Assignment Lab 1
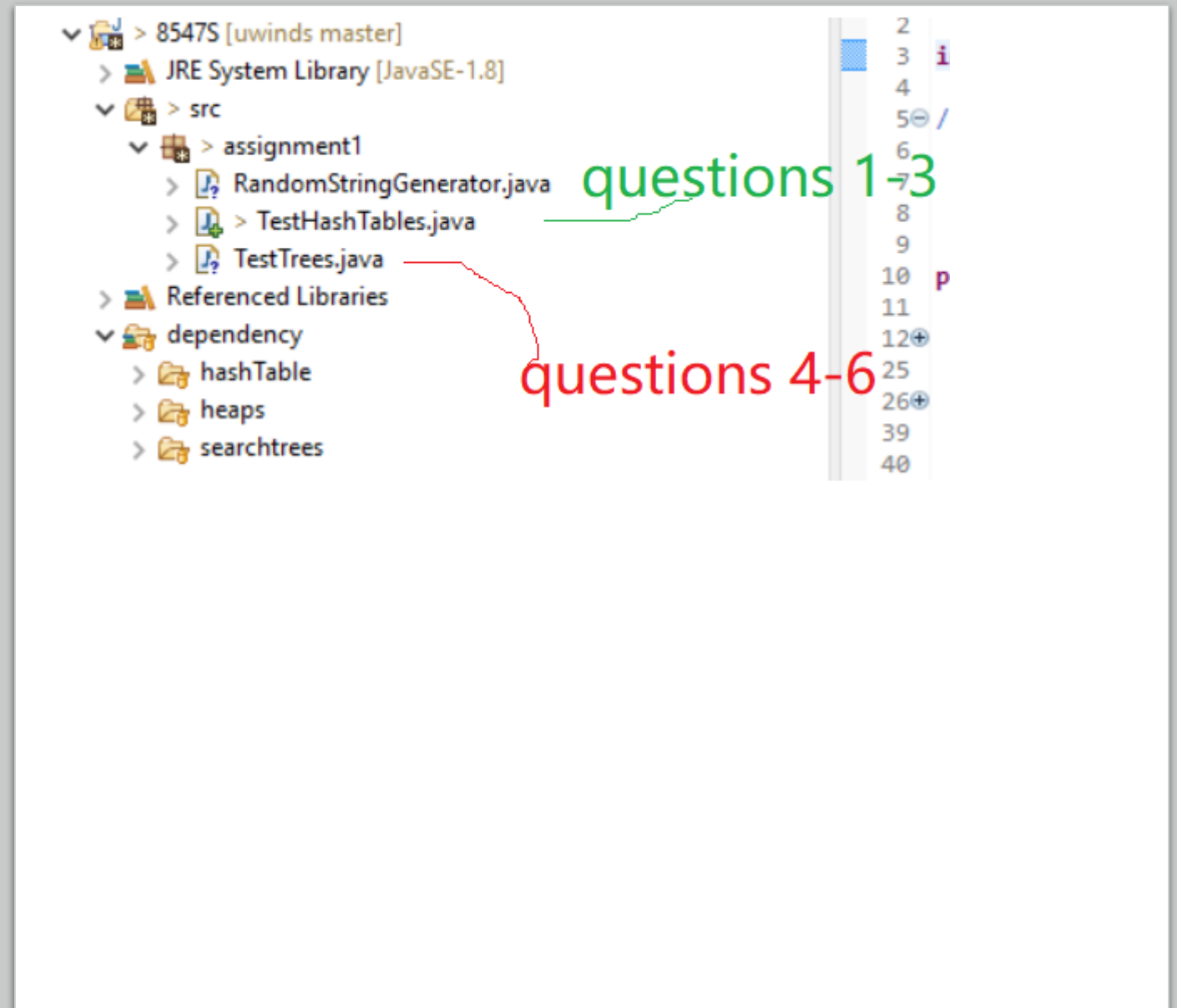
This is the source code structure corresponding to all the questions.
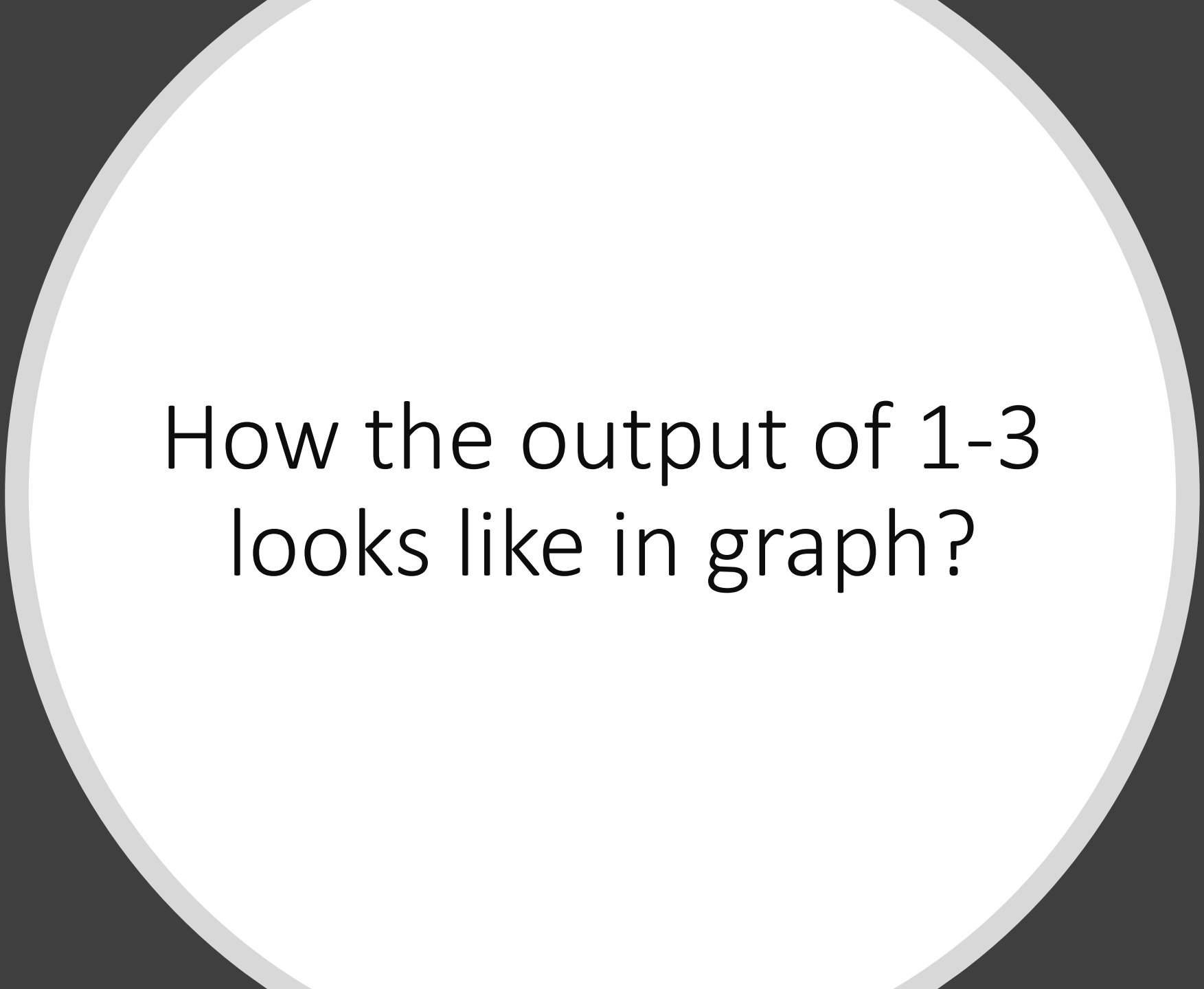
a.Run TestHashTales.java for questions 1-3;

b. Run TestTrees.java for questions 4-6

# Output of the program for questions 1-3

- Cuck I: Cuckoo hash able Insertion average time in **nano** seconds, likewise for "Quad I" and "SCh I" in the table headers.

- Cuck D: Cuckoo hash table Search or Deletion average time in **nano** seconds, likewise for "Quad D" and "SpCh D" in the table headers.

| i	Cuck I	Cuck D	Quad I	Quad D	SpCh I	SpCh D
1	33712	12420	32727	6112	93055	4929
2	11040	7196	1873	1380	2464	1281
3	110355	7343	37606	936	3795	1133
4	18877	6530	1010	739	1601	788
5	53501	2341	1121	837	1306	690
6	3012	3135	2248	2636	1164	708
7	3376	2150	2760	431	13252	877
8	4018	2542	999	696	5141	349
9	1286	679	1133	408	4336	428
10	1569	4079	1026	820	2938	309
11	5240	550	1009	255	2317	292
12	2148	530	1515	1511	9599	223
13	540	248	433	438	2461	270
14	547	209	202	93	428	131
15	1072	306	278	161	958	242
16	1093	645	726	276	1813	281
17	1200	640	770	212	1171	225
18	1637	673	873	261	1667	187
19	2289	826	1345	296	1697	248
20	1413	940	1126	322	1538	306

How the output of 1-3 looks like in graph?

Insertion average time in nano seconds
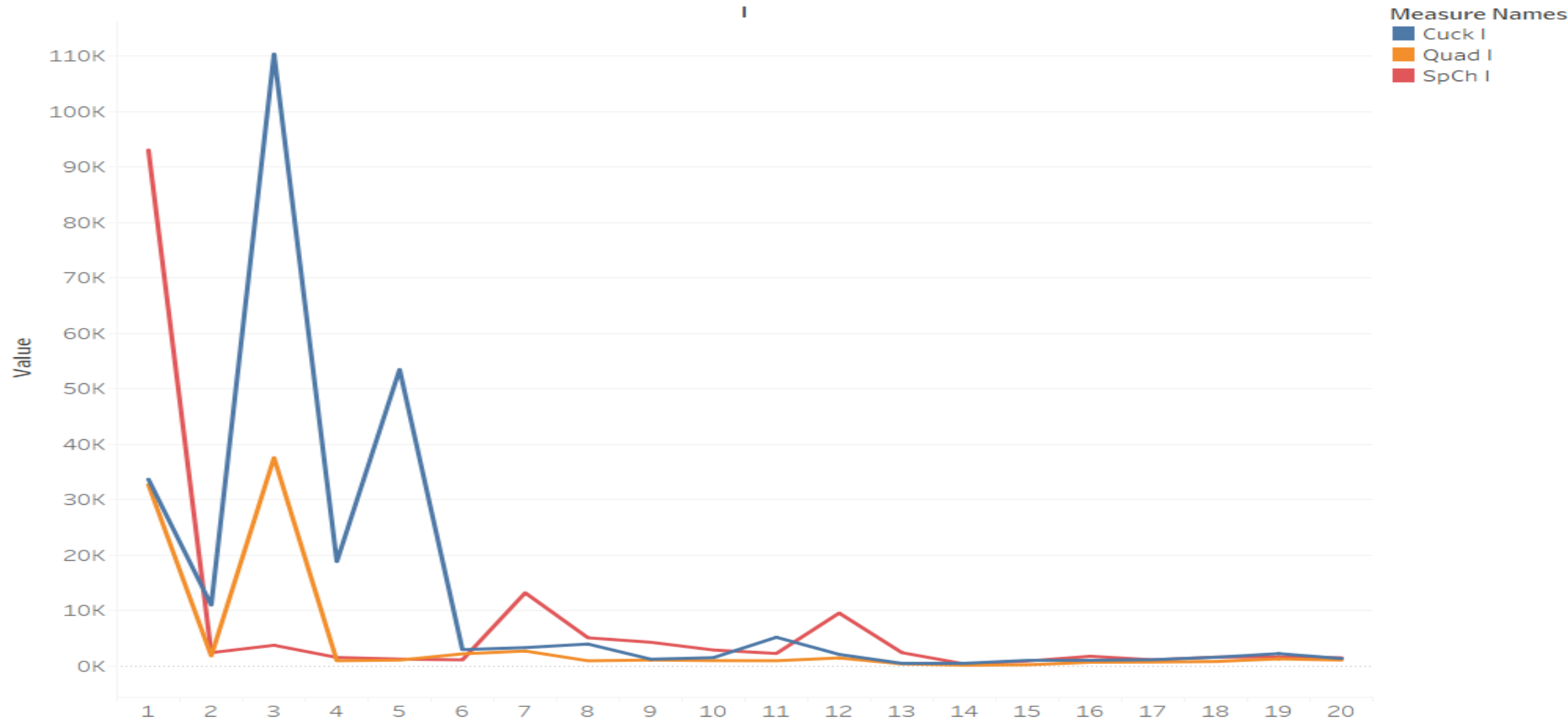
The trends of Cuck I, Quad I and SpCh I for I Day. Color shows details about Cuck I, Quad I and SpCh I.

Search / deletion average time in nano seconds
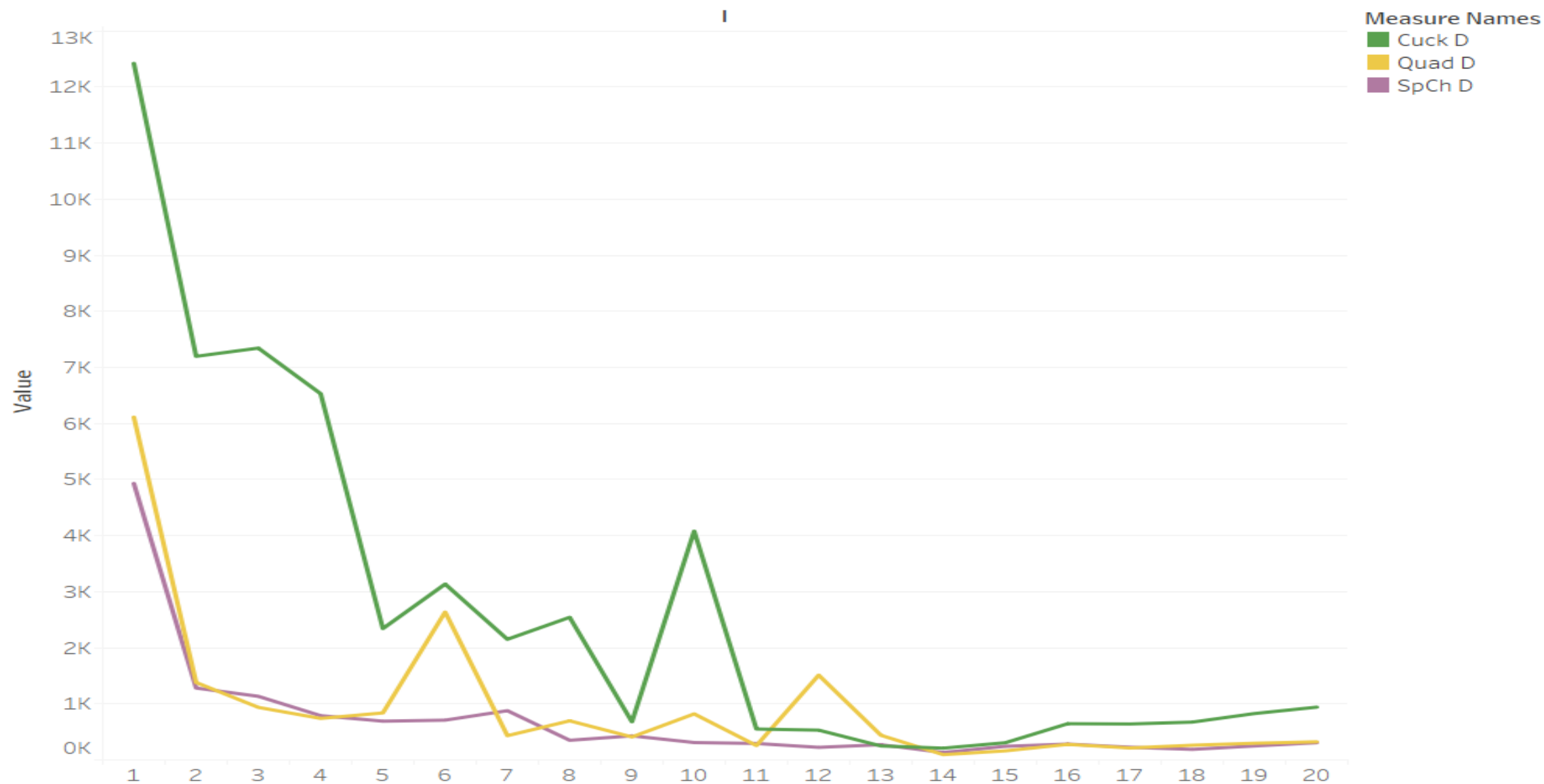
The trends of Cuck D, Quad D and SpCh D for I Day. Color shows details about Cuck D, Quad D and SpCh D.

Measure Names
- Cuck D
- Quad D
- SpCh D

Result analysis

For all Hash tables, expected running time of insertion, search, and deletion is O(1), as showing in the two preceding slides all the average time are approaching a horizontal line.
Whereas it displays unstable with high average time at the beginning for each line, that is owing to two reasons:
1. Overhead caused by environmental factors are augmented for average time when N is small
2. Overhead caused by accessory actions are augmented for average time when N is small, for example, increasing table capacity, or rebuilding hash table, et cetera.

Output of the program for questions in 4 and 5

Tree	4.a	4.b	4.c	5.a	5.b	5.c
BST	1159902	655528	1159699	961	1162	1162
AVL	736	1144	953	1613	854	1337
RedBlck	898	1016	1899	1784	1212	3413
Splay	313	1345	460	2090	1947	1440

4.a, 4.b, 4.c represent insertion, search and deletion average time in nano seconds for **sequential** inputs in question 4
5.a, 5.b, 5.c represent insertion, search and deletion average time in nano seconds for **random** inputs in question 5

Result analysis for trees test

As we learned from class

Review and comparison

Search tree	Average case			Worst case		
	Search	**Insertion**	**Deletion**	**Search**	**Insertion**	**Deletion**
BST	O(log n)*	O(log n)*	O(log n)*	O(n)	BO(n)	O(n)
AVL	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)
Red-black	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)
wavl	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)
Splay	O(log n)	O(log n)	O(log n)	O(n)	SO(n)	O(n)

* Average case for a BST assumes a series of insertions and deletions of n randomly generated keys.

Binary Search Tree – sequential inputs

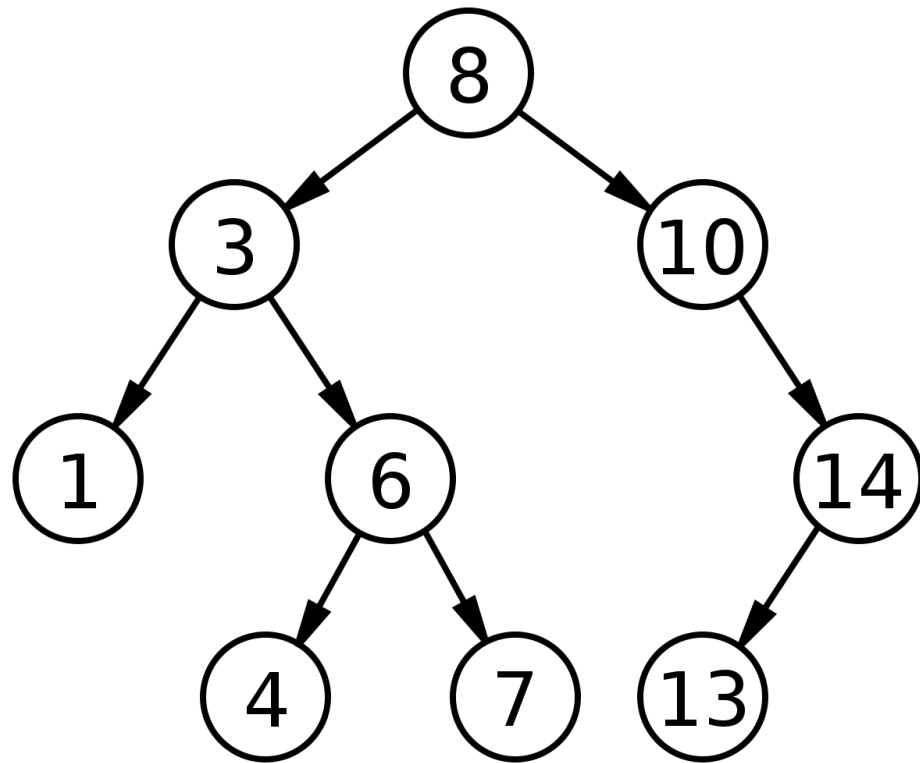| Tree	4.a(insert)	4.b(search)	4.c(deletion)	5.a(insertion)	5.b(search)	5.c(deletion)
BST	1159902	655528	1159699	961	1162	1162

4.a - insertion integers [1, 100000] in order, will result in worst performance case O(n) and end up with a tree of height N (100000).

4.c - likewise deletion [100000,1] in order from the constructed tree by the insertions would perform in worst case O(n) as well, because each deletion needs to traverse to the rightest/highest node.

4.b - actual deletion average time for random numbers is O(n/2) as the average depth is n/2, which is O(100000/2)
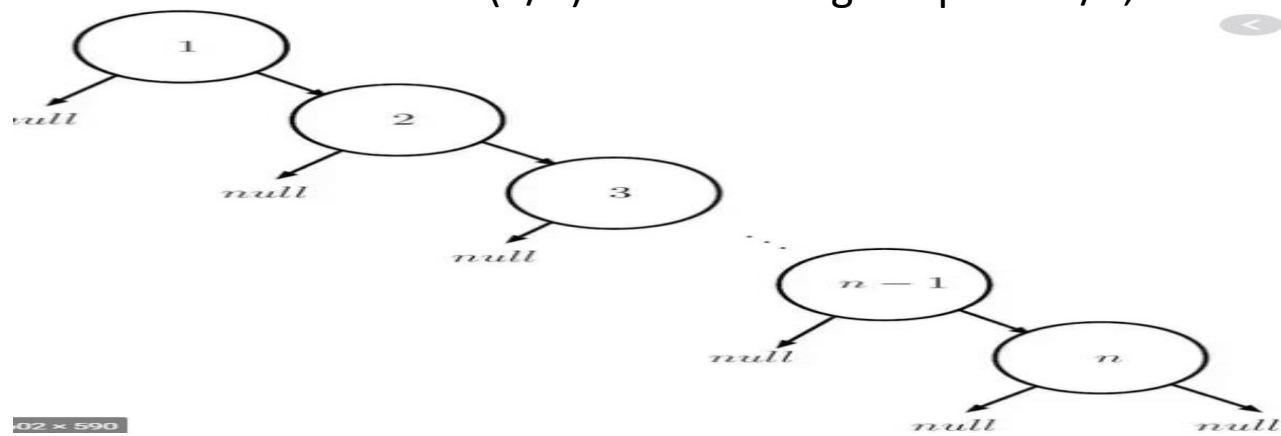
Binary Search Tree – random inputs

- 5.a – insertion of **random** numbers will result in a tree of height approaching O(log n), running time is O(log n), it is same true for 5.b search and 5.c deletion.

Tree	4.a	4.b	4.c	5.a	5.b	5.c
BST	111261	60310	107332	1255	1177	1587

AVLTree & Red-Black Tree

AVLTree and Red-Black trees are balanced trees, with height log(n), so Insertion, search, and deletion are all of time complexity O(log n).

Tree	4.a(insert)	4.b(search)	4.c(deletion)	5.a(insertion)	5.b(search)	5.c(deletion)
BST	1159902	655528	1159699	961	1162	1162
AVL	736	1144	953	1613	854	1337
RedBlck	898	1016	1899	1784	1212	3413
Splay	313	1345	460	2090	1947	1440

Splay Tree

As we learned, SplayTree's insertion, search and deletion are of O(log n) and at worst O(n), it can be analyzed case by case.

For 4.a (insertion), the splaying will make each insertion of ordered numbers as the root's child therefore the time complexity is approaching O(1)

For 4.b (search), the tree is getting more and more balanced due to splaying along with search, so complexity is O(log n)

For 4.c, deletion of reversely ordered numbers [100000,1] looks better then O(log n), because when a key is deleted the slightly smaller parent will get splayed, which is likely to benefit the next deletion.

5.a,b,c – all the actions all are of O(log n) for random inputs as the numbers show.

Tree	4.a(insert)	4.b(search)	4.c(deletion)	5.a(insertion)	5.b(search)	5.c(deletion)
BST	1159902	655528	1159699	961	1162	1162
AVL	736	1144	953	1613	854	1337
RedBlck	898	1016	1899	1784	1212	3413
Splay	313	1345	460	2090	1947	1440

Answer of question 6

Tree	4.a(insert)	4.b(search)	4.c(deletion)	5.a(insertion)	5.b(search)	5.c(deletion)
BST	1159902	655528	1159699	961	1162	1162
AVL	736	1144	953	1613	854	1337
RedBlck	898	1016	1899	1784	1212	3413
Splay	313	1345	460	2090	1947	1440

Let's render them in graph for comparison.
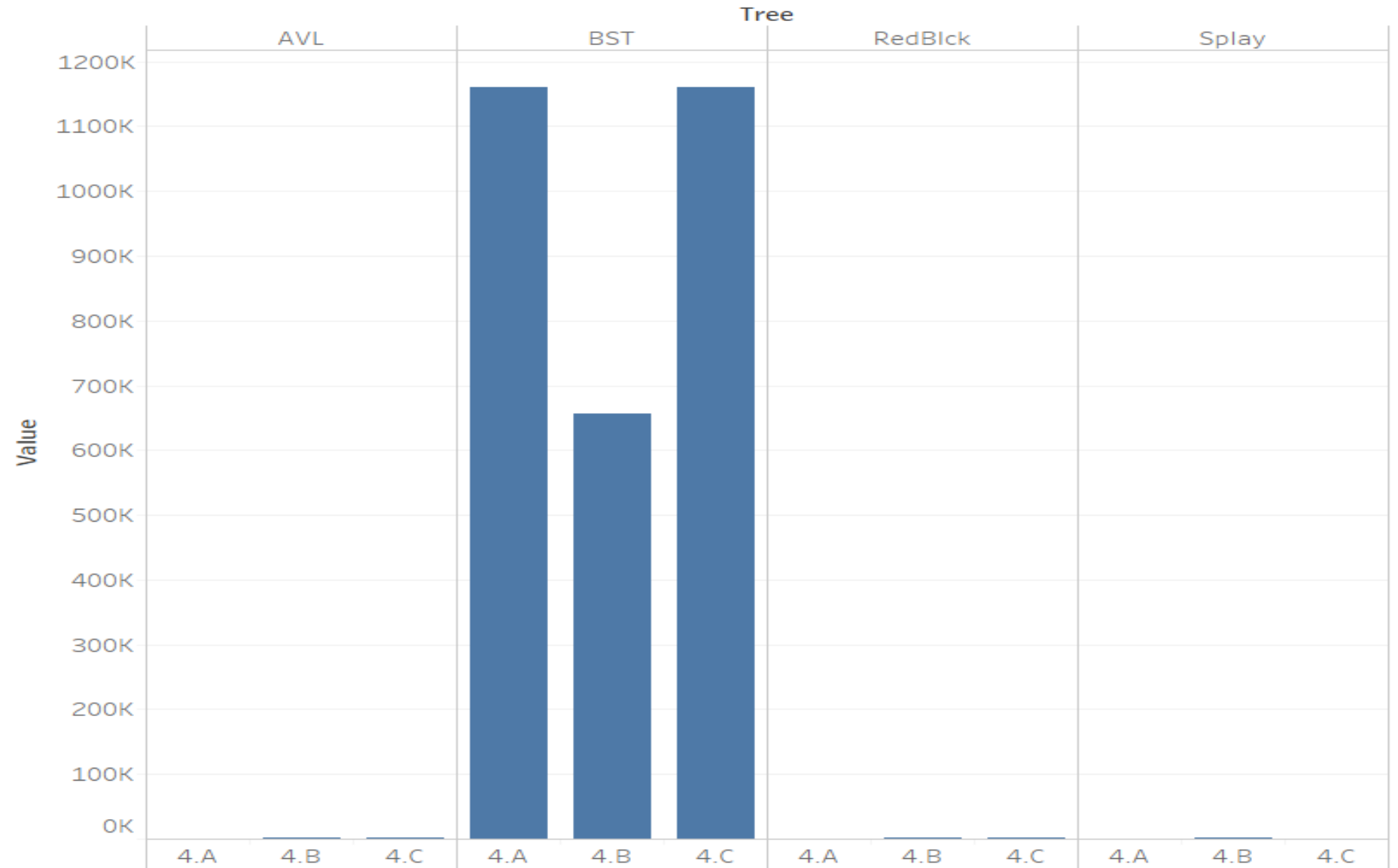
For random inputs, average time for all are decent at average complexity O(log)

average time for random inputs

Answer of question 6

Tree	4.a(insert)	4.b(search)	4.c(deletion)	5.a(insertion)	5.b(search)	5.c(deletion)
BST	1159902	655528	1159699	961	1162	1162
AVL	736	1144	953	1613	854	1337
RedBlck	898	1016	1899	1784	1212	3413
Splay	313	1345	460	2090	1947	1440

we can see actions on all the tables are of average-time complexity O(log n), except Binary Search tree for sequential inputs which Performs at worst case O(n). For real problems, if the inputs are random data, I will choose Binary Search Tree or AVLTree, otherwise for sequential inputs I will choose AVLTree, and choose Splay tree especially when there is many reoccurrences of search for a same key.

(* average time in nano seconds.)