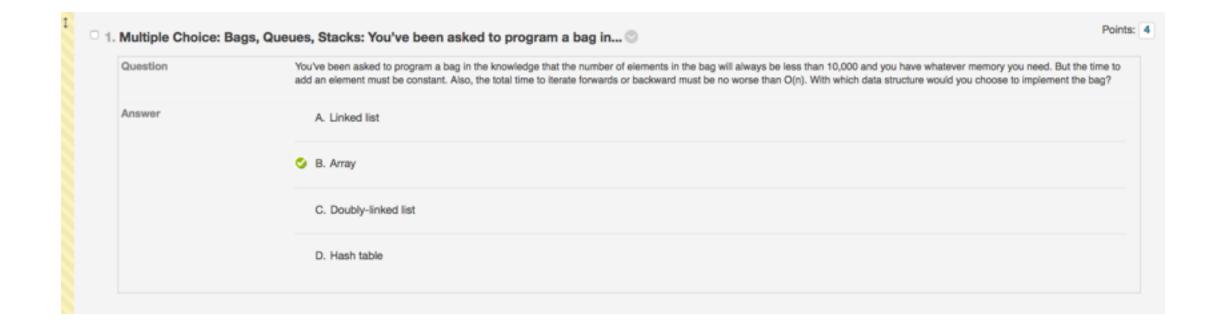
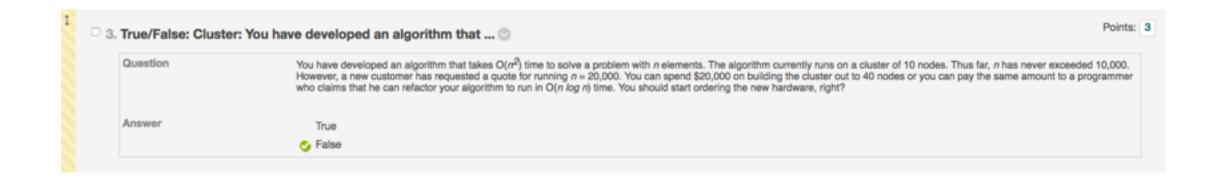
Mid-term review







My answer:

- By building out the cluster to 40 nodes, we will be able to run the algorithm in the same time that we have been accustomed to previously.
- But by employing the programmer to reduce the algorithm to O(n lg n), we can run the new algorithm on the existing cluster in less than 0.1% of the time that we have been taking up until now!!

To score full marks:

- Reasonable explanation plus
- Some sort of back-of-the-envelope calculation

n	t	log n	n log n	n^2	log t
8	12.4	3.00	24	64	3.63
16	32.2	4.00	64	256	5.01
32	79.4	5.00	160	1024	6.31
64	192.2	6.00	384	4096	7.59
128	443.7	7.00	896	16384	8.79

log t = 1.5 * log n - 1 i.e. t = $\frac{1}{2}$ * n $\frac{3}{2}$ True relationship is t = n log n

n	t	log n	n log n	n^2	log t	M	2^M
8	12.4	3.00	24	64	3.63	3.5	11.31
16	32.2	4.00	64	256	5.01	5	32.00
32	79.4	5.00	160	1024	6.31	6.5	90.51
64	192.2	6.00	384	4096	7.59	8	256.00
128	443.7	7.00	896	16384	8.79	9.5	724.08
256	443.7	8.00	2048	65536	8.79	11	2,048.00

This uses the (incorrect) model but comes up with a reasonable answer. True model predicts 1024. The point is that with only four doublings, we can't be 100%

Q7, Q8, Q9

- Some ambiguity in wording
- Q7: correct answer is n+1
- Q8: lg n or lg (n+1)
- Q9: question is incorrect so pretty much everyone gets full mark
- There are three regions into which new element can go:
 - Left (1): 1 comparison
 - Right (1): 2 comparisons
 - Middle (n-1): 2 comparisons plus log n

Q10, Q11

- **Prove** that sum of the numbers 1 through n is equal to n(n+1)/2
 - By Induction:
 - Base case: n = 1, $S_1 = 1(2)/2 = 1$ (correct)
 - Inductive case: the increment Δ from S_n to S_{n+1} should be n+1
 - Δ equals $\{(n+1)(n+2) n(n+1)\}/2$
 - $\Delta = n+1$
 - QED -- 证毕 -- इति सिद्धम
 - Or by arithmetic progression
 - $2 * S_n = (1 + 2 + ... + n) + (n + n-1 + ... + 1)$
 - $2 * S_n = n * (n+1)$
 - Therefore $S_n = n * (n+1) / 2$
- Tilde notation:
 - $\sim n^2$ or $\sim n^2/2$

- Question:
 - Stirling's approximation for n!, the number of permutations of a list of n different objects, expressed in entropy, as bits, is: $n \lg n/e + \ln(2pin)/2$. Express this in \sim ("tilde") notation.
- Answer:
 - ~ n lg n
- Notes: it's important not to include any other terms in the tilde notation.



- You are required to implement a method for the storage of up to 1 million elements. Each element has a key which represents a total ordering. It is desired to be able to select any element according to its key. However, you expect that the total number of different possible keys is somewhat greater than 4 billion (the number of possible *int* values). You don't want to search for these elements by traversing the list and comparing keys. But obviously you don't want to assign array storage to have one open slot for every possible key value! The way to do this is to implement a hash table. When you add an element to the table, you compute its *hashCode* and map that (typically by shifting bits to the right) into an index value which points to an element of the table. When it's possible (as it usually is) for many values to map to the same element (called a *collision*) there are several schemes to deal with that. The simplest is to use the next higher empty slot.
- However, for lookup (in the table) and comparison purposes, you will use the hashCode as a surrogate for the key itself. The hashCode is monotonically increasing with the key. That's to say that if hashCode(x) > hashCode(y), then x>y. However, since the hashCode is essentially a 32-bit digest of the key, the hashCode itself does not qualify as a total order for the elements. That's because it is possible that x > y or x < y while, at the same time, hashCode(x) = hashCode(y).
- If there are 100 million possible different values in the domain of the values (that's to say there are 100 million possible values of the key), what is the (approximate) probability that two different elements will have the same hash key (assume that the hashCode is uniformly distributed over the domain)?
- Answer: 1 in 43 (2³² / 100M)

```
// Sort the arrays indices and hashes by comparing hashes
private void insertionSort(int n, int[] indices, int[] hashes) {
    // TODO implement me (8 points)
    for( int i=1; i< n; i++) {
       for( int j=i; j>0 && (hashes[j]<hashes[j-1]); j-- ) {
          exchange (hashes, j);
          exchange(indices, j);
 // Verify that the arrays are in true order according to natural ordering on X
private void verify(int n, int[] indices, int[] hashes, List<X> a) {
  // TODO implement me (8 points)
    for(int i=0; i<n-1; i++) {
       if( hashes[i]==hashes[i+1] ) {
 if( a.get(indices[i]).compareTo( a.get(indices[i+1]) )<0 )
             continue;
          else
             exchange(indices,i);
```