INFO1105 assignment 1 report --- Shuwei Zhang

1. Code implementation overview:

The code I have used is a nested data structure, with main structures being **arraylist** and nested **binary search tree**.

1. Data structure usage:
   1. The **Assignment** class includes a **<studentData>arraylist** called **‘UnikeyCollection**’, which is the basis of each Assignment object.
   2. The ‘**studentData**’ object contains a String variable ‘**unikey**’ and a binary search tree’s **root**. (it acts like a map, with **unikey** being the key, and **tree root** being the value.)
   3. The **nodes** of binary search trees are the **submission** objects. Each submission object represents a distinct submission, and it has its unique **unikey, time, and grade**.
   4. In the **binary search tree of submissions**, the **time is key**, and **grade is value**. All nodes that belongs to a same tree have the **same unikey as the studentData** object that tree belongs to, which helps to filter out the submissions that belongs to the other students.
   5. **In conclusion**, the data structure is an arraylist that represents different students, containing binary search trees whose nodes are different submissions of this student.
2. Method analysation:

Assume: there are ***m* different unikeys**, **each unikey have *n* different submissions**. So the ***total number of submission is mn,* the basic running time is O(mn).**

* 1. getBestGrade : **loop through m** **unikeys** to find the matching unikey, then **traversal through n nodes** under that unikey to find the best grade.

So the running time is **O(m+n)**. (less than O(mn)).

* 1. getSubmissionFinal : **loop through m unikeys** to find the matching unikey, then use **binary tree search method** to find that node that has biggest key(Date variable).

So the running time is **O(m + log(n))**.(less than O(mn)).

* 1. getSubmissionBefore : **loop through m unikeys**, then use **binary tree search method** to find that node that has biggest key less or equal to the deadline.

So the running time is **O(m + log(n))**.(less than O(mn)).

* 1. Add : **loop through m unikeys**, then **use binary tree search method** to find the new added key’s position it should be.

So the running time is **O(m + log(n))**.(less than O(mn)).

* 1. Remove : **loop through m unikeys**, then **use tree search method** to find the key and remove, if the node has 2 children, use its predecessor to replace it.

So the running time is **O(m + log(n))**.(less than O(mn)).

* 1. listTopStudents : **loop through m unikeys**, in each unikey **find its best grade** to put the ones got higher than current highest mark into the list. If one unikey has the higher mark, clear the list and put the high mark unikey into it,

So the running time is **O(mn)**. (still, less than O(m2n2)).

* 1. listRegressions : **loop through m unikeys**, **in each unikey find its best grade and its latest grade**, if the best grade is lower, put the student into the list.

So the running time is **O(m\*(n+log(n)))**.(less than O(m2n2).

Note that in b), c), d), e), and g), we **assume** all the **binary search trees are balanced**. For the **worst case scenario**, all of the **log(n)** operations will cost time **O(n).**This will happen if all the nodes are ordered in one single branch.