CSCI 136
Data Structures & Advanced Programming

Lecture 8

Fall 2019

Instructors: B&S

Administrative Details

- Remember: First Problem Set is online
- Due at beginning of class on Friday
- Lab 3
  - You *may* work with a partner
    - Fill out the Google Form by 4 pm today (check email!)
  - Come to lab with a plan!
  - Answer questions before lab

Last Time

- Measuring Computational Complexity
- Introduction to Recursion

Today

- More Recursion
- Mathematical Induction (Weak)
- Mathematical Induction (Strong)

Mathematical Induction : Version 2

Principle of Mathematical Induction (Weak)

Let P_0, P_1, P_2, … Be a sequence of statements, each of which could be either true or false.

Suppose that

1. P_0 and P_1 are true, and
2. Whenever P_{n-1} and P_{n-2} are true, then so is P_n.

Then all of the statements are true!

Other versions:

- Can have $k > 2$ base cases
- Doesn't need to start at 0

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[]

```
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
                                        //second 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 $n = high - low + 1$, then recBinSearch performs at most $c\,(1+\log n)$ 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. 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 $n/2$, and by induction, at most $c(1 + \log(n/2))$ operations are performed.
    - This gives a total of at most $c + c(1 + \log(n/2)) = c + c(\log(2) + \log(n/2)) = c + c(\log n) = c(1 + \log n)$ statements

Longest Increasing Subsequence

- Given an array a[] of positive integers, find the length of the largest subsequence of (not necessary consecutive) elements such that for any pair a[i], a[j] in the subsequence, if i<j, then a[i] < a[j].
- Example 10 7 12 3 5 11 8 9 1 15 has 3 5 8 9 15 as its longest increasing subsequence (LIS), so the length is 5.
- How could we find the LIS length of a[]?
- How could we prove our method was correct?
- Let's think....

Longest Increasing Subsequence

- We'll assume all numbers are positive
- (Brilliant) Observation: A LIS for a[1 ... n] either contains a[1] … or it doesn't.
- Therefore, a LIS for a[1 ... n] either
  - Doesn't contain a[1] and is just a LIS for a[2 ... n]
  - Does contain a[1], along with an LIS for a[2 ... n] such that every element in the LIS is > a[1], or
- So the LIS length is either
  - Or the LIS length for a[2..n]
  - 1 + LIS length for a[2..n] where every element in LIS is > a[1]
- So the problem to solve is: find the LISL for a[] given that every element in LIS is at least some *threshold* value

Longest Increasing Subsequence

```
// Pre: curr < arr.length
// Post: returns length of LIS of arr[curr…] having all > threshold
public static int lisIHelper(int[] arr, int curr, int threshold ) {
    if(curr == arr.length -1)
        if (return arr[curr] > threshold) return 1;
        else return 0;
    else
        int usingFirst = 0;
        if(arr[curr] > threshold)
            usingFirst  = 1 + lisIHelper(arr, curr+1, arr[curr]);
        int notUsingFirst = lisIHelper(arr, curr+1, threshold);
        return Math.max(usingFirst, notUsingFirst);
  }
```

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 Preview: 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 substantially sorted
- Time complexity
  - $O(n^2)$
- Space complexity
  - $O(n)$

Sorting Preview: 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 3 5 7 4 2 6 1
- 0 3 4 5 7 2 6 1
- 0 2 3 4 5 7 6 1
- 0 2 3 4 5 6 7 1
- 0 1 2 3 4 5 6 7

Sorting Preview: Selection Sort

- Similar to insertion sort
- Performs worse than insertion sort in general
- 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)

Sorting Preview: Selection Sort

- 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:
  - $O(n)$