Analysis and Design of Algorithms
Homework 1

Arturo Fornés Arvayo A01227071

January 27, 2017

1. Bounding Summations

   Give tight bounds on the following summations. Assume r $\geq$0 and s $\geq$0 are constants.

   (a) $\sum_{k=1}^{n} k^r$

   We can use integrals to find an upper and lower bound to the summation:

$$\int_0^n k^r dk \leq \sum_{k=1}^{n} k^r \leq \int_1^{n+1} k^r dk$$

$$\frac{k^{r+1}}{r+1}|_0^n \leq \sum_{k=1}^{n} k^r \leq \frac{k^{r+1}}{r+1}|_1^{n+1}$$

$$\frac{n^{r+1}}{r+1} \leq \sum_{k=1}^{n} k^r \leq \frac{(n+1)^{r+1}}{r+1} - \frac{1^{r+1}}{r+1}$$

   We take away the constants to get something like this:

$$n^{r+1} \leq \sum_{k=1}^{n} k^r \leq (n+1)^{r+1}$$

$$\Omega(n^{r+1}) \leq \sum_{k=1}^{n} k^r \leq O(n^{r+1})$$

   Because Ω is equal to O, we can then say that the tight bound of the summation is: $\Theta(n^{r+1})$

   (b) $\sum_{k=1}^{n} lg^s k$

   Expanding the summation we get the following terms:

$$lg^s 1 + lg^s 2 + lg^s 3 + ... + lg^s n$$

As it is a sum of logarithms it can be put like this:

$$lg^{\hat{s}}(1 * 2 * 3 * ... * n) = lg^s n!$$

Let's prove this by mathematical induction:

$$\sum_{k=1}^{n+1} lg^s k = \sum_{k=1}^{n} lg^s k + lg^s(n+1)$$
$$= lg^s n! + lg^s(n+1)$$

$$= lg^s(1 * 2 * ... * n * (n+1))$$

$$= lg^s(n+1)!$$

So we can say that the complexity of this summation is: $\Theta(lg^s(n!))$

(c) $\sum_{k=0}^{lgn} \lceil \frac{n}{2^k} \rceil$

This summation can be bounded using a geometric series, because the following is true:

$$\frac{a_{k+1}}{a_k} \leq r \text{ , } 0 < r < 1 \, is \, constant$$

$$\frac{\frac{n}{2^{k+1}}}{\frac{n}{2^k}} = \frac{n * 2^k}{n * 2^{k+1}} = \frac{2^k}{2^{k+1}} = 2^{k-(k+1)} = 2^{-1} = \frac{1}{2}$$

Then the following applies:

$$\sum_{k=0}^{lgn} \lceil \frac{n}{2^k} \rceil \leq \sum_{k=0}^{\infty} a_0 r^k \text{ , } where \, a_0 \, is \, the \, first \, term \, of \, the \, series$$

$$a_o = \frac{n}{2^0} = n$$

$$\sum_{k=0}^{lgn} \lceil \frac{n}{2^k} \rceil \leq \sum_{k=0}^{\infty} n * r^k$$

$$\leq n * \frac{1}{1 - \frac{1}{2}} = 2n$$

Havin 2n as a bounding expression, we can define constants c_0 and c_1 such that the conditions for Θ are met:

$$c_0 n \leq \sum_{k=0}^{lgn} \lceil \frac{n}{2^k} \rceil \leq c_1 n$$

Because each term of the series is the previous one divided over two, the series won't ever result in $2n$. So having $c_1 = 2$ would do, and $c_0 \leq 1$ would do, such that:

$$\sum_{k=0}^{lgn} \lceil \frac{n}{2^k} \rceil = \Theta(n)$$

2. Given the Tree Sort Algorithm:

   (a) Give the step count

TreeSort(A):	
1. $Tree = BinaryTree$	1
2. for $i = 0$ to $A.length$:	n
3. $\quad Tree.insert(A[i])$	$(n-1)(log(n))\ or\ (n-1)(n)$
4. $A = Tree.inOrder()$	n

   (b) Complexities

The complexities of Tree Sort depend on the form of its input, a randomized array will take less time than an already sorted array.

Best Case Analysis:
In the best case of Tree Sort the input is an array such that the resulting tree of lines 2 - 3 is a balanced tree, in which each insertion takes $O(log(n))$. Such that we have the following summation:

$$1 + n + \sum_{k=0}^{n-1} log(n) + n = 1 + n + (n-1)log(n) + n$$

This case takes $O(nlog(n))$.

Worst Case Analysis:
The worst case of the Tree Sort is when the input is an array already sorted or almost sorted, this results in a degenerate tree, where child

nodes hang either only on the left or right side of the Tree, an insertion in this Tree takes $O(n)$. Such that we have the following summation:

$$1 + n + \sum_{k=0}^{n-1} n + n = 1 + n + (n-1)n + n$$

This case takes $O(n^2)$.

Average Case:
The average case is when we have a randomized array, per se. For this case we can assume that we have a uniform distribution, meaning every element of the array has the same probability of being in the correct position to result in a balanced tree:

For every element i in the original array, we have i possible spaces in the tree for i to be placed in. We define an indicator function $I(A)\begin{cases} P(A) & P(\widetilde{A}) \end{cases}$ in which $P(A)$ would be $\frac{1}{i}$ given that we are at element i.

To get the expected value of an insertion we must take into account insertions 1 through n, we can do so by adding up the expected value at each step, where $X_i = I(A_i)$: $E[x_n] = E[x_1]+E[x_2]+...+E[x_n] = E[\sum_{i=1}^{n} x_i] = 1 + \int_2^n \frac{1}{i} di = 1 + logn - log2 = logn$.

So the average case for inserting n elements would be: $O(nlogn)$.

(c) Prove the correctness of the algorithm

To prove the correctness of Tree Sort using loop invariants we must also prove the correctness of the method $Tree.insert(item)$ inside Tree Sort in order to prove the loop invariant set for lines 2 - 3.

BinaryTree
Insert(item):

1. $node = Tree.root$
2. while $node \neq null$:
3.   if $item < node$:
4. $node = node.leftSubtree$
5.   else:
6. $node = node.rightSubtree$
7. $node = item$

We define the following invariant for the while loop of lines 2-6:

"Before iteration of the while loop of lines 2 - 6 $node.leftSubtree$ contains elements less than *node* and $n.rightSubtree$ contains elements greater than *node*"

Initialization: before the first iteration, *node* is the root of a Binary Tree, if no insertions have been made, the root is null and the properties of a Binary Tree are true.
Maintenance: before each iteration, *node* is placed as the root of another subtree, either right or left depending on the value of item and so *node* is now the root of a Binary Tree, so the invariant is validated.
Termination: at termination, *node* is null, so we have arrived at the place where *item* should be, as we ended up here through a Binary Search, *item* is now the root of a right or left subtree of it's parent node. The invariant is validated.

The loop invariant for the for loop of lines 2 - 3 is thus:

"Before each iteration of the for loop of lines 2 - 3 $Tree$ is ordered in such way that each node is a root of a Binary Tree, where any node's left subtree has values smaller than the root, and the right subtree has values bigger than the root."

Initialization: before the first iteration of the loop, $Tree$ is an empty tree, no nodes and whose root is null, so it follows the properties of a Binary Tree.
Maintenance: before each iteration i each element of the sub-array $A[1..i-1]$ is now a node in $Tree$ through $Tree.insert(item)$ such that it follows the properties of a Binary Tree. Each node is the root of a Binary Tree, where the right subtree has values bigger than the root and the left subtree has values smaller than the root.

Termination: at termination, $i = A.length$ and the elements of the array $A[1...A.length-1]$ are nodes in $Tree$ sorted in since their insertion using $Tree.insert(item)$, resulting in a Binary Tree where each node is the root of a Binary Tree, the right subtree having values greater than the root and the left subtree having values smaller than the root. Such that in line 4, $Tree.inOrder()$ outputs the sorted version of array A.

3. Book Problems

   (a) Excercise 2.3-3:

   Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence $T(n)$ is $nlgn$

   $T(n) = 2 \quad$ if $n = 2$,
   $T(n) = 2T(\frac{n}{2}) + n \quad$ if $n = 2^k$, for $k > 1$

   $T(n) = \sum_{i=0}^{lgn} (\frac{2}{2})^i n = \sum_{i=0}^{lgn} n = nlg(n)$
   $T(2^k) = \sum_{i=0}^{lg2^k} 2^k = 2^k lg(2^k) = k2^k$, where k is $logn$ when $n = 2^k$
   $T(2^{k+1}) = \sum_{i=0}^{lg2^{k+1}} 2^{k+1} = (k+1)2^{k+1}$, where $k+1$ is $logn$ when $n = 2^{k+1}$

   Thus, $T(n)$ is exactly $nlgn$ when n is an exact power of 2.

   (b) Excercise 3.1-7:

   Prove that $o(g(n)) \cap \omega(g(n))$ is the empty set.

   We can prove this by looking at the definition of each $o(g(n))$ and $\omega(g(n))$.

   $o(g(n))$ is defined as the set $\{f(n)$ such that $0 \leq f(n) < cg(n)$ for any $c > 0$, $n_0 > 0\}$
   $\omega(g(n))$ is defined as the set $\{f(n)$ such that $0 \leq cg(n) < f(n)$ for any $c > 0$, $n_0 > 0\}$

   So we get the following equation:

   $$\omega(g(n)) < f(g(n)) < o(g(n))$$

   Then $\omega(g(n)) < o(g(n))$. Thus $\omega(g(n))$ is not a subset of $o(g(n))$ and $o(g(n)) \cap \omega(g(n)) = \emptyset$.

4. Give asymptotic upper and lower bounds for $T(n)$ in:

   (a) $T(n) = 9T(\frac{n}{81}) + logn$

Using the master theorem we can make this fit in one of the cases:

$$f(n) = O(n^{log_b a - \varepsilon}),\ for\ some\ \varepsilon > 0$$

$$f(n) = logn,\ b = 81\ ,\ a = 9$$

$$log_b a = log_{81} 9 = \frac{1}{2}$$

$$So: \quad log(n) = O(n^{\frac{1}{2}-\varepsilon})\ for\ some\ \varepsilon > 0$$

$$For\ cn^k\ to\ be\ greater\ than\ log(n)\ k\ must\ be\ greater\ than\ 0:$$

$$\frac{1}{2} - \varepsilon > 0$$

$$\varepsilon < \frac{1}{2} \quad ,\ and\ \varepsilon > 0\ so:$$

$$0 < \varepsilon < \frac{1}{2}$$

Because there is an ε that satisfies the condition for the case, then, following the master theorem $T(n) = \Theta(n^{log_b a})$, so in this case $T(n) = \Theta(n^{log_{81} 9}) = \Theta(n^{\frac{1}{2}}) = \Theta(\sqrt{n})$

(b) $T(n) = T(n-1) + n^c$, where $c \leq 1$ is a constant.

Expanding $T(n)$ we get the following:

$$T(n-3) + (n-2)^c + (n-1)^c + n^c$$

So we get the following summation:

$$\sum_{k=0}^{n-1} (n-k)^c$$

We get the following bounds using the minimum and maximum term of the summation:

$$a_{max} = n^c \quad a_{min} = 1^c$$

$$\sum_{k=0}^{n-1} 1^c \leq \sum_{k=0}^{n-1} (n-k)^c \leq \sum_{k=0}^{n-1} n^c$$

$$(n-1) * 1^c \leq \sum_{k=0}^{n-1} (n-k)^c \leq n^c * (n-1)$$

$$n - 1 \leq \sum_{k=0}^{n-1} (n-k)^c \leq n^c n - n^c$$

$$n - 1 \leq \sum_{k=0}^{n-1} (n-k)^c \leq n^{c+1} - n^c$$

And so we get:

$$\Omega(n) \leq \sum_{k=0}^{n-1} (n-k)^c \leq O(n^{c+1})$$

(c) $T(n) = T(n^{\frac{1}{10}}) + 1$

Using substitution we get the following:

$$m = log_{10} n$$

$$10^m = 10^{log_{10} n}$$

$$10^m = n^{log_{10} 10}$$

$$10^m = n$$

$$(10^m)^{\frac{1}{10}} = 10^{\frac{m}{10}}$$

$$S(m) = T(10^m)$$

$$S(m) = S(\frac{m}{10}) + 1$$

Now we use the recursion tree structure to solve $S(m)$:
There'll be a point where a division would be equal to 1, we're going to get that value to get the last term of the series:

$$\frac{m}{10}, \frac{m}{10^{\mathbf{2}}}, \frac{m}{10^{\mathbf{3}}} \cdots \frac{m}{10^h}, \; where\, h\, is\, the\, height\, of\, the\, tree.$$

$$\frac{m}{10^h} = 1$$

$$m = 10^h$$

$$log_{10}m = log_{10}10^h$$

$$log_{10}m = h$$

So we can solve $S(m)$ as follows:

$$\sum_{k=0}^{log_{10}m} 1 = \Theta(log_{10}m)$$

Then we substitute to get back to terms of n:

$$\Omega(log_{10}(log_{10}n)) \leq \sum_{k=0}^{log_{10}m} 1 \leq O(log_{10}(log_{10}n))$$

$$\sum_{k=0}^{log_{10}m} 1 = \Theta(log_{10}(log_{10}n))$$