4999406

Problem 3:

Part A:

Hash Method: *h(key) = (2*key + 5) mod TableSize* ***Collision:*** *Separate Chaining*

h(12) = 7
h(44) = 5
h(13) = 9
h(88) = 5
h(23) = 7
h(94) = 6
h(11) = 5
h(39) = 6
h(20) = 1
h(16) = 4
h(5) = 4

Surface		20			16	44	94	12		13	
Chain 1					5	88	39	23			
Chain 2						11					
Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Part B:

Method: *Linear Probing*

h(12) = 7
h(44) = 5
h(13) = 9
h(88) = 5 Collision= 6
h(23) = 7 Collision = 8
h(94) = 6 Collision = 10
h(11) = 5 Collision = 0
h(39) = 6 Collision = 1
h(20) = 1 Collision = 2
h(16) = 4
h(5) = 4 Collision = 3

	11	39	20	5	16	44	88	12	23	13	94
Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Part C:
Method: *Quadratic Probing*

h(12) = 7
h(44) = 5
h(13) = 9
h(88) = 5 Collision = 6
h(23) = 7 Collision = 8
h(94) = 6 Collision = 10
h(11) = 5 Collision = 3
h(39) = 6 Collision = 4
h(20) = 1
h(16) = 4 Collision = 2
h(5) = 4 Collision = 0

	5	20	16	11	39	44	88	12	23	13	94
Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Part D:
Method: *Double Hasing*
New Hash Function: *v(key) = 7 - (key mod 7)*

h(12) = 7
h(44) = 5
h(13) = 9
h(88) = 5 Collision: v(88) = 3; 5 + 3 = 8
h(23) = 7 Collision: v(23) = 5; 7 + 5 = 2
h(94) = 6
h(11) = 5 Collision: v(11) = 3; 5 + 3 = 8… + 3 = 0
h(39) = 6 Collision: v(39) = 3; 6 + 3 = 9…+3 = 2…+3 = 5…+3 = 8…+ 3 = 1
h(20) = 1 Collision: v(20) = 1; 1+1 = 2+1 = 3
h(16) = 4
h(5) = 4 Collision v(5) = 2; 4+2 = 6 + 2 = 8 + 2 = 10

	11	39	23	20	16	44	94	12	88	13	5
Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Part E:

```
IsSubset(Integers S, Integers T)

Array STHash of size T.n.

For index = 0 and index <= n:
        Insert element at index from T into position of STHash derived from hash.

For index = 0 and index <= m:
        If element of S at index is not found in STHash:
                Return result as false.

Return result as true.
```

The first for loop is O(n) and populates the array with T's elements.
The second is O(m) and goes through the elements of S and attempts to find them in the table.
The total complexity is thus O(n+m).