zachary_hightower_433_hwk_7

1. Exercises 7.1: 1, 5, 6

   Exercises 7.2: 2

   Exercises 7.3: 1, 5(Honors only), 6, 8

   Due Apr. 24th.

7.1

Q1

1. Is it possible to exchange numeric values of two variables, say, u and v, without using any extra storage?

ANS-1

If we assume we have variables u and v with something like integers, or ints, stored in them, we can use the following short algorithm to swap without using any additional storage.

ALGORITHM `swap(u,v):`

```
u = u + v
v = u - v
u = u - v
```

This uses addition and subtraction tricks in order to swap the two without needing to instantiate a third variable. Similar tricks can be designed to work with other sorts of variables.

Q5

5. Design a one-line algorithm for sorting any array of size n whose values are n distinct integers from 1 to n.

ANS-5

ALGORITHM `oneLineSort (A)`

```
for i = 0, i < n – 1, i++ {A[i] -> B[A[i] – 1]}
```

Q6

6. The ***ancestry problem*** asks to determine whether a vertex u is an ancestor of vertex v in a given binary (or, more generally, rooted ordered) tree of n vertices. Design a $O(n)$ input-enhancement algorithm that provides sufficient information to solve this problem for any pair of the tree's vertices in constant time.

ANS-6

We want to write an algorithm that will manage this determination in linear time. To manage this, we can use the facts of post order and pre order traversal. Because we know that if vertex u is an ancestor of vertex v in the tree, it will have to obey the following conditions

$$pre(u) \leq pre(v)$$

$$post(u) \geq post(v)$$

This is because the preorder traversal visits the root, then the subtrees left to right, which is what establishes the truth of the first statement.

The second statement for postorder is true due to the reverse. It visits the subtrees, then the roots, from left to right.

If these two statements hold, that means u must be an ancestor of v. Since the time efficiencies for both of these types of traversal is $O(n)$ we can write the time complexity as.

$$n + n = 2n = O(n)$$

Since we can drop the constant.

ALGORITHM `boolean checkAncestor:`

```
if {pre(u) <= pre(v)} AND {post(u) >= post(v)}
return true
else
return false
```

7.2

Q2

2. Consider the problem of searching for genes in DNA sequences using Horspool's algorithm. A DNA sequence is represented by a text on the alphabet {A, C, G, T}, and the gene or gene segment is the pattern.

   a. Construct the shift table for the following gene segment of your chromosome 10:

   TCCTATTCTT

   b. Apply Horspool's algorithm to locate the above pattern in the following DNA sequence:

   TTATAGATCTCGTATTCTTTTATAGATCTCCTATTCTT

ANS-2-A

For the pattern in **a.** we want to build a shift table. It will look like this.

c	A	C	G	T
t(c)	5	2	10	1

ANS-2-B

So we're looking for the pattern TCCTATTCTT
Using Horspool's algorithm.
So we start the comparisons and shifts

Position in big string	actions
T - 10 (counting from 1)	Do first two comparisons fail, shift 1
C - 11	Do first comparison, fail, shift 2
T - 13	Do first two comparisons, fail, shift 1
A - 14	Do first comparison, fail, shift 5
T - 19	Do first 8 comparisons, fail, shift 1
T - 20	Do first 3 comparisons, fail, shift 1
T - 21	Do first 3 comparisons, fail, shift 1
A - 22	Do first comparison, fail, shift 5
T - 27	Do first 2 comparisons, fail, shift 1
C - 28	Do first comparison, fail, shift 2
C - 30	Do first comparison, fail, shift 2
T - 32	Do first 2 comparisons, fail, shift 1
A - 33	Do first comparison, fail, shift 5
T - 38	Do ten comparisons, found!

7.3

Q1

1. For the input 30, 20, 56, 75, 31, 19 and hash function $h(K) = K \mod 11$
   a. construct the open hash table.
   b. find the largest number of key comparisons in a successful search in this table.
   c. find the average number of key comparisons in a successful search in this table.

ANS-1-A

We can construct the hash table with our list of keys and the hash function, the operations of which are shown in the following table

K	h(K)
30	8
20	9
56	1
75	9
31	9
19	8

Where we just use the hash function on our key to get h(K) column
Then we can put them into our hash table

Position	In-1	In-2	In-3
0			
1	56		
2			
3			
4			
5			
6			
7			
8	30	19	
9	20	75	31

Position	In-1	In-2	In-3
10			

ANS-1-B

We want to find the largest number of key comparisons in a successful search through the hash table. We can look and see that it would be 3 comparisons as we look for the number 31, going from the first, to the second, to the third position in bucket 9

ANS-1-C

The average number of key comparisons in a successful search through the table is found by the sum of the following formula

$$\frac{1}{6} \times V_1 + \dots$$

Where we assume that a search for each of the six keys is equally likely and that V is the value of how far into the hash table we have to look for the particular key.

$$\frac{1}{6} \times 1 + \frac{1}{6} \times 1 + \frac{1}{6} \times 1 + \frac{1}{6} \times 2 + \frac{1}{6} \times 2 + \frac{1}{6} \times 3$$

$$= \frac{10}{6}$$

Q6

6. Answer the following questions for the separate-chaining version of hashing.

a. Where would you insert keys if you knew that all the keys in the dictionary are distinct? Which dictionary operations, if any, would benefit from this modification?

b. We could keep keys of the same linked list sorted. Which of the dictionary operations would benefit from this modification? How could we take advantage of this if all the keys stored in the entire table need to be sorted?

ANS-6-A

In this version using separate chaining we can always be sure of inserting a new key at the beginning of the linked lists. This means that when we operate under those conditions, insertion will always be constant time of $\Theta(1)$. Though that is the only operation that will be changed by this modification.

ANS-6-B

In this one, we've made modifications that will affect the search, as we can stop the search in a sorted list whenever we find a key larger than the one we're looking for. This also affects deletion and insertion, as both operate along the same principles as a search in order to perform. To sort our dictionary and apply this idea, we can merge all the non empty linked lists for a sort.

Q8

8. Fill in the following table with the average-case (as the first entry) and worst-case (as the second entry) efficiency classes for the five implementations of the ADT dictionary:

	unordered array	ordered array	binary search tree	balanced search tree	hashing
search					
insertion					
deletion					

ANS-8

Average case

Avg cases	unordered array	ordered array	binary search tree	balanced search tree	hashing
search	$\Theta(n)$	$\Theta(log\ n)$	$\Theta(log\ n)$	$\Theta(1)$	$\Theta(1)$
insertion	$\Theta(1)$	$\Theta(n)$	$\Theta(log\ n)$	$\Theta(1)$	$\Theta(1)$
deletion	$\Theta(1)$	$\Theta(n)$	$\Theta(log\ n)$	$\Theta(1)$	$\Theta(1)$

Worst case

Worst cases	unordered array	ordered array	binary search tree	balanced search tree	hashing
search	$O(n)$	$O(n)$	$O(n)$	$O(log\ n)$	$O(n)$
insertion	$O(1)$	$O(n)$	$O(n)$	$O(log\ n)$	$O(n)$
deletion	$O(1)$	$O(n)$	$O(n)$	$O(log\ n)$	$O(n)$