6.006 Homework	**Problem Set 1, Part A**	# 1 – Asymptotic Growth
Xola Ntumy		February 14, 2011
Collaborators: Adwoa Boakye		

(a) Group 1:

$$f_3 = 2^{2^{100000}} \qquad f_4 = \log n \qquad f_1 = 8\sqrt{n} \qquad f_2 = \binom{n}{2}$$

(b) Group 2:

$$f_1(n) = 1/n \qquad f_2(n) = \log\log\log n \qquad f_3(n) = n/\log n \qquad f_4(n) = n^{0.99}$$

(c) Group 3:

$$f_3(n) = \log n^n \qquad f_4(n) = \sum_{i=0}^{n} i \qquad f_2(n) = n \cdot 2^{n/2} \qquad f_1(n) = 2^n$$

6.006 Homework	**Problem Set 1, Part A**	# 2 – Unimodal Maximum
Xola Ntumy		February 14, 2011
Collaborators: None		

To find the maximum element in a unimodal array in an efficient manner, first take the middle element of the array at index (n/2), where n is the length of the array. If this element is greater than or equal to the elements on its right and left, then this element is the unimodal maximum. Return that element. Otherwise, if the element to the left of the middle element is greater, recurse this algorithm using the left half of the array. Else, use the right half.

There is guaranteed to be only one unimodal maximum. The running time of this algorithm is O(log n) because each time the ”search space” (array) is divided into two and hence the list is traversed a total of log n times.

6.006 Homework	**Problem Set 1, Part A**	# 3 – (Re)writing History
Xola Ntumy		February 14, 2011
Collaborators: None		

(a) No. This will keep the document distance the same as before. This is because the same dot product will be computed when the words are expressed as vectors even though the order of the words is different from before. The length of the document will remain the same and hence the document distance will not be affected.

(b) Yes. This will increase the document distance. Though the dot product between matching words in both documents may remain the same, the document lengths will vary. The product of the document lengths will decrease. This will give rise to a greater document distance.

(c) Yes. This should increase the document distance. Less matching words and a smaller product from the document lengths will also give rise to a greater document distance.

6.006 Homework	**Problem Set 1, Part A**	# 4 – Augmented Binary Search Trees
Xola Ntumy		February 14, 2011
Collaborators: none		

(a) To find the number of items (in this case textbooks) in a regular, non-augmented AVL tree which fall within a specific range, start at the root of the tree and search the tree linearly. Initialize a counter. Traverse all nodes of the tree and check whether the prices to which the nodes point to are within the range (ie. are they less than or equal to b and greater than or equal to a), if they are, increment the counter.

Otherwise, move to the next node. Since this algorithm would imply visiting and expanding each and every node, the running time would be O(n) where n is the number of nodes in the AVL tree.

(b) A more efficient way of finding the number of textbooks would involve expanding nodes only if we have to. Since an augmented tree has pointers to its price, parents and children, the children of a node can be examined before that node is expanded. Starting with the root, the augmented AVL tree may be traversed such that each time a node is visited, its price and the price of its children is checked.

First off, if the root's price is less than a, the right tree is recursed, and similarly if the root's price is greater than b, the left-tree is recursed. If the price is within the range, comparisons are made as to which child of the tree should be traversed.

In pseudocode:

Start with root. Initialize counter. Define an auxiliary recursive function. Recurse(T): If root is less than a: recurse right-tree

if root is greater than b: recurse left-tree

if the root is within the range (a,b): if left-child is less than a and the right-child is greater than b: increment counter return counter

else if root has no children: increment counter check parent's right-child

else if the left-child is within the range(a,b): recurse left-tree

else if the right child is within the range(a,b): recurse right-tree

This would give a running time of O(log n) because at each decision or comparison, the number of nodes is reduced by half.