8.1-1

What is the smallest possible depth of a leaf in a decision tree for a comparison sort?

n-1. for the leaf, best case is to compare the n-1 time.
A=[1,2,3,4] is sorted, we compare, 1-2, 2-3,3-4

8.2-1

Using Figure 8.2 as a model, illustrate the operation of COUNTING-SORT on the array $A = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$.

```
1 2 3 4 5 6 7 8 9 10 11
~~[illegible]~~
```

A = {6,0,2,0,1,3,4,6,1,3,2}

C = , 2, 2, 2, 1, 0, 2}   C = {2. 4. 6. 8. 9. 9, 11}

```
     1 2 3 4 5 6 7 8 9 10 11            1 2 3 4 5 6 7 8 9 10 11
B =            2                   B =            2   3
```

```
C = 2  4  5  8  9  9  11           C = 2  4  5  7  9 9 11
```

```
     1  2  3  4  5  6  7  8  9  10  11
B =             1     2     3
```

```
C = 2  3  5  7  9 9 11             B = 0 0 1 1 2 2 3 3 4 6 6
```

8.2-2

Prove that COUNTING-SORT is stable.

the items with the same key are placed in the destination array index-decreasingly. If we iterate these items in the original array also index-decreasingly, the sorting will be stable.

8.3-2

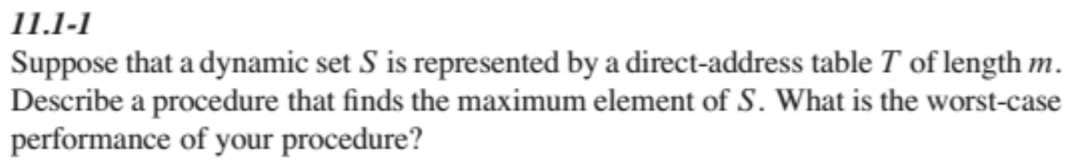
Which of the following sorting algorithms are stable: insertion sort, merge sort, heapsort, and quicksort? Give a simple scheme that makes any sorting algorithm stable. How much additional time and space does your scheme entail?

Insertion sort is stable. When inserting A[j] into the sorted sequence A[1...j-1]. Compare A[j]t o A[i], starting with i = j - 1 and going down to i = 1. Continue at long as A[j] < A[i].

Merge sort is stable. When two elements compared are equal, the tie is broken by taking the element from array L which keeps them in the original order

Heap-sort and quick-sort are not.

To add to each key the position in the initial array and to sort using the additional secondary key. This requires O(n) additional space and has the same time requirement

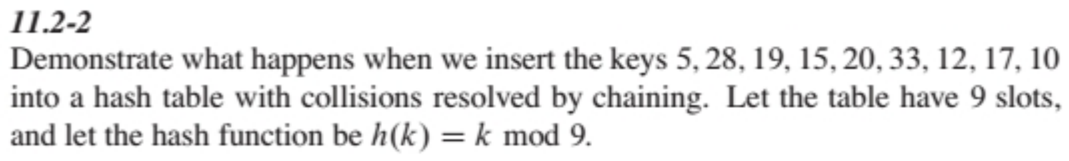


11.1-1

Suppose that a dynamic set S is represented by a direct-address table T of length m. Describe a procedure that finds the maximum element of S. What is the worst-case performance of your procedure?

```
FindMax (T, m)
{
  max = $-\infty$
  for i = 1 to m {
  if   T[i] != NIL && max < T[i]
  max = T[i]}
  return max
}
```

The worst case is O(m)



11.2-2

Demonstrate what happens when we insert the keys 5, 28, 19, 15, 20, 33, 12, 17, 10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \bmod 9$.

.

K	h(k)
5	5
28	1
19	1
15	6
20	2
33	6
12	3
17	8
10	1

Hash Table			
0	Null		
1	28	19	10
2	20		
3	12		
4	Null		
5	5		
6	15	33	
7	Null		
8	17		

11.2-3

Professor Marley hypothesizes that he can obtain substantial performance gains by modifying the chaining scheme to keep each list in sorted order. How does the professor's modification affect the running time for successful searches, unsuccessful searches, insertions, and deletions?

11.3-1

Suppose we wish to search a linked list of length n, where each element contain a key k along with a hash value $h(k)$. Each key is a long character string. Hov might we take advantage of the hash values when searching the list for an elemen with a given key?

use h(k) to create a bloom filter of strings in the linked list. This is an $\Theta(1)$ check to determine if it is possible that a string appears in the linked list

11.4-1

Consider inserting the keys 10, 22, 31, 4, 15, 28, 17, 88, 59 into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h'(k) = k$. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $c_1 = 1$ and $c_2 = 3$, and using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \bmod (m - 1))$.

set of key $\{10, 22, 31, 4, 15, 28, 17, 88, 59\}$

$m=11$

$c_1=1$ $c_2 3$ $h(k,i) = (h'(k) + c_1 i + c_2 i^2)$ $(h_1(k) + i h_2(k))$

Linear probing!

0	22
1	88
2	N
3	N
4	4
5	15
6	28
7	17
8	59
9	31
10	10

quadratic probing

0	22
1	N
2	88
3	17
4	4
5	N
6	28
7	59
8	15
9	31
10	10

double hashing

0	22
1	N
2	59
3	17
4	4
5	15
6	28
7	88
8	N
9	31
10	10