Problem Set 3

CSCI 3104 Spring 2014
Cristobal Salazar
07/22
Partner: Alex Tsankov

Problem 1

a) If we assume a majority of the chips are false, we can't use any strategy to solve this problem because the only strategy that does allow us to select a majority of good chips, throwing away anything that doesn't pass T_1, Test 1, fails with a plurality of bad chips.

All of these tests work by simply magnifying the previous bias of the set. If we put anything that passes $T_1 \in \mathcal{G}$, we know from the law of large numbers that as n approaches infinity, that either $\mathcal{G}$ or $\mathcal{B}$, will ultimately be filled with more bad than good chips. Even if one of them does have more good chips than the other, we won't ever be able to find out because we will always have to contend with the fact that the chips could be lying to us as we are testing, and if more of them lie to us than don't, the scenario at hand, none of the tests work.

b) To find the best way to reduce the problem in half, it helps to put all of the chips in an array. If we go down the list, comparing the i^{th} and $i^{th} + 1$ element, until we find a pair of chips that both return *good*. Using one of those chips, we test the other chips, and we discard the chips that return *good-bad*, *bad-good*, and *bad-bad*. We will at most discard 1 good chip each time we do this, but there is a chance we get rid of 2 bad chips. This means we are getting rid of twice as many bad chips as we are good chips, so only $n/2$ pairwise tests are required to reduce this problem in half.

c) The recurrence relation will be given in the following form: $T(n) = a * T(n/b) + f(n)$. To identify the good chips, we use the process above to reduce the problem in half in $n/2$ pairwise tests. This only requires 1 recursive call, so $a = 1$. $b = 2$ because we are reducing the problem in half. The non-recursive cost is given by $f(n) = n/2$. Given this we find the recurrence relation to be $T(n) = 1 * T(n/2) + n/2$. Using the Master Theorem, we see that $f(n) = \Theta(n^c)$, where $c = 1$. We must test to see if $log_b(a) < c$, where $log_2(1) < 1$ shows us $0 < 1$, so we use case 3 of the Master Theorem. Given this, we plug our $f(n)$ into $\Theta(f(n))$. This gives us $\Theta(n)$.

Problem 2

a) The answer to this question is $O(n^3)$. This is because Dumbledore's algorithm contains a total of three loops. The first loop is on Line 1, and iterates through n. This is $O(n)$. The next loop is inside of the first loop and brings the run time up to $O(n^2)$. The final loop is inside the second one and this requires us to iterate through an array of numbers to add them together and ultimately brings the total run time up to $O(n^3)$.

b) This algorithm can be modelled mathematically with the following summation:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} (j - i + 1) = \frac{n^3}{6} + \frac{n^2}{2} - \frac{2n}{3}$$

With the answer to this summation at hand we can then show that as $n \to \infty$ we get a loosely bounded $\Omega(n^3)$ and paired with our $O(n^3)$ give us an ultimate answer of $\Theta(n^3)$

c) To improve Dumbledore's algorithm we want to minimze the number of times we have to fully sum $A[i] + ... + A[j]$, by only summing it once, then just adding the following $A[j]$'s to the already calculated sum.

```
for i=0 to n
     s = 0
     for j = i+1 to n
          if(s == 0)
               s = sum of array elements A[i] through A[j]
          else
               s += A[j]
          B[i,j] = s
     end
end
```

Problem 3

When we are studying randomized algorithms, we generally use average cases because those are most likely to occur. We can use randomized quicksort as an example, which we randomize to prevent potentially harmful input. The worst case-scenario with quicksort is if we have a last place pivot, n, but this is much less likely than having a pivot in position $n-2$, or $n-100$, both of which are considered average cases. To summarize: we use average case scenarios with randomized algorithms because when it comes to working with randomness, we want to work with what is *most likely*, which is by definition, the average case scenario. The worst-case scenario is too rare for it to be useful for us.

Problem 4

a) See back for recurrence tree.

b) The answer to this question is $\Theta(log(n))$. We can use the Master Theorem to confirm this. This represents a **binary search**. See back for recurrence tree.

c) The answer to this question is $\Theta(n * log(n))$. We can use the Master Theorem to confirm this. This represents a **mergesort** recurrence relation, similar to problem 3. See back for recurrence tree.

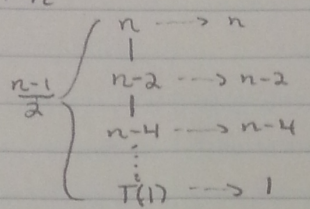
d) The answer to this question is $\Theta(n)$. We can use the Master Theorem to confirm this. This recurrence relation represents **traversing a binary tree and adding each element to an array** .See back for recurrence tree.

Sources

- http://www.bowdoin.edu/ ltoma/teaching/cs231/fall07/Lectures/reccurences.pdf
- http://www.cs.unm.edu/ saia/561-f13/lec/lec3.pdf

Problem 4

a) $T(n) = T(n-2) + n$



$n - 2i = 1$

$i = \frac{n-1}{2}$

The number of nodes a depth i is $\boxed{1}$

The cost of node at depth i is $\boxed{c(n-2i)}$

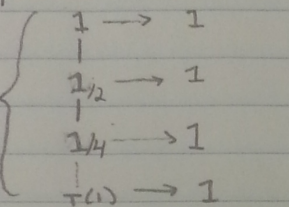
The overall cost at depth i is $\boxed{1 * c(n-2i)}$

given this,

$T(n) = n + (n-2) + (n-4) + \cdots + 1$

which simplifies to $\boxed{\Theta(n^2)}$

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



$\frac{n}{2^i} = 1$

$i = \log_2 n$

# of nodes at depth i is $\boxed{1}$

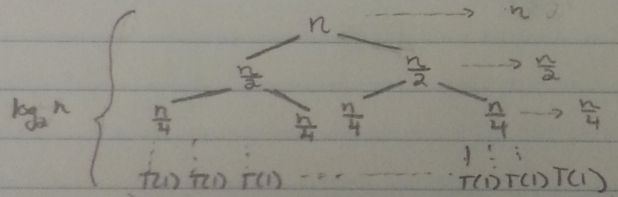
cost of node at depth i is $\boxed{1}$

Total # of node = $\log_2 n$

Because total node is $\log_2 n$ all contributing $T(1)$,

$T(n) = \boxed{\Theta(\log_2 n)}$

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



$\frac{n}{2^i} = 1$

$i = \log_2 n$

# of nodes at i $\boxed{2^i}$

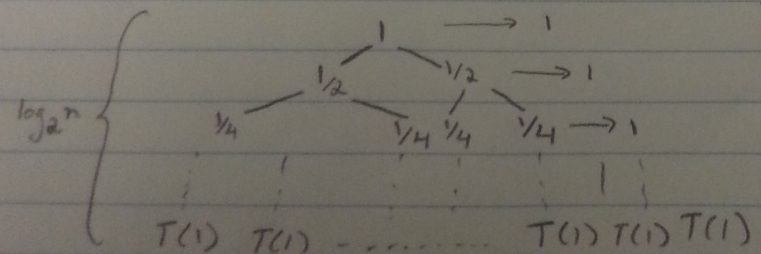
cost of node at i $\boxed{\frac{n}{2^i}}$

Total cost of node of depth i is $2^i * \frac{n}{2^i}$,

$2^{\log_2 n} = n$ # of $T(1)$

$\boxed{T(n) = \Theta(n * \log_2 n)}$

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



$\frac{n}{2^i} = 1$

$i = \log_2 n$

# of nodes at $i = \boxed{2^i}$

cost of node at $i = \boxed{1}$

Total cost at depth i $\boxed{1 * 2^i}$

Total cost = $2^{\log_2 n} = n$

$\boxed{T(n) = \Theta(n)}$