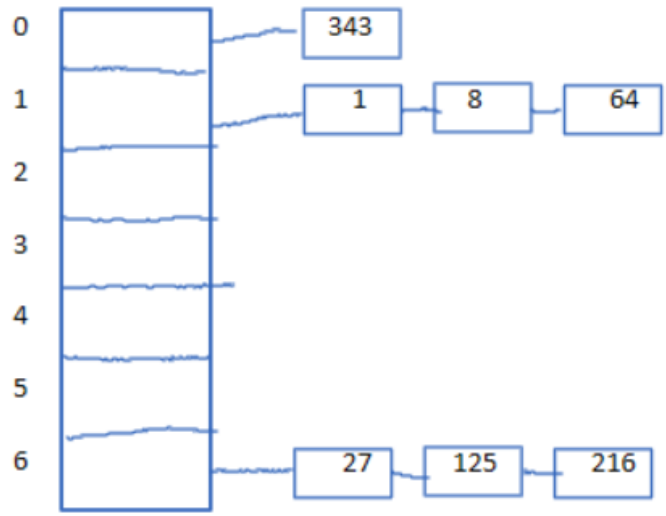
Steven Kravitsky
Homework 4

1) The maximum height of the tree must be n-1 since. The tallest binary tree you can create is one where each node only has one child, which means that there is one less edge than there is vertex since the root has no parent. The minimum height is log(n+1) -1 with the base of the log being 2. The minimum size is when each node has two children so the total number of nodes + 1 is a power of 2. You have to subtract one though since there is only one root node.
2) The minimum height of a tree is 1 since every node connected to the root can be a leaf. The maximum height is n-1, meaning each node only has one child.
3)

	Open hash	Closed hash	Unsorted list	Fixed length
MAKENULL	O(n)	O(n)	O(n)	O(n)
UNION	O(n*m)	O(n*m)	O(n*m)	O(n*m)
INTERSECTION	O(n*m)	O(n*m)	O(n*m)	O(n*m)
MEMBER	O(c)	O(c)	O(n)	O(n)
MIN	O(n)	O(n)	O(n)	O(n)
INSERT	O(c)	O(c)	O(c)	O(c)
DELETE	O(c)	O(c)	O(n)	O(n)

4) H(i) = i mod 7
1 % 7 = 1
8 % 7 = 1
27 % 7 = 6
64 % 7 = 1
125 % 7 = 6
216 % 7 = 6
343 % 7 = 0



b) I used 3 as the number of jumps to go if a block is full

5) Using the length of a string as a hash function is not a good idea because there are multiple inputs that will output the same hash, especially since most words have a length less than 10 characters. It is not smart to use a random value to bucket the data because then you have no way to find that data later, since inputting the value will generate a new random value
6) An efficient data structure to use would be a bit vector. If the subset of S is integers from 1 to n then the size of the bit vector would only need to be n. To remove integer i from the set all you need to do is bitwise and the entire vector with 1's except for the index where you want to remove, that would be a 0. To insert a value to the bit vector you would bitwise or the bit vector with all 0's except the index you want to insert, that would be a one. The inset and delete functions could both be done in constant time as well as searching.
7) The program is linear since it is going through all values of n and only performing constant time operations.

```
X = LargePrimeNumber
Size = n * hashfactor
biggerTable = NewTable(Size)

For n in Table:
        Index = hash(Table[n])
        While biggerTable[index] is None:
                Index += X
        biggerTable[Index] = Table[n]
```