Category	Total Marks	Remarks	Partial Marks
Correct Working	**3**	Test-1) Once launched, the program must ask user for following options: It would first ask to enter the total number of nodes. After this it should ask a) input the set of nodes to search b) specify the technique the user would like to use for the above query processing, i.e., explicit multithreading or ForkJoinPool, and c) total number of threads that should be used in parallel processing.	**0.3**
		Output for: a) Time taken for tree construction/traversal, b) height of the tree, and c) height of the found goal nodes	**0.3**
		Test-2) Enter number of nodes as 500000 and repeat this below test for both the approaches of multithreading using threads 2 and 4 (total 2 x 2 x 2 tests): "**Input single and multiple random integer(s) within the supported range (each value representing the data of a node in tree) to search and program should return the found status (true/false) for each specified input**". Program should not hang in either case. No need to check for speedup/efficiency.	**0.4x6**
Tree creation	**4**	Correct sequential recursive implementation for construction the tree (both BFS/DFS approach is fine)	**1.5**
		Tree should be an unbalanced tree, i.e., total children at each node should not be same. It should be randomly chosen within the range provided in documentation.	**2**
		Correct approach for calculating the height of the tree	**0.5**
Design pattern	**3**	Flyweight design pattern must be used for adding a new node in the tree such that only unique values are added in the tree	**3**
Tree Traversal using explicit multithreading	**4**	Total threads mentioned by the user would be created [using thread.start()] and each thread would be given a different portion of the tree to search. It is hard to perform an optimal work division in this approach so any form of work division among threads should be fine as long as each thread got some portion of tree to explore	**3**
		Thread.join() called for each of the threads in explicit multithreading	**1**
Tree Traversal using ForkJoinPool	**4**	Recursive task creation (tree traversal) using fork() inside compute method. Both BFS or DFS approach for tree traversal is fine	**2**
		All parallel tasks should be joined using helpQuiesce() API instead of join() API	**2**
Speculative Parallelism (its fine if student has implemented in only one of the approaches for multithreading instead of both)	**2**	Global check if all the goal nodes specified by ther user are found ---(OR)--- if a node is searched already, program should reuse the previous result instead of searching again	**0.5**

		For explicit multithreading, threads / tasks should stop exeution as soon as all the nodes are found. For ForkJoinPool implmentation, consider two cases - a) once all the nodes are found, to stop further searching, shutdownNow() API should be used with try/catch block for catching CancellationException. ---(OR)--- b) If the program is implemented by searching each number one by one (instead of all nodes together) implementatoin for ForkJoinPool, shutdownNow() API should be used once that node is found. (Effectively stopping further search for that node by other threads)	**1.5**
Total	**20**		**20**