Joseph Martinez
PID: 3816842

**Problem #1: (35 pts)**

Consider the problem of inserting the **keys 10, 22, 31, 4, 15, 28, 17, 88, and 59** into a hash table of length 11 (**TableSize = 11**) using open addressing with the standard hash function h(k) = k mod TableSize. Illustrate the result of inserting these keys using:

(a) Linear probing.

Keys = { 10, 22, 31, 4, 15, 28, 17, 88, 59}

( h (key) + i ) % TableSize

| value | 22 | 88 | | | 4 | 15 | 28 | 17 | 59 | 31 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

h(10) % 11 = 10

h(22) % 11 = 0

h(31) % 11 = 9

h(4) % 11 = 4

h(15) % 11 = 4

h (15) + 1 % 11 = 5

h(28) % 11 = 6

h(17) % 11 = 6

h(17) + 1 % 11 = 7

h(88) % 11 = 0

h(88) + 1 % 11 = 1

h(59) % 11 = 4

h(59) + 1 % 11 = 5

h(59) + 2 % 11 = 6

h(59) + 3 % 11 = 7

h(59) + 4 % 11 = 8

Joseph Martinez
PID: 3816842

(b) Quadratic probing with quadratic probe function c(i) = 3i2 + i.

Keys = { 10, 22, 31, 4, 15, 28, 17, 88, 59}

( h (key) + 3i2 + i ) % TableSize

| value | 22 | | 28 | 59 | 4 | 17 | 15 | 88 | | 31 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

h(10) % 11 = 10

h(22) % 11 = 0

h(31) % 11 = 9

h(4) % 11 = 4

h(15) % 11 = 4

h (15) + 3(1)2 + 1 % 11 = 0

h (15) + 3(2)2 + 2 % 11 = 6

h(28) % 11 = 6

h (28) + 3(1)2 + 1 % 11 = 22

h(17) % 11 = 6

h (17) + 3(1)2 + 1 % 11 = 2

h (17) + 3(2)2 + 2 % 11 = 9

h (17) + 3(3)2 + 3 % 11 = 5

h(88) % 11 = 0

h (88) + 3(1)2 + 1 % 11 = 7

h(59) % 11 = 4

h (59) + 3(1)2 + 1 % 11 = 0

h (59) + 3(2)2 + 2 % 11 = 7

h (59) + 3(3)2 + 3 % 11 = 3

Joseph Martinez
PID: 3816842

(c) Double hashing with u(k) = k and v(k) = i + (k mod(TableSize − 1)).

Keys = { 10, 22, 31, 4, 15, 28, 17, 88, 59}

u(k) = k and v(k) = 1 + (k % (TableSize – 1 ))

u(k) + i * (1 + (v(k) % (TableSize – 1)) % TableSize

| value | 22 | 28 | 59 | 17 | 4 | 15 | | 28 | 88 | 31 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

u(10) % 11 = 10

u(22) % 11 = 0

u(31) % 11 = 9

u(4) % 11 = 4

u(15) % 11 = 4

u(15) + (1) * (1 + (v(15) % (11 -1)) % 11 = 10

u(15) + (2) * (1 + (v(15) % (11 -1)) % 11 = 5

u(28) % 11 = 6

u(28) + (1) * (1 + (v(28) % (11 -1)) % 11 = 4

u(28) + (2) * (1 + (v(28) % (11 -1)) % 11 = 7

u(17) % 11 = 6

u(17) + (1) * (1 + (v(17) % (11 -1)) % 11 = 3

u(88) % 11 = 0

u(88) + (1) * (1 + (v(88) % (11 -1)) % 11 = 8

u(59) % 11 = 4

u(59) + (1) * (1 + (v(59) % (11 -1)) % 11 = 3

u(59) + (2) * (1 + (v(59) % (11 -1)) % 11 = 2

Joseph Martinez
PID: 3816842

**Problem #2: (30 pts)**

(a) Given a sorted array of N distinct integers that has been rotated an unknown number of times. Implement (in Java) an efficient algorithm that finds an element in the array.

**PLEASE REVIEW JAVA IMPLEMENTATION**

(b) What is the running time complexity of your algorithm? Note: You may assume that the array was originally sorted in increasing order. Example:

**The running time is O( log n )**

**Finding the pivot is: O( log n )**

**Calling the binary search on the partitioned array: O( log n )**

Input:

Find 5 in array (15 16 19 20 25 1 3 4 5 7 10 14)

Output:

8 (the index of 5 in the array)

Joseph Martinez
PID: 3816842

**Problem #3: (35 pts)**

(a) Given an unsorted array A[1...n] of distinct integers numbers and is given a nonnegative integer number, k, k < n. You need to find an element from A such that its rank is k, i.e., there are exactly (k-1) numbers less than or equal to that element.

Example:

Input:

A = [1*, -3*, 4*, 3*, 12*, 20*, 30*, 7*, 14*, -1*, 0*] and k =8.

Output:

-3, -1, 0, 1, 3, 4, 7, 12, 14, 20, 30

12, since 7, -3, -1, 0, 4, 3, 1 (8-1=7 numbers) are all less than or equal to 12

Suggest **a sub-quadratic** running time complexity algorithm (only pseudo code) to solve this problem. Justify.

```
//We can first sort the array using any O( n log n ) algorithm. We use a Merge Sort, as this algorithm provides a warranty that the worst case is O( n log n ). The printing of the element will take O( n ).

//Thus we have O( n log n ) + O ( n )

public static void function ( list , k)
        list sortedList = mergeSort(list);  // let us assume that merge sort return a sorted copy array
        if k  > sortedList.length  // check if k is larger than the list size.
                break
        print( k is equal to sortedList[ k -1 ] )
        while n is less than the k
                print ( sortedList[ n ] )  // print all the numbers that rank less or equal to k -1
        n ++
```

Joseph Martinez
PID: 3816842

(b) Given an array A of n + m elements. It is known that the first n elements in A are sorted and the last m elements in A are unsorted.

Suggest an algorithm (only pseudo code) to sort A in O( m log m + n) worst case running time complexity. Justify.

```
// Given the problem we can use HeapSort sort to solve this situation in O ( m log m + n) time.

// first we can check if the array is sorted in O( n )

public static void function( alist )

for i = 0; i < alist.length - 1; i++

        j = i + 1

        if alist[ i ] > alist[ j ]

                heapsort( alist ) // If the array is not completely sorted we call Heapsort O (n log n)

                 break


public static void heapsort( alist )

        for i = 0 ; i < alist.length ;  i++

                blist[ I ] = insert ( alist [ i ] )  // Build the heap

        for i = 0 ; i < blist.length ;  i++

                alist[ i ] = deleteMin()   // Print the popped values
```

(c) The processing time of an algorithm is described by the following recurrence equation (assume the c is a positive constant):

$T(n) = 3T(n/3) + 2cn; T(1) = 0$

What is the running time complexity of this algorithm? Justify.

$T(n) = 3T(n/3) + 2cn; T(1) = 0$

$a = 3 , b = 3, d = 1$

$\log_3 3 = 1$

$3 = 3^1$ True thus T(n) is O( n log n )

Joseph Martinez
PID: 3816842

(d) You decided to improve insertion sort by using binary search to find the position p where the new insertion should take place.

(d.1) What is the worst-case complexity of your improved insertion sort if you take account of **only the comparisons made by the binary search**?

O( n log n)

Binary search is O( log n) * number of shifting elements O( n )

(d.2) What is the worst-case complexity of your improved insertion sort **if only swaps/inversions of the data values are taken into account**?

Worst case is for the while whole algorithm is:

Inserting * ( Binary Search + Shifting)

$n_1$ * (O( log n) + O($n_2$))

if only the swaps are taken into account we would have inserting number of element (n) and number of swaps (n):

O(n) * O(n) = O(n^2)