Topics covered thus far - we will focus more on second half

1.) Induction
2.) Asymptotic Analysis
3.) Divide and Conquer
4.) Greedy Algorithms
5.) Dynamic Programing
6.) Graph representation
7.) Proofs: for greedy choice and suboptimality

Proof by induction

1.) 5pts

$$\sum_{i=1}^{n+1} i \cdot 2^i = n2^{n+2} + 2 \qquad \text{For all integer n >= 0}$$

Solution:

Base cases: n=0, L.S.= 2 =R.S.

Assume f(k) is true, i.e $\sum_{i=1}^{k-1} i2^i = k2^{k+2} + 2$

When n = k+1 $\sum_{i=1}^{k+2} i2^i$

$$\begin{aligned}
&= (k+2)2^{k+2} + \sum_{i=1}^{k-1} i2^i \\
&= (k+2)2^{k+2} + k2^{k+2} + 2 \\
&= k2^{k+2} + 2^{k+3} + k2^{k+2} + 2 \\
&= k2^{k+3} + 2^{k+3} + 2 \\
&= (k+1)2^{k+3} + 2
\end{aligned}$$

Asymptotic Analysis

2.) 5pts

By definition

Find c and M to prove that $2n^5 + 3n^3 + 6 = O(n^5)$.

Solution:

Note $2n^5 + 3n^3 + 6n \leq 2n^5 + 3n^5 + 6n^5$ for $n \geq 1$

$= 11n^5$

So picking $c = 11$, $M = 1$ shows

$2n^5 + 3n^3 + 6 = O(n^5)$

3.) 10 pts Asymptotic analysis continued

T(n)= T(n/3)+n

a.) By recursion tree method (4 pts)

Provide the tightest bound for **T(n)= T(n/3)+n**

$T(n) = T(n/3) + n$

n

$T(n/3)$ $\quad n/3$

$T(n/9)$ $\quad n/9$

I see the pattern that for level i there is $\frac{n}{3^i}$ amount of operations

the depth is: BASE CASE

$$\frac{n}{3^i} = 1$$

$$\text{depth} = i = \log_3 n$$

$$\sum_{i=0}^{\log_3 n} n/3^i = n \sum_{i=0}^{\log_3 n} \frac{1}{3^i} \Rightarrow \text{geo } \frac{1}{1-a} = \frac{1}{2/3} = 3/2$$

$$= n(3/2) \Rightarrow O(n)$$

b.) By substitution (3 pts)

Prove the above with substitution from the solution you found in a c.) By Master theorem (2pts)

$T(n) = T(n/3) + n$

$T(n) \leq cn$

$T(n) \leq c(\frac{n}{3}) + n \leq cn$

$c(\frac{n}{3}) + n \leq cn$

$\frac{c}{3} + 1 \leq c$

$1 \leq 2/3c$

$3/2 \leq C$ $\quad C = 2$

$T(n) = T(n/3) + n$

$f(n) = \Theta(n^1)$

$a = 1 \quad b = 3$

$\log_b a = \log_3 1 \; ? \; 1$

$<$

CASE 3

$\Theta(n)$

4.) **Divide and Conquer**

a.)

. Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

ANSWER: Best algorithm (fastest) asymp. in worst case: Algo. C, Algo A, ~~log~~ Algo B

~~[illegible]~~

WORK:

Algo A $5T(n/2) + O(n)$ $\Rightarrow$ $O(n^{2\,32})$

Algo B $2T(n-1) + O(1)$ $O(2^n)$

Algo C $9T(n/3) + O(n^2)$ $O(n^2 \log n)$

How did I come up with the answer:

Algo A: use MASTERS THM CASE 1

$O(n^{\log_2 5}) \approx O(n^{2.32})$

Algo B: MASTERS DOES NOT WORK, USE Tree Method

Tree — Work at level

$T(n-1)\ T(n-1)$ — 1, 2

$T(n-2)\ T(n-2)\ T(n-2)\ T(n-2)$ — 2^2

work at level i: 2^i

depth: $n - i = 1$, $i = n-1$

$$\sum^{depth} \text{work at level} = \sum_{i=0}^{n-1} 2^i = 2^{(n-1)+1} - 1 = 2^n - 1$$

Algo C: MASTERS THEOREM CASE 2

$O(n^2 \log n)$

How do I know that n^log_2(5) >n^2*logn

proof: Algo C is smaller asympt. then Algo A

$$\lim_{n \to \infty} \frac{n^2 \log n}{n^{2.32}} = \lim_{n \to \infty} \frac{\log n}{n^{0.32}}$$

B.C. its $\frac{\infty}{\infty}$ we must use l hopes rule

$$\lim_{n \to \infty} \frac{\log n}{n^{0.32}} = \frac{1}{n^1} \cdot \frac{1}{n^{-.78}} = \frac{1}{n^{.32}} = \infty$$

b.)

Create a divide and conquer algorithm that finds the 2 largest number of an array in the fastest asymptotic time bound. Provide the recurrence and the closed form solution

Exact same as HW solution

5.) Greedy Algorithm example

(a) Provide a greedy algorithm for the following problem

Selecting breakpoints.

- Road trip from Princeton to Palo Alto along fixed route.
- Refueling stations at certain points along the way.
- Fuel capacity = C.
- Goal: makes as few refueling stops as possible.

(b)

Solution

Greedy Breakpoint Selection Algorithm

```
Sort breakpoints by increasing value:
0 = b_0 < b_1 < b_2 < ... < b_n.

S ← {0}          ⟸ S = breakpoints selected.
x = 0
while (x ≠ b_n)
   let p be largest integer such that b_p ≤ x + C
   if (b_p = x)
      return "no solution"
   x ← b_p
   S ← S ∪ {p}
return S
```

Proof 2 Parts

A. **Greedy choice**
   Goal: Show that given an optimal solution i can insert the greedy choice and have the solution remain optimal. Alternatively every optimal solution has a greedy choice (this is the case with coin changing)

Proof (by contradiction)

1. Let A be the breakpoints chosen by our greedy algorithm such that
2. A= $g_{1,} < g_2 < \ ... \ < g_p$
3. Assume A is not optimal
4. Let OPT be a set of break points in the optimal solution; OPT= $f_{1,} < f_2 < \ ... \ < f_q$
5. |OPT| < |A| based of assumption in line 3
6. $\mathbf{f_0 = g_0, f_1 = g_1, \ldots, f_r = g_r}$ **for largest possible value of r.** (could be r=0)
7. We know that the stop choose by algorithm A will be furthest possible give capacity so it is further then optimal stop chosen.
8. Replace $fr \ with \ g_r \quad OPT' = f_{1,} < f_2 \ < \ ... \ < f_q \ - \{f_r\} + \{g_r\}$. The solution is still feasible and still optimal $|OPT'| = |OPT|$. q.e.d

B. **Suboptimality**

Goal:Show that given an optimal solution if you remove a subset of the solution the remaining solution is an optimal solution for the subset of the problem.

Let OPT= $f_{1,} < f_2 < \ldots < f_{q-1} < f_q$ be the optimal solution to reach f_q . Lets assume for contradiction that A'= $f_{1,} < f_2 < \ldots < f_{q-1}$ is not optimal to reach f_{q-1} (we want to prove that it is). Let B=$g_{1,} < g_2 < \ldots < g_p$ be the optimal way to reach f_{q-1} . Hence |B|<|A'|. Let B'= B U $\{f_q\}$ is a possible way way to reach f_q . Hence |B'| <OPT ===> contradiction and hence A' must be optimal

Given the following greedy algorithm, prove that it is optimal or provide a counterexample:

Given a set of activities and their start and finish times, and you create a greedy algorithm that adds activities based on latest start times.

Solution Proof:
Same as HW

6.) Dynamic

Consider a modification of the rod-cutting problem in which, in addition to a price p_i for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic-programming algorithm to solve this modified problem.

Prove suboptimality

Solution

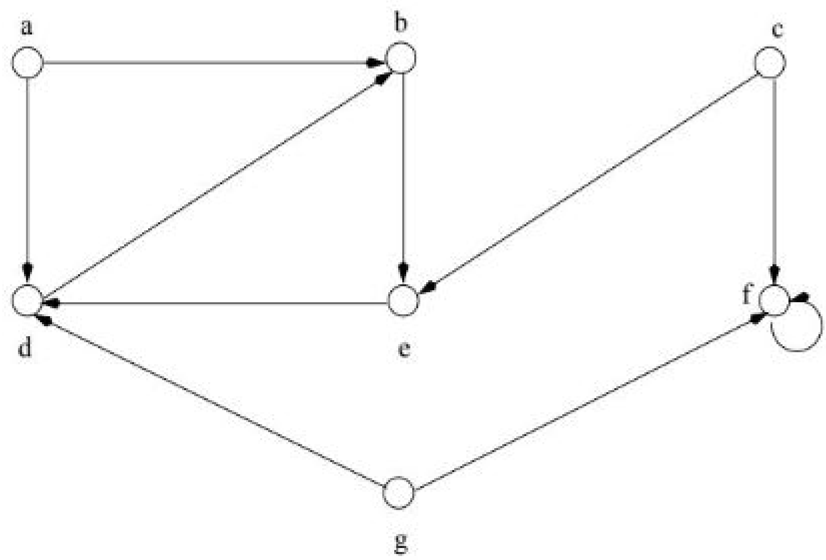
```
MODIFIED-CUT-ROD(p, n, c)
    let r[0..n ] be a new array
    r[0] = 0
    for j = 1 to n
        q = p[j]
        for i= 1 to j-1
                q = max(q, p[i]+r[j-i]-c)
        r[j] = q
    return r[n]
```

Graph questions:

1.)

```
For (v  in   Adj[u])
             Print "v";
```

What is the runt time of the code above if the Graph is stored as an adj. List ?
What about Adj. Matrix



2.)

Please provide the adj list and adj. Matrix of the graph above.