Lecture 1

Fibonacci

```
def fib(n):
  if n == 0: return 0
  elif n == 1: return 1
  else: return fib(n-1) + fib(n-2)
```

- `T(0) = 2` (if, return)
- `T(1) = 3` (if, elif, return)
- `T(n) = T(n-1) + T(n-2) + 7` (if, elif, else, +, fib, fib, return)

Notations

O −notation

- *Upper Bound*, that is function grows no faster than $cg(n)$
- $f \in O(g)$ if there is $c > 0$ and $n_0 > 0$ such that $\forall n \geq n_0 : 0 \leq f(n) \leq cg(n)$
  - The intuition is that for a large enough n, there is a function g and constant c, such that $f(n)$ is always lesser than g.

Ω −notation

- *Lower Bound*, that is function grows at least as fast as $cg(n)$
- $f \in O(g)$ if there is $c > 0$ and $n_0 > 0$ such that $\forall n \geq n_0 : 0 \leq cg(n) \leq f(n)$

Θ −notation

- Both upper and lower bounded by $cg(n)$
- $f \in O(g)$ if there is $c_1, c_2 > 0$ and $n_0 > 0$ such that $\forall n \geq n_0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$

o-notation

- Strict upper bound, $0 \leq f(n) < cg(n)$

ω-notation

- Strict lower bound, $0 \leq cg(n) < f(n)$

Orders of Common Functions

- $O(1)$
- $O(\log \log n)$
- $O(\log n)$
- $O((\log n)^c)$
- $O(n^c), 0 < c < 1$
- $O(n)$
- $O(n \log^* n)$
- $O(n \log n) = O(\log n!)$
- $O(n^2)$
- $O(n^c)$
- $O(c^n)$
- $O(n!)$

Limits

- $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = 0 \Rightarrow f(n) = o(g(n))$
  - By defn of limits, $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = 0$ means:
    - $\forall \varepsilon > 0, \exists n_0 > 0$, s.t. $\forall n \geq n_0, \frac{f(n)}{g(n)} < \varepsilon$
    - Hence, $f(n) < c * g(n)$
- $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) < \infty \Rightarrow f(n) = O(g(n))$
- $0 < \lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) < \infty \Rightarrow f(n) = \Theta(g(n))$
- $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) > 0 \Rightarrow f(n) = \Omega(g(n))$
- $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \infty \Rightarrow f(n) = \omega(g(n))$

Lecture 2

Merge Sort

- MERGE-SORT $A[1..n]$
  1. If $n = 1$, done
  2. Recursively sort $A\left[1..\left\lceil\frac{n}{2}\right\rceil\right]$ and $A\left[\left\lceil\frac{n}{2}\right\rceil + 1..n\right]$
  3. Merge the 2 sorted lists
- $T(n) =$
  - $\Theta(1)$ if $n = 1$
  - $2T\left(\frac{n}{2}\right) + \Theta - (n)$ if n > 1

Solving Recurrances

Telescoping Method

- For a sequence $\sum_{k=0}^{n-1}(a_k - a_k + 1) = \begin{matrix} a_0 - a_1 \\ a_1 - a_2 \\ a_{n-1} - a_n \end{matrix} = a_0 - a_n$
- E.g. $T(n) = 2T\left(\frac{n}{2}\right) + n$
  - $\frac{T(n)}{n} = \frac{T\left(\frac{n}{2}\right)}{\frac{n}{2}} + 1$
  - $\frac{T(n)}{n} = \frac{T(1)}{1} + \log n$
  - $T(n) = n \log n$
- General Solution
  - $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
  - $\frac{T(n)}{g(n)} = \frac{T\left(\frac{n}{b}\right)}{g\left(\frac{n}{b}\right)} + h(n)$

- And then sum up occurances of $h(n)$.

Recursion Tree
- Draw the tree, where each node is the $f(n)$ value.
- Figure out the height of the tree and number of leaves

Master Theorem
- Put recurrance in the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- $a \geq 1, b > 1, f$ is asymptotically positive
- Compare $f(n)$ and $n^{\log_b a}$

Cases
1. $f(n) = O\left(n^{\log_b a - \varepsilon}\right), \varepsilon > 0$ (If $\varepsilon = 0$, case 2)
   - $f(n)$ grows polynomically slower than $n^{\log_b a}$
   - $\therefore T(n) = \Theta\left(n^{\log_b a}\right)$
2. $f(n) = \Theta\left(n^{\log_b a} \log^k n\right), k \geq 0$
   - $f(n)$ and $n^{\log_b a}$ grow similar rates
   - $\therefore T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
3. $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right), \varepsilon > 0$
   - $f(n)$ satisfies $af\left(\frac{n}{b}\right) \leq cf(n), c < 1$
   - The regularity condition satisifes that sum of subproblems $< f(n)$
   - $f(n)$ grows polynomically faster than $n^{\log_b a}$
   - $\therefore T(n) = \Theta(f(n))$

Substitution
- For Upper bound: $T(n) \leq cf(n)$
- For Tight bound: $T(n) \leq c_2 n^2 - c_1 n$

Solve $T(n) = 4T\left(\frac{n}{2}\right) + n$
- Guess $T(n) = O(n^3)$
  - Constant c s.t. $T(n) \leq cn^3, n \geq n_0$
- By induction
  - $c = \max\{2, q\}, n_0 = 1$
  - Base Case ($n = 1$): $T(1) = q \leq c(1)^3$
  - Recursive Case ($n > 1$):
    - Strong induction, assume $T(k) \leq ck^3, n > k \geq 1$
    - $T(n) = 4T\left(\frac{n}{2}\right) + n \leq 4c\left(\frac{n}{2}\right)^3 + n = \left(\frac{c}{2}\right)n^3 + n \leq cn^3$

Tutorial 2

Telescoping
- $T(n) = 4T\left(\frac{n}{4}\right) + \frac{n}{\log n}$
- $\frac{T(n)}{n} - \frac{T\left(\frac{n}{4}\right)}{\frac{n}{4}} = \frac{1}{\log n}$
- Let $a_i = \frac{T(4^i)}{4^i}, i = \log_4 n, a_i - a_{i-1} = \frac{1}{2i}$
- $\sum_{m=0}^{i-1}(a_{m+1} - a_m) = \frac{1}{2i} + \frac{1}{2(i-1)} + \ldots + \frac{1}{2} = \frac{1}{2}(H_i)$ (H_i is harmonic)
- $a_i - a_0 = O(\log i)$
- $T(4^i) = O(4^i \log i), T(n) = O(n \log \log n)$

Lecture 3

Correctness of Iterative Algos using Invariants
- Initialization: Invariant is true before first iteration of loop
- Maintenance: Invariant is true at start of loop iteration, it is true at start of the *next iteration* (Induction)
- Termination: Algo at the end gives correct answer

Insertion Sort
- Invariant 1: $A[1..i-1]$ are sorted values of $B[1..i-1]$
- Invariant 2:

Recursive Algorithms
- Usually use mathematical induction on *size of problem*

Binary Search
- Induction on length $n = \text{ub} - \text{lb} + 1$
- Base Case
  - `if n <= 0: return False`
- Induction Step: if $n > 0$, `ub >= lb`
  - Assume algo works for all values `ub-lb+1 < n`
  - `x == A["mid"]: return True`, answer returned correctly
  - `x > A[mid]`, then x is in the array iff it is in `A[mid + 1..ub]`, as A is sorted. Thus by induction answer must be `BinarySearch(A, mid+1, ub, x)`
  - `x < A[mid]`, then x is in the array iff it is in `A[lb..mid-1]`, as A is sorted. Thus by induction answer must be `BinarySearch(A, lb, mid-1, x)`
  - Thus, answer returned is correct

Divide and Conquer
1. Divide problem into smaller subproblems
2. Solve subproblems recursively (conquer)

3. Combine / Use subproblem to get solution to full problem

- $T(n) = aT(\frac{n}{b}) + f(n)$
  - a subproblems
  - Each subproblem is size of atmost $\frac{n}{b}$
  - $f(n)$ is time needed to *divide* problem into subproblems + time to get solution from subproblems *(combine)*

Merge Sort

1. `MergeSort(A[lb..ub])`
2. `If ub = lb` return
3. `If ub > lb` mid $= \frac{\text{ub+lb}}{2}$ $(O(1))$
4. `MergeSort(A[lb,mid]), MergeSort(A[mid+1,ub])` (2 Subproblems, each $\lceil \frac{n}{2} \rceil$)
5. Merge 2 sorted list (Combining: $\Theta(n)$)

$T(n) = 2T(\frac{n}{2}) + O(n)$ Complexity: $\Theta(n * \log n)$

Powering

- $F(a,n) = F(a, \lfloor \frac{n}{2} \rfloor)^2$ if n is even
- $F(a,n) = F(a, \lfloor \frac{n}{2} \rfloor)^2 * F(a,1)$ if n is odd

$T(n) = T(\frac{n}{2}) + O(1)$

Fibonacci

Tutorial 3

Lecture 4

Lower bound for sorting

- Any comparison based sorting runs in $\Omega(n \log n)$
- The tree must contain at least $n!$ leaves for every possible permutation
- Height of binary tree = $\log(n!) = \Omega n \log n$

Tutorial 4

Lecture 5

- Las Vegas Algorithms
  - Output always correct
- Monte Carlo
  - Answer may be incorrect with small probability

Tutorial 5

Lecture 6

LCS

Brute Force

- Check all possible subsequences of A and check if its a subsequence of B and output longest one
- 2^n possible subsequences, Total Time $O(m2^n)$

Recursive

Base Case:

- $\text{LCS}(i, 0) = \emptyset$ (i is index A, j is index B)
- $\text{LCS}(0, j) = \emptyset$ (i is index A, j is index B)

If $a_n = b_m$, then $\text{LCS}(n-1, m-1) :: a_n$ Proof by contradiction

- If last symbol in $S = \text{LCS}(n,m)$ is not same as a_n, then last symbol must be a past of$a_1, ...a_{n-1}$, and $b_1, ...b_{n-1}$.
- S is subsequence of $a_1, ...a_{n-1}$, and $b_1, ...b_{n-1}$.
- Append a_n with S i.e. $S :: a_n$ and get subsequence of length 1 more
- Thus S canont be largest subsequence (Contradiction)
- So far, we only argued a_n must be last symbol in $\text{LCS}(n,m)$
- It is ifne to match a_n with b_m (since a_n is last symbol)
- Therefore $\text{LCS}(n,m) = \text{LCS}(n-1, m-1) :: a_n$

If $a_n \neq b_m, \text{LCS}(n,m) = \max(\text{LCS}(n-1,m), \text{LCS}(n,m-1))$

Tutorial 6

Greedy vs DP

Lecture 7 Greedy

DP Recap

- Express solutions recursively
- Small number(polynomial) of subproblems
- Huge overlap among subproblems, so recursive may take exponential time
- Compute recursive iteratively in bottom up fashion

Greedy

Recast the problem so that only 1 subproblem needs to be solved at each step.

Lecture 8 Amortized Analysis