Problem 1

(a) Pick $c = 37$ and $n_0 = 1$. We know that

$$2n^2 + 15n + 20 \leq 2n^2 + 15n^2 + 20n^2 \text{ for all } n \geq 1.$$

From this we see that $2n^2 + 15n + 20 \leq 37n^2$ for all $n \geq 1$. This shows that $f = O(n^2)$.

(b) Pick $c = 2$ and $n_0 = 1$. We know that

$$2n^2 + 15n + 20 \geq 2n \text{ for all } n \geq 1.$$

This shows that $f = \Omega(n)$.

Problem 2

We know that $\log_2 n \leq n$ for all $n \geq 1$. Therefore, $7n \log_2 n \leq 7n^2$ for all $n \geq 1$. This establishes that $f = O(n^2)$. We know that $\log_2 n \geq 1$ for all $n \geq 2$. Therefore, $7n \log_2 n \geq 7n$ for all $n \geq 2$. This establishes that $f = \Omega(n)$.

Problem 3

1) Since $2^{\log_2 n} = n$, $2^{\sqrt{\log_2 n}} < 2^{\log_2 n} = n$ for all $n \geq 1$. Thus we have:

$$2^{\sqrt{\log_2 n}} = O(n \cdot (\log_2 n)^3).$$

2) Since $(\log_2 n)^3 = O(n^{1/3})$, we have:

$$n \cdot (\log_2 n)^3 = O(n^{4/3}).$$

3) Since $4/3 < 2$, we have:

$$n^{4/3} = O(100n^2).$$

4) Since $100 = O(\log_2 n)$, we have:

$$100n^2 = O(n^2 \log_2 n).$$

5) Since $\log_2 n \cdot \log_3 n < n$ when n is sufficiently large, $\log_2 n = O(n/\log_3 n)$. Then we will have:

$$n^2 \log_2 n = O(n^3/\log_3 n).$$

6) We know that $n^3/\log_3 n = O(n^3)$ and also that $n^3 = O(2^n)$. Thus we have:

$$n^3/\log_3 n = O(2^n).$$

Thus, the final sequence of complexity from low to high is as below:

$$2^{\sqrt{\log_2 n}}, n \cdot (\log_2 n)^3, n^{4/3}, 100n^2, n^2 \log_2 n, n^3/\log_3 n, 2^n$$

Problem 4

(a) The inner loop runs $\lfloor \log_2 i \rfloor + 1$ times for each value of i. Thus the total number of basic operations are:

$$\lfloor \log_2 1 \rfloor + \lfloor \log_2 2 \rfloor + \cdots + \lfloor \log_2 n \rfloor + n = n + \Theta(\log_2(1 \cdot 2 \cdot 3 \cdots n) = n + \Theta(\log_2(n!)).$$

We can change the base for log function using the change of base formula: $\ln(n) = \log_2 n / \log_2 e$. Then we can apply Stirling's Approximation to the equation above. Based on Stirling's Approximation, we have the final running time as:

$$n + \Theta(\log_2(n!)) = n + \Theta(\ln(n!)) = \Theta(n \ln n + n - n + O(\ln n)) = \Theta(n \ln n).$$

(b) The inner loop runs $n/3$ times always. If we count the steps of the outer loop, the total steps will be:

$$\frac{n}{3}(n + n - 1 + \cdots + 1).$$

Using the arithmetic series formula, we have:

$$\frac{n}{3}(n + n - 1 + \cdots + 1) = \frac{n}{3} \cdot \frac{n(n+1)}{2} = \Theta(n^3)$$

Problem 5

First we use *merge sort* to sort the list. This runs in $\Theta(n \log n)$ time. The resulting list is sorted, with identical numbers bunched together. Then we go through the list in linear time, and count the number of times each distinct element appears in the list. This will take $\Theta(n)$ steps. Suppose that there are s distinct elements in the list with $k_1, k_2, \ldots, k_s$ being their frequecies in the list. Then the total number of pairs is:

$$\binom{k_1}{2} + \binom{k_2}{2} + \cdots + \binom{k_s}{2}.$$

Note that we have computed $k_1, k_2, \ldots, k_s$ during the linear scan mentioned above. The calculation above also takes linear time since $s < n$. Thus the total running time of this algorithm is $\Theta(n \log n)$.

Problem 6

(a) We can see that the outer loop execute exactly n times. The inner loop will execute at most n times every time it is executed. Adding up items from i to j takes at most $O(n)$ steps as well. Storing the result in $B[i, j]$ takes only constant time. Thus the running time of this algorithm is $O(n^3)$.

(b) Now consider values of $i \leq n/4$ and $j \geq 3n/4$. The amount of work the algorithm does for these values of i and j is a lower bound on the total amount of work done by the algorithm. The variables i and j take on $n^2/16$ values together. Now note that for each of these values, $j - i \geq n/2$ and hence the summation will take at least $n/2$ basic operations. The total number of basic operations is $n^3/32$. This means that the algorithm has running time bounded below by $\Omega(n^3)$.

(c) Consider the following algorithm:

Problem 6 continued on next page...

for $i = 1, 2, \ldots, n$
    $B[i, i+1] \leftarrow A[i] + A[i+1]$

for $size = 2, 3, \ldots, n-1$
    for $i = 1, 2, \ldots, n - size$
        $j \leftarrow i + size$
        $B[i, j] \leftarrow B[i, j-1] + A[j]$

This algorithm works since the values $B[i, j]$ were already computed in the previous iteration of the for-loop. It first computes $B[i, i+1]$ for all i by summing $A[i]$ with $A[i+1]$. This requires $O(n)$ steps. For each value of $size = 2, 3, \ldots$, the algorithm computes $B[i, j]$ for $j = i + size$ by setting $B[i, j] = B[i, j-1] + A[j]$. For each $size$, the algorithm runs $O(n)$ steps since there are at most n $B[i, j]$'s of that "size" (i.e., such that $j = i + size$). There are also less than n values of $size$. Thus the algorithm runs in $O(n^2)$ time.