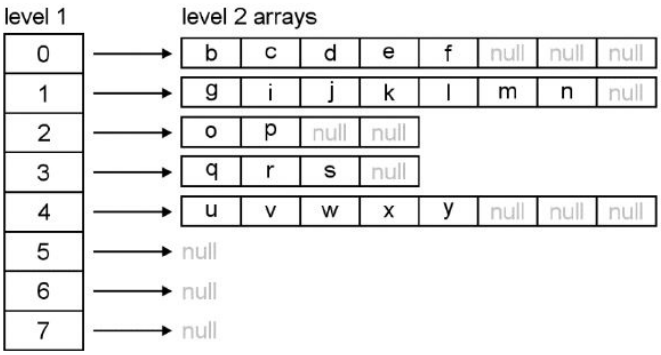
CSCI-290 *Project Part 4*

Ragged Array List - Add

Objectives: Building a new data structure (the add method). Now we are going to write the add() method. Your findEnd() method needs to be working perfectly because you will use it to find the insertion point. For example findEnd("h") would return ListLoc(1,1), and then "h" would be inserted at that location. The following values "i" through "n" would all be pushed up to make room.



More Data Structure Details:

- Remember there is ALWAYS room to add one element.
- If you add an item to a level 2 array, and that array becomes full, you either split that array into 2 level 2 arrays of the same size but each only half full after the split, or you double the size of that level 2 array.
- The decision of whether to split or double is based on the sizes of the level 1 and level 2 arrays. If the level 2 array is smaller than the level 1 array, then the level 2 array should be doubled, otherwise it should be split.
- For example if an "h" were added, its level 2 array would become full and be split into two level 2 arrays of size 8, each with four items. The l2arrays in slots 2 through 4 of the level1 array would all be moved down to make room.
- If another "r" was added, that level2 array of size 4 would become full, and so it would be doubled to size 8 to make room for future additions.
- If you split an L2 array, it's possible that the level 1 array becomes full, it would then be doubled.
- If things work well, as the data structure grows large both the level 1 array and the level 2 arrays will be approximately of size $\sqrt{N}$
- The initial array size is 4. This is the minimum size for any array. Initially, the level 1 array should have 4 slots, and the first slot will hold a reference to a level 2 array, also with 4 slots.

Setting up the testing code:

1. On Brightspace you will find a new testing file:
   - RALtester2.jar
2. Remove RALtester.jar from the list of jar files for your project.
3. Copy the new jar file into the project folder (where the other jar files are).
4. Configure the build path: File - Project Properties - Libraries - Run [Add JARs]
5. The RALtester2.jar and Scaffold.jar contain code that will test your add() method that you will be writing for this assignment.
6. You can use the same run configuration you used for the RALtester. Internally RALtester2 has the same package and class names. It just has new test cases.
7. Run the testing code. It should run and has 15 test cases.

Task 1: Implement add()

- Duplicates are allowed. When adding a duplicate item, it should always be added after the last matching item. That is the position that findEnd() finds.
- An invariant of the design is that there is always space available to insert another item. In both level 1 and level 2 arrays there is at least one unused slot. This invariant will make add() a little easier to write because you can trust that there is room for the insertion, and then you can deal with the doubling or splitting afterwards.
- For loops are not necessary for splitting. Some useful Array methods:
- Arrays.copyOf(origArray, newLength) is useful for growing an array.
- System.arraycopy(srcArray, srcPos, destArray, destPos, length) is useful for inserting or splitting. srcArray and destArray can be the same array and it will still copy correctly.
- Arrays.fill(array, fromIndex, toIndex, value) is useful for cleaning up after splitting
- Suggestion: Build and test incrementally! Don't try to write this all at once. Do little pieces that you can test. Write enough code just to insert into the level 2 array. Once that is tested and working, write the code to double a level 2 array. Continue developing and testing in little steps like this.
- Keep it simple. My code used only 25 lines (not counting comments, blanks, and }'s).

Task 2: Analysis of best and worst cases The ragged array list may look a bit sloppy because some rows are fuller than others. We are going to calculate how much of a difference that could make between the best and worst cases.

1. Best Case: We will soon be adding 10514 songs into a ragged array list. Calculate the smallest possible size of a level 1 array that could accommodate 10514 items. Your calculation should be based on the rules for the ragged array list. Show your work and explain your answer.

2. Worst Case: Now calculate that largest possible size of a level 1 array resulting from inserting 10514 items. Hint: this worst case will occur if the items are added in sorted order, so that each new item is added onto the last level 2 array at that point during the building process. Show your work and explain your answer.

What to turn in:

Written part:

1. You may need to set your pdf printer as the default before running the application to print the results.
2. Include all member names at the top of the writeup. Everyone should submit on Brightspace.
3. Upload a pdf printout of the Report tab in the RALtester. Use the print button.
4. Upload a pdf of your results for test13.txt in the RALtester. Use the print button and select Page Setup - Landscape and fit to page so that I can read it.
5. Your analyses from Task 2.
6. Any incomplete parts, or any known bugs in your code should be documented.

Electronic submission:

- I will create groups and put you into them. Anyone in the group can upload files.
- Print your write up to pdf name it Project4.pdf to the project folder be sure that the names of the team members are in the write up on the first page.
- Upload the Project4.pdf, Report.pdf, test13.pdf and RaggedArrayList.java to Brightspace. The pdfs can be combined into one pdf if you have that capability.

Grading:

Written part - report and specified test case	10 points
Task 1 add()	60 points
Task 2 analyses of best case and worst case sizes	30 points

I expect your programs to be properly commented and use good style. Points may be deducted for egregious disregard of these matters. Every method in every class must have complete documentation including authorship. A full an complete revision history should be at the top of every class file!