**CS 3050 Homework # 3. Name :**

**Submitted to Blackboard, due at 11:59pm on Feb. 28, 2017.**

1. Demonstrate what happens when we insert the keys 3, 11, 80, 74, 92, 1024, 32, 59, 503, 293, 2010, 22, 104 into a hash table with collisions resolved by chaining. Let the table have 11 slots, and let the hash function be h(k) = k mod 11.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| v | 11 | 1024 |  | 3 | 92 | 104 |  | 293 | 74 |  | 32 |
|  | 22 |  |  | 80 | 59 |  |  |  | 503 |  |  |
|  |  |  |  |  |  |  |  |  | 2010 |  |  |

1. Consider a hash table of size m = 128 and a corresponding hash function h(k) = floor( m (kA mod 1) ) for A = (sqrt(5)-1)/2. Compute the locations to which the keys 1000, 1001, 1002, 1003, and 1004 are mapped..

h(1000) = floor( 128 (1000A mod 1) ) =

floor( 128 (0.0339887) )=floor(4.3505536) =4

h(1001) = floor( 128 (1001A mod 1) ) =

floor( 128 (0.6520227) )=floor(83.4589056)=83

h(1002) = floor( 128 (1002A mod 1) ) =

floor( 128 (0.2700567) )=floor(34.5672576)=34

h(1003) = floor( 128 (1003A mod 1) ) =

floor( 128 (0.8880907) )=floor(113.6756096)=113

h(1004) = floor( 128 (1004A mod 1) ) =

floor( 128 (0.5061247) )=floor(64.7839616)=64

3. Write pseudocode for HASH-DELETE as outlined in the textbook, and modify HASH-INSERT to handle the special value DELETED.

HASH-DELETE(T,k)

i ← 0

j ← h(k, i)

while i < m-1 and T[j] ≠ null and T[j] ≠ DELETED

i ← i +1

j ← h(k, i)

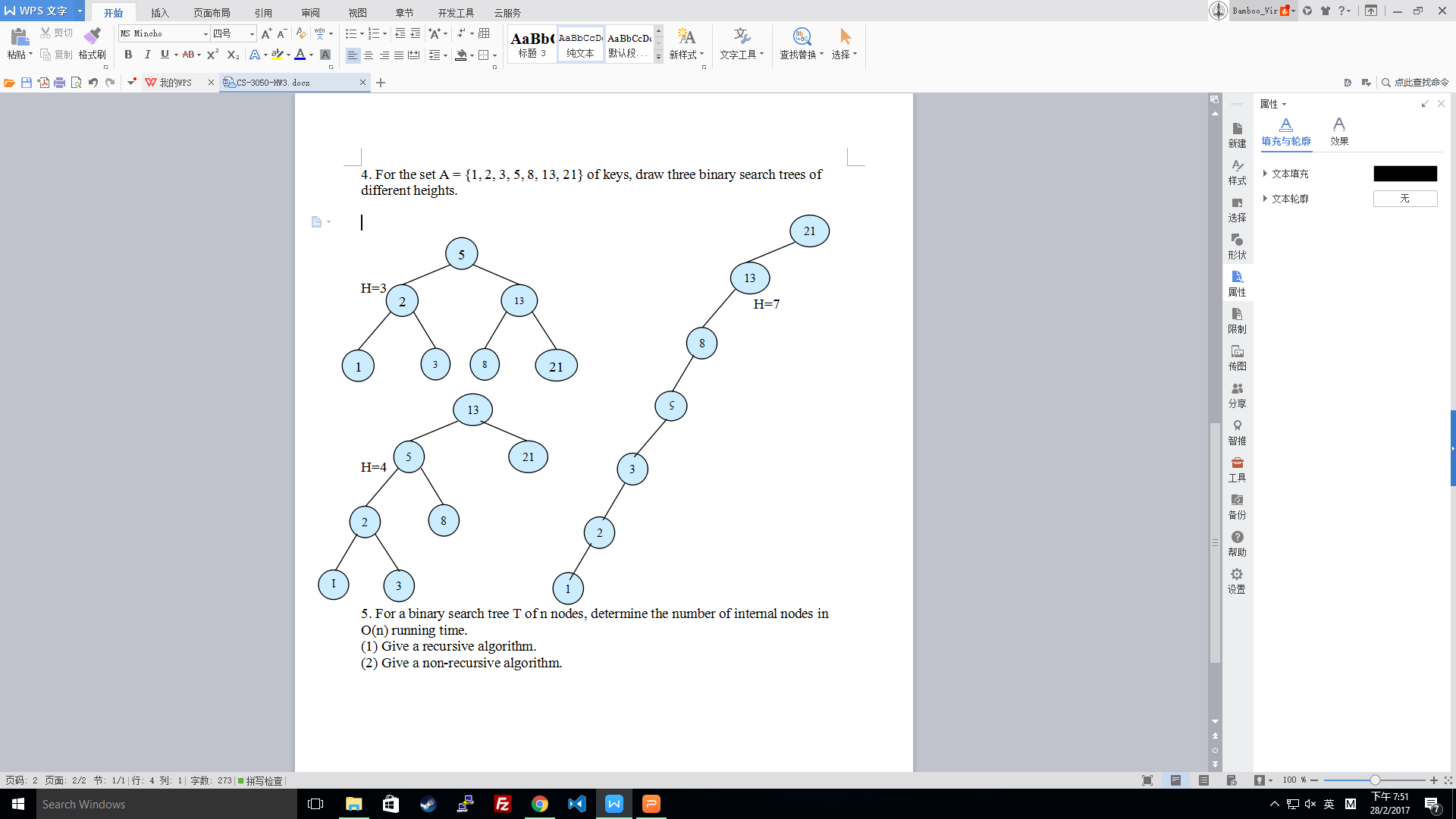
if T[j] = k

T[j]=DELETED

else

error”Fail to delete”

4. For the set A = {1, 2, 3, 5, 8, 13, 21} of keys, draw three binary search trees of different heights.



5. For a binary search tree T of n nodes, determine the number of internal nodes in O(n) running time.

(1) Give a recursive algorithm.

(2) Give a non-recursive algorithm.

Recursive :

Preorder (TreeNode X , I = 0) {

If X != NIL {

I ++ ;

If ((I = Preorder ( X . Get\_Left() , I))!=NIL and

((X.Get\_Left().Get\_Left()==NIL

Or

X.Get\_Left().Get\_Right == NIL))->i--)

or

(X.Get\_right().Get\_Left()==NIL or X.Get\_Right().Get\_Right == NIL))->i--) {} ;

If ((I = Preorder ( X . Get\_Right() , I))!=NIL and ((X.Get\_Left().Get\_Left()==NIL

Or

X.Get\_Left().Get\_Right == NIL)) -> i--)

Or

((X.Get\_right().Get\_Left()==NIL

Or

X.Get\_Right().Get\_Right == NIL))->i--) {};

}

Return I;

}

Non-recursive :

Preorder\_no\_recursive(TreeNode X) {

I = 0;

INIT Stack<TreeNode> ;

While(X != NIL || ! Stack.empty()) {

If X != null {

I++;

Stack.push(X);

If ((X=X.Get\_Left())!=NIL and

(X.Get\_Left().Get\_Left()==NIL

Or

X.Get\_Left().Get\_Right == NIL)) -> i--)

or

(X.Get\_Right().Get\_Left()==NIL

or

X.Get\_Right().Get\_Right == NIL)) -> i--)

{}

} else {

X = Stack.pop();

If((X = X.getRight()) != NIL and

(X.Get\_Right().Get\_Left()==NIL

Or

X.Get\_Right().Get\_Right == NIL))->i--)

or

(X.Get\_left().Get\_Left()==NIL or X.Get\_Left().Get\_Right == NIL))->i--)

{}

}

}

}