6.00.1x Week 6 FAQ

- **Mistake in lecture/subtitle ?**

  We value improvements. For mispronounced words or typographical errors, please check the errata against the concerned lecture prior to post.

- **How does the computer know if `L` is an empty list while counting the number of steps ?**

  We need to realize that counting the number of steps can be somewhat arbitrary. In any event, the professor is pretty consistent in not counting a `for` going through an empty list (as long as *there is no assigment*).

  Apparently checking the list to see if it has zero element has been optimized. That means that certain operations are running at the speed of a compiled C language program, rather than being implemented using the slower Python interpreter.

- **How do we know there's 2^k possible cases for k elements in `L` ?**

  For a subset of $n-1$, the last element will be used to create a new subset with each $n-1$ subsets, thus doubling the number of subsets. Therefore, for each element in the set, the number of subsets doubles, so the number of possible subsets is 2^k.

  Here's an example for various sized lists as input.

  Table 1: Number of possible cases for k elements in `L`

List	Subsets	Number of sample(s)
[]	[]	1
[1]	[], [1]	2
[1, 2]	[], [1], [2], [1,2]	4
[1, 2, 3]	[], [1], [2], [1,2], [3], [1,3], [2,3], [1,2,3]	8

- **How does the function `recurPowerNew` work ?**

  What this code does is raise `a` to the power of `b`. If we call it with $a =$ any number and $b = 0$, then it returns 1 since anything raised to 0 is 1.

  Next if we call it with $a =$ any number and $b =$ an even number, then it recursively calls itself with the new $a = a \times a$ and $b = b \div 2$.

  To understand this consider `a = 5` and `b = 4`. Then this function calls itself again with `a = 25` and `b = 2` and then again with `a = 625` and `b = 1`. This makes sense because we can write 5^4 as $(5^2)^2$. This is also the reason that the answer was $\mathcal{O}(\log(b))$ since `b` gets halved every recursion.

  And then in the third case if `a` = any number and `b` = odd number, then it returns $a \times$ another recursion of this function with `a` being same and `b = b-1` so that `b` is now even. This is same as writing a^5 as $a \times (a^4)$.

- **Why the two recursive bisection search algorithms have different time complexity ?**

  The `bisect_search1` algorithm makes a copy of the list which increases the space that is being used but also takes additional time (recall that the complexity of the copy is $\mathcal{O}($ `len(L)` $)$.

  In `bisect_search2`, the inner function `bisect_search_helper` only needs the beginning and ending index that needs to be searched. The entire list `L` is passed each time it is called. Then the only portion searched is from the beginning index through the ending index. So the complexity is $\mathcal{O}(\log n)$ and no extra time is used up making any copies.