CS 367 - Introduction to Data Structures
Week 14, 2017

Final Exam
- Saturday, May 6th, 12:25 pm to 2:25 pm
- UW ID required
- See posted exam information

Program 5 due 10 pm Thursday, May 4th

Homework 10 due 10 pm Tuesday, May 2nd

Last Week Graphs: topological order, Dijkstra’s Algorithm, Hashing: terminology, good hash functions, choosing a table size

This Week
Hashing
- choosing table size (from last time)
- expanding a hash table
- handling collisions
Java Support for Hashing
Tree Map vs. Hash Map

Sorting
Intro
Basic Sorts
- bubble sort
- insertion sort
- selection sort
Better Sorts
- heap sort
- merge sort

Next Week
Read: continue Sorting
Better Sorts
- heap sort
- merge sort
- quick sort
Course Evaluations
Final Exam Review
Resizing the Hash Table

Naïve Expand

30  17  88

Rehashing

1.

2.

Complexity
Collision Handling using Open Addressing

Open Addressing

Linear Probing

166
359
263

440 266 124 246 337 351
Collision Handling using Open Addressing

Quadratic Probing

166
359
263

440 | 266 | 124 | 246 | 337 | 351

Double Hashing

probe sequences assuming $H_k$ is index 0:

<table>
<thead>
<tr>
<th>Step size</th>
<th>Table size 10</th>
<th>Table size 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Collision Handling using Buckets

Buckets

Array Buckets

“Chained” Buckets

Tree Buckets
Java API Support for Hashing

hashCode method
- method of Object class
- returns an int
- default hash code is BAD - computed from object’s memory address

Guidelines for overriding hashCode:

Hashtable<K, V> and HashMap<K, V> class
- in java.util package
- implement Map<K, V> interface
  \[ K \]
  \[ V \]
  operations:

- constructors allow you to set
  initial capacity (default = 16 for HashMap, 11 for HashTable)
  load factor (default = 0.75)
- handles collisions with chained buckets
- HashMap only:
- Hashtable only:
## TreeMap vs HashMap

<table>
<thead>
<tr>
<th>TreeMap</th>
<th>HashMap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sorting

Problem

Solution

Complexity

In-Place Sorts

Basic In-Place Comparison Sorts
Bubble Sort

Idea

Psuedocode

```java
int passes = A.length-1;
for (int i = 0; i < passes; i++) {
    for (int j = A.length–1; j > i; j--) {
        if (A[j] < A[j-1]) {
            swap(A[j], A[j-1]);
        }
    }
}
```

Analysis

<table>
<thead>
<tr>
<th>kind of array</th>
<th>best case</th>
<th>worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td># comparisons</td>
<td></td>
<td></td>
</tr>
<tr>
<td># swaps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Insertion Sort

Idea

Psuedocode (linear insertion)

```java
for (int i = 1; i < A.length; i++) {
    int temp = A[i];
    int j;
    for (j = i-1; j >= 0 && A[j] > temp; j--)
    A[j+1] = temp;
}
```

Analysis

<table>
<thead>
<tr>
<th>kind of array</th>
<th>best case</th>
<th>worst case</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th># comparisons</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th># shifts</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>total</th>
</tr>
</thead>
</table>
Selection Sort

Idea

Psuedocode

```java
int passes = A.length-1;
for (int i = 0; i < passes; i++) {
    int minIndex = i;
    for (int j = i+1; j < A.length; j++) {
        if (A[j] < A[minIndex])
            minIndex = j;
    }
    swap(A[minIndex],A[i]);
}
```

Analysis

<table>
<thead>
<tr>
<th>kind of array</th>
<th>best case</th>
<th>worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td># comparisons</td>
<td></td>
<td></td>
</tr>
<tr>
<td># swaps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>