**After lecture03 & lecture04 -** Answer any questions on HW1

Practice Problems (all taken from previous exams)

1. What is time complexity of fun()?

int fun(int n) {

  int count = 0;

  for (int i = 0; i < n; i++)

     for (int j = i; j > 0; j--)

        count = count + 1;

  return count;

}

A O(n)

B O(n^2)

C O(n\*Logn)

D O(nLognLogn)

2. Consider the following function

int unknown(int n) {

int i, j, k = 0;

for (i = n/2; i <= n; i++)

for (j = 2; j <= n; j = j \* 2)

k = k + n/2;

return k;

}

What is the returned value of the above function?

A Theta(n^2)

B Theta(n^2 log n)

C Theta(n^3)

D Theta(n^3 log n)

3. What is the worst-case auxiliary space complexity (including stack space for recursion) of merge sort?

a) O(1)

b) O(log n)

c) O(n)

d) O(n log n)

4. Choose the incorrect statement about merge sort from the following?

a) it is a comparison based sort

b) it’s runtime is dependent on input order

c) it is not an in place algorithm

d) it is stable algorithm

5. Use definition of big O to prove or disprove.  
5a) is 2^(n+1) ?=? O(2^n)  
5b) is 2^(2n) ?=? O(2^n)

6. Although merge sort runs in Θ(n lg n) worst-case time and insertion sort runs in Θ(n^2) worst-case time, the constant factors in insertion sort make it faster for small n. Thus, it makes sense to use insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which n/k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

6a) Show that the n/k sublists, each of length k, can be sorted by insertion sort in Θ(nk) worst-case time.

6b) Show that the sublists can be merged in Θ(n lg (n/k) worst-case time.

6c) Given that the modified algorithm runs in Θ(nk + n lg (n/k)) worst-case time, what is the largest asymptotic (Θ notation) value of k as a function of n for which the modified algorithm has the same asymptotic running time as standard merge sort?

6d) How should k be chosen in practice?

7. The Fibonacci sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... is defined recursively as

|  |  |
| --- | --- |
| HW1Img04 | This mathematical definition leads naturally to a recursive algorithm FIB(n)      if n <= 1           then return(n)           else return(FIB(n-1) + FIB(n-2))      endif end FIB |

7a) Write the recurrence relation, T(n), for the asymptotic runtime for procedure FIB(n) shown above, and solve the recurrence relation to show that T(n) = O(2^(n-2)).

7b) Another recursive procedure which computes the nth Fibonacci number is below.  
F1(n)  
     if n < 2  
          then return(n)  
          else return(F2(2,n,1,1))  
     endif  
end F1   
F2(i,n,x,y)  
     if i <= n  
          then call F2(i+1,n,y,x+y)  
     endif  
     return x  
end F2  
Trace out the algorithm as it computes F1(1), F1(2), F1(3), F1(4), explain how the algorithm works, and then compare its asymptotic runtime to the time for procedure FIB(n).

8. Use mathematical induction to show that when n is an exact power 2, the solution of the recurrence

T(n) = 2 if n =2

2T(n/2) +n if n = 2k, for k>1

is T(n) = n\* lg n