Maxwell Cunningham

1) Get is faster for an array list because array lists allow for the random access of specific elements. In a linked list, you have to traverse the list from the beginning until you find what you are looking for.
2) B passes at at n=13.
3) You need 10 doublings of an array to get from 1 element to 1024. 2^10=1024. For large n, you need Logbase2(n) doublings to reach n elements.
4) Search: O(n) because what you're looking for may be at the back
   Get: O(1) in an array list, you can jump straight to the element you want to access

5) Output:

```
 Time taken for size=10000: 105.0
 Constant of proportionality: 1.05E-6
 Time taken for size=30000: 897.0
 Constant of proportionality: 9.966666666666667E-7
 Time taken for size=50000: 1029.0
 Constant of proportionality: -5.732694976076043E-7
 Time taken for size=70000: 4801.0
 Constant of proportionality: 7.93510824829727E-6
 Time taken for size=90000: 3995.0
Constant of proportionality: -8.154149686985157E-6
```

6) Optimized Output of 5:

```
 Time taken for size=10000: 34.0
 Constant of proportionality: 3.4E-7
 Time taken for size=30000: 221.0
 Constant of proportionality: 2.455555555555556E-7
 Time taken for size=50000: 576.0
 Constant of proportionality: -3.2089721148880473E-7
 Time taken for size=70000: 1129.0
 Constant of proportionality: 1.8660148328114178E-6
 Time taken for size=90000: 1859.0
Constant of proportionality: -3.7943840470852076E-6
```

7) Insert and search take O(n) operations because you have to traverse the list to find what you are looking for and where you are going to insert. The worst case scenario is that what you are searching for is at the end and you are inserting at the end. This does not change if the list is sorted or not, since you still have to go to the end in the worst case and there is no random access like an array.

9) In order to insert into an array, you must find the bucket where the element will go. Then, you must move everything behind it one bucket to the right. In the worst case, this is n operations; this is known as O(n). Assuming an array list is sorted, you can use binary search to search it. That means you split the data in half until you are looking at one element. You would get logbase2(n) operations. This is also known as O(log(n)).

11) The total amount of work is the work done by the outer loop multiplied by the work done by the inner. This total then needs to be added to the work done during the swap. (n(n-1)/2)+(n-1) can be reduced to ((n^2-n)/2) + (n-1). The highest degree in this function is 2. In Big-O notation, this is known as O(n^2).

12) The numbers got so large for 2^n that my function started printing 0 instead of the number and said the ratio was infinity.

13) An exponent a^n can be modeled as a*a*a…*a. A is multiplied by itself n times. A factorial of n, however, can be modeled as n*(n-1)*(n-2)*…*1. A factorial will be multiplying by greater values than a, so it will always grow at a faster rate than an exponential function.

14) F(r) has greater values than 2^r, but is not as bad as r!. 10! is an order of magnitude greater than F(10) at roughly 3.6 million. F(9) is three times lower than 9!.

F(1)=4
F(2)=12
F(3)=40
F(4)=140
F(5)=504
F(6)=1,848
F(7)=6,864
F(8)=25,740
F(9)=97,240
F(10)=369,512