CPSC 223
Algorithms & Data Abstract Structures

Lecture 9:
Midterm Overview, Exercises

Today …
• Quiz 5
• Midterm overview
• Exercises 1-3
• Homework 4
Midterm Overview …

Midterm

- There will be about **8 questions**
- Open book / open notes
- Closed computer / smart phone / etc.
- Be sure you understand:
  - Answers to quizzes and exercises
  - Answers to questions on slides
- Be sure you have done the *reading assignments*
- Worth 10% of your grade
Topics we’ve covered

• ADT vs. Data Structure
  – Also, modularity and data abstraction

• Inheritance
  – Basic is-a relationships
  – Private vs. public vs. protected
  – Constructors
  – Hiding and Overriding
  – Dynamic vs. Static Binding
  – Basic templates

Topics we’ve covered

• Dynamic memory (pointers)
  – Basic use of pointers
  – Linked lists
  – Constructors, destructors, copy constructors

• Operator overloading
  – Signatures, lvalues/rvalues
  – Operator==, operator<, etc.
  – Assignment operator
Topics we’ve covered

- Constants
  - Function constants
  - Const vs. non-const arguments
  - Passing and returning values by reference

- Algorithm analysis
  - Basic detailed (nitty-gritty) analysis
  - Basic definitions for big $O$
  - Some examples

Topics we’ve covered

- Binary search

- Sorting
  - Selection sort
  - Bubble sort
  - Insertion sort (linear and binary)
  - Best, worst, average big $O$ costs
Questions?

Sorting Exercises
Evaluating Sorting Algorithms

Number of comparisons
- How many pairs of items are compared within a pass

Number of moves
- How many times do we move list items
- Within a list or to a temporary variable

• We often do “swaps”

```c
void swap(int& x, int& y) // here just integers
{
    int tmp = x;
    x = y;
    y = tmp;
}
```

How many moves in a swap?
3

Selection Sort

The basic idea
- Select (i.e., “find”) the largest item
- Swap it with the last item in the list
- Repeat with items 0 to n – 2
- Stop when only one item left

<table>
<thead>
<tr>
<th>Initial list: (n=4)</th>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 10 14 13</td>
<td>13 10 14 29</td>
<td>13 10 14 29</td>
<td>13 10 14 29</td>
</tr>
<tr>
<td>13 10 14 29</td>
<td>13 10 14 29</td>
<td>10 13 14 29</td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort (based on textbook)

```java
void selectionSort(Entry theArray[], int n) {
    for(int i = n - 1; i >= 1; i--) {   // passes 1..n-1
        int index = 0;
        for(int j = 1; j < i + 1; j++) {   // passes 1..i
            if(theArray[j] >= theArray[index])
                index = j;                       // not 1 move
        } // end inner for
        swap(theArray[index], theArray[i]);     // 3 moves
    } // end outer for
}
```

Bubble Sort

The basic idea:

– Compare adjacent items …
– Exchange items if they are out of order
– Repeat for $n - 1$ passes
– Each pass $p$ requires $n - p$ comparisons

<table>
<thead>
<tr>
<th>Initial list: $(n=4)$</th>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>29 10 14 13</td>
<td>10 14 13 29</td>
<td>10 13 14 29</td>
</tr>
<tr>
<td>10 29 14 13</td>
<td>10 14 13 29</td>
<td>10 13 14 29</td>
<td></td>
</tr>
<tr>
<td>10 14 29 13</td>
<td>10 13 14 29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 14 13 29</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Bubble Sort (based on textbook)**

```c
void bubbleSort(Entry theArray[], int n)
{
    bool sorted = false;
    for(int i = 1; i < n && !sorted; i++) {
        sorted = true;
        for(int j = 0; j < (n - i); j++) {
            if(theArray[j] > theArray[j+1]) {
                swap(theArray[j], theArray[j+1]);
                sorted = false;
            }
        } // end inner for
    } // end outer for
} // end outer for
```

**Insertion Sort**

**The basic idea:**

- Partition list into sorted and unsorted regions
- Select first item in unsorted region
- Insert item into the right location of sorted region
- Shift larger items one location forward in the list

---

**Initial list: (n=4)**

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 10 14 13</td>
<td>10 29 14 13</td>
<td>10 14 29 13</td>
</tr>
<tr>
<td>29 10 14 13</td>
<td>10 29 14 13</td>
<td>10 14 29 13</td>
</tr>
<tr>
<td>10 29 14 13</td>
<td>10 14 29 13</td>
<td>10 13 14 29</td>
</tr>
</tbody>
</table>
Insertion Sort (based on textbook)

```java
void insertionSort(Entry theArray[], int n) {
    for(int i = 1; i < n; i++) {
        Entry next = theArray[i];
        int j = i; // insertion index
        while(j > 0 && theArray[j-1] > next) {
            theArray[j] = theArray[j-1]; // shift
            j = j - 1;
        } // end inner for
        theArray[j] = next;
    } // end outer for
}
```

Binary Search

- Given a sorted list, **binary search** finds an element in $O(\log n)$ time

- Search for an element:
  1. Pick middle element of the list
  2. If middle element == key, then found match
  3. If middle element > key, search left half of list
  4. If middle element < key, search right half of the list

- Can implement using recursion or looping
Evaluating Sorting Algorithms

An example of why counting comparisons separately from moves is useful …

– A variant of insertion sort is “binary” insertion sort
– Here we find the insert location using a binary search
– Comparisons go from $O(n)$ on each pass to $O(\log n)$!
– The algorithm is still $O(n^2)$ … why?

• Many of the algorithms we discuss have variants