1 Developing an API (for Assignment 5):

Instead of reading in an expression and generating SaM code, you will read in an expression and generate an ETree. Start with the SaM parser/compiler code; you shouldn’t have to change very much at all!

**Internal node class:**

1. what instance variables will it need?
2. what should the constructor(s) look like?
3. what accessor methods will you need to later traverse this tree?

**External node (leaf) class:**

1. what instance variables will it need?
2. what should the constructor(s) look like?
3. what accessor methods will you need to later find out what’s stored in the leaf?

**For each method you write, think:**

1. What is the return type of the method?
2. What is the base case?
3. What is the recursive case?

2 Quicksort

Basic algorithm: (assuming ASCENDING sort)

1. Base case: array has one item. Return (it’s sorted).
2. Otherwise:
   
   (a) Pick a pivot element
   
   (b) Walk through array, swapping items so that:
       
       Red balls = items less than or equal to pivot (to left)
       
       Blue balls = items greater than pivot (to right)
   
   (c) Recurse on Red Ball sub-array
   
   (d) Recurse on Blue Ball sub-array

**To consider:**

1. Where is the pivot element while this happens?
2. How will you pick the pivot element?

3. How would this change to sort in DESCENDING order?

**Two pivot elements:**

1. Now we have white balls as well (greater than 1st pivot, less than or equal to 2nd pivot)

2. How does the algorithm need to change?

### 3 Recursion

Are you comfortable with recursion yet?

DO look at the solutions for Assignment 4, problem 1 in particular. The recursive solution is quite simple. (It would be even simpler without the hashSet, e.g.):

```java
int bestFit(int T, int[] v) {
    // Base case: one element in the array. Check whether it fits.
    if (v.length == 1) return (v[0] <= T) ? v[0] : 0;

    // Recursive case: either use v[0] or don’t.
    // First make a copy of v that doesn’t have v[0].
    int[] v2 = new int[v.length-1];
    for (int i=0; i<v.length-1; i++) v2[i] = v[i+1];

    // 1. Use v[0] and recurse on the remaining space and v2
    int useItem = v[0] + bestFit(T - v[0], v2);

    // 2. Don’t use v[0] and recurse on v2
    int dontUseItem = bestFit(T, v2);

    // Return the best option.
    return (useItem > dontUseItem) ? useItem : dontUseItem;
}
```

**Invariant:** array gets shorter each time a recursive call is made. Therefore, it will eventually reach the base case and terminate.

This solution is completely *recursive* (no for loop needed over v). Can you convince yourself that this works? If not, talk to a TA and/or spend some more time writing recursive programs. Recursion is a difficult concept at first, but quickly becomes a very easy and natural tool to use, especially for problems like this one. This solution only required six lines of code.

Also, if you are not familiar with the

```java
expr ? expr : expr
```

notation in Java, you should review it. It’s very useful! (And similar to the conditional expression you handled for SaM in Assignment 4).