Run-time analysis (16 pts).

1. (2pts each) In each of the following cases, indicate if $f = O(g)$, $f = \Omega(g)$, $f = \Theta(g)$.

$f(n)$	$g(n)$	O	Ω	Θ
2^{n+10}	2^{2n}			
$\log_5 \sqrt{n}$	$\log_2 n$			
$n \log_2 n$	$n^{1.5}$			

2. (5 pts each) Please solve the following recurrence relations.

   a. $T(n) = 2T(n-1) + c$
   b. $T(n) = T(n/2) + cn$

Section	total	score
Runtime	16	

Proof by Induction (8 pts) Prove the following statement by induction.

A tree with n nodes has $n-1$ edges.

Note that here a tree refers to a connected graph that contains no cycles.

Please clearly state the base case, inductive assumption and the inductive step:
Base case: (1pt)

Inductive assumption: (1pt)

Inductive step: (6pts)

Section	total	score
Proof	8	

Divide and Conquer. (15 pts)
Given an array of n objects $A[1], A[2], ..., A[n]$, you cannot sort them (think of the objects as images, or files), but can compare if two objects are equal in $O(1)$ time. An element of A is called the majority if it occurs more than $\frac{n}{2}$ times (not just that it is the most common) in A. For example $[1, 2, 2, 4]$ does not have a majority element.

a. (10 pts) Design an algorithm that returns either the majority element or return Null if no majority element exists. For full credit, the algorithm must run in $O(n \log n)$ time. You may describe your algorithm in plain English, or in pseudocode. You can assume n is a power of 2. (Hint: if you know the majority elements of the left and right sub-arrays of A, how to figure out the majority element of A?)

b. (5 pts) Provide the recurrence relation describing the run time of your algorithm. You don't need to solve it.

Section	total	score
D&C	15	

3. (15 pts)
You are given a sequence of n binary bits $x_1, \cdots, x_n \in \{0, 1\}$. Your output is to be either

- any i such that $x_i = 1$ or
- the value 0 if the input is all 0s.

For example, if your input is $\{0, 0, 1, 1\}$, the output could be either 3 or 4.

The only operation you are allowed to use to access the inputs is a function Group-Test where Group-Test(i, j) returns 1 if any bit in $x_i, x_{i+1}, \cdots, x_j$ has value 1, and returns 0 otherwise. For example, for the given input sequence $\{0, 0, 1, 1\}$, Group-Test$(2, 4) = 1$ and Group-Test$(1, 2) = 0$.

(Historical Note: In World War I, the army was testing recruits for syphilis which was rare but required a time-consuming but accurate blood test. They realized that they could pool the blood from several recruits at once and save time by eliminating large groups of recruits who didn't have syphilis.)

a) (10 points) Design a divide and conquer algorithm to solve the problem that uses only $O(\log n)$ calls to Group-Test in the worst case. Your algorithm should never access the x_i directly.

b) (5 points) Briey justify your bound on the number of calls to Group-Test.

4. Dynamic programming. (20 pts)

   Tomorrow is the big dance contest you've been training for your entire life. You've obtained a list of n songs that will be played during the contest, in chronological order. For the k-th song, if you dance to it, you will get exactly $score[k]$ points, but then you will be physically unable to dance for the next $wait[k]$ songs.

   Here is an example input, we have:

song	1	2	3	4	5
score	5	10	4	8	5
wait	1	2	1	2	1

   That is, if you choose to dance to the first song in the schedule, you will get 5 points but have to skip the next song. If you choose to dance to the second song, you will get 10 points, but have to skip the next two songs. You can choose as many songs as you can subject to the wait-time constraints. The goal is to maximize your total score.

   Describe and analyze an efficient algorithm to compute the maximum total score you can achieve given a pair of input arrays $score[1 \ldots n]$ and $wait[1 \ldots n]$.

   You must:

   (a) (5pts) Define (in words) the subproblems you solve.

   (b) (10pts) Give a recurrence relation for the subproblems. Also please provide the *base case*, and state the *final output* of your dynamic programming algorithm.

   (c) (5pts) Give the running time of your algorithm. Explain.