Week 4: Time Complexity, Recurrence, Recursive Algorithms

June, 8, 2017

1. **Complexity Review (From CSC165)**

   - Measure running time by counting steps in algorithm
   - Measure as a function $T(n)$ of input size n.
   - $T(n)$ measures **worst-case** running time: maximum number of steps over all input size n
   - Don't care exact step counts
   - Most of the time, we prove bounds on $T(n)$ using asymptotic notation
   - Upper bound $T(n) \in \mathcal{O}(f(n)) \Leftrightarrow \exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow T(n) \leq cf(n)$
   - Lower bound $T(n) \in \Omega(f(n)) \Leftrightarrow \exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow T(n) \geq cf(n)$
   - Tight bound $T(n) \in \Theta(f(n)) \Leftrightarrow T(n) \in \mathcal{O}(f(n)) \wedge T(n) \in \Omega(f(n))$

   Example

```
def LS(A, x):
    """ Given a list A, return an index i such that x == A[i]. Otherwise return -1"""
    i = 1
    while i < len(A):
        if A[i] == x:
            return i
        i = i + 1
    return -1
```

   $T(n) =$ _____ $\in$

```
def foo(n):
    for i in range(n):
        pass
    return 42
```

   $T(n) =$ _____ $\in$

   Is $T(n)$ in polynomial time?

2. **Recursive Algorithms**

```
def recLinearSearch(A, x):
    if not A:
        return -1
    elif A[0] == x:
        return 0
    else:
        return 1 + recLinearSearch(A[1:], x)
```

Recurrence: _____
Unwind your recurrence and find a closed form.

Prove your closed form by induction.

Divide-and-Conquer: divide the input and "conquer" the pieces
Recursively breaking down a problem into two or more sub-problems of the same or related type
until these become simple enough to be solved directly.
The solutions to the sub-problems are then combined to give a solution to the original problem.

```
def MergeSort(A, b, e):
    if b == e:
        return
    m = (b + e) // 2
    MergeSort(A, b, m)
    MergeSort(A, m + 1, e)
    # merge sorted A[b..m] and A[m+1..e] back into A[b..e]
    B = [i for i in A]
    c = b
    d = m + 1
    for i in range(b, e + 1):
        if d > e or (c <= m and B[c] < B[d]):
            A[i] = B[c]
            c = c + 1
```

```
        else: # d <= e and (c > m or B[c] >= B[d])
            A[i] = B[d]
            d = d + 1
```

Recurrence: _____

Unwind (using repeated substitution) your recurrence and find a closed form.

Prove upper bound

Prove lower bound

```
def recBinSearch(x, A, b, e):
    if b == e:
        if x <= A[b]:
            return b
        else:
            return e + 1
    else:
```

```
    m = (b + e) // 2 # midpoint
    if x <= A[m]:
        return recBinSearch(x, A, b, m)
    else:
        return recBinSearch(x, A, m + 1, e)
```

Recurrence: _____

Unwind (using repeated substitution) your recurrence and find a closed form.

Prove upper bound

Prove lower bound

Master Theorem

$$T(n) = \begin{cases} k & \text{if} \quad n \leq B \\ a_1 T(\lceil \frac{n}{b} \rceil) + a_2 T(\lfloor \frac{n}{b} \rfloor) + f(n) & \text{if} \quad n > B \end{cases}$$

If $a = a_1 + a_2$ and $f \in \Theta(n^d)$, then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if} \quad n \leq B \\ \Theta(n^d \log n) & \text{if} \quad a = b^d \\ \Theta(n^{\log_b a}) & \text{if} \quad a > b^d \end{cases}$$