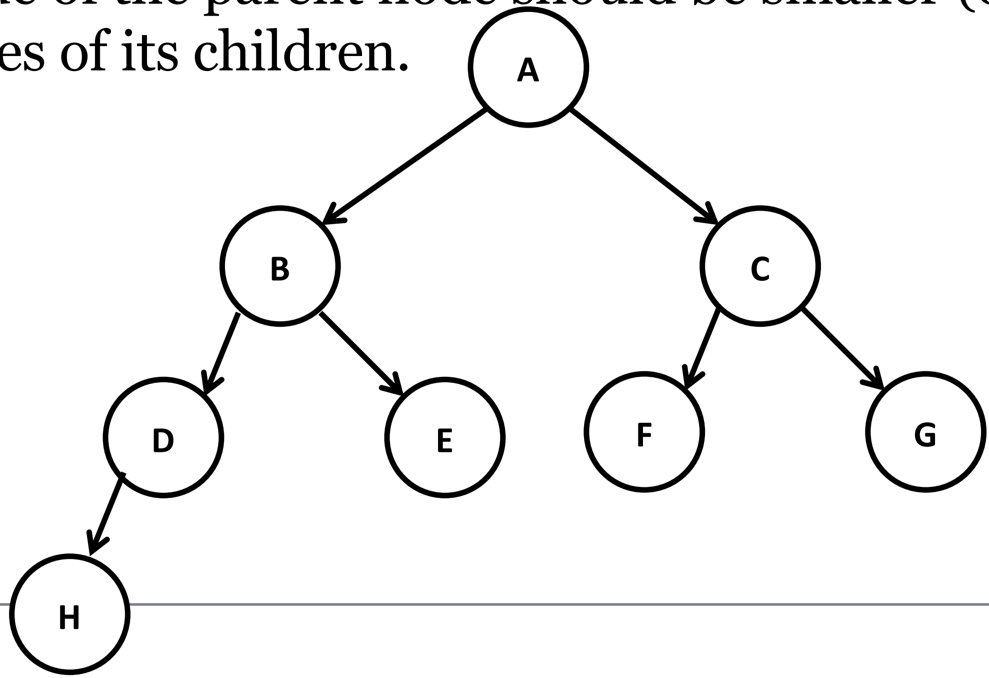
  - Or

  - delete
  - insertWithOrder

- Another application of binary tree is the implementation of priority queue.

Binary Heap

- The properties of a binary heap:
- Structure property:
  - A binary tree that is completely filled with the possible exception of the bottom level, which is filled from left to right.
- Heap order property:
  - The value of the parent node should be smaller (or greater) than the values of its children.



Binary Heap

- Basic operations:
  - findMin
    - The minimum value is stored in the root node
    - Complexity of O(1)

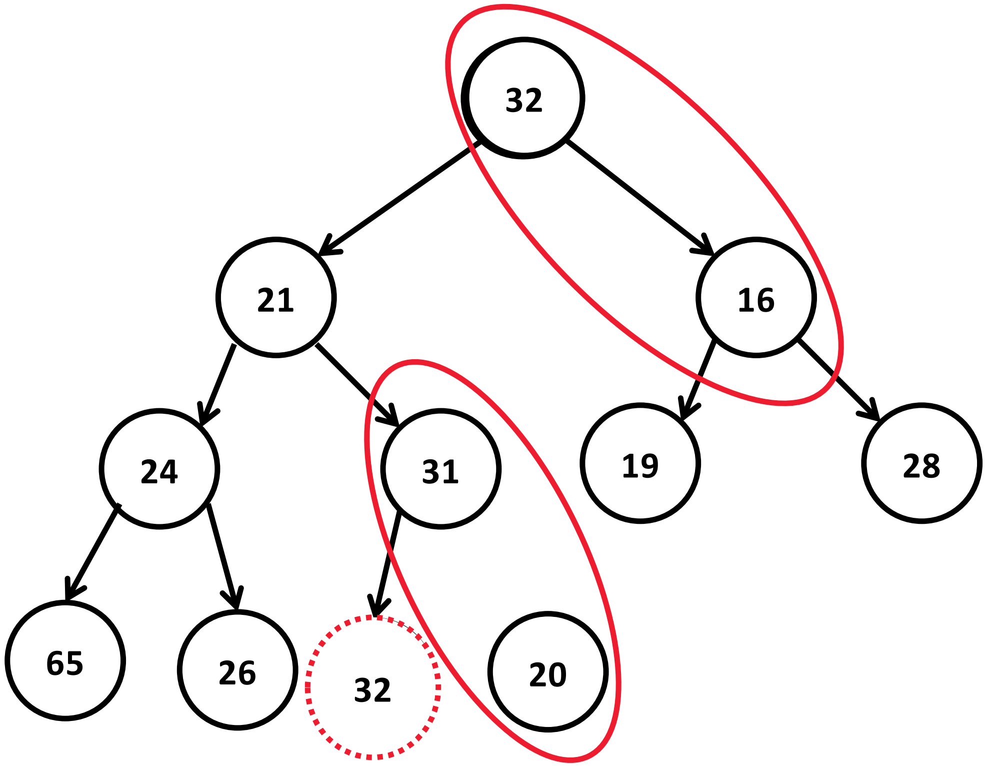
  - insert
  - deleteMin
    - For these two operations, we need to restore the properties.
    - What is the worst case in complexity?

Binary Heap

- Basic operations:
  - findMin
  - insert
  - deleteMin

  - Example: add 20

  - Example: deleteMin



- Basic operations of Priority Queue
  - Insert worst case O(log n), Average case O(1)
  - Delete worst case O(log n)

Heap Sort

- Do you remember selection sort?
  - Find the smallest item
  - Place it at the beginning of the list
  - Find the next smallest item and place it next
  - repeat above step

  - Placing takes place in O(1)
  - Costly part is finding the minimum.
  - What if that could be done in O(log n)?
  - Total procedure:
    - Make the heap (what is the worst case complexity?) Can be O(n)
    - Repeat for n times
      - Remove the item with minimum value and place it in sorted list
    - Total complexity O(n log n)

Heap Sort

- Can you store a heap in an array?
  - It is possible! We will not need to save the links! Note that it is a complete binary tree (except for the last level)
  - Think of the first item as the root, followed by items of each level from left to right
  - For a node with index i,
    - Left child has index 2i+1
    - Right child has index 2i+2
    - Parent has index $\lfloor (i-1)/2 \rfloor$
- Same process
  - Make the heap and remove elements from the root one by one
  - Can be done in-place

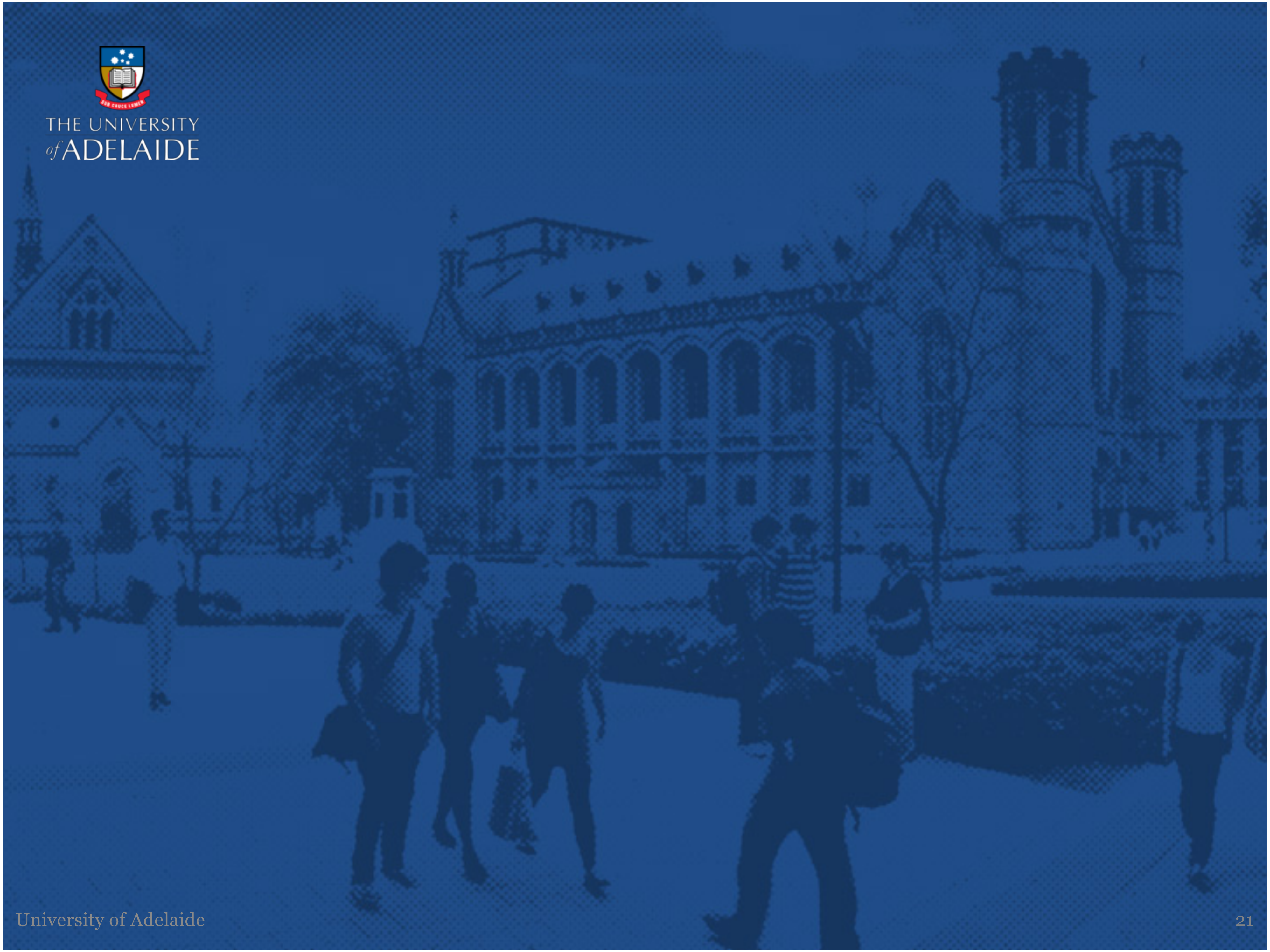
Compare Heap Sort with others

- Can be done on an array
  - But not on a linked list (similar to quick sort)
    - We should add another link to the node structure that we have and make a binary heap, then we can do heap-sprt!
  - Merge sort can be done on both arrays and linked lists
- In-place – no need for auxiliary list
  - Merge sort needs auxiliary array when sorting an array, but not when sorting a linked list
- Worst case complexity
  - Like merge sort, O(n log n)
  - Quick sort O(n^2)
- Not stable
  - Merge sort is stable

Do we have extra time?

- Let us watch some cartoon!
- Topic:
  Can the machine solve every problem?

https://www.youtube.com/watch?v=92WHN-pAFCs


THE UNIVERSITY
of ADELAIDE