Data Type Comparisons

Mac Decker

December 8, 2019

1 Introduction

Given two different hashing algorithms and four types of collision resolution: linear probing, chaining with a linked list, chaining with a binary search tree and cuckoo hashing; our goal is to compare the time performance of the insertion, deletion and lookup of a group of 100 elements at various load factors.

2 Procedure and Methodology

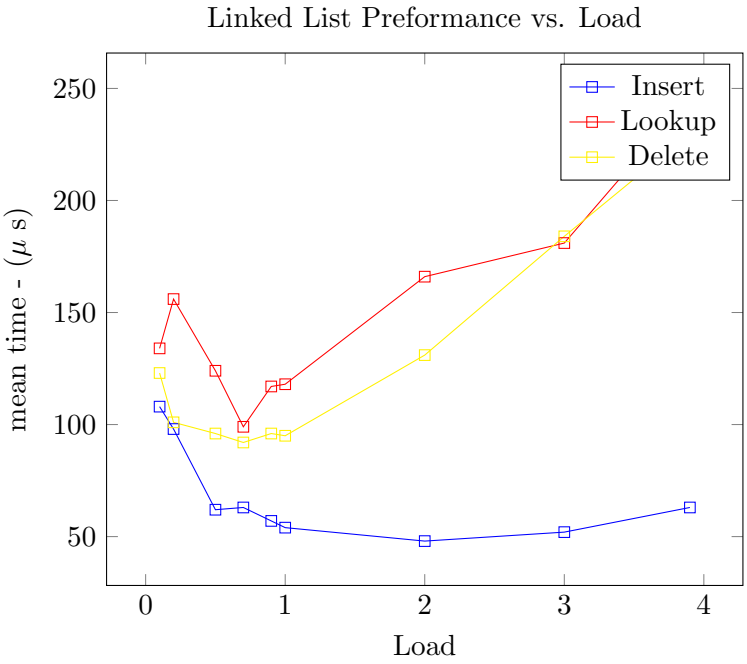
We will use an array of binary search trees with a left and right pointer to store our hashed data for each type of collision resolution.

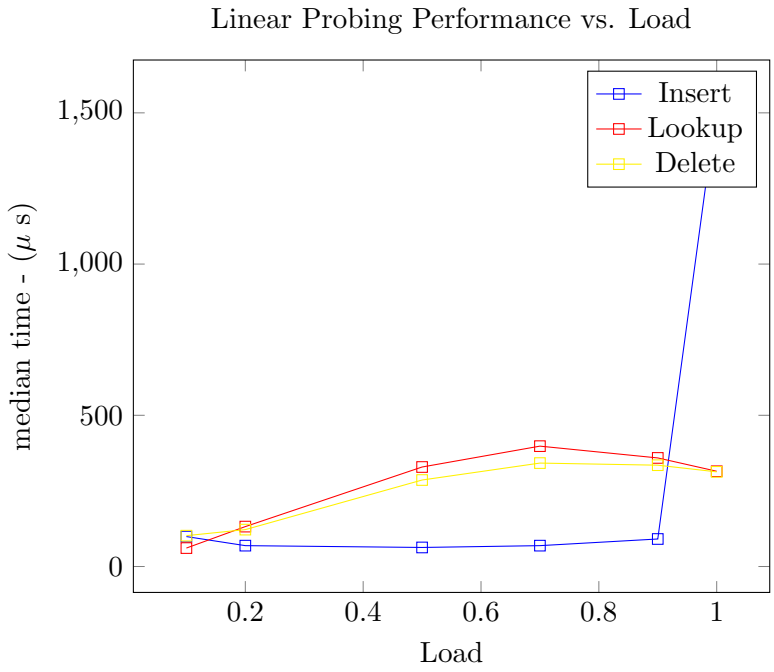
- The linear probing implementation only needs the base array to store our data.
- Utilizing only the left branch will give us the functionality of a linked list.
- Adding the right branch and forcing balancing will give us the functionality of our binary search tree.
- For the cuckoo hashing we first try to put the hashed value in our array and if there is a collision we try the second hash function, this so far only requires the base array. If there is a collision again at the second hash value, the new item overwrites the original collision, and the overwritten number is moved to the second hash value. This can cause a cycle if something would eventually move itself. Keeping track of what moved an item will let us know when to resize the array.

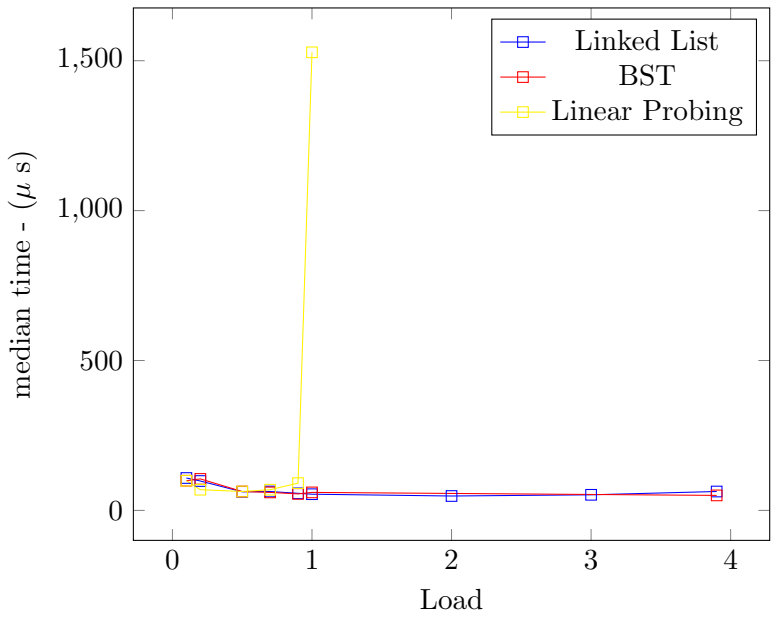
We started by creating an associative array of numbers pulled at most once randomly from the data set.
After that, program takes the desired hashing algorithm, collision resolution and load input from the user, inserts data until the desired load value is achieved and then runs the different variations of the insert, lookup and delete functions in groups of 100 to get more measurable results.

3 Graphs

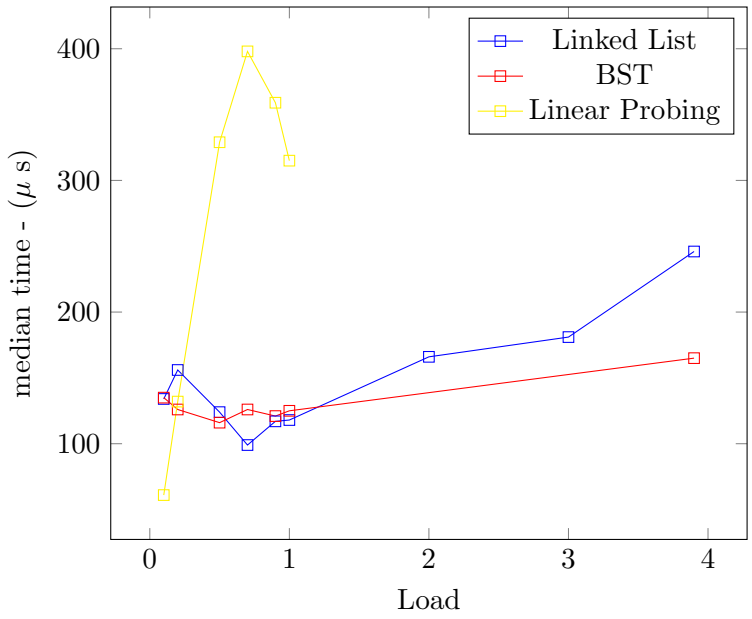
3.1 Data Set 1:



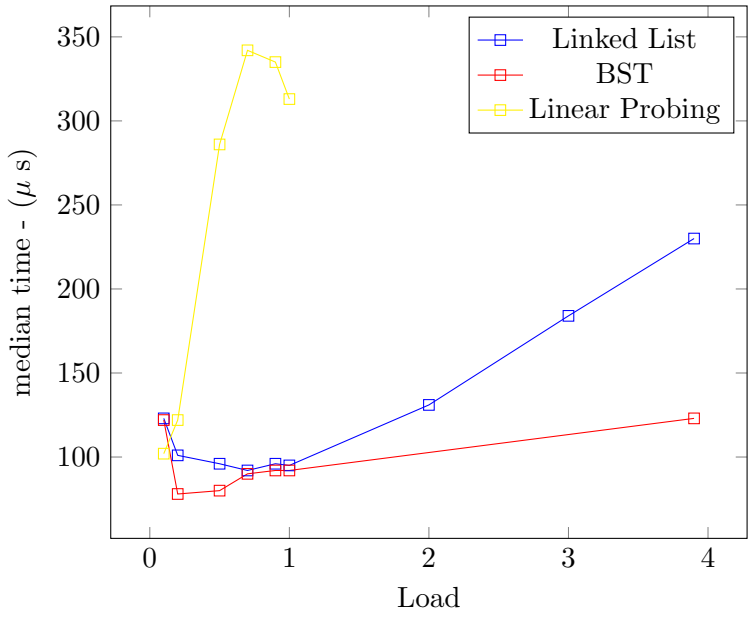

Linear Probing Performance vs. Load
median time - (μ s)
1,500
1,000
500
0
0.2
0.4
0.6
0.8
1
Load
Insert
Lookup
Delete


Insert Performances vs. Load
median time - (μ s)
1,500
1,000
500
0
0
1
2
3
4
Load
Linked List
BST
Linear Probing

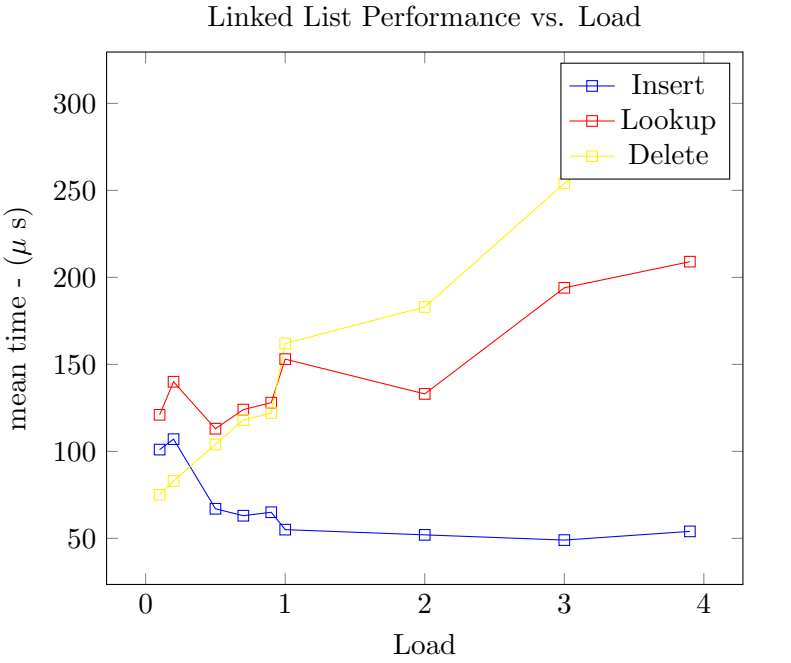
Lookup Performances vs. Load

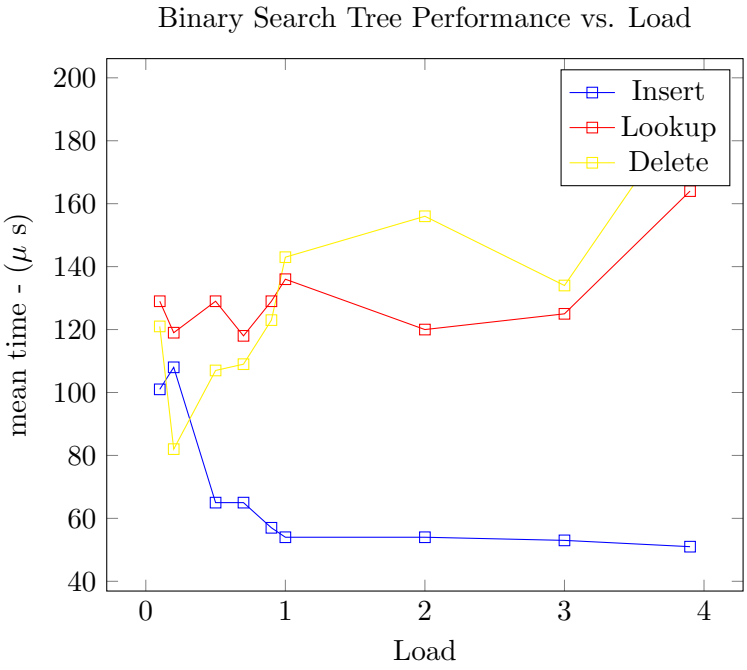


Delete Performances vs. Load

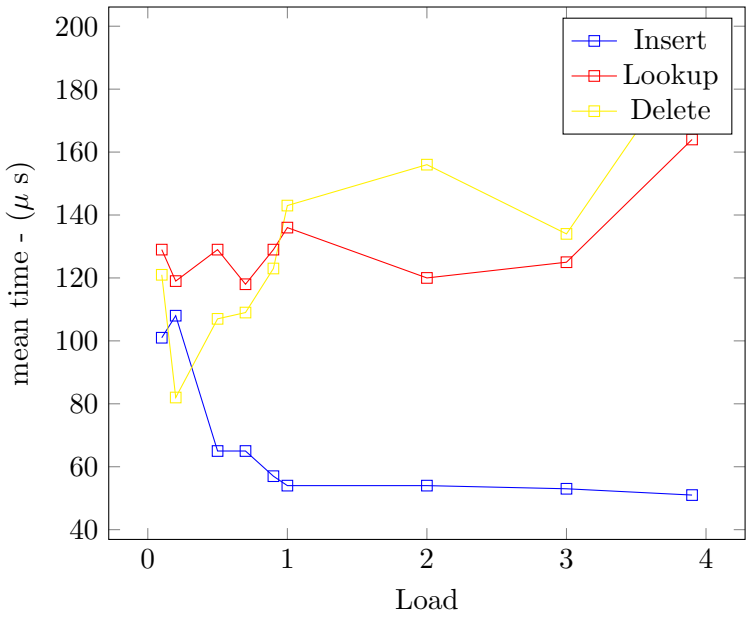


3.2 Data Set 2:

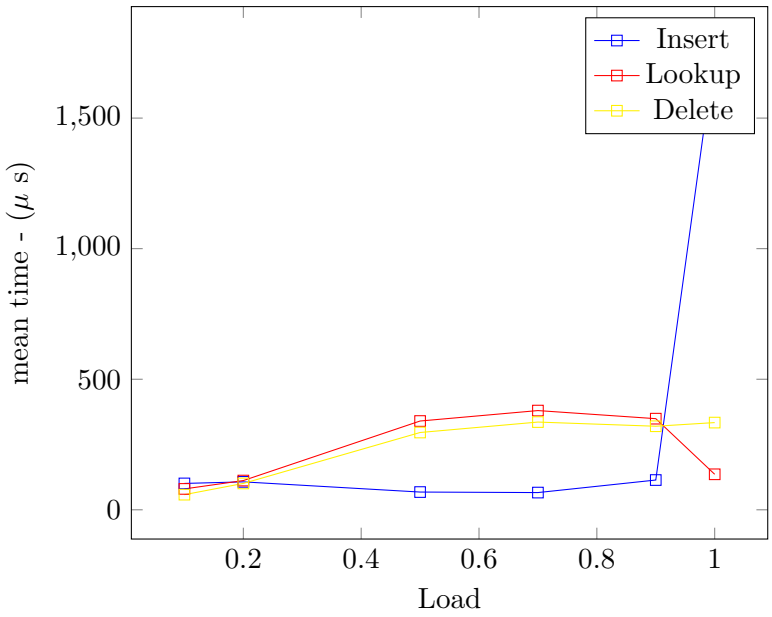

Linked List Performance vs. Load
Insert
Lookup
Delete
mean time - (μ s)
300
250
200
150
100
50
0
1
2
3
4
Load


Binary Search Tree Performance vs. Load
Insert
Lookup
Delete
mean time - (μ s)
200
180
160
140
120
100
80
60
40
0
1
2
3
4
Load

Binary Search Tree Performance vs. Load

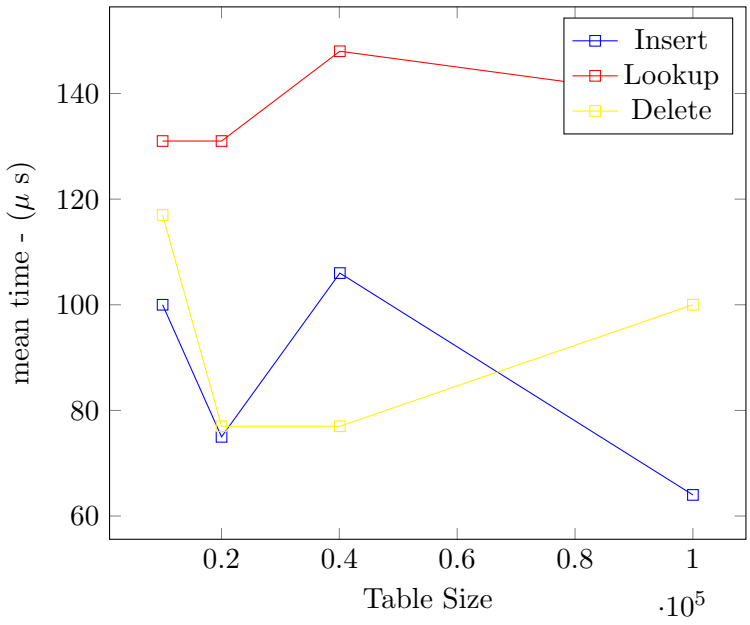


Linear Probing Performance vs. Load



3.3 Cuckoo Hashing

Cuckoo Hashing Performance vs. Table Size at .1 load



4 Results

A close analysis of the graphs shows a few things.

- Inserting a value appears faster than deleting and searching, unless trying to insert into a full array.
- Linear probing and cuckoo hashing scales the worst with load.
- Binary search tree outperforms linked list for collision resolution.
- The second hashing function appears to perform better overall.

The general performances of inserting, deleting and searching is consistent with what was expected from the code. The insert function typically can insert the value, and if not it resolves the collision until it finds a spot and continues, so part of the time there is never a second step. The delete function and search function both search in the initial array location, and have to compare that to their key before being able to attempt any collision resolution, so it makes sense that they take longer to perform.

For the cuckoo hashing, the load never was able to get above .2 before hitting a loop, so it was interesting see how performances changed with table size at a constant load. From what data was collected it appears that lookup of a cuckoo hash stays fairly consistent in this regard, where insert and delete might gain an advantage from a larger hash table.

The decrease in linear probings performance after the load was around .8 is unexpected, as well as the increase in performance time at smaller loads for all of the resolution types. The data around a load of .2 was more likely to be inconsistent with what I was expecting, usually rising before an expected dip. Though that could easily be from a small data size, as sometimes it felt like my laptops charge also had an affect. It was also unusual that deleting appeared faster than

lookups in the first data set, but it was reversed in the second. A larger data set, and better equipment, would help us see if the numbers are consistent, or an artifact of the setup.