Notice

- By completing and submitting the "giveup.txt" file, you will receive 40 points for the assignment. Note that failure to submit will result in a score of 0.

  While this option may not be ideal, it is a better alternative to cheating, which can lead to more severe consequences. Please be aware that it is not possible to partially complete the assignment and receive a partial score; you must complete and submit the entire assignment or opt out by submitting the "giveup.txt" file.

- We cannot accept late submissions for **any excuse**.

- Please visit the AR website on academic integrity and the COMP syllabi webpage to learn the consequence of cheating in COMP2011.

- Please read the comments carefully about what should and should not be changed. In particular, list all the resources you have used in finishing the assignment and running times of your methods, with the smallest possible functions.

- No imports (`java.*`) anywhere except in `Tester.java`. The package `java.lang` (e.g., `String`, `Comparable`) is OK since it's auto-imported.

- Submission procedure:

  1) Name the .java file as `<class_name>_<secret_number>.java`. Your secret number can be found on Blackboard. Since the files will be released on Blackboard, please double check that they do not contain any identification information.
  2) Put files `DaryHeap_<secret_number>.java` and `PolyuTree_<secret_number>.java` into a folder with name `A2_<secret_number>`. Please double check that nothing else (especially the `.class` files) is there.[1]
  3) Create a .zip or .jar file to contain this folder, similar as the distributed file.
  4) Each deviation from 1) or 2) will result in a deduction of 10 points; a submission not in zip/jar format will be treated as "giving up," i.e., 40 points without grding.

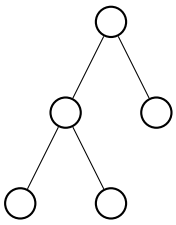
[1] Mac OS X users may find this link useful.

1. (60 points) Implement a binary search tree (BST) to store University students. Each node stores one student, keyed by the student's full name in the order "Surname, Given name" (e.g., "Chan, Alice"). The BST must support fast lookup.

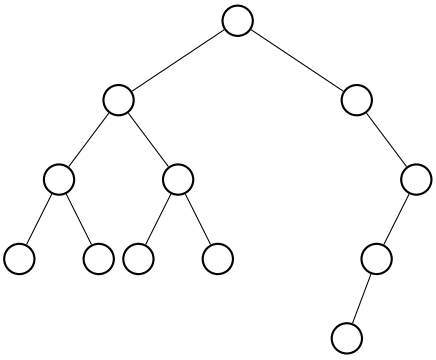
   The first part evaluates the shape of your tree. While there is no single measure of how close a binary tree is to a perfect tree, report the following three quantities:

   - (10 points) The total number of nodes that have exactly one child.
   - (15 points) The maximum difference between the depths of the left and right subtrees over all nodes.
   - (5 bonus points) The maximum difference between the sizes (number of nodes) of the left and right subtrees over all nodes.

   For illustration, these values are $(0, 1, 2)$ for tree (a) and $(3, 3, 3)$ for tree (b).



(a)



(b)

   The second part is about search. It supports three ways of search.

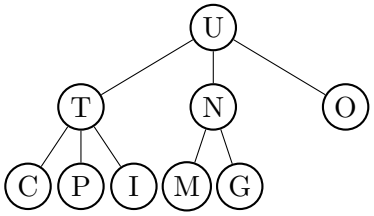
   - (10 points) Exact full-name search: return a student whose full name matches exactly the query string. If multiple students share the same name, you may return any one of them.
   - (15 points) Surname search: return all students with the specified surname. You may return `null` or an empty array if no matches. For simplicity, treat the first word as the surname (e.g., "Au Yeung" is treated as surname "Au").

   Implement the following methods. Do not change their signatures; you may add private helper methods as needed.

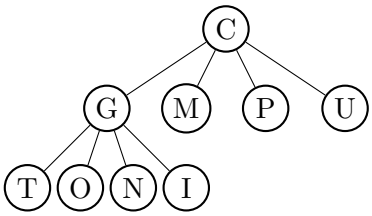
```
void insert(Student s)
int maxDepthDiff()
int maxSizeDiff()
int nodesWithOneChild()
Student searchFullname(String name)
Student[] searchSurname(String surname)
```

   Analyze the running time of your implementation.

2. (40 points) A *d-ary max heap* generalizes the binary heap by allowing each node to have up to d children (instead of 2). A d-ary min-heap is defined analogously. For example, inserting the nine letters of "COMPUTING" into a tertiary (3-ary) max heap and a quaternary (4-ary) min heap yields the trees below.



(a) A 3-ary max heap



(b) A 4-ary min heap

Let d be the last digit of your secret number. If d is odd, implement a 3-ary max heap; otherwise, implement a 4-ary min heap.

1. Implement the class `DaryHeap`.

```
class DaryHeap <T> {
    DaryHeap(int capacity) {}
    void insert(T x) {}
    T removeRoot () {}
    void up(int c) {}
    void down(int ind) {}
}
```

2. Implement a method to merge another d-ary heap into the current heap.

```
void merge(DaryHeap<T> heap) {}
```