This entire assignment was very helpful in getting all my concepts cleared when it comes to theta, omega and big O complexity problems. It also gave me quite a bit of insight into how to solve problems related to finding the time complexity of various algorithms, functions and recurrences.

In Q1, I applied the plug and chug method as it was a very straightforward question, with a simple summation as our final result. This was an easy question.

Q2 was a very good question. I tried applying substitution and induction but it didn't work. I tried applying tree recursion and got very far too, but my final equations were too complex to solve and get a closed form solution for(it was a highly combinatorial solution). Even the master method is not usable in questions like these. In the end, I decided to brute force my way through the recursion, by trying to get the numbers inside the function smaller and smaller, just to try and find some semblance of a pattern somewhere.
I ended up figuring out that as we use recurrence to simplify the terms, similar terms got added in successive steps, giving us successive coefficients which were those of a corresponding fibonacci sequence. Hence I applied the results of a fibonacci sequence derived in class and solved this question.

In Q3, as the answer was fairly visible by guesswork, I guessed the answer and used mathematical induction to prove that my guess was correct.

In Q4, many of my concepts with regard to the master theorem got cleared. All the asymptotic bounds , and complexity notations became quite clear to me. All 3 parts were not very tough to solve, provided we know the definitions of theta, omega and big-O complexity well.

In Q5, I found out the total number of j iterations per iteration of i, and added them up. This was an interesting question as it required me to split the sum into a sum of Hn's(Harmonic sums). I then found out a bound on this sum, using the approximation of integrals by rectangles concept, thus getting a higher bound on them. I got a sum of logarithms, which was less than nlog(n). Thus I got an upper bound on the function, hence getting its Big-O complexity.