# Homework 7

## 1 Part I: Searching

I have provided a driver program that you can use to test your solutions to #1-2 and 5-7. I have also provided a modified `Alist` class.

- You must use this method `Alist` class - not previous versions from other assignments.
- I have commented the places in `Alist` where your code should go. Remove the placeholder `return` statements and write your code in those places.
- As always, the use of the driver program is optional and you might want to include additional checks to make sure your methods work properly

1. Revise the recursive method `search`, as given in Segment 16.6, so that it looks at the last entry in the array instead of the first one.

    - You can access `list` directly in your method

    Refer to AList.java

2. When searching a sorted array sequentially, you can ascertain that a given item does not appear in the array without searching the entire array. For example, if you search the array

    [2 5 7 9]

    for 6, you can use the approach described in Segment 16.8. That is, you compare 6 to 2, then to 5, and finally to 7. Since you did not find 6 after comparing it to 7, you do not have to look further, because the other elements in the array are greater than 7 and therefore cannot equal 6. Thus, you do not simply ask whether 6 equals an array element, you also ask whether it is greater than the element. Since 6 is greater than 2, you continue the search. Likewise 5. Since 6 is less than 7, you have passed the point in the array where 6 would have had to occur, so 6 is not in the array.

    - You can access `list` directly in your method.

    (a) Write an iterative method `contains` to take advantage of these observations when searching a sorted array sequentially.

    Refer to AList.java

    (b) Write a recursive method `search` that a method `contains` can call to take advantage of these observations when searching a sorted array sequentially.

    - You may need more than one method to accomplish this recursively.

    Refer to AList.java

4. Trace the method `binarySearch`, as given in Segment 16.13, when searching for 4 in the following array of values:

   [5 8 10 13 15 20 22 26 30 31 34 40]

   Repeat the trace when searching for 34.

   - List the values of `first`, `last`, and `mid` for each call to `binarySearch`

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 8 | 10 | 13 | 15 | 20 | 22 | 26 | 30 | 31 | 34 | 40 |

Search for 4

(a) `binarySearch(0, 11, 4)`

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 8 | 10 | 13 | 15 | 20 | 22 | 26 | 30 | 31 | 34 | 40 |
| first | | | | | mid | | | | | | last |

(b) `binarySearch(0, 4, 4)`

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 8 | 10 | 13 | 15 | 20 | 22 | 26 | 30 | 31 | 34 | 40 |
| first | | mid | | last | | | | | | | |

(c) `binarySearch(0, 1, 4)`

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 8 | 10 | 13 | 15 | 20 | 22 | 26 | 30 | 31 | 34 | 40 |
| first mid | last | | | | | | | | | | |

(d) `binarySearch(0, -1, 4)`
`return found = false`

Search for 34

(a) `binarySearch(0, 11, 34)`

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 8 | 10 | 13 | 15 | 20 | 22 | 26 | 30 | 31 | 34 | 40 |
| first | | | | | mid | | | | | | last |

(b) `binarySearch(6, 11, 34)`

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 8 | 10 | 13 | 15 | 20 | 22 | 26 | 30 | 31 | 34 | 40 |
| | | | | | | first | | mid | | | last |

(c) `binarySearch(9, 11, 34)`

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 8 | 10 | 13 | 15 | 20 | 22 | 26 | 30 | 31 | 34 | 40 |
| | | | | | | | | | first | mid | last |

`return found = true`

5. Modify the method `binarySearch` in Segment 16.13 so that it returns the index of the first array element that equals `desiredItem`. If the array does not contain such an element, return `-(belongsAt + 1)`, where `belongsAt` is the index of the array location that should contain `desiredItem`. At the end of Segment 16.13, Question 7 asked you to return `-1` in this case. Notice that both versions of the method return a negative integer if and only if `desiredItem` is not found.

   Refer to AList.java

6. Implement a binary search of an array iteratively. Model your methods after the ones given in Segment 16.13.

   Refer to Alist.java

7. Write a recursive method to find the largest object in an array-based list of `Comparable` objects. Like the binary search, your method should divide the array into halves. Unlike the binary search, your method should search both halves for the largest object. The largest object in the array will then be the larger of these two largest objects.

   - You can directly access `list` in your method

   Refer to AList.java

8. Suppose that you are searching an unsorted array of objects that might contain duplicates. Devise an algorithm that returns a list of indices of all objects in the array that match a given object. If the desired object is not in the list, return an empty list.

   - Pseudocode only is required.

   Refer to Homework7Driver.java

9. Repeat the previous exercise for a sorted array. Your algorithm should be recursive and efficient.

   - Pseudocode only is required.

   Refer to Homework7Driver.java

## 2 Part II: Hashing

Consider a system that stores street addresses (perhaps for direct mailing). The key is the street address (e.g., 50 Phelan Avenue). The data would contain other information (e.g., name, zip code, etc.). You have a table of size 11.

1. Define a hash function for these keys.

   Since the key is a string, we could use Horner's method to create the hash code. Since the size of the table is 11, which is a prime number greater than 2, we can use the modulus of the hash code to create the hash function for the table. Producing the hash code can result in an overflow if the length of the string is long enough. To produce a non-negative index in the hash function, we would add the length of the table to a negative modulus so that it lies within the index of the table.

```java
// would go inside the class that holds the key
@Override
public int hashCode()
{
    int hash = 0;
    int n = address.length(); //address is a String that is the key
    final int g = 31;
    for(int i = 0; i < n; i++)
        hash = g * hash + address.charAt(i);

    return hash;

    // alternatively can just return address.hashCode()
}


// would go inside the class that holds the hashtable
// and implements DictionaryInterface
private int getHashIndex(K key)
{
    int hashIndex = key.hashCode() % hashTable.length;
    if(hashIndex < 0)
        hashIndex = hashIndex + hashTable.length;

    return hashIndex;
}
```

2. List five addresses that don't **all** hash to the same location but that would result in collisions. Using separate chaining to resolve collisions, draw your hash table. (Assume elements are sorted.)

   Refer to Homework7Driver.java

   Addresses:

   - "1717 Harrison Street", index 2
   - "50 Phelan Avenue", index 2
   - "1 Dr. Carlton B. Goodlett Place", index 6
   - "2095 Harrison Street", index 6

- "2406 Bryant Street", index 7

0
1
2
3
4
5
6
7
8
9
10

0
1
2 $->$ "1717 Harrison Street"
3
4
5
6
7
8
9
10

0
1
2 $->$ "1717 Harrison Street" $->$ "50 Phelan Avenue"
3
4
5
6
7
8
9
10

0
1
2 $->$ "1717 Harrison Street" $->$ "50 Phelan Avenue"
3
4
5
6 $->$ "1 Dr. Carlton B. Goodlett Place"
7
8
9
10

0

1
2 − > “1717 Harrison Street” − > “50 Phelan Avenue”
3
4
5
6 − > “1 Dr. Carlton B. Goodlett Place” − > “2095 Harrison Street”
7
8
9
10

0
1
2 − > “1717 Harrison Street” − > “50 Phelan Avenue”
3
4
5
6 − > “1 Dr. Carlton B. Goodlett Place” − > “2095 Harrison Street”
7 − > “2406 Bryant Street”
8
9
10

3. Now assume you are using open addressing with linear probing to resolve collisions. Revise your list if necessary so that the five addresses again don’t **all** hash to the same location, but they result in **both** collisions and clustering. Draw your hash table. Make sure you show what is stored in **each** position of the table.

0 − > null
1 − > null
2 − > null
3 − > null
4 − > null
5 − > null
6 − > null
7 − > null
8 − > null
9 − > null
10 − > null

0 − > null
1 − > null
2 − > “1717 Harrison Street”
3 − > null
4 − > null
5 − > null
6 − > null
7 − > null
8 − > null
9 − > null
10 − > null

0 − > null
1 − > null

2 − > “1717 Harrison Street”
3 − > “50 Phelan Avenue”
4 − > null
5 − > null
6 − > null
7 − > null
8 − > null
9 − > null
10 − > null

0 − > null
1 − > null
2 − > “1717 Harrison Street”
3 − > “50 Phelan Avenue”
4 − > null
5 − > null
6 − > “1 Dr. Carlton B. Goodlett Place”
7 − > null
8 − > null
9 − > null
10 − > null

0 − > null
1 − > null
2 − > “1717 Harrison Street”
3 − > “50 Phelan Avenue”
4 − > null
5 − > null
6 − > “1 Dr. Carlton B. Goodlett Place”
7 − > “2095 Harrison Street”
8 − > null
9 − > null
10 − > null

0 − > null
1 − > null
2 − > “1717 Harrison Street”
3 − > “50 Phelan Avenue”
4 − > null
5 − > null
6 − > “1 Dr. Carlton B. Goodlett Place”
7 − > “2095 Harrison Street”
8 − > “2406 Bryant Street”
9 − > null
10 − > null

Indexes 2-3 is one cluster, and indexes 6-8 is another cluster

4. Assume a new hash table for the same kind of keys and using the hash function defined in #1. The hash table is now of size 7. The hash table uses open addressing and linear probing. Show the table after inserting the following keys in this order. Make sure you show what is stored in **each** position of the table.

Refer to Homework7Driver.java

Addresses:

- “15 Irving”, index 2
- “700 Ocean”, index 6
- “65 California”, index 4
- “135 Greenwhich”, index 4
- “940 Mason”, index 4
- “778 Judah”, index 4
- “89 Brannan”, index 2

0 − > null
1 − > null
2 − > null
3 − > null
4 − > null
5 − > null
6 − > null

0 − > null
1 − > null
2 − > “15 Irving”
3 − > null
4 − > null
5 − > null
6 − > null

0 − > null
1 − > null
2 − > “15 Irving”
3 − > null
4 − > null
5 − > null
6 − > “700 Ocean”

0 − > null
1 − > null
2 − > “15 Irving”
3 − > null
4 − > “65 California”
5 − > null
6 − > “700 Ocean”

0 − > null
1 − > null
2 − > “15 Irving”
3 − > null
4 − > “65 California”

5 − > "135 Greenwhich"
6 − > "700 Ocean"

0 − > "940 Mason"
1 − > null
2 − > "15 Irving"
3 − > null
4 − > "65 California"
5 − > "135 Greenwhich"
6 − > "700 Ocean"

0 − > "940 Mason"
1 − > "778 Judah"
2 − > "15 Irving"
3 − > null
4 − > "65 California"
5 − > "135 Greenwhich"
6 − > "700 Ocean"

0 − > "940 Mason"
1 − > "778 Judah"
2 − > "15 Irving"
3 − > "89 Brannan"
4 − > "65 California"
5 − > "135 Greenwhich"
6 − > "700 Ocean"

5. Now assume you want to retrieve each of these seven keys. How many locations do you look in before finding each one? How many locations do you look in if you try to find the location 210 Jones?

   Addresses:

   - "15 Irving", index 2, 1 location
     (a) "15 Irving"
   - "700 Ocean", index 6, 1 location
     (a) "700 Ocean"
   - "65 California", index 4, 1 location
     (a) "65 California"
   - "135 Greenwhich", index 4, 2 locations
     (a) "65 California"
     (b) "135 Greenwhich"
   - "940 Mason", index 4, 4 locations
     (a) "65 California"
     (b) "135 Greenwhich"
     (c) "700 Ocean"
     (d) "940 Mason"
   - "778 Judah", index 4, 5 locations
     (a) "65 California"
     (b) "135 Greenwhich"

  (c) "700 Ocean"
  (d) "940 Mason"
  (e) "778 Judah"

- "89 Brannan", index 2, 2 locations
  (a) "15 Irving"
  (b) "89 Brannan"
- "210 Jones", index 5, 7 locations
  (a) "135 Greenwhich"
  (b) "700 Ocean"
  (c) "940 Mason"
  (d) "778 Judah"
  (e) "15 Irving"
  (f) "89 Brannan"
  (g) "65 California"

6. What does the table look like if you remove the key 940 Mason? How many location do you now look in if you try to find the location 210 Jones?

   0 − > available
   1 − > "778 Judah"
   2 − > "15 Irving"
   3 − > "89 Brannan"
   4 − > "65 California"
   5 − > "135 Greenwhich"
   6 − > "700 Ocean"

   Address:

   - "210 Jones", index 5, 7 locations
     (a) "135 Greenwhich"
     (b) "700 Ocean"
     (c) available
     (d) "778 Judah"
     (e) "15 Irving"
     (f) "89 Brannan"
     (g) "65 California"

7. What if you had a hash table of size 11 (instead of 7)? Draw the table after inserting the same seven addresses list above. Make sure you know what is stored in **each** position of the table.

   Refer to Homework7Driver.java

   Addresses:

   - "15 Irving", index 7
   - "700 Ocean", index 8
   - "65 California", index 5
   - "135 Greenwhich", index 7

- “940 Mason”, index 0
- “778 Judah”, index 5
- “89 Brannan”, index 0
- “210 Jones”, index 3

0 − > null
1 − > null
2 − > null
3 − > null
4 − > null
5 − > null
6 − > null
7 − > null
8 − > null
9 − > null
10 − > null

0 − > null
1 − > null
2 − > null
3 − > null
4 − > null
5 − > null
6 − > null
7 − > “15 Irving”
8 − > null
9 − > null
10 − > null

0 − > null
1 − > null
2 − > null
3 − > null
4 − > null
5 − > null
6 − > null
7 − > “15 Irving”
8 − > “700 Ocean”
9 − > null
10 − > null

0 − > null
1 − > null
2 − > null
3 − > null
4 − > null
5 − > “65 California”
6 − > null
7 − > “15 Irving”
8 − > “700 Ocean”

9 − > null
10 − > null

0 − > null
1 − > null
2 − > null
3 − > null
4 − > null
5 − > "65 California"
6 − > null
7 − > "15 Irving"
8 − > "700 Ocean"
9 − > "135 Greenwhich"
10 − > null

0 − > "940 Mason"
1 − > null
2 − > null
3 − > null
4 − > null
5 − > "65 California"
6 − > null
7 − > "15 Irving"
8 − > "700 Ocean"
9 − > "135 Greenwhich"
10 − > null

0 − > "940 Mason"
1 − > null
2 − > null
3 − > null
4 − > null
5 − > "65 California"
6 − > "778 Judah"
7 − > "15 Irving"
8 − > "700 Ocean"
9 − > "135 Greenwhich"
10 − > null

0 − > "940 Mason"
1 − > "89 Brannan"
2 − > null
3 − > null
4 − > null
5 − > "65 California"
6 − > "778 Judah"
7 − > "15 Irving"
8 − > "700 Ocean"
9 − > "135 Greenwhich"
10 − > null

8. With this new table, how many locations do you look in before finding each one? How many locations do you look in if you try to find the location 210 Jones?

Addresses:

- “15 Irving”, index 7, 1 location
  (a) “15 Irving”
- “700 Ocean”, index 8, 1 location
  (a) “700 Ocean”
- “65 California”, index 5, 1 location
  (a) “65 California”
- “135 Greenwhich”, index 7, 3 locations
  (a) “15 Irving”
  (b) “700 Ocean”
  (c) “135 Greenwhich”
- “940 Mason”, index 0, 1 location
  (a) “940 Mason”
- “778 Judah”, index 5, 2 locations
  (a) “65 California”
  (b) “778 Judah”
- “89 Brannan”, index 0, 2 locations
  (a) “940 Mason”
  (b) “89 Brannan”
- “210 Jones”, index 3, 1 location
  (a) null