Algorithms Assignment Three - Sorting and Hashing

Charlie Schmitz

October 28th, 2020

---

https://github.com/charlesschmitz2/CMPT435—Algorithms/tree/Assignment-Three—Searching-and-Hashing-Magic-Items

1 Results

Each time the program is run there will be 42 random items selected from the magicitems list, these selections will be different each run:

Linear Search searches through an UNSORTED magicitems list to find the randomly selected items (Could be sorted if you want but does not need to be)

Binary Search searches through a SORTED magicitems list to find the randomly selected items

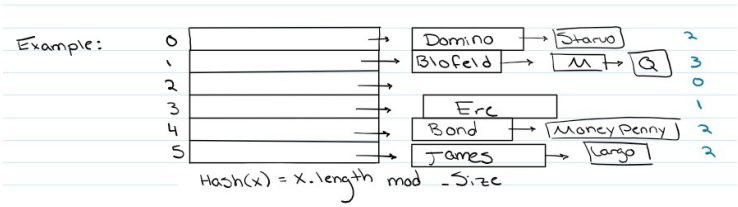
The Hashing Algorithm performs a hash on each value and stores them in 'buckets' of linked lists. The comparisons are based on the get method in the HashTable class

Below I show some sample output when running the program. While it is a lot I do this mainly for myself to be able to literally see each step of the process. This enables me to have a better understanding of why and how things occur and in what order. Each run of the program will be a bit different as 42 random items are selected each time, despite this many of the average comparison times remain within the same interval. For linear search it typically lands with around 12-13 thousand comparisons for the 42 items with an average of around 300 per item. This is a typical amount for Linear Search. Binary Search greatly improves upon this needing only around a total of 3-4 hundred comparisons for all 42 items combined, averaging out to around 7-10 per item. It can easily be seen why binary search is much more efficient than linear search.

2 Run Time/Analysis of Hashing

Hashing, with chaining in this case, takes again those same 42 random items and gets them from within a hash table containing all 666 magic items. Overall it took my hash method a total of 50-65 comparisons on average to get each of the 42 items. This is fairly good considering that is an average of just 1.3 - 2 per item. Considering that we have 666 items with a hash table of 250, collisions are not a huge issue. There is never more than 4 given our set of items, but it can be noted that as you gain more and more and more items there can be some issues that arise. In this case, the space of our hash table is not an issue but when there are say thousands or millions of items, the efficiency of getting each item can become slower and slower as they are buried deeper within the linked lists.

'Alpha', as we discussed in class represents the average length of chains in the hash table or the 'load factor.' From this is we derived that the overall asymptotic running time of this method of hashing is $O(1 + ALPHA)$.



In the picture above I took an example that we discussed in class to help illustrate the running time of hashing with chaining. To find the running time here you would take the amount of total items divided by the average (ALPHA). In this case $10/6 = 1.67$, so ALPHA would be 1.67. Plugging this into the asymptotic running time described before it can be seen that our running time here would be $O(1 + 10) -> O(11)$. In conclusion, hashing with this method can be quite efficient if space is not an issue so it is a very viable solution in some cases. There are many many different types of hashing so its best to first analyze the data and the constraints you are working within to be most efficient.

```
--------Linear Search----------

The Element 'Amulet of mighty fists +2' was Found at Key 100
Comparisons: 101

The Element 'Amulet of mighty fists +5' was Found at Key 159
Comparisons: 160

The Element 'Amulet of natural armor +1' was Found at Key 315
Comparisons: 316

The Element 'Aquasword' was Found at Key 609
Comparisons: 610

The Element 'Book of the Necromancer' was Found at Key 209
Comparisons: 210

The Element 'Brooch of shielding' was Found at Key 342
Comparisons: 343

The Element 'Candle of truth' was Found at Key 113
Comparisons: 114

The Element 'Casters Aid' was Found at Key 28
Comparisons: 29

The Element 'Circlet of Superiority' was Found at Key 447
Comparisons: 448

The Element 'Cup of Change' was Found at Key 404
Comparisons: 405

The Element 'Daggers of V' was Found at Key 261
Comparisons: 262

The Element 'Darkskull' was Found at Key 433
Comparisons: 434

The Element 'Delacour' was Found at Key 569
Comparisons: 570

The Element 'Doom Horn' was Found at Key 174
Comparisons: 175

The Element 'Dust of disappearance' was Found at Key 133
Comparisons: 134

The Element 'Elixir of truth' was Found at Key 21
Comparisons: 22

The Element 'Erikson Ring' was Found at Key 286
Comparisons: 287

The Element 'Falchion' was Found at Key 335
Comparisons: 336

The Element 'Frictionless Shield' was Found at Key 140
Comparisons: 141

The Element 'Garment of Yveth' was Found at Key 415
Comparisons: 416

The Element 'Handbook of the Magus' was Found at Key 60
Comparisons: 61

The Element 'Hat of disguise' was Found at Key 646
Comparisons: 647

The Element 'Healing Totem' was Found at Key 249
Comparisons: 250

The Element 'Helm of Debilitation' was Found at Key 224
Comparisons: 225

The Element 'Ioun stone, lavender and green ellipsoid' was Found at Key 265
Comparisons: 266

The Element 'Manual of gainful exercise +3' was Found at Key 584
Comparisons: 585

The Element 'Manual of quickness of action +1' was Found at Key 543
Comparisons: 544

The Element 'Maul of the titans' was Found at Key 531
Comparisons: 532

The Element 'Nebel Orbs' was Found at Key 200
Comparisons: 201

The Element 'Pipes of haunting' was Found at Key 324
Comparisons: 325

The Element 'Razor Leaf' was Found at Key 278
Comparisons: 279

The Element 'Robe of the archmagi' was Found at Key 87
Comparisons: 88

The Element 'Rope of Elemental Binding' was Found at Key 231
Comparisons: 232

The Element 'ropper' was Found at Key 39
Comparisons: 40

The Element 'Scimitar' was Found at Key 483
Comparisons: 484

The Element 'Shield of Tenser' was Found at Key 605
Comparisons: 606

The Element 'Short Sword' was Found at Key 201
Comparisons: 202

The Element 'Talisman of Heroic Returns' was Found at Key 155
Comparisons: 156

The Element 'The Circlet of Zahnlok' was Found at Key 232
Comparisons: 233

The Element 'Twig of a Dozen Uses or The Handy Stick' was Found at Key 555
Comparisons: 556

The Element 'UFO tofu' was Found at Key 190
Comparisons: 191
```

```
--------Binary Search----------

The Element 'Amulet of mighty fists +2' was Found at Key 13
Comparisons: 9

The Element 'Amulet of mighty fists +5' was Found at Key 16
Comparisons: 9

The Element 'Amulet of natural armor +1' was Found at Key 17
Comparisons: 8

The Element 'Aquasword' was Found at Key 28
Comparisons: 9

The Element 'Book of the Necromancer' was Found at Key 74
Comparisons: 8

The Element 'Brooch of shielding' was Found at Key 106
Comparisons: 9

The Element 'Candle of truth' was Found at Key 109
Comparisons: 9

The Element 'Casters Aid' was Found at Key 117
Comparisons: 10

The Element 'Circlet of Superiority' was Found at Key 129
Comparisons: 7

The Element 'Cup of Change' was Found at Key 163
Comparisons: 8

The Element 'Daggers of V' was Found at Key 171
Comparisons: 7

The Element 'Darkskull' was Found at Key 172
Comparisons: 9

The Element 'Delacour' was Found at Key 176
Comparisons: 6

The Element 'Doom Horn' was Found at Key 182
Comparisons: 9

The Element 'Dust of disappearance' was Found at Key 193
Comparisons: 8

The Element 'Elixir of truth' was Found at Key 207
Comparisons: 4

The Element 'Erikson Ring' was Found at Key 211
Comparisons: 10

The Element 'Falchion' was Found at Key 221
Comparisons: 10

The Element 'Frictionless Shield' was Found at Key 244
Comparisons: 9

The Element 'Garment of Yveth' was Found at Key 249
Comparisons: 3

The Element 'Handbook of the Magus' was Found at Key 285
Comparisons: 7

The Element 'Hat of disguise' was Found at Key 289
Comparisons: 9

The Element 'Healing Totem' was Found at Key 292
Comparisons: 9

The Element 'Helm of Debilitation' was Found at Key 299
Comparisons: 9

The Element 'Ioun stone, lavender and green ellipsoid' was Found at Key 327
Comparisons: 7

The Element 'Manual of gainful exercise +3' was Found at Key 367
Comparisons: 10

The Element 'Manual of quickness of action +1' was Found at Key 370
Comparisons: 10

The Element 'Maul of the titans' was Found at Key 381
Comparisons: 8

The Element 'Nebel Orbs' was Found at Key 394
Comparisons: 10

The Element 'Pipes of haunting' was Found at Key 425
Comparisons: 10

The Element 'Razor Leaf' was Found at Key 448
Comparisons: 9

The Element 'Robe of the archmagi' was Found at Key 482
Comparisons: 9

The Element 'Rope of Elemental Binding' was Found at Key 493
Comparisons: 10

The Element 'ropper' was Found at Key 495
Comparisons: 9

The Element 'Scimitar' was Found at Key 507
Comparisons: 8

The Element 'Shield of Tenser' was Found at Key 520
Comparisons: 5

The Element 'Short Sword' was Found at Key 524
Comparisons: 10

The Element 'Talisman of Heroic Returns' was Found at Key 584
Comparisons: 9

The Element 'The Circlet of Zahnlok' was Found at Key 589
Comparisons: 9

The Element 'Twig of a Dozen Uses or The Handy Stick' was Found at Key 627
Comparisons: 8
```

```
--------Hashing----------

Bucket 0 :  | **0 - Eyes of petrification | | **0 - Ranger's Sword |
Bucket 1 :  | **1 - Cursed Sword of Ratha-Zim | | **1 - Hand of Tyr | | **1 - Ring of Flickering |
Bucket 2 :
Bucket 3 :  | **3 - Dust of appearance | | **3 - Gloves of Far Reaching | | **3 - Hard Leather Armor | | **3 - Nigrals Book of Amassment |
Bucket 4 :  | **4 - Guardian Armor | | **4 - Helm of underwater action | | **4 - Pheonix Spellbook | | **4 - Shadowskill Armor | | **4 - Terro Piccapoc's Halfling Longblades |
Bucket 5 :  | **5 - Foolkiller | | **5 - Paladin | | **5 - Winter's Night |
Bucket 6 :  | **6 - Earring of Cooling | | **6 - Efreeti bottle | | **6 - Ring of Fire Absorbing |
Bucket 7 :  | **7 - Velunaedor |
Bucket 8 :  | **8 - Arenea Wand | | **8 - Circlet of persuasion | | **8 - Gloves of Dexterity +2 | | **8 - Stone Bloodhounds |
Bucket 9 :  | **9 - Book of Spelling Mistakes | | **9 - Helm of teleportation |
Bucket 10 :  | **10 - Ioun stone, orange | | **10 - Restorative ointment |
Bucket 11 :
Bucket 12 :  | **12 - Bag of tricks, gray | | **12 - Disenchanter/Unenchanter | | **12 - Field Plate | | **12 - Gloves of Dexterity +6 | | **12 - Ioun stone, lavender and green ellipsoid |
Bucket 13 :  | **13 - Gem of Hoodwinking |
Bucket 14 :  | **14 - Bowl of Purity | | **14 - Harpy Blade |
Bucket 15 :  | **15 - Cloak of the Undead | | **15 - Elondel | | **15 - Warding Stakes |
Bucket 16 :  | **16 - Elixir of sneaking |
Bucket 17 :  | **17 - Bedroll of Comfort | | **17 - Clevershot | | **17 - Horseshoes of Stealth |
Bucket 18 :
Bucket 19 :  | **19 - Aerewens armor | | **19 - Dragon Helm | | **19 - Frying Pan of Whacking | | **19 - Ice Mace | | **19 - Potion of Dreaming | | **19 - Ring of Treasure Location | | **1
Bucket 20 :  | **20 - Buckler | | **20 - Po-Tien's Stoney Blade | | **20 - Spectral Darts |
Bucket 21 :  | **21 - Casters Aid | | **21 - Potion of the Hero's Heart | | **21 - Salve of slipperiness |
Bucket 22 :  | **22 - Potion of Infestation |
Bucket 23 :  | **23 - Belt of Keeping | | **23 - Ring of Defense | | **23 - Sovereign glue | | **23 - Strand of prayer beads | | **23 - The Circlet of Zahnlok |
Bucket 24 :  | **24 - Dust of tracelessness | | **24 - Kite Shield | | **24 - Snuff of Poison Detection | | **24 - Spiked Club |
Bucket 25 :  | **25 - Flail of Armor Disruption |
Bucket 26 :  | **26 - Boots of elvenkind | | **26 - Carpet of flying, 5 ft. by 10 ft. | | **26 - Kidnapper's Bag | | **26 - Mantle of faith | | **26 - Ring of Redemption |
Bucket 27 :  | **27 - Eversol's Innebriator |
Bucket 28 :  | **28 - Horseshoes of Water-Striding | | **28 - Mace | | **28 - Rod of Substituted Spell Components |
Bucket 29 :  | **29 - Chopper |
Bucket 30 :
Bucket 31 :  | **31 - Cape | | **31 - Ioun stone, vibrant purple prism |
Bucket 32 :  | **32 - Mats of Travelling | | **32 - Potion of Icewater |
Bucket 33 :  | **33 - Dagger of the wilds |
Bucket 34 :  | **34 - Censer of controlling air elementals | | **34 - Cloak of Charisma +2 | | **34 - Fleen's Reagent | | **34 - Flytrap Liquid | | **34 - Mimpot's Anti-magic Girdle | | **
Bucket 35 :  | **35 - Hand of the mage |
Bucket 36 :  | **36 - Arrows of Eroch the Archer | | **36 - Crystal ball with true seeing | | **36 - Full Plate Mail | | **36 - Sword of Mastery +2 | | **36 - Thumb Ring of Grasping |
Bucket 37 :  | **37 - Bag of holding type III |
Bucket 38 :  | **38 - Cloak of Charisma +6 |
Bucket 39 :  | **39 - Hat of disguise | | **39 - Snapping Purse |
Bucket 40 :  | **40 - Aibohphobia |
Bucket 41 :  | **41 - Bow of Doubling | | **41 - Bracers of archery, lesser | | **41 - Broad Sword | | **41 - Horn of Holding | | **41 - Sack of Plunder | | **41 - Spider Ring | | **41 - T
Bucket 42 :  | **42 - Box of Delights | | **42 - Circlet of blasting, minor | | **42 - Zales Might |
Bucket 43 :  | **43 - Candle of truth | | **43 - Elixir of swimming | | **43 - Feather token, bird | | **43 - Star of Landryn Teriak | | **43 - Taco cat |
Bucket 44 :  | **44 - Club | | **44 - Helm | | **44 - Long Battle Bow | | **44 - Necklace of fireballs type III | | **44 - Weapon ring |
Bucket 45 :  | **45 - Book of Stealth |
Bucket 46 :  | **46 - Manacadam Rings | | **46 - Robe |
Bucket 47 :  | **47 - Cloak of the bat | | **47 - Folding Catapult (or Ballista) | | **47 - Splint Mail |
Bucket 48 :
Bucket 49 :  | **49 - Bow of Called Shots | | **49 - Circlet of Demonic Protection |
Bucket 50 :
Bucket 51 :  | **51 - Coronet of Gith | | **51 - Horseshoes of Cleanliness | | **51 - Tear of Life | | **51 - Throwing Stone | | **51 - Torque of Aerie | | **51 - Wind fan |
Bucket 52 :  | **52 - Animal lord's helms | | **52 - Arrows of Outrageous Fortune | | **52 - Fire Stones | | **52 - Great Sword |
Bucket 53 :  | **53 - Blindfold of Wakeful Sleep | | **53 - Hammer of Wonderous Works | | **53 - Maul |
Bucket 54 :  | **54 - Mug of Infinite Thirst | | **54 - Stone salve |
Bucket 55 :  | **55 - Gloves of the Pugelist | | **55 - Hectorius's Twin Maces of The Heavens |
Bucket 56 :  | **56 - Backhand | | **56 - Stone Of Acid Arrow |
Bucket 57 :  | **57 - Pearl of power, two spells |
Bucket 58 :  | **58 - Daggers of V | | **58 - Feather token, tree | | **58 - Globe of War |
Bucket 59 :
Bucket 60 :  | **60 - Poultice of Ori | | **60 - Tongue Twister |
Bucket 61 :  | **61 - Gutbuster's Smelly Mithral Armor | | **61 - Seeds of Plenty |
Bucket 62 :  | **62 - Robot Tobor | | **62 - Wings of flying |
Bucket 63 :  | **63 - Gloves of Wizardry | | **63 - Horseshoes of a zephyr | | **63 - Razor of Bluntness | | **63 - Sorcerer's robe | | **63 - Wartexx |
Bucket 64 :  | **64 - Mortis Ring |
Bucket 65 :  | **65 - Headband of intellect +2 | | **65 - Item of Instant Armour Appearance | | **65 - Walking Stick +3 |
Bucket 66 :  | **66 - Feather token, whip | | **66 - White sword |
Bucket 67 :  | **67 - Skin of Fresh Water |
Bucket 68 :  | **68 - Blade of Shattering | | **68 - Handy haversack | | **68 - Long Bow |
Bucket 69 :  | **69 - Robe of eyes |
Bucket 70 :  | **70 - Carpet of flying, 10 ft. by 10 ft. | | **70 - The Thain Soul ring |
Bucket 71 :  | **71 - Brooch of shielding |
Bucket 72 :
Bucket 73 :  | **73 - Sword of Melnor |
Bucket 74 :  | **74 - Boots of the Wraith | | **74 - Hide of Transformation | | **74 - Tome of leadership and influence +2 |
Bucket 75 :  | **75 - Shrouds of disintegration | | **75 - Tome of leadership and influence +3 |
Bucket 76 :  | **76 - Eyes of doom | | **76 - Tome of leadership and influence +4 |
Bucket 77 :  | **77 - Brazier of commanding fire elementals | | **77 - Dragoliness | | **77 - Sword of the Kauhns | | **77 - Tome of leadership and influence +5 |
Bucket 78 :  | **78 - Pipes of the sewers | | **78 - War Drum |
Bucket 79 :
Bucket 80 :  | **80 - Exploding Caltrops | | **80 - Falchion | | **80 - Link Tabbard |
Bucket 81 :  | **81 - Book of the Past | | **81 - Elixir of truth | | **81 - Mantle of spell resistance | | **81 - Short Sword |
Bucket 82 :  | **82 - Book of the Necromancer | | **82 - Book of Untruth | | **82 - Eversmoking bottle |
Bucket 83 :  | **83 - Dacad | | **83 - Sleepytime Bear |
Bucket 84 :  | **84 - Long War Bow | | **84 - Tome of clear thought +1 | | **84 - UFO tofu |
Bucket 85 :  | **85 - Broach of insect repulsion | | **85 - Dust of dryness | | **85 - Saddle of Stability | | **85 - The healing ring of Sheldon | | **85 - Tome of clear thought +2 |
Bucket 86 :  | **86 - Amulet of Deception | | **86 - Large Shield | | **86 - Potion of Liquefaction | | **86 - Tome of clear thought +3 |
Bucket 87 :  | **87 - Dragon Cloak | | **87 - Hwoctul's Masks | | **87 - Soap of Cleanliness | | **87 - Tome of clear thought +4 | | **87 - Well of many worlds |
Bucket 88 :  | **88 - Periapt of Wisdom +2 | | **88 - Quicksilver Amulet | | **88 - Robe of blending | | **88 - Robe of the archmagi | | **88 - Tome of clear thought +5 |
Bucket 89 :  | **89 - Angel's Fury | | **89 - Hectorius's Twin Rings |
Bucket 90 :  | **90 - Silversheen | | **90 - Sword of Enlightenment |
Bucket 91 :  | **91 - Blessed (Cursed) Oil | | **91 - Delacour | | **91 - Figurine of wondrous power, silver raven | | **91 - Pearl of power, 2nd-level spell |
Bucket 92 :  | **92 - Periapt of Wisdom +6 |
Bucket 93 :
Bucket 94 :  | **94 - Blade | | **94 - Crystal ball with detect thoughts |
Bucket 95 :  | **95 - Composite Staff |
Bucket 96 :  | **96 - Amber Spider |
```

```
----Analysis/Get Comparisons----

0. Amulet of mighty fists +2 - Comparisons to Get : 0 || Bucket Value : 185

1. Amulet of mighty fists +5 - Comparisons to Get : 0 || Bucket Value : 188

2. Amulet of natural armor +1 - Comparisons to Get : 0 || Bucket Value : 245

3. Aquasword - Comparisons to Get : 0 || Bucket Value : 195

4. Book of the Necromancer - Comparisons to Get : 0 || Bucket Value : 82

5. Brooch of shielding - Comparisons to Get : 0 || Bucket Value : 71

6. Candle of truth - Comparisons to Get : 0 || Bucket Value : 43

7. Casters Aid - Comparisons to Get : 0 || Bucket Value : 21

8. Circlet of Superiority - Comparisons to Get : 0 || Bucket Value : 110

9. Cup of Change - Comparisons to Get : 2 || Bucket Value : 117

10. Daggers of V - Comparisons to Get : 0 || Bucket Value : 58

11. Darkskull - Comparisons to Get : 1 || Bucket Value : 185

12. Delacour - Comparisons to Get : 1 || Bucket Value : 91

13. Doom Horn - Comparisons to Get : 0 || Bucket Value : 146

14. Dust of disappearance - Comparisons to Get : 1 || Bucket Value : 227

15. Elixir of truth - Comparisons to Get : 1 || Bucket Value : 81

16. Erikson Ring - Comparisons to Get : 0 || Bucket Value : 125

17. Falchion - Comparisons to Get : 1 || Bucket Value : 80

18. Frictionless Shield - Comparisons to Get : 0 || Bucket Value : 140

19. Garment of Yveth - Comparisons to Get : 0 || Bucket Value : 139

20. Handbook of the Magus - Comparisons to Get : 1 || Bucket Value : 183

21. Hat of disguise - Comparisons to Get : 0 || Bucket Value : 39

22. Healing Totem - Comparisons to Get : 2 || Bucket Value : 179

23. Helm of Debilitation - Comparisons to Get : 1 || Bucket Value : 145

24. Ioun stone, lavender and green ellipsoid - Comparisons to Get : 4 || Bucket Value : 12

25. Manual of gainful exercise +3 - Comparisons to Get : 2 || Bucket Value : 185

26. Manual of quickness of action +1 - Comparisons to Get : 2 || Bucket Value : 136

27. Maul of the titans - Comparisons to Get : 2 || Bucket Value : 240

28. Nebel Orbs - Comparisons to Get : 1 || Bucket Value : 200

29. Pipes of haunting - Comparisons to Get : 5 || Bucket Value : 204

30. Razor Leaf - Comparisons to Get : 0 || Bucket Value : 210

31. Robe of the archmagi - Comparisons to Get : 3 || Bucket Value : 88

32. Rope of Elemental Binding - Comparisons to Get : 2 || Bucket Value : 225

33. ropper - Comparisons to Get : 2 || Bucket Value : 222

34. Scimitar - Comparisons to Get : 3 || Bucket Value : 104

35. Shield of Tenser - Comparisons to Get : 1 || Bucket Value : 119

36. Short Sword - Comparisons to Get : 3 || Bucket Value : 81

37. Talisman of Heroic Returns - Comparisons to Get : 5 || Bucket Value : 101

38. The Circlet of Zahnlok - Comparisons to Get : 4 || Bucket Value : 23

39. Twig of a Dozen Uses or The Handy Stick - Comparisons to Get : 2 || Bucket Value : 129

40. UFO tofu - Comparisons to Get : 2 || Bucket Value : 84

41. Universal solvent - Comparisons to Get : 6 || Bucket Value : 34

Total Comparisons : 60
```

3 Main Class

```
import java.io.*;
import java.lang.reflect.Array;
import java.security.SecureRandom;
import java.text.DecimalFormat;
import java.util.*;
import java.util.HashMap;

public class AlgorithmsAssignmentThree {

    private static int totalComparisonsBinarySearch;
    private static int tempComparisonsBinarySearch;
    private static final int LINES_IN_FILE = 666;
    private static final int HASH_TABLE_SIZE = 250;


    public static void main(String[] args){

        System.out.println("\nEach time the program is run there will be 42 random items selected from the magicitems list, these selections will be different each run:");
        System.out.println("Linear Search searches through an UNSORTED magicitems list to find the randomly selected items ( Could be sorted if you want but does not need to be)");
        System.out.println("Binary Search searches through a SORTED magicitems list to find the randomly selected items");
        System.out.println("The Hashing Algorithm performs a hash on each value and stores them in 'buckets' of linked lists. The comparisons are based on the get method in the HashTable class ");

        //Declare Variables, in this case the strings that will be used throughout main
        String[] magicitems = readArray("magicitems.txt");
        String[] randomMagicItems = new String[42];

        /*
            System.out.println("All Elements in Array: ");
            for(int i = 0; i < magicitems.length; i++){
                System.out.println(magicitems[i]);
            }
        */



        //Randomly select 42 items within the magic items list, hold onto these values in a new array
        //Unique Numbers will be an array of value 0-666 all unique and in a random order, from this the first 42 can be used as an index value for the magic items to have 42 random and unique value
        //Each time the program is rerun new random values are generated but during the duration of the program the random
```

```
    elements remain the same
            int[] uniqueNumbers = createUniqueRandomNumber(0,665);

            for(int i = 0; i < randomMagicItems.length; i++){
                int temp = uniqueNumbers[i];
                String temp2 = magicitems[temp];

                randomMagicItems[i]= temp2;
            }
        /* Prints out all of the randomly selected items
            System.out.println("\n--------Randomly Selected 42
    Elements in Array-------- ");
            for (int i = 0; i < randomMagicItems.length; i++){
                System.out.print(randomMagicItems[i] + ", ");
            }//for
        */
        //Taken from Assignment two, adapted from arrayList to work
     with arrays. Using quicksort. Magicitems remains unsorted
            int n = randomMagicItems.length;
            quickSort(randomMagicItems, 0, n-1);
        /* Prints out all the random items in sorted order
            System.out.println("\n--------Sorted Array-----------")
    ;
            for (int i = 0; i < randomMagicItems.length; i++){
                System.out.print(randomMagicItems[i] + ", ");
            }//for
        */
        //Linear Search - Here I have a sorted randomMagicItems
    array and an unsorted magicitems array. I am searching for each
     of the randomly selected items
        //within the larger magicitems array which again is
    unsorted.

            // Using Scanner for Getting Input from User so have
    the option to search for whatever individual items if desired
            System.out.println("\n\n--------Linear Search
    -----------");
            //System.out.println("Enter the Word to be Searched for
    : ");
            //Scanner in00 = new Scanner(System.in);
            //String searchLinear = in00.nextLine();

            String searchLinear = "";
            int[] result = new int[3];
            int totalComparisons = 0;
            for(int i = 0; i < randomMagicItems.length; i++){
                searchLinear = randomMagicItems[i];
                result = linearSearch(magicitems, searchLinear);
                int temp = result[2];
                totalComparisons = totalComparisons + temp;

                if(result[0] == 1) {
                    System.out.println("\n\tThe Element '" +
    searchLinear + "' was Found at Key " + result[1]);
                    System.out.println("\tComparisons: " + temp);
                }//if
                else{
```

```
86                  System.out.println("\n\tThe Element '"+
    searchLinear + "' was NOT Found in the Array");
87              }//else
88          }//for
89
90          System.out.println("\nTotal Number of Comparisons: " +
    totalComparisons);
91          System.out.println("\nAverage Number of Comparisons (
    Total/42): " + totalComparisons/42);
92
93
94
95
96      //Binary Search - Here I have the same sorted
    randomMagicItems, but now I have to sort the full magicitems
    array in order to perform a binary search within it.
97
98          // Using Scanner for Getting Input from User so have
    the option to search for whatever individual items if desired
99          System.out.println("\n\n--------Binary Search
    -----------");
100         //System.out.println("Enter the Word to be Searched for
    : ");
101         //Scanner in00 = new Scanner(System.in);
102         //String searchBinary = in00.nextLine();
103
104     //sorting the magic items array to be used in binary search
105             int m = magicitems.length;
106             quickSort(magicitems, 0, m-1);
107
108
109         String searchBinary = "";
110         int start = 0;
111         int stop = magicitems.length-1;
112
113
114         for(int i = 0; i < randomMagicItems.length; i++){
115             searchBinary = randomMagicItems[i];
116             int result2 = 0;
117             tempComparisonsBinarySearch = 0;
118             result2 = binarySearch(magicitems, 0, magicitems.
    length-1, searchBinary);
119
120                 if(result2 == -1) {
121                 System.out.println("\nThe Element '" +
    searchBinary + "' was NOT Found in the Array");
122             }//if
123             else{
124                 result2 = result2+1;
125                     //adding a + 1 to the index key simply
    because we know that the list is 0-665 but in a text document
    or to an avg
126                     //person the list of magicitems goes from
    1-666.
127                 System.out.println("\n\tThe Element '"+
    searchBinary + "' was Found at Key " + result2);
128                 System.out.println("\tComparisons: " +
```

```
    tempComparisonsBinarySearch);

                }//else
            }//for

            System.out.println("\nTotal Number of Comparisons: " +
totalComparisonsBinarySearch);
            System.out.println("\nAverage Number of Comparisons (
Total/42): " + totalComparisonsBinarySearch/42);


        System.out.println("\n\n--------Hashing-----------");
        int[] hashValues = new int[magicitems.length];
        // Print the array and hash values.
            int hashCode = 0;
            for (int i = 0; i < magicitems.length; i++) {
                //System.out.print(i);
                //System.out.print(". " + magicitems[i] + " - ");
                hashCode = makeHashCode(magicitems[i]);
                //System.out.format("%03d%n", hashCode);
                hashValues[i] = hashCode;
            }

        // Analyze the distribution of hash values.
        //analyzeHashValues(hashValues);

        //Initialize the Hash Table
            HashTable hashTable = new HashTable(HASH_TABLE_SIZE);

        //Functions of the hash table:
            //hashtable.put(key, value)
            //hashtable.remove(key)
            //hashtable.get(key)
            //hashtable.getSize()
            //hashtable.empty()
            //hashtable.printHashTable();
            //hashtable.makeHashCode(); --> this function is also
in the main class if wanted to be used that way but for my
needs I found it to be better in
            //the hast table class. I left it in main because the
example given was oriented that way but could be editied a bit
better to avoid repetition and
            //be better organized and implemented
        //Here I set the key as the string form of the word,
example: "Zales Might".
        //The value represents the value of the hashcode of that
String so for "Zales Might", its value would be 42.

        //Adding all of the Magic Items Values into the hash table
            System.out.print(" ");
            int hashValuesPut = 0;

            for(int i = 0; i < magicitems.length; i++){
                hashValuesPut = hashTable.makeHashCode(magicitems[i
]);
                hashTable.put(magicitems[i],hashValuesPut);
            }
```

```
176
177         //Printing out the hash table - Could also use the given
    Analyze Hash Table Function but I took that function and
    simplified it down a bit
178         //to have the less of the analyzing and more just printing
    out the value and key for each bucket.
179         //Each entry is divided by a "|" so it gives a bit more
    information than just stating how many counts there are at each
     bucket since that is
180         //already done by you for us in the analyze function so you
     know how many are at each value but it more so shows you what
    each value within each linked list
181         //bucket.
182             hashTable.printHashTable();
183
184             //For comparisons here I basically test how deep within
     the linked list at the bucket value. TO get the value the time
     can take anywhere from
185             //0 meaning it is the first element of that Entry/
    linked List or in this case a max of 3 or 4.
186             System.out.println("\n\n----Analysis/Get Comparisons
    ---- ");
187             String print = " ";
188             String tempValue = " ";
189             for(int i = 0; i < randomMagicItems.length; i++){
190                 System.out.print("\n");
191                 print = randomMagicItems[i] + " - ";
192                 tempValue = randomMagicItems[i];
193                 System.out.print(i + ". ");
194                 System.out.print(print);
195                 System.out.println(" || Bucket Value : " +
    hashTable.get(tempValue));
196             }//for
197
198         System.out.println("\nTotal Comparisons : " + hashTable.
    getTotalComparisons());
199             double avgHashTable = hashTable.getTotalComparisons()
    /42.0;
200
201         System.out.println("\nAverage Number of Comparisons (Total
    /42): " + new DecimalFormat("0.00").format(avgHashTable));
202
203
204
205
206
207
208
209
210
211
212
213     }//main
214
215     //A function that will take in a string parameter that is the
    name of the file and copy the contents into an array
216     //Assumes one element per line
```

```
217     public static String[] readArray(String file){
218         int count = 0;
219 
220         try{
221             Scanner s1 = new Scanner(new File(file));
222 
223             //Count how many elements are in the file
224             while(s1.hasNextLine()){
225                 count++;
226                 s1.nextLine();
227             }//while
228 
229             //Create the array and copy elements into it
230             String[] words = new String[count];
231 
232             Scanner s2 = new Scanner(new File(file));
233             for(int i = 0; i < count; i++){
234                 words[i]=s2.nextLine();
235             }
236 
237             return words;
238         }//try
239         catch (FileNotFoundException e){
240 
241         }
242 
243         return null;
244     }//readString
245 
246     //A function that will create a list of random unique numbers
    between the desired range
247     public static int[] createUniqueRandomNumber(int from, int to){
248         //Number of integers need to generate
249             int n = to - from + 1;
250 
251         //Create an array to store all the numbers from, from - to
252             int array[] = new int[n];
253             for (int i = 0; i < n; i++){
254                 array[i] = i;
255             }//for
256 
257         //Create an array to store the result
258             int result[] = new int[n];
259 
260             int x = n;
261             SecureRandom rd = new SecureRandom();
262             for(int i = 0; i < n; i++){
263                 //k is a random index in [0, x]
264                 int k = rd.nextInt(x);
265 
266                 result[i] = array[k];
267 
268                 //Replace value from a[k] by the value from the
    last index
269                 //so that there will not get the value array[k]
    again
270                 array[k] = array[x-1];
```

```
271
272                 //Decrease x by 1 to get a random index from 0 to x
    only
273                 x--;
274             }
275
276             return result;
277     }//UniqueRandomNumber
278
279     //Linear Search Function
280     public static int[] linearSearch(String[] arr, String search){
281         int[] result = new int[3];
282         //the 0 index of the result array will either be
    initialized to a 0 representing a false value, if the search
    string is found then it is set
283         //to a 1 representing a true value.
284         // The 1 index of the result array represents the key, this
     is the index where the element is found within the random
    array
285         //The 2 index of the result array represents the number of
    comparisons
286         int key = -1;
287         int comparisons = 0;
288         result[0] = 0;
289         result[1] = key;
290         result[2] = comparisons;
291
292         int count = 0;
293         boolean end = true;
294         while((count < arr.length) && (end == true)){
295             if(arr[count].compareToIgnoreCase(search) == 0){
296                 result[0] = 1;
297                 key = count;
298                 comparisons++;
299                 count++;
300                 end = false;
301             }//if
302             comparisons++;
303             count++;
304         }//for
305         result[1] = key+1;
306         result[2] = comparisons;
307         return result;
308     }//linearSearch
309
310     //Binary Search Function
311     public static int binarySearch(String[] arr, int start, int
    stop, String search){
312
313         if(start > stop){
314             tempComparisonsBinarySearch++;
315             totalComparisonsBinarySearch++;
316             return -1;
317         }//if
318
319         //could round or take floor or ceiling of midpoint, I did't
     explicitly do that here but it seems to calculate it correctly
```

```
320         //so im just going to leave it as is
321         int midPoint = start + (stop-start) / 2;
322 
323         if(search.equalsIgnoreCase(arr[midPoint])){
324             tempComparisonsBinarySearch++;
325             totalComparisonsBinarySearch++;
326             return midPoint;
327         }//if
328         else if(search.compareToIgnoreCase(arr[midPoint]) < 0){
329             tempComparisonsBinarySearch++;
330             totalComparisonsBinarySearch++;
331             return binarySearch(arr, start, midPoint - 1, search);
332         }
333         else{
334             tempComparisonsBinarySearch++;
335             totalComparisonsBinarySearch++;
336             return binarySearch(arr, midPoint + 1, stop, search);
337         }
338 
339     }//BinarySearch
340 
341 
342 
343     private static int makeHashCode(String str) {
344         str = str.toUpperCase();
345         int length = str.length();
346         int letterTotal = 0;
347 
348         // Iterate over all letters in the string, totalling their
    ASCII values.
349         for (int i = 0; i < length; i++) {
350             char thisLetter = str.charAt(i);
351             int thisValue = (int)thisLetter;
352             letterTotal = letterTotal + thisValue;
353 
354             // Test: print the char and the hash.
355 /*
356            System.out.print(" [");
357            System.out.print(thisLetter);
358            System.out.print(thisValue);
359            System.out.print("] ");
360            //
361 */
362 
363             }
364 
365         // Scale letterTotal to fit in HASH_TABLE_SIZE.
366         int hashCode = (letterTotal * 1) % HASH_TABLE_SIZE;  // %
    is the "mod" operator
367         // TODO: Experiment with letterTotal * 2, 3, 5, 50, etc.
368 
369         return hashCode;
370         }//makeHashCode
371 
372     private static void analyzeHashValues(int[] hashValues) {
373         System.out.println("\nHash Table Usage:");
374 
```

```
375         // Sort the hash values.
376         Arrays.sort(hashValues);    // This is a "dual-pivot"
    quicksort
377                                     // See https://zgrepcode.com/
    java/oracle/jdk-8u181/java/util/dualpivotquicksort.java
378                                     // Actually, look at that JDK
    source code; it's a bunch of sorts.
379
380         // Test: print the sorted hash values.
381         /*
382 56         for (int i=0; i < LINES_IN_FILE; i++) {
383 57            System.out.println(hashValues[i]);
384 58         }
385 59      // */
386
387         // Create a histogram-like report based on the count of
    each unique hash value,
388         // count the individual entry size,
389         // the total space used (in items),
390         // and the standard deviation of their distribution over
    the hash table.
391         int asteriskCount = 0;
392         int[] bucketCount = new int[HASH_TABLE_SIZE];
393         int totalCount = 0;
394         int arrayIndex = 0;
395
396         for (int i=0; i < HASH_TABLE_SIZE; i++) {
397             System.out.format("%03d ", i);
398             asteriskCount = 0;
399             while ( (arrayIndex < LINES_IN_FILE) && (hashValues[
    arrayIndex] == i) ) {
400                 System.out.print("*");
401                 asteriskCount = asteriskCount + 1;
402                 arrayIndex = arrayIndex + 1;
403                 }
404             System.out.print(" ");
405             System.out.println(asteriskCount);
406             bucketCount[i] = asteriskCount;
407             totalCount = totalCount + asteriskCount;
408         }
409
410         System.out.print("Average load (count): ");
411         float averageLoad = (float) totalCount / HASH_TABLE_SIZE;
412         System.out.format("%.2f%n", averageLoad);
413
414         System.out.print("Average load (calc) : ");
415         averageLoad = (float) LINES_IN_FILE / HASH_TABLE_SIZE;
416         System.out.format("%.2f%n", averageLoad);
417
418         System.out.print("Standard Deviation: ");
419         // TODO: Refactor this into its own method.
420         double sum = 0;
421         for (int i=0; i < HASH_TABLE_SIZE; i++) {
422             // For each value in the array...
423             // ... subtract the mean from each one ...
424             double result = bucketCount[i] - averageLoad;
425             // ... and square the result.
```

```
426             double square = result * result;
427             // Sum all of those squares.
428             sum = sum + square;
429         }
430         // Divide the sum by the number of values ...
431         double temp = sum / HASH_TABLE_SIZE;
432         // ... and take the square root of that.
433         double stdDev = Math.sqrt(temp);
434         System.out.format("%.2f%n", stdDev);
435     }
436
437
438
439
440     //Quicksort and partition functions taken from assignment two
        using array's instead of array list
441     public static void quickSort(String arr[], int begin, int end)
        {
442         if (begin < end) {
443             int partitionIndex = partition(arr, begin, end);
444
445             quickSort(arr, begin, partitionIndex-1);
446             quickSort(arr, partitionIndex+1, end);
447         }
448     }//quicksort
449     private static int partition(String arr[], int begin, int end)
        {
450         String pivot = arr[end];
451         int i = (begin-1);
452
453         for (int j = begin; j < end; j++) {
454             if (arr[j].compareToIgnoreCase(pivot) < 0) {
455                 i++;
456
457                 String swapTemp = arr[i];
458                 arr[i] = arr[j];
459                 arr[j] = swapTemp;
460             }
461         }
462
463         String swapTemp = arr[i+1];
464         arr[i+1] = arr[end];
465         arr[end] = swapTemp;
466
467         return i+1;
468     }//partition
469
470
471
472 }//AssignmentThree
```

4 HashTable Class

```
public class HashTable {
    private int HASH_TABLE_SIZE;
    private int size;
    private Entry[] hashTable;
    private int totalComparisons;
    private int tempComparisons;

    /* Constructor */
    public HashTable(int tableSize)
    {
        size = 0;
        HASH_TABLE_SIZE = tableSize;
        hashTable = new Entry[HASH_TABLE_SIZE];
        tempComparisons = 0;
        totalComparisons = 0;
        for (int i = 0; i < HASH_TABLE_SIZE; i++)
            hashTable[i] = null;
    }//Constructor

    //Gets the size of the HashTable
    public int getSize()
    {
        return size;
    }//getSize

    //Clears/Empties the hash table by setting all values in it to
    null, could also be used to initialize all values to null
    public void empty()
    {
        for (int i = 0; i < HASH_TABLE_SIZE; i++)
            hashTable[i] = null;
    }//empty

    //Gets the Value of a Key
    public int get(String key)
    {
        tempComparisons = 0;
        int hash = (makeHashCode( key ) % HASH_TABLE_SIZE);
        if (hashTable[hash] == null) {
            tempComparisons++;
            totalComparisons += tempComparisons;
            System.out.print("Comparisons to Get : " +
    tempComparisons);
            return -1;
        }//if
        else
        {
            Entry entry = hashTable[hash];
            while (entry != null && !entry.key.equals(key)) {
                tempComparisons++;
                entry = entry.next;
            }//while

```

```
52              totalComparisons += tempComparisons;
53
54              if (entry == null) {
55                  System.out.print("Comparisons to Get : " +
    tempComparisons);
56                  return -1;
57              }//if
58              else {
59                  System.out.print("Comparisons to Get : " +
    tempComparisons);
60                  return entry.value;
61              }//else
62          }//else
63      }//get
64
65      //Inserts a new entry into the hashTable
66      public void put(String key, int value)
67      {
68          int hash = (makeHashCode( key ) % HASH_TABLE_SIZE);
69
70          if (hashTable[hash] == null) {
71              hashTable[hash] = new Entry(key, value);
72          }//if
73          else
74          {
75              Entry entry = hashTable[hash];
76              while (entry.next != null && !entry.key.equals(key)) {
77                  entry = entry.next;
78              }//while
79              if (entry.key.equals(key)) {
80                  entry.value = value;
81              }//if
82              else {
83                  entry.next = new Entry(key, value);
84              }//else
85          }//else
86
87          size++;
88      }//put
89
90      //Removes an entry from the hashTable
91      public void remove(String key)
92      {
93          int hash = (makeHashCode( key ) % HASH_TABLE_SIZE);
94          if (hashTable[hash] != null)
95          {
96              Entry prevEntry = null;
97              Entry entry = hashTable[hash];
98
99              while (entry.next != null && !entry.key.equals(key))
100             {
101                 prevEntry = entry;
102                 entry = entry.next;
103             }//while
104             if (entry.key.equals(key))
105             {
106                 if (prevEntry == null) {
```

```
                    hashTable[hash] = entry.next;
                }//if
                else {
                    prevEntry.next = entry.next;
                }//else

                size--;
            }//if
        }//if
    }//remove

    public int getTempComparisons(){
        return tempComparisons;
    }
    public int getTotalComparisons(){
        return totalComparisons;
    }


    //Prints the entire hashTable out listing the values in each bucket
    public void printHashTable()
    {
        for (int i = 0; i < HASH_TABLE_SIZE; i++)
        {
            System.out.print("\nBucket "+ (i) +" : ");

            Entry entry = hashTable[i];

            while (entry != null)
            {
                System.out.print(" | **" + entry.value + " - " + entry.key + " |");
                entry = entry.next;
            }//while
        }//for
    }//printHashTable


    public int makeHashCode(String str) {
        str = str.toUpperCase();
        int length = str.length();
        int letterTotal = 0;
        //System.out.println(" ");
        //System.out.print(str + " - ");

        // Iterate over all letters in the string, totalling their ASCII values.
        for (int i = 0; i < length; i++) {
            char thisLetter = str.charAt(i);
            int thisValue = (int)thisLetter;
            letterTotal = letterTotal + thisValue;
/*
            // Test: print the char and the hash.

            System.out.print(" [");
            System.out.print(thisLetter);
```

```
            System.out.print(thisValue);
            System.out.print("] ");
            //
*/

        }//for

        // Scale letterTotal to fit in HASH_TABLE_SIZE.
        int hashCode = (letterTotal * 1) % 250;  // % is the "mod" operator
        // TODO: Experiment with letterTotal * 2, 3, 5, 50, etc.

        return hashCode;
    }//makeHashCode
}
```

5 Entry Class

Based off of Linked List Code from previous assignment but shrunken down to really just contain what we need in regard to hash entries being chained together

```
public class Entry {

    String key;
    int value;
    Entry next;

    /* Constructor */
    Entry(String key, int value)
    {
        this.key = key;
        this.value = value;
        this.next = null;
    }

}
```