1 Problem 1

a) True

Using the limit method and algebra we can see that the limit evaluates to a constant.

$$\lim_{n\to\infty}(\frac{2^{n+1}}{2^n}) = \lim_{n\to\infty}(2^{(n+1)-(n)}) = \lim_{n\to\infty}(2^1) = 2$$

Therefore, it must be true that $2^{n+1} = O(2^n)$.

b) False

Using the limit method and algebra we can see that the limit evaluates to infinity.

$$\lim_{n\to\infty}(\frac{2^{2n}}{2^n}) = \lim_{n\to\infty}(2^{(2n)-(n)}) = \lim_{n\to\infty}(2^{n(2-1)}) = \lim_{n\to\infty}(2^n) = \infty$$

Therefore, it must not be true that $2^{2n} = O(2^n)$.

2 Problem 2

Proof. Direct proof. Assume for functions f, g, and h that $f(n) = O(g(n))$ and $g(n) = O(h(n))$. Therefore it must be true that for constants c and d, that $f(n) \leq c * g(n)$ and $g(n) \leq d * h(n)$. Substituting in $g(n)$, you get $f(n) \leq c(d * h(n))$. This is the same as $f(n) \leq (c * d)h(n)$. Because $c * d$ is a constant we have shown that $f(n) = O(h(n))$ for sufficiently large n. $\square$

3 Problem 3

a) Yes

50% - identify time required to read input.
The time required to read in the entire array input is $\Omega(n)$ because you must iterate over every item in the list once.
50% - write conclusion.
Therefore its impossible to make an algorithm that is faster than being able to see every item in the list.

b) $O(n \log n)$

50% - identify complexity of A and B.
The complexity of A is $O(n \log n)$ and B is $O(n)$
50% - identify combined complexity.
Using the limit definition, we can see that $n \log n$ is an upper bound for n.

$$\lim_{n\to\infty}(\frac{n}{n \log n}) = \lim_{n\to\infty}(\frac{1}{\log n}) = 0$$

Therefore, the overall complexity of $A + B$ is $O(n \log n)$.

c) $O(n^2)$

The two different parts are also run sequentially, so their complexities add together as well. This means the overall complexity looks like $(n * \log n) + (n * n) = (n \log n) + (n^2)$. This time complexity is bounded by the polynomial which makes the overall complexity $O(n^2)$.

d) $O(a * b)$

Starting with the inner for-loop, it is run b times in all cases. Then the outer for-loop is also run a times in every case. Since they are nested for loops, the complexities multiply. Therefore the overall complexity is $O(a * b)$.

4 Problem 4

Proof. In order to prove $(n + a)^d = \Theta(n^d)$ we can use the limit strategy.

$$\lim_{n\to\infty}(\frac{(n+a)^d}{n^d}) = \lim_{n\to\infty}((\frac{(n+a)}{n})^d) = \lim_{n\to\infty}((1 + \frac{a}{n})^d) = \lim_{n\to\infty}((1+0)^d) = 1^d$$

Since this limit tends to a constant, $(n + a)^d = \Theta(n^d)$. $\square$