CSCI 314 – Fall 2012 Handout: asgn02
Algorithms
Michael H. Goldwasser
Saint Louis University Friday, 7 September 2012

Homework #2: Asymptotics, Recurrences, Amortization
Due Date: Friday, 14 September 2012

Guidelines

Please make sure you adhere to the policies on collaboration and academic honesty as outlined in the syllabus.

Reading

Chapters 3, 4, 17 of CLRS.

Problems

There are four required problems worth 25 points each, and one extra credit challenge worth 10 points.

Problem A (25 points)

"Work entirely on your own."

Rank the following functions by order of growth, *i.e.*, arrange the functions as $g_1, g_2, \ldots, g_{25}$ such that $g_1 = \Omega(g_2), g_2 = \Omega(g_3), \ldots, g_{24} = \Omega(g_{25})$. Partition your list into equivalence classes such that $f(n)$ and $g(n)$ are in the same class iff $f(n) = \Theta(g(n))$.

$(\sqrt{2})^{\lg n}$	n^2	$n!$	$(\lg n)!$	$(3/2)^n$
n^3	$\lg^2 n$	$\lg(n!)$	2^{2^n}	$n^{1/\lg n}$
$\ln \ln n$	$n \cdot 2^n$	$n^{\lg \lg n}$	$\ln n$	1
$2^{\lg n}$	$(\lg n)^{\lg n}$	e^n	$4^{\lg n}$	$(n+1)!$
$\sqrt{\lg n}$	$2^{\sqrt{2 \lg n}}$	n	2^n	$n \lg n$

[Note: We do not require formal proof for this problem, simply the ordering and the equivalence classes.]

Problem B (25 points)

"Work entirely on your own."

Let $f(n)$ and $g(n)$ be functions such that $f(n) = \Omega(1)$ and $g(n) = \Omega(1)$. Prove or disprove each of the following conjectures.

i. $f(n) + g(n) = \Omega(\min(f(n), g(n)))$.

ii. $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.

iii. $f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$.

iv. $f(n) = O((f(n))^2)$.

v. $f(n) = \Theta(f(n/2))$.

[Your proofs must be formal, although for false conjectures a specific counterexample constitutes a valid proof.]

Problem C (25 points)

"Work entirely on your own."

For each of the following recurrences, give a function, $f(n)$ such that $T(n) = \Theta(f(n))$. Assume that $T(n)$ is constant for $n \leq 2$.

For some of these, you may use the Master Theorem as a proof so long as you justify that it applies. For others, you should prove your bounds by induction, using the substitution method. Since the claim involves a Θ bound, you must prove both upper and lower bounds.

i. $T(n) = 2T(n/2) + n^3$.

ii. $T(n) = 7T(n/2) + n^2$.

iii. $T(n) = 2T(n/4) + \sqrt{n}$.

iv. $T(n) = T(n-1) + n$.

v. $T(n) = T(\sqrt{n}) + 1$.
As a huge hint, I'll tell you that $T(n) = \Theta(\lg \lg n)$.
Your job is to prove that tight bound.

Problem D (25 points)

"You may discuss ideas with other students."

Assume that we are interested in a data structure which supports two operations: SEARCH and INSERT. If we store the items in an unordered array, then INSERT can be done in $O(1)$ time but SEARCH would require $\Omega(n)$ time, where n is the current number of items.

Alternatively, if we keep the array sorted, then we can use *binary search* to implement SEARCH in $O(\lg n)$ time (if you are not familiar already with binary search, please consult one of the recommended readings or see a brief discussion in Exercise 2.3-5 of CLRS). Unfortunately, if we insist on keeping the array sorted, INSERT will require $\Omega(n)$ time in the worst case, as we may have to shift many items around.

In this problem, we are going to develop a way to accomplish these tasks while better balancing the time required for SEARCH and INSERT. Specifically, suppose that we wish to support SEARCH and INSERT on a set of n elements. Let $k = \lceil \lg(n+1) \rceil$, and let the binary representation of n be $\langle n_{k-1}, n_{k-2}, \ldots, n_0 \rangle$. We can keep k sorted arrays $A_0, A_1, \ldots, A_{k-1}$, where for $i = 0, 1, \ldots, k-1$, the length of array A_i is 2^i. Each array is either full or empty, depending on whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all k arrays is therefore $\sum_{i=0}^{k-1} n_i 2^i = n$. Although each individual array is sorted, there is no particular relationship between elements in different arrays.

i. (10 points)
Describe how to perform the SEARCH operation for this data structure in $O(\lg^2 n)$ worst-case time. (justify the bound on the running time)

ii. (15 points)
Describe how to perform the INSERT operation for this data structure in $O(\lg n)$ *amortized* time. (justify the amortized bound on the running time).

Problem E **(EXTRA CREDIT – 10 points)**
"You may discuss ideas with other students."

Consider what happens if we wish to include a DELETE operation in the data structure developed in Problem D. (we will assume that the parameter to DELETE is a reference to the exact location in the structure which holds the item to be deleted – that is we will not need to perform a search to find the item being deleted.)

i. Argue that if we insist on using a structure where all of the arrays are either empty or full, there will always be a sequence of t operations, for any t, which require $\Omega(tn)$ time, and thus $\Omega(n)$ amortized time.

ii. If we allow you to relax the restriction that all arrays are either full or empty, show how to implement DELETE in $O(1)$ amortized time, while still maintaining the previous time bounds for SEARCH and INSERT. Make sure that you justify not only the analysis of DELETE, but also that you re-justify the previous bounds for the other operations on the new technique for the data structure.