1. **Goal**

The goal of this assignment is to gain some experience with Java nested classes, generics, and the implementation of a tree sort.

2. **Problem Statement** [client’s statement of their need]

Our lads, Sponge Bob and Patrick, have decided to help Sandy sort her nuts, using, wait for it, a *tree sort*. The algorithm is described from 17:45 to 20:30 in the amazing video found at https://www.youtube.com/watch?v=SJwEwA5gOkM. A word of warning you are not implementing a sort using a BST.

3. **Analysis** [What is Sandy’s problem?]

Q: Hey Sandy, **what**’s a tree sort?
A: Tree sort starts with the array to be sorted as the leaves of a binary tree. It then promotes the smaller of each pair to be the parent. The smaller of each pair of parents is then promoted to be the grandparent and so on until the smallest value is at the root of the tree. After each promotion the vacancy left by the promoted value is immediately filled by promoting the smaller value of its two children.

The sorted data is extracted by repeatedly removing the root value, appending it to the sorted output, and then filling in the vacancy.

Re-watch the video and work some simple examples with playing cards until you grok the algorithm.

Q: Hey Sandy, **what** kind of interface do you want?
A: A command-line interface that supports the following options:
   
   - `n <number list>` sorts the numbers of number list
   - `r <count>` sorts count random numbers
   - `-t` is an optional flag that causes the time taken while sorting to be printed
   - `-d` is an optional flag that turns on debug mode, which outputs at least the tree before each extraction.
   
   Note that `-t` and `-d` can appear alone or in either order but always before `-n` or `-r`. Also exactly one of `-n` or `-r` will be present.

Q: Hey Sandy, any other requirements?
A: Yes, your tree sort must be able to sort objects of any comparable class and must include a nested node class. (Think ```class Node { protected Node left, right; ... }```).

4. **Design** [How will you as a software engineer solve this problem.]

Q: (ask yourself) **How** should I represent the tree and its nodes.
A: While I’ll leave the bulk of the *how* decisions to you, I will emphasize that you must implement a *generic* tree of `Comparable` objects with a nested node class.

**Assignment Requirements** [part of being a course rather than part of software development]

- Each source code file must start with

  ```
  /**
   * This is my code! It’s goal is to ....
   * CS 312 - Assignment 6
   * @author Your Name
   * @version
   */
  ```
• An application program interface (API) enables other programs to interact with your program. Check out https://www.webopedia.com/TERM/A/API.html. To support its grading your solution to Assignment 6 must provide the following two methods as part of its API.

```java
/**
 * purpose: construct a tree from a List of values
 * input: the list, values, to make the tree of, and a debug flag
 * output: a tree and if debug == true print status information
 */
public Tree(List<E> values, boolean debug)

/**
 * purpose: perform a tree sort
 * input: just the tree (which is an attribute of the class)
 * output: the sorted list
 */
public List<E> sort()

Here is an example call
Tree<Integer> t = new Tree<>(ns, false);
List<Integer> it = t.sort();
```

• Let’s do some empirical data analysis! Let $N$ be the largest number of values your program can sort in 5 seconds (using `-t -r N`). Output the time taken to sort $N, 2N, 4N,$ and $8N$ numbers. Do you see a pattern between the $N$’s and the time taken?

   Record your (nicely formatted!) empirical data in README.md and then add a sentence or two, explaining any patterns.

5. What to hand in

[ By 6am Monday push a one paragraph summary of the requirements, and optionally a UML class diagram, OOA1, OOA2, and OOD1. I will comment on however much you have pushed. ]

   (1) A well-formatted 2-up printout of your source code.

   (2) A GitHub repo that includes (you must use these names as the grading script assumes them!)

      • README.md with the sections plateau schedule, test plan, and data analysis; and

      • your Java source code with main in <your repo>/Driver.java.

6. Notes

• Consider using the testing tool JUnit to help with test automation.

• Reread the general notes regarding style, braces, and header comments.

• You can get the amount of memory that Java is using with `long memoryUsed = Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()`

• You can compute the time taken using `System.currentTimeMillis()` before and after the code to be timed.

• The grading rubric extends that of past assignments to include correct implementation of generics and use of nested classes.