CS 311: Discrete Structures  
Spring 2003

Homework 8. Due Tuesday, November 25, 2003

1) Give the best possible bound for each function below using big-O notation and justify your answer.
   For example: \( T(n) = 7n + 3 \)
   \( T(n) \) is \( O(n) \). To justify this answer we must find constants \( c \) and \( k \) such that \( T(n) \leq cn \) for all \( n \geq k \)
   Let \( c = 10 \) then we want \( 7n + 3 \leq 10n \), which means
   \( 3 \leq 3n \) or \( n \geq 1 \). Thus \( t(n) = 7n + 3 \) satisfies the
definition of \( O(n) \) for constants \( c = 10 \) and \( k = 1 \)
   (a) \( t(n) = 1000n + 100 \)
   (b) \( t(n) = 10n^2 + 3n + 6 \)
   (c) \( t(n) = 3 \cdot \log_2(n) + 6n + 5 \)
   (d) \( t(n) = n\log_2 n + 4n + 10 \)
   (e) \( t(n) = 4n\log_2 n + 3n(n - 1) \)

2) Consider the following code:
   read n
   i = 1
   while (i <= n)
   i = 2*i
   print i
   (a) If \( n = 50 \) what does the code print out?
   (b) As a function of \( n \), how many times is the loop executed?
   (i.e., how many numbers does the code print out?) Do some small examples
to figure this out.

3) Answer the following questions.
   (a) How do you show that a function \( f(n) \) is \( O(g(n)) \)?
   (b) Show that \( t(n) = n^2 \) is not \( O(n) \).
   (c) Why big-O notation? Why does it make sense to ignore constant factors?

4) Give recurrence relation for running time of the following functions:
(a) int myst(int n)
{
    if (n == 0) return 1;
    if (n == 1) return 2;
    else return myst(n-2)*4;
}

(b) int find_min(int A[], int start, int end)
{
    int tmp;

    if (start == end) return A[start];
    else {
        tmp = find_min(A, start + 1, end);
        if (tmp < A[start]) return tmp;
        else return A[start];
    }
}

/* note this is not a good use of recursion */

(c) int process(int A[], int start, int end)
{
    int mid;

    if (start == end) { /* do something in 1 step */
        printf("%d\n", A[start]);
        return;
    }
    else {
        mid = (start + end)/2;
        process(A, start, mid);
        process(A, mid+1, end);
    }
}
You may assume for part (c) that \( n \) is a power of 2.