# 13 B: Summary of CS1102S

CS1102S: Data Structures and Algorithms

Martin Henz

April 16, 2010

- 1 Highlights of CS1102S
- 2 Java API Support for Data Structures
- Outlook to Other Modules

#### Highlights of CS1102S

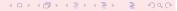
Java API Support for Data Structures Outlook to Other Modules

#### Algorithm Analysis

Lists, Stacks, Queues Trees, Hashing, Priority Queues Sorting Graph Algorithms Algorithm Design Techniques

# Algorithm Analysis (Chapter 2)

- Asymptotic behavior: Big-oh, Big-theta, Big-omega
- Analysing loops
- Recurrences



#### Highlights of CS1102S

Java API Support for Data Structures Outlook to Other Modules

#### Algorithm Analysis

Lists, Stacks, Queues Trees, Hashing, Priority Queues Sorting Graph Algorithms Algorithm Design Techniques

# Example of Recurrence

### Runtime T(N)

$$T(1) = 1$$
  
 $T(N) = 2T(N/2) + N$ 



#### Highlights of CS1102S

Java API Support for Data Structures Outlook to Other Modules

#### Algorithm Analysis

Lists, Stacks, Queues Trees, Hashing, Priority Queues Sorting Graph Algorithms Algorithm Design Techniques

## Example of Recurrence

#### Runtime T(N)

$$T(1) = 1$$
  
 $T(N) = 2T(N/2) + N$ 

#### Theorem

$$T(N) = O(N \log N)$$



# Lists, Stacks, Queues (Chapter 3)

- Collections (add, contains, remove)
- Lists: indexed elements
  - ArrayList: Implementation based on resizable arrays
  - LinkedList: Implementation based on chains of objects
- Stacks and queues: position-oriented
  - Stack: Last-In, First-Out (LIFO)
  - Queue: First-In, First-Out (FIFO)

# Trees (Chapter 4)

- Trees ubiquitous in CS (e.g. expression trees)
- Search trees for efficient collections of ordered elements
- Average insertion/retrieval time:  $O(\log N)$
- Worst case: O(N) (linked list)

# Hashing (Chapter 5)

- Collections that exploit mapping of elements (keys) to (nearly) unique hash values
- Separate chaining: keep linked lists of colliding elements
- Linear/quadratic probing; double hashing



## Priority Queues (Chapter 6)

- Collection of ordered elements with efficient deleteMin and insert
- Idea: use complete binary tree with heap property (implemented by an array)
- insert:  $O(\log N)$  and on average using 2.607 comparisons
- deleteMin: O(log N)

## Sorting (Chapter 7)

- Insertion sort, bubble sort: exchanging adjacent elements:  $O(N^2)$
- Shellsort: use larger step size:  $\Theta(N^{3/2})$
- Heapsort: Use priority queue for sorting, re-using shrinking array: O(N log N)
- Mergesort: Divide-and-conquer: Split in half, and merge: O(N log N)
- Quicksort: Divide-and-conquer: Split using pivot, merge trivial: average and best O(N log N), worst: O(N<sup>2</sup>)



# **Graph Algorithms (Chapter 8)**

Definitions: Section 9.1

Topological sort: Section 9.2

Shortest-Path: 9.3 (excluding 9.3.4, 9.3.5, 9.3.6)

Network Flow: 9.4

Minimum Spanning Tree: 9.5.1

Intro to NP-Completeness: 9.7

## Algorithm Design Techniques

- Greedy algorithms: example Huffman codes
- Divide and conquer: sorting, closest-points
- Dynamic programming: optimal binary search tree
- Backtracking algorithms: turnpike reconstruction, games

## **External Algorithms**

- B-trees: 4.7
- External sorting: 7.10 (excluding 7.10.5 and 7.10.6)

- Highlights of CS1102S
- Java API Support for Data Structures
  - Collections, Lists, Iterators
  - Trees
  - Hashing
  - PriorityQueue
  - Sorting
- Outlook to Other Modules

## The Top-level Collection Interface

```
public interface Collection <Any>
       extends Iterable <Any>
    int size();
    boolean isEmpty();
    void clear();
   boolean contains (Any x);
    boolean add(Any x); // sic
    boolean remove(Any x); // sic
    java.util.lterator <Any> iterator();
```

### The List Interface in Collection API

```
public interface List<Any>
       extends Collection < Any>
 Any get(int idx);
 Any set(int idx, Any newVal);
 void add(int idx, Any x);
 void remove(int idx);
  ListIterator < Any | listIterator (int pos);
}
```

## ArrayList and LinkedList

### **Iterators**

```
public interface Iterator <Any> {
  boolean hasNext( );
  Any next( );
  void remove( );
}
```

### ListIterators

### **TreeSet**

- Implements Collection
- Guarantees  $O(\log N)$  time for add, remove and contains

# AbstractMap<K,V>

#### **Basic operations**

- V get(K key): Returns the value to which the specified key is mapped.
- V put(K key, V value): Associates the specified value with the specified key in this map.

## AbstractMap<K,V>

#### **Basic operations**

- V get(K key): Returns the value to which the specified key is mapped.
- V put(K key, V value): Associates the specified value with the specified key in this map.

#### Other operations

containsKey(key), containsValue(val), remove(key)

## TreeMap

- Extends AbstractMap
- Guarantees O(log N) time for put, get, containsKey, containsValue, remove

## HashMap

- Extends AbstractMap
- Uses separate chaining with rehashing
- Rehashing is governed by initial capacity and load factor, set in constructor

### HashSet

Implements Collection using HashMap

# **PriorityQueue**

- Implements Collection
- Efficient implementation of heap data structure
- Operation names:
  - deleteMin is called "poll"
  - insert is called "add" (of course)

# Sorting

Generic sorting supported by class Collections

# Sorting

- Generic sorting supported by class Collections
- Uses mergesort in order to minimize number of comparisons

## Sorting

- Generic sorting supported by class Collections
- Uses mergesort in order to minimize number of comparisons
- Sorting of built-in numerical types supported by class Arrays

## Sorting

- Generic sorting supported by class Collections
- Uses mergesort in order to minimize number of comparisons
- Sorting of built-in numerical types supported by class Arrays
- Uses efficient implementation of quicksort, to take advantage of tight inner loop.

## Algorithm-related Modules

- CS3233 Competitive Programming
- CS3230 Design and Analysis of Algorithms
- CS4231 Parallel and Distributed Algorithms
- CS5206 Foundation in Algorithms

# Programming-languages-related Modules

- CS2104 Programming Language Concepts
- CS3210 Parallel Computing
- CS3211 Parallel and Concurrent Programming
- CS4215 Programming Language Implementation
- CS4216 Constraint Logic Programming
- CS5205 Foundation in Programming Languages