CSCI 201 Friday, November 21, 2025 Lab 8

DATA STRUCTURES

Hash Table • Hash Function • Collision • Linear Probing

LinearProbing

Insert the following keys into four different hash tables (`T1`, `T2`, `T3`, `T4`). Start each hash table with a capacity of `8`, and use linear probing to handle collisions. Double the capacity when the load factor α `> 0.5`*. Each table should have a capacity of 16 after all keys have been added.*

`T1`: 6, 14, 35, 16, 18, 32, 4, 17
`T2`: 15, 11, 9, 45, 23
`T3`: 0, 2, 48, 12, 67
`T4`: 6, 14, 35, 18, 33, 4, 17

key	letter
0	t
2	e
4	w
6	r
9	a
11	k
12	r
14	k
15	s
16	t
17	o
18	a
23	m
32	e
33	e
35	m
45	e
48	h
67	d

Then use the table on the right to convert each key from the resulting tables to a letter. Note that different keys can map to the same letter. Decode the (non-`null`) keys from left to right in each table (in the order of `T1`, `T2`, `T3`, `T4`):

RuntimeComplexity

microseconds

The tables on the right show experimental timing data (in µs) for a few methods of certain `Java` Collections (`C1`, `C2`, `C3`). These collections are either an `ArrayList`, `HashSet` or `TreeSet`. Use the timing data to determine which collection `C1`, `C2`, `C3` might be.

- *Treat the `add` data as the total time to `add` `n` items to the collection.*
- *The `contains` and `remove` tables report the total time to call these methods 100 times for a collection of `size` `n`.*

add

n	C1	C2	C3
100	38	182	126
1000	409	2586	1256
10000	4370	124389	19928

contains

n	C1	C2	C3
100	394	121	35
1000	4619	178	40
10000	65689	320	39

remove

n	C1	C2	C3
100	287	89	32
1000	7730	182	34
10000	99444	334	33

*Hint: create a copy of the shared google sheet from the website. Then, create charts for each method and set both x- and y- axes to a **Log Scale**.*