CSCI 136
Data Structures & Advanced Programming

Lecture 10
Fall 2018
Instructors: Bill & Bill
Last Time

- Mathematical Induction
  - For algorithm run-time and correctness
- More About Recursion
  - Recursion on arrays; helper methods
Today’s Outline

• Finish Binary Search & Induction
• Basic Sorting
  • Bubble, Insertion, Selection Sorts
  • Including proofs of correctness
• The Comparable Interface
Example: Binary Search

• Given an array $a[]$ of positive integers in increasing order, and an integer $x$, find location of $x$ in $a[]$.
  • Take “indexOf” approach: return -1 if $x$ is not in $a[]$

```java
protected static int recBinarySearch(int a[], int value, int low, int high) {
    if (low > high) return -1;
    else {
        int mid = (low + high) / 2; //find midpoint
        if (a[mid] == value) return mid; //first comparison
        else if (a[mid] < value) //search upper half
            return recBinarySearch(a, value, mid + 1, high);
        else //search lower half
            return recBinarySearch(a, value, low, mid - 1);
    }
}
```
Binary Search takes $O(\log n)$ Time

Can we use induction to prove this?

• Claim: If $n = \text{high} - \text{low} + 1$, then recBinSearch performs at most $c \times (1 + \log n)$ operations, where $c$ is twice the number of statements in recBinSearch.

• Base case: $n = 1$: Then low = high so only $c$ statements execute (method runs twice) and $c \leq c(1 + \log 1)$

• Assume that claim holds for some $n \geq 1$, does it hold for $n+1$? [Note: $n+1 > 1$, so low < high]

• Problem: Recursive call is not on $n$—it’s on $n/2$.

• Solution: We need a better version of the PMI….
Mathematical Induction

Principle of Mathematical Induction (Strong)

Let P(0), P(1), P(2), ... Be a sequence of statements, each of which could be either true or false. Suppose that, for some $k \geq 0$

1. P(0), P(1), ... , P(k) are true, and
2. For all $n \geq k$, whenever P(1), P(2), ... , P(n) are true, then so is P(n+1).

Then all of the statements are true!
Binary Search takes $O(\log n)$ Time

Try again now:

- Assume that for some $n \geq 1$, the claim holds for all $k \leq n$, does claim hold for $n+1$?
- Yes! Either
  - $x = a[mid]$, so a constant number of operations are performed, or
  - RecBinSearch is called on a sub-array of size at most $n/2$, and by induction, at most $c(1 + \log(n/2))$ operations are performed.
    - This gives a total of at most $c + c(1 + \log(n/2)) = 2c + c \log(n/2)) = 2c + c(\log n - \log 2) = c(1 + \log n)$ statements
Notes on Induction

• Whenever induction is needed, strong induction can be used
• The numbering of the propositions doesn’t need to start at 0
• The number of base cases depends on the problem at hand
  • Enough are needed to guarantee that the smallest non-base case can be proven using only the base cases
Bubble Sort

• First Pass:
  • (5 1 3 2 9) → (1 5 3 2 9)
  • (1 5 3 2 9) → (1 3 5 2 9)
  • (1 3 5 2 9) → (1 3 2 5 9)
  • (1 3 2 5 9) → (1 3 2 5 9)

• Second Pass:
  • (1 3 2 5 9) → (1 3 2 5 9)
  • (1 3 2 5 9) → (1 2 3 5 9)
  • (1 2 3 5 9) → (1 2 3 5 9)

• Third Pass:
  • (1 2 3 5 9) → (1 2 3 5 9)
  • (1 2 3 5 9) → (1 2 3 5 9)

• Fourth Pass:
  • (1 2 3 5 9) → (1 2 3 5 9)

http://www.youtube.com/watch?v=lyZQPjUT5B4
http://www.visualgo.net-sorting
Sorting Intro: Bubble Sort

• Simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order
• Repeated until no swaps are needed
• Gets its name from the way larger elements "bubble" to the end of the list

• Time complexity?
  • $O(n^2)$

• Space complexity?
  • $O(n)$ total (no additional space is required)

• Let’s write it!
# Sorting Intro: Insertion Sort

- 5 7 0 3 4 2 6 1
- 5 7 0 3 4 2 6 1
- 0 5 7 3 4 2 6 1
- 0 5 7 3 4 2 6 1
- 0 3 5 7 4 2 6 1
- 0 3 5 7 4 2 6 1
- 0 3 4 5 7 2 6 1
- 0 3 4 5 7 2 6 1
- 0 2 3 4 5 7 6 1
- 0 2 3 4 5 7 6 1
- 0 1 2 3 4 5 6 7

http://www.visualgo.net/sorting
Sorting Intro: Insertion Sort

- Simple sorting algorithm that works by building a sorted list one entry at a time
- Less efficient on large lists than more advanced algorithms
- Advantages:
  - Simple to implement and efficient on small lists
  - Efficient on data sets which are already mostly sorted
- Time complexity
  - $O(n^2)$
- Space complexity
  - $O(n)$
Sorting Intro : Selection Sort

http://www.visualgo.net/sorting
(demo is “min” version)

• 11  3  27  5  16
• 11  3  16  5  27
• 11  3  5  16  27
• 5   3  11  16  27
• 3   5  11  16  27

• Time Complexity:
  • $O(n^2)$

• Space Complexity:
  • $O(n)$
Sorting Intro: Selection Sort

- Similar to insertion sort
- Noted for its simplicity and performance advantages when compared to complicated algorithms
- The algorithm works as follows:
  - Find the maximum value in the list
  - Swap it with the value in the last position
  - Repeat the steps above for remainder of the list (ending at the second to last position)
Selection sort uses two utility methods

Uses a swap method

```java
private static void swap(int[] A, int i, int j) {
    int temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}
```

And a max-finding method

```java
// Find position of largest value in A[0 .. last]
public static int findPosOfMax(int[] A, int last) {
    int maxPos = 0;        // A wild guess
    for (int i = 1; i <= last; i++)
        if (A[maxPos] < A[i]) maxPos = i;
    return maxPos;
}
```
Some Skill Testing!

An Iterative Selection Sort
public static void selectionSort(int[] A) {
    for(int i = A.length - 1; i>0; i--)
        int big= findPosOfMax(A,i);
        swap(A, i, big);
    }
}

A Recursive Selection Sort (just the helper method)
public static void recSSHelper(int[] A, int last) {
    if(last == 0) return; // base case
    int big= findPosOfMax(A, last);
    swap(A,big,last);
    recSSHelper(A, last-1);
}
Some Skill Testing!

• Prove: recSSHelper (A, last) sorts elements A[0]…A[last].
  • Assume that maxLocation(A, last) is correct

• Proof:
  • Base case: last = 0.
  • Induction Hypothesis:
    • For k<last, recSSHelper sorts A[0]…A[k].
  • Prove for last:
    • Note: Using Second Principle of Induction (Strong)
Some Skill Testing!

- After call to findPosOfMax(A, last):
  - ‘big’ is location of largest A[0..last]
- That value is swapped with A[last]:
  - Rest of elements are A[0]..A[last−1].
- Since last − 1 < last, then by induction
  - recSSHelper(A, last−1) sorts A[0]..A[last−1].
- Thus A[0]..A[last−1] are in increasing order
Making Sorting Generic

• We need *comparable* items
• Unlike with equality testing, the Object class doesn’t define a “compare()” method 😞
• We want a uniform way of saying objects can be compared, so we can write generic versions of methods like binary search
• Use an interface!
• Two approaches
  • Comparable interface
  • Comparator interface
Comparable Interface

• Java provides an interface for comparisons between objects
  • Provides a replacement for “<“ and “>” in recBinarySearch
• Java provides the Comparable interface, which specifies a method compareTo()
  • Any class that implements Comparable must provide compareTo()

```java
public interface Comparable<T> {
    //post: return < 0 if this smaller than other
    //      return 0 if this equal to other
    //      return > 0 if this greater than other
    int compareTo(T other);
}
```
Comparable Interface

• Many Java-provided classes implement Comparable
  • String (alphabetical order)
  • Wrapper classes: Integer, Character, Boolean
  • All Enum classes

• We can write methods that work on any type that implements Comparable
  • Example: RecBinSearch.java and BinSearchComparable.java
We could write

```java
class CardRankSuit implements Comparable<CardRankSuit> {
    public int compareTo(CardRankSuit other) {
        if (this.getSuit() != other.getSuit())
            return getSuit().compareTo(other.getSuit());
        else
            return getRank().compareTo(other.getRank());
    }
    // rest of code for the class....
}
```
Comparable & compareTo

- The Comparable interface (Comparable<T>) is part of the java.lang (not structure5) package.
- Other Java-provided structures can take advantage of objects that implement Comparable
  - See the Arrays class in java.util
  - Example JavaArraysBinSearch
- Users of Comparable are urged to ensure that compareTo() and equals() are consistent. That is,
  - x.compareTo(y) == 0 exactly when x.equals(y) == true
- Note that Comparable limits user to a single ordering
- The syntax can get kind of dense
  - See BinSearchComparable.java: a generic binary search method
  - And even more cumbersome….