CS 367 - Introduction to Data Structures  
Week 15, 2017

Final Exam
- Saturday, May 6th, 12:25 to 2:25 pm
- Lecture 1: Room B10 of Ingraham Hall
- Lecture 2: Room 105 of Psychology Building
- Lecture 3: Room 1351 of Chemistry Hall
- UW ID required
  IMPORTANT: Bring your UW Photo ID card and go to correct exam room!
  Students without their UW-ID must wait until students with their ID have been checked in.
- See posted exam information
- Solution to sample questions will be posted on Canvas by 8am on Study Day (Friday)

Homework 10 due 10 pm Tuesday, May 2nd

Program 5 due 10 pm Thursday, May 4th
P5 quiz (is survey on Team experiences) – complete individually by midnight Monday May 8th

Verify that your scores are correct on Canvas
Send your instructor an email if there is an inconsistency.

Last Week:
Finish Hashing, Tree Map vs. Hash Map, Basic Sorts, bubble sort, insertion sort, and selection sort

This Week:
Read: finish Sorting
Better Sorts
- heap sort
- merge sort
- quick sort
Stable Sorts
Sorting in Java

Radix Sort
Sorting out Sorting
Course Overview Sheets
Final Exam Info
Evaluations (please bring a device that can connect to Internet – online eval form)
Heap Sort

Idea

Analysis
Merge Sort

Idea

Analysis
Quick Sort

Idea

Analysis
Quick Sort (cont.)

Choosing a Good Pivot

Quick Sorting the Array with Partitioning

6  1  5  9  3  5  4  3  7  6  2  8  2
Stable Sorts

What do you notice about the sorting of the following three lists of names?

UNSORTED!

Jane Jetson    Barney Rubble    Stewie Griffin
Elroy Jetson   Elroy Jetson   Elroy Jetson
Homer Simpson  George Jetson  George Jetson
Marge Simpson  Homer Simpson  Jane Jetson
Stewie Griffin Jane Jetson   Judy Jetson
Judy Jetson    Judy Jetson   Barney Rubble
George Jetson  Marge Simpson  Homer Simpson
Barney Rubble  Stewie Griffin  Marge Simpson
Sorting in Java

In java.util

    Collections.sort(List)

Arrays.sort(array_to_sort)
Radix Sort

Assumptions

number of items (N):

range of unique digits (RANGE):

length of item’s sequence of digits (LEN)

Idea

Sort the following integers:

121 367 354 873 777 333 123 222 411 262 897

What is N? RANGE? LEN?

Pass 1:

0 1 2 3 4 5 6 7 8 9

Pass 2:

0 1 2 3 4 5 6 7 8 9

Pass 3:

0 1 2 3 4 5 6 7 8 9
Radix Sort

Algorithm

```
List[] digitQ = new List[RANGE]

for (i = 0; i < RANGE; i++)
    digitQ[i] = new Queue()

for (pos = LEN-1; pos >= 0; pos--)

    for (j = 0; j < N; j++)
        let x = digit in pos position of the item in A[j]
        digitQ[x].enqueue(A[j])

index = 0
for (j = 0; j < RANGE; j++)

    while (!digitQ[j].isEmpty())
        A[index] = digitQ[j].dequeue()
    index++
```

Complexity
Abstract Data Types (ADTs) and Data Structures (DS)

ADT
DS

Layout of Collection

- Linear

  List
  array, SimpleArrayList, shadow array
  chain of nodes, Listnode, SimpleLinkedList
tail, header, doubly linked, circularly

  Stack
  Queue
  Deque
  circular array

- hierarchical

  general tree, Treenode
  binary tree, BinaryTreenode
  binary search tree, BSTnode
  IntervalTree (from Program 4)
balanced search tree
red-black tree

  PriorityQueue
  heap

- graphical

  Graph
  Graphnode
  adjacency matrix
  adjacency list

Orientation of Operations

- position oriented - operations occur at a specified position

  list, stack (top), queue (front/rear), deque (“double ended”)

- value oriented - operations occur at position determined by item’s key value

  sorted list
  search trees

  Map
  hash table

- hybrid?
  PriorityQueue
  heap
  hash table
Algorithms

Operations on ADTs/data structures

insert, lookup, delete

Recursion

vs. iteration
rules, guiding questions
call stack trace
execution tree trace

Traversing

list
tree level pre/in/post
graph DFS (stack) BFS (queue) spanning trees

Searching

linear O(N)
binary O(logN)

Hashing

hash function: hash code (extracting, weighting, folding) → hash index (compressing)
table size: prime size, load factor, rehashing
collisions: open addressing, buckets

Graphs

topological ordering
Dijkstra’s (priority queue)

Sorting

basic O(N^2): bubble, insertion, selection
better O(NlogN): heap, merge, quick
stable sorts
Complexity

Complexity

1, logN, N, NlogN, N^2, N^3, 2^N, N!

time: abstract, dominant ops

space: memory

worst/average/best-case

big-O

Determining Complexity

informal
constant
linear
quadratic

code
loops
method calls

time equation
simplify

recurrence equations
base T() =
recursive T(N) = + T(

equations → table, guess solution → verify → complexity

Caveats

small problem size
same complexity
Java Concepts

Primitives vs. References

Command-line Arguments

Exceptions

throw
try/catch/finally
throws (checked vs unchecked)
defining

Programming for Generality

Object
generics

Interfaces

Comparable, compareTo
ADTs

Iterators

Iterable: iterator()
Iterator: hasNext(), next()

indirect
direct

Package Visibility

Java Collections Framework

Iterable<T>, Iterator<E>
List<T>: ArrayList<T>, LinkedList<T>
Vector<E>, Stack<E>
Hashtable<K,V>
Map<K,V>: TreeMap<K,V>, HashMap<K,V>
Set<E>: TreeSet<E>, HashSet<E>
Course Evals

Go to: http://aefis.wisc.edu

Course: CS 367 lecture 001 1 pm TR (Deb -- afternoon)
Course: CS 367 lecture 002 11am TR (Deb -- morning)
Course: CS 367 lecture 003 11am MWF (Alexi -- MWF)

Questions to Consider

What was effective in helping you learn?

What wasn’t effective in helping you learn?

What would you use to evaluate the extent of your learning?

If you were instructor what would you do differently?