|  |
| --- |
| Assignment |
| Lesson 6 to 9 |
| Rizk Hanna |

**Lesson 6**

**R-4.14**

Both bubble-sort and merge-sort are stable algorithms.

**R-4.16**

No, because it uses an additional sequence for the buckets which is equal to O (n + N) space.

**C-4.13**

Algorithm CompareSet(A, B)

Input two sequences

Output true if the two sequences have the same set of numbers and false vise versa

Create new dictionary D over a hash table using chaining for conflicts

for i ← 1 to A.size() do

element ← A.last()

A.remove(element)

D.insertItem(element, NIL)

for I ← 1 to B.size() do

element ← B.last()

if ¬D.findElement(element) then

return false

return true

**R-5.4**

(a)T (n) = 2T (n/2) + logn a = 2, b = 2, f(n) = logn, logb a = 1

From the first case, logn is O (n1-e)so T (n) is  (n)

(b) T (n) = 8T(n/2) + n2a = 8, b = 2, f(n) = n2, logb a = 3

From the first case, logn is O (n3-e)so T (n) is  (n3)

(c) T(n) = 16T(n/2) + (nlogn)4  a = 16, b = 2, f(n) = n4 log4n, logb a = 4, k = 4

From the first case, n4 log4n is **not** O (n4-e)

From the second case, n4 log4n is  n4 log4n so; T (n) is n4 log5n

(d) T(n) = 7 T(n/3) + n a = 7, b = 3, f(n) = n, logb a = log37, k = 0

From the first case, n is **not** O (n (log37) - e)

From the second case, n is **not**  (nlog37)

From the third case, n is Ω (n (log37) + e) and

af(n/b) <= Vf(n)

7(n/3) <= Vn so

T (n) is (n)

(e) T (n) = 9T (n/3) + n3logn a = 9, b = 3, f(n) = n3logn, logb a = 2, k = 1

From the first case, n3logn is **not** O (n2-e)

From the second case, n3logn is **not**  (n2 logn)

From the third case, n3logn is Ω (n2+e) and

af(n/b) <= Vf(n)

9((n/3)3 logn/3) <= Vn then

n3/3 (logn – log3) <= Vn so

T (n) is  (n3logn)

**Lesson 7**

**R-2.19**

Chaining

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

**R-2.20**

Liner Probing

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

**R-2.21**

Quadratic Probing

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

Can’t insert 5

**R-2.22**

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

Algorithm Remove(k)

Input key to be removed

Output success or failure

counter ← 0

a ← h(k)

while ¬counter = N do

if H[a].element = Ø then

return NO\_SUCH\_KEY

else if H[a].element = k then

H[a].element ← Available

return true

else

a ← ( a + 1 ) % N

counter ← counter + 1

return NO\_SUCH\_KEY

**Lesson 8**

**R-3.8**

No. One property of a (2, 4) tree is that all external nodes are at the same depth. The multi-way search tree of the example does not adhere to this property.

**R-3.10**

Suppose we are inserting a sequence of 5 numbers in two different orders

1. “1, 2, 3, 4, 5”
2. “5, 4, 3, 2, 1”

**C-4.11**

We can use the AVL tree to store the available IDs of the candidates in addition to an integer counter associated with each ID, then we traverse the sequence of the votes, incrementing the counter for the related ID. So each search and update consumes O (log k) -while k is the number of the IDs-; so the total time is O (n log k).

**C-4.22**

Using a balanced search tree like AVL tree to store the first sequence A, and through iterating over the sequence B, we calculate and search for the expected value of “a” according to the currently available information of both “x” and “b”. So each search operation consumes O (log n), n times means O (n log n).

**R-1.10**

Algorithm Loop1(n)

s ← 0 O(1)

for i ← 1 to n do O(n)

s ← s + I O(n)

Running time = O(n).

**C-2.1**

Algorithm findMiddle()

if header = trailer V header.next = trailer then

return header

while header ¬= trailer do

header ← header.next

trailer ← trailer.prev

return header

The worst case running time is O(n).

**Lesson 9**

**R.3.11**

**a)**

1- 2-

3- 4-

5- 6-

7- 8-

9- 10-

11-

**b)**

1- 2-

3- 4-

5- 6-

7- 8-

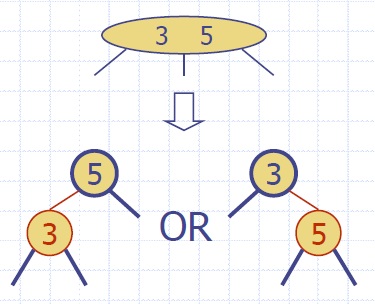
9- 10-

11- 12-

13- 14-

**R-3.14**

1. False, for instance in the above tree, the sub tree starts from node 22 is not a valid tree, as the root should be black
2. True, as the inserted node must be in red, and the Red Black tress is balanced, that’s mean every node should have either Red or external node.
3. True. As there is no role for the conversion gives two different results.
4. False, for instance in the following 2-4 tree, It can be converted into a Red Black tree with two ways



**C-3.10**

Algorithm FindAllInRange(k1, k2)

Input k1 and k2

Output a sequence with all all the elements in D with key k such that k1 ≤ k ≤ k2

return getRange(k1, k2, T.root())

Algorithm getRange(k1, k2, parent)

Input k1, k2 and parent node to start from

Output a sequence with all all the elements in D with key k such that k1 ≤ k ≤ k2.

Create sequence S

if parent.val >= k1 And k2 >= parent.val then

S.add(parent)

if(k1 < parent AND !T.isExternal(parent.leftChild())) then

getRange(parent.leftChild)

if(parent < k2 AND !T.isExternal(parent.rightChild())) then

getRange(parent.rightChild)

return S