I. Math

In the following problems, we will look at how rapidly different functions grow. We are interested in figuring out when two functions grow at the same rate. Our definition of “the same” is: f(n) grows no faster than g(n) if there is some minimum value for the argument (we’ll call it n0) after which f(n) is no larger than a constant multiple of g(n) -- call it k\*g(n). For example, if f(n) is 367n+1098 and g(n) is 2n, n0 could be 40 and k could be 200: 367n+1098 <= 200\*2n for n>40.

For each of the following pairs of functions, identify a multiplier and a minimum value of n for which the first is no larger than the specified multiple of the second. If you cannot -- if there are no such values -- explain why not.

1. 367n+1098 and 2n [n0 = 40; k=200]  
   *367n+1098 <* ***200****\*2n, n>40*
2. n^2 + 2n + 6 and 6n^2 - 25  
   *n^2 + 2n + 6 <* ***1****\*(6n^2 - 25), n > 3  
   Note that this is possible because (n^2 + 2n + 6)/(6n^2 - 25) grows as ⅙ in the limit, i.e, once we get past small numbers (where the -25 matters) the second order term dominates, and 6n^2 is larger than n^2 (for n>1, at least).*
3. n^3 + n^2 - 2n and 6n^2 - 25  
   *There is no constant that will make this true, because (*n^3 + n^2 - 2n*)/(6n^2 - 25) grows as n/6 in the limit, which will exceed any constant as n gets sufficiently large.*
4. 869,438 and 923  
   *Any constant greater than 869,438/923, which is a little smaller than 1000, will do. Neither of these quantities changes when n grows.*
5. 2^n and 3^n  
   *No constant will make this true, because (2^n)/(3^n) is (⅔)^n, which gets smaller as n gets larger, so there’s nothing to multiply 2^n by to make it larger in the limit.*
6. 3^n and 2^n  
   *This is the opposite of the previous problem, so any constant will do (for large enough n). (2^n)/(3^n) is (3/2)^n, which is > 1 for n > 1. So for n>1, 1 is a good choice of constant.*
7. log\_2 n and log\_10 n [NB: these are intended to be log-base-2 of n and log-base-10 of n]  
   *Since (log\_2 n)/log\_10 n) = log\_10 2 for all n, these two expressions grow at the same rate/remain in a constant ratio. As long as we set k > log\_10 2, the left-hand expression will be larger.*
8. log\_10 n^2 and log\_2 n  
   *This answer is almost the same as above. log\_10 n^2 is 2 \* log\_10 n, so these expressions remain in a constant ratio as well. (In this case, we want k > 2 log\_2 10, which is about 7.)*

II. Linear data structures

For the following problem, assume that it costs 1 unit to access a structure -- an item in an array, a field in a record you are holding, the first element in a linked list -- whether reading or writing that element. [Accessing the second element in a linked list costs 2 units if you have a pointer to the list: 1 unit to access the first element/read its “next” pointer, and another unit to access the second element.]

Assume that you have n elements stored in an array and the same n elements stored in a linked list; you have no other references to these data structures. You may assume that n > 10. How much does it cost to perform each of the following actions?

1. Read the value of the first element in the array? In the list?  
   *Array: 1. List: 2.*
2. Read the value of the seventh element in the array? In the list?  
   *Array: 1. List: 14.*
3. Add an element to the array? [Where is the cheapest place to add it?] To the list? [Where is the cheapest place to add it?]  
   *Array: 1 if you know the last active index and can insert it there (+1 to update that index); n otherwise.  
   List: 2 if you insert it at the beginning.*
4. Delete the fifth element from the array? From the list?  
   *Array: 1 to access it, then n-5 to move each the elements that follow one position to the left.  
   List: 10. 3\*2+1 to find the element that points to it, 1 to find it, 1 to find its next pointer, and 1 to update the element that used to point to it so that it points to the (former) fifth element’s successor instead.*
5. Confirm that a particular element -- say x -- is present in the array? In the list?   
   *Array: As many accesses as it takes to find the element, which would average n/2 in an array of n elements. [NB: Faster if the array is sorted (perhaps as fast as log n).]  
   List: As many accesses as it takes to find the element, which would average 2\*n/2 in an list of n elements. [NB: No particular benefit if the array is sorted.]*
6. Confirm that a particular element -- say y -- is NOT present in the array? In the list?  
   *Array: Checking every element in the array would cost n. [NB: We can go faster if the array is sorted -- average cost n/2 if we use a linear search and just give up when we have gone “too far”, or log n with binary search.]  
   List: Again, as many accesses as it takes to (not) find the element, which is 2n in an list of n elements. [NB: In this case, sorting would reduce this to an average of 2\*n/2, because we could tell when we’d gone “too far”.]*

Now assume that you can maintain one extra piece of storage for the data structure. In the case of the array, you may want to maintain a length reference.

1. In the case of the linked list, what would be similarly helpful?  
   *A pointer to the last element could be similarly useful. Perhaps even better would be to make this a doubly linked list (i.e., give each element a pointer to the previous element as well as the next one).*
2. How much would it cost now to add an element to the end of the array? To add an element to the end of the list?  
   *As noted above, a length counter makes this cost 2 for the array. For the linked list, scanning it to find the last element would cost 2n, but a last-element-pointer makes this constant cost (2).*
3. How much would it cost now to delete an element from the array? From the list? What if it were the final element? The penultimate element?  
   *Deletion remains the same for the array, +1 to update the length count. For the list, deletion remains a linear finding operation plus some logistics to actually remove it. If you* know *you are looking for the last element, though, a last pointer plus a doubly linked list makes this much less expensive (5).*

Finally, assume that both the array and the linked list are sorted, i.e., that their elements are in some predictable order (say, comparable by < or <=).

1. How much would it cost to confirm that a particular element -- say x -- is present in the array? In the list? To confirm that a particular element -- say y -- is not present in the array? In the list?  
   *See above.*