CRICOS PROVIDER 00123M

School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Sorting

adelaide.edu.au

seekLIGHT

Sorting Algorithms

- Insertion Sort
  - Complexity
    - Worst and average-case $O(n^2)$
- Selection Sort
  - Complexity
    - Worst and average-case $O(n^2)$
- Today:
  - Bubble sort
  - Quick sort

Bubble Sort

- Bubble sort is another classic sorting algorithm:
- Larger values ‘bubble up‘ to the top of the array.
- Example

Bubble Sort

- Bubble sort is another classic sorting algorithm:

```
v[] <- {numbers}                    // Start with an array of numbers
n <- length{v}                      // Get the length
  for (i = n-1; i > 0 ; i--){       // Start from the end
    for(j = 0; j < i ; j++){        // Scan up to this point
      if(v[j] > v[j+1]){            // If this value is bigger than the next one
        temp <- v[j]                // Swap them
        v[j] <- v[j+1]
        v[j+1] <- temp
      }
    }end for loop
  }end for loop
```

- *Terminate outer loop once we don't swap anything*
- Complexity
  - worst-case $O(n^2)$
  - average-case $O(n^2)$
  - best-case $O(n)$

Quicksort

- Quicksort is another divide-and-conquer sorting algorithm.
- We divide the larger list into two smaller lists, the low elements and the high elements.
- We then recursively sort the sub-lists until we reach the base case, lists of length 0 or 1.
- Compared to what, do we determine if something is lower or higher?
  - We pick a value for comparison - the pivot.

Quicksort

- Steps for quicksort, starting with a list of values:
  - Pick an element, the pivot, from the list.
  - Reorder the list so that everything smaller than the pivot comes before it, and everything greater comes after it. (The partitioning step)
  - Recursively sort the ‘smaller’ list and the ‘greater’ list.

Quicksort

function quicksort(ref to myList, lIndex, rIndex) {

- If (lIndex>=rIndex)
  return;
- Choose pivot;
- Partition myList (must be done in-place) to leftList + pivot +rightList
  Such that elements of leftList are <= pivot and elements of rightList are >= pivot (or >= and <=, respectively, for descending order)
  Let leftListEnd be the index of the last element of leftList and rightListStart be the index of the first element of rightList
- quicksort(myList, lIndex, leftListEnd)
- quicksort(myList, rightListStart, rIndex)
- return;

}

Quicksort

- The choice of pivot is essential.
- To discuss this, let's look at the best and worst case performance for quicksort.
  - What is the worst case? $O(n^2)$
    - The pivot choice can make all the difference.
  - What is best case? $O(n \log n)$
    - $T(n)=2T(n/2)+cn$
  - What is the average case? $O(n \log n)$
    - Idea: in half of the situations the pivot's value is less than ¼ of the elements and also more than 1/4 of the elements
- If the list is small, insertion sort may even take less time!
  - Good to combine quick and insertion sort

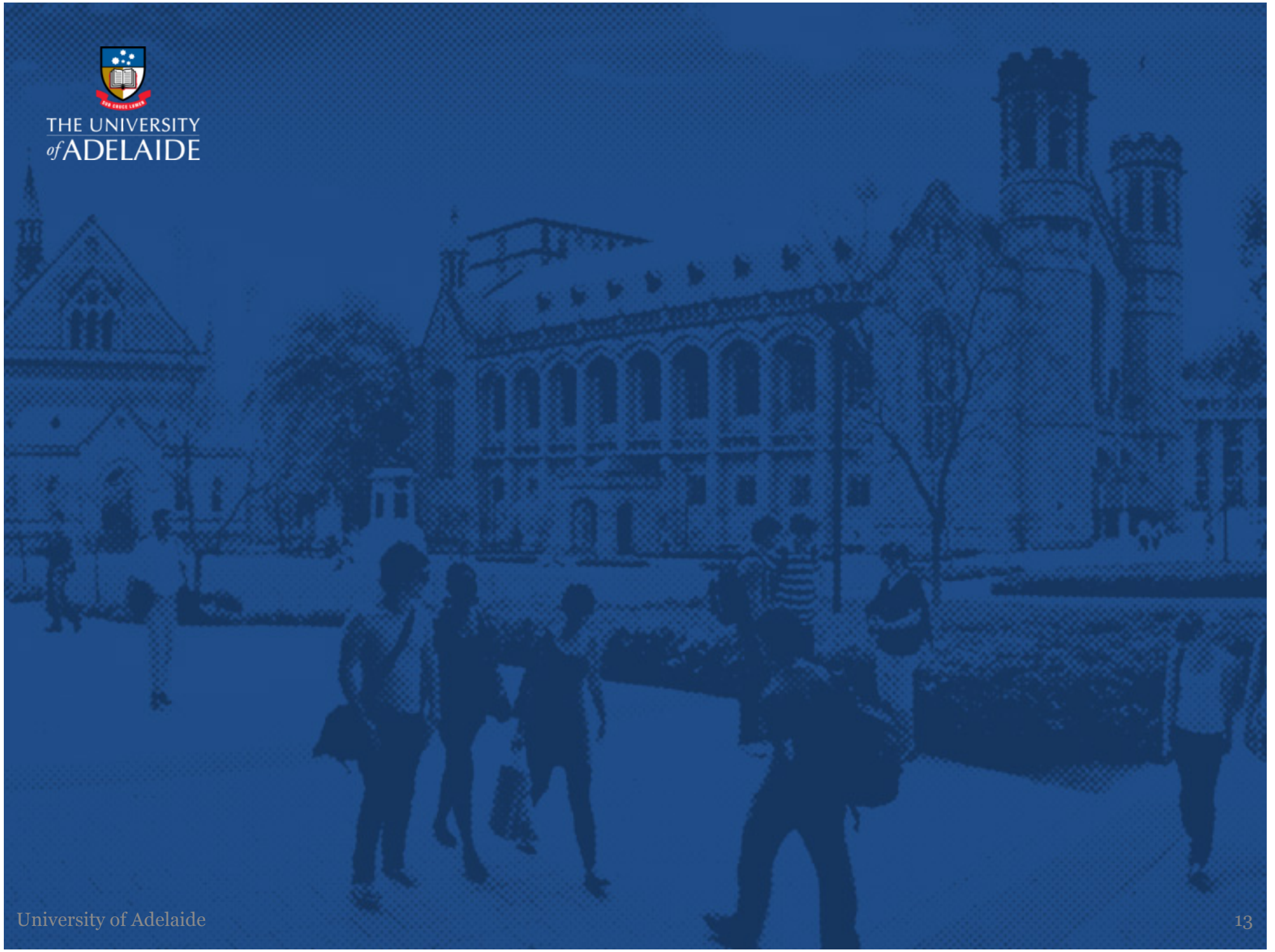
Quicksort

- How to implement?!

- It must be done in place: Do not make new lists to keep the sublists.

Complexity of divide and conquer algorithms

- **Master's theorem**
- T(n)= aT(n/b)+ f(n)
- $log_b a$ is important
  - If f(n)= O(n) and $log_b a$= 1 then we have T(n)= O(n log n)


THE UNIVERSITY
of ADELAIDE