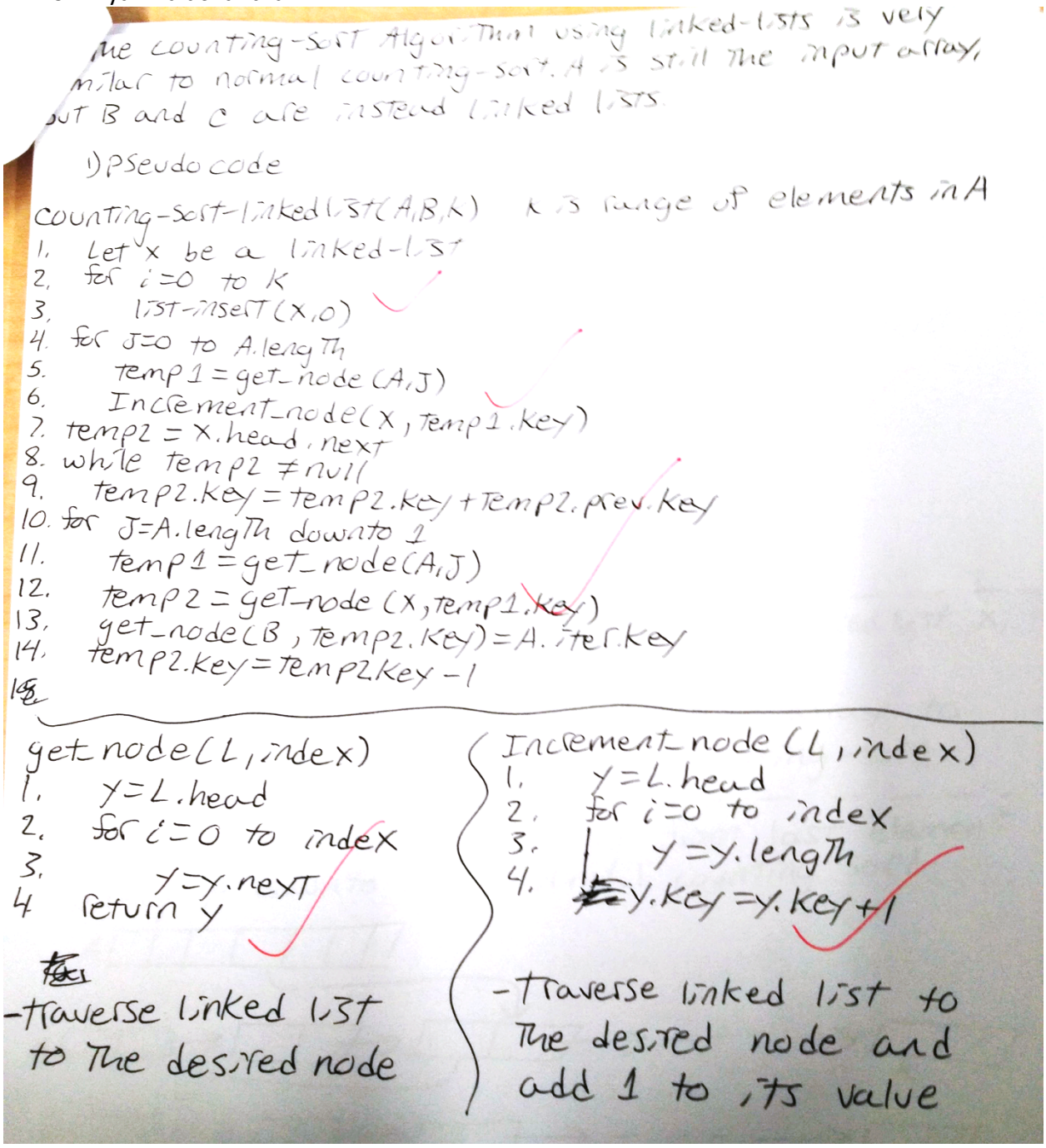
1. From Ryan Balachandran.



The counting-sort Algorithm using linked-lists is very similar to normal counting-sort. A is still the input array, but B and C are instead linked lists.

1) Pseudo code

```
COUNTING-SORT-LINKED-LIST(A,B,k)   k is range of elements in A
1.  Let x be a linked-list
2.  for i=0 to k
3.      LIST-INSERT(x,0)
4.  for j=0 to A.length
5.      temp1 = get_node(A,j)
6.      Increment_node(x, temp1.key)
7.  temp2 = x.head.next
8.  while temp2 ≠ null
9.      temp2.key = temp2.key + temp2.prev.key
10. for j=A.length downto 1
11.     temp1 = get_node(A,j)
12.     temp2 = get_node(x, temp1.key)
13.     get_node(B, temp2.key) = A.iter.key
14.     temp2.key = temp2.key - 1
15.
```

```
get_node(L, index)
1.  y = L.head
2.  for i=0 to index
3.      y = y.next
4.  return y
```

-traverse linked list to the desired node

```
Increment_node(L, index)
1.  y = L.head
2.  for i=0 to index
3.      y = y.length
4.  y.key = y.key + 1
```

-traverse linked list to the desired node and add 1 to its value

pseudocode explanation

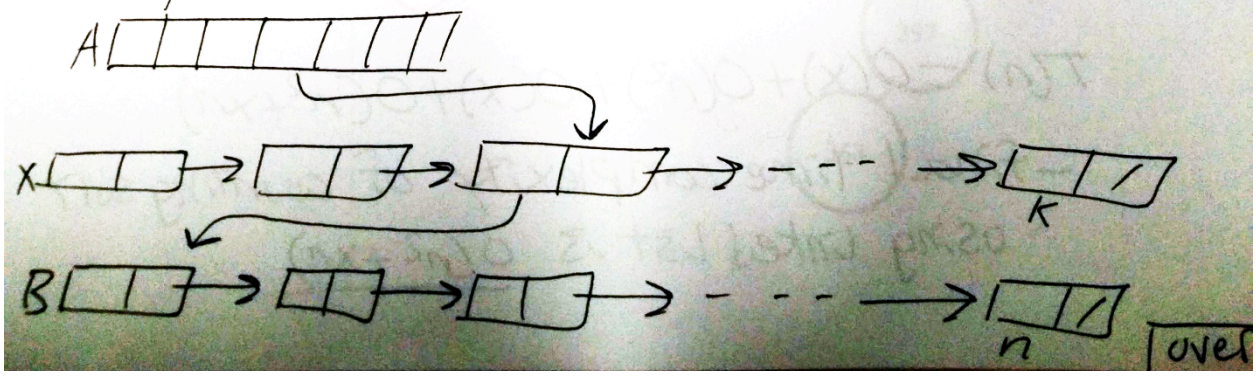
counting-sort-linkedlist(A, B, k)

- create a linked-list X
- takes O(x) time to make and fill with 0's

- when counting the # of different values in A, takes $O(n^2)$ time
- need to traverse ~~[illegible]~~ linked-list up to element in A every iteration
- index 1 in Array A
- index 0 in ~~[illegible]~~ linkedlist X
- index 2 in array A
- index 0, then index 1 in linked list X

- temp2 value starts at index 1 of linked list X, value at index 0 remains the same
- same concept when changing counting array to index position array in normal counting-sort.

- same concept when starting from last element in array A down to 1 in normal counting sort



- takes $O(n^2+xn)$ time
- no constant time
- Have to iterate to n^{th} node to access a key node

---

2) $O(x)$: creating linked list x

$O(n^2)$: iterating through A, and traversing through x to the desired index for every iteration on A.

$O(x)$: changing the counting ~~list~~ linked list of x into the index position linked list x.

Counting linked list: # times elements in A appear
index position: where to place elements in A, into linked list B

$O(n^2+xn)$: have to traverse/iterate through A backwards $[O(n)]$, traverse through x every iteration $[O(n^2)]$, and ~~its~~ traverse linked list B.

~~$T(n) = O(n)$~~

$T(n) = O(x)+O(n^2)+O(x)+O(n^2+xn)$

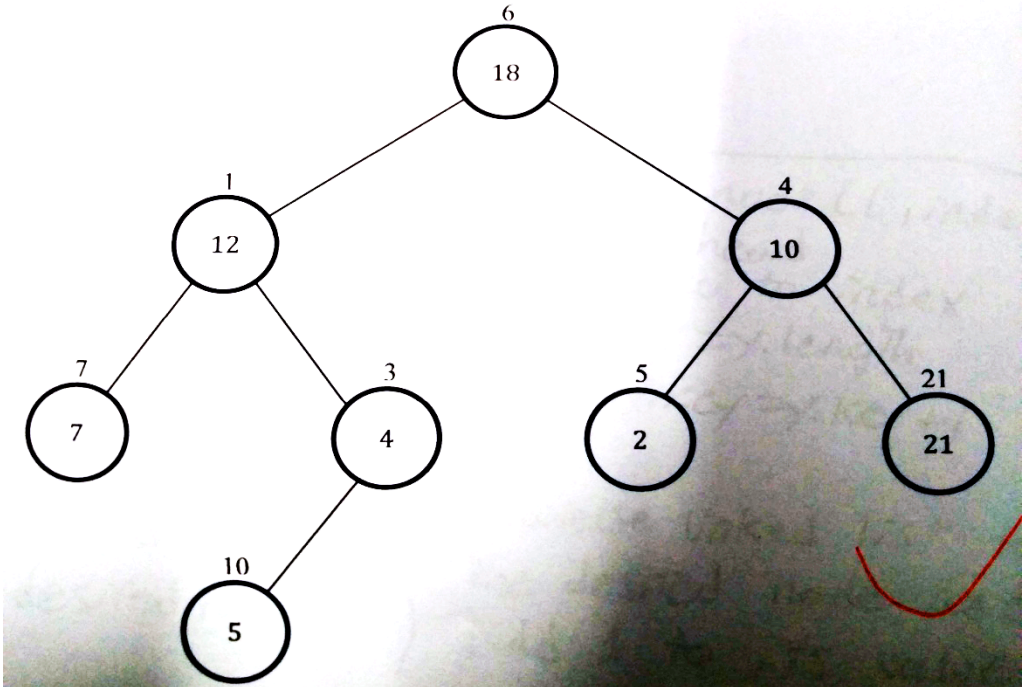
- final time complexity of counting sort using linked list is $\underline{O(n^2+xn)}$

2. From Ryan Balachandran.

2. Exercise 10.4-1, textbook page 248

Q. Draw the binary tree rooted at index 6 that us represented by the following attributes:

Index	Key	Left	Right
1	12	7	3
2	15	8	-
3	4	10	-
4	10	5	9
5	2	-	-
6	18	1	4
7	7	-	-
8	14	6	2
9	21	-	-
10	5	-	-

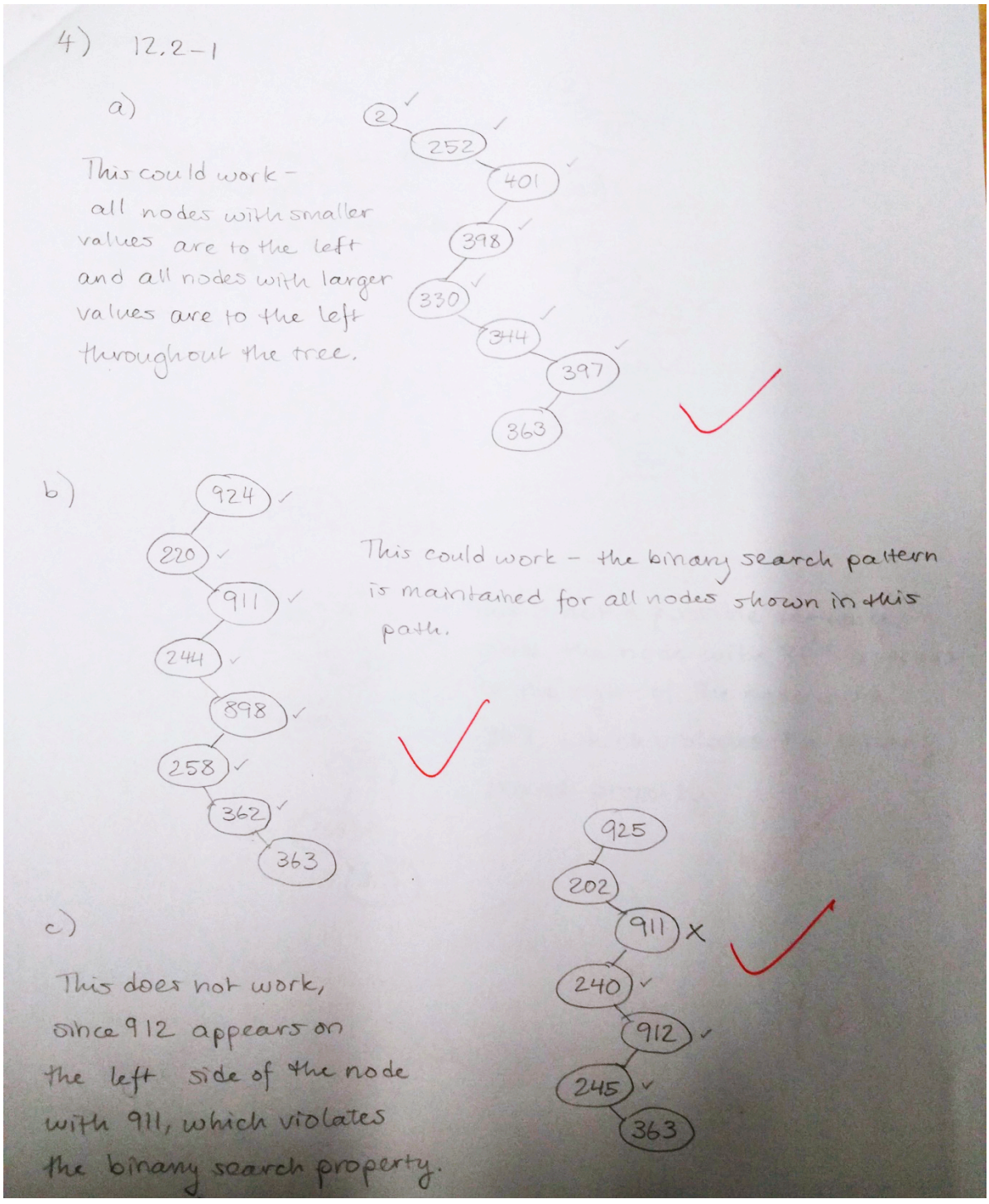


3. From Yufeng Tan.

3. Binary Search Tree and Heap(Exercise 12.1-2)

The difference between binary-search-tree and min-heap is that, in a binary tree, the left child is smaller than its parent, and the right child is larger than its parent; for the min-heap, both left child and right child are larger than their parent.
The min-heap property can't be used to print out the keys of an n-nodes tree in sorted order in O(n) time, since in the min-heap, the larger element can be in the left child or right child. To sort the min-heap, we need to sort it in O(nlgn) time, which is more than O(n).

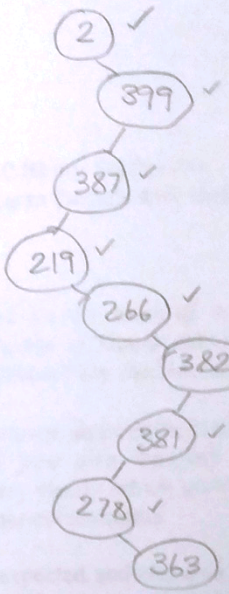
4. From Saara Luna.



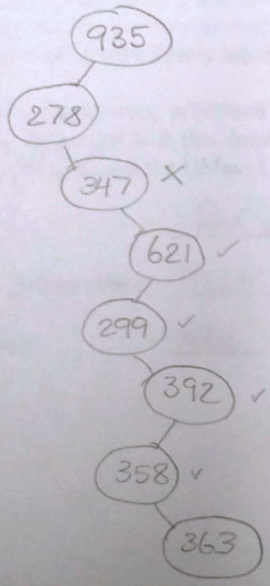
1, continued)

d)

This is possible, since the binary search property is maintained for all nodes in the path.



e)



This is not a possible sequence, since the node with 299 appears to the right of the node with 347, which violates the binary search property.