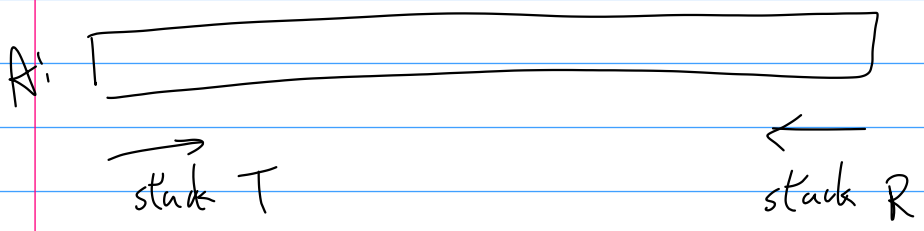
10.1-1

Consider an $m \times n$ matrix in row-major order, where both m and n are powers of 2 and rows and columns are indexed from 0. We can represent a row index i in binary by the $\lg m$ bits $\langle i_{\lg m-1}, i_{\lg m-2}, \ldots, i_0 \rangle$ and a column index j in binary by the $\lg n$ bits $\langle j_{\lg n-1}, j_{\lg n-2}, \ldots, j_0 \rangle$. Suppose that this matrix is a 2×2 block matrix, where each block has $m/2$ rows and $n/2$ columns, and it is to be represented by a single array with 0-origin indexing. Show how to construct the binary representation of the $(\lg m + \lg n)$-bit index into the single array from the binary representations of i and j.

C24106082 陳宏勳

10.1-3

Explain how to implement two stacks in one array $A[1:n]$ in such a way that neither stack overflows unless the total number of elements in both stacks together is n. The PUSH and POP operations should run in $O(1)$ time.



initially T.top = 0, R.top = n+1

```
Push(S, x){
  switch(S){
  case T:
      if(T.top+1 == R.top) return "overflow"
      else A[++T.top] = x

  case R:
      if(R.top-1 == T.top) return "overflow"
      else A[--R.top] = x
}

Pop(S){
  switch(S){
  case T:
      if(T.top==0) return "underflow"
      else return A[T.top--]
  case R:
      if(R.top == n+1) return "underflow"
      else return A[R.top++]
}
```

10.1-7

Show how to implement a queue using two stacks. Analyze the running time of the queue operations.

```
stack A, B
push(z){
    A.push(z)          → O(1)
}
pop(){
    for(x in A.pop()) B.push(x)   → O(n)
    z = B.pop()
    for(x in B.pop()) A.push(x)   → O(n)
    return z
}
```

} O(n)

10.2-1

Explain why the dynamic-set operation INSERT on a singly linked list can be implemented in $O(1)$ time, but the worst-case time for DELETE is $\Theta(n)$.

Insert x:

```
x.next = L.head
L.head = x
```

→ $O(1)$

Delete x:

```
z = head, j = nil
while (z.key != x.key) {
  j = z
  z = z.next
}
j.next = z.next
```

$O(n)$

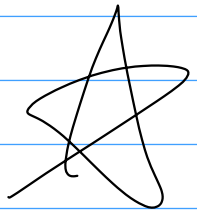
10.2-4

The dynamic-set operation UNION takes two disjoint sets S_1 and S_2 as input, and it returns a set $S = S_1 \cup S_2$ consisting of all the elements of S_1 and S_2. The sets S_1 and S_2 are usually destroyed by the operation. Show how to support UNION in $O(1)$ time using a suitable list data structure.

$S_1.tail.next = S_2.head \rightarrow O(1)$

★ *10.2-6*

Explain how to implement doubly linked lists using only one pointer value $x.np$ per item instead of the usual two (*next* and *prev*). Assume that all pointer values can be interpreted as k-bit integers, and define $x.np = x.next$ XOR $x.prev$, the k-bit "exclusive-or" of $x.next$ and $x.prev$. The value NIL is represented by 0. Be sure to describe what information you need to access the head of the list. Show how to implement the SEARCH, INSERT, and DELETE operations on such a list. Also show how to reverse such a list in $O(1)$ time.



```
typedef struct {
    int key
    node * np
} node

search ( L, val ){
    node *prev = nil , *curr = L.head , * next = nil ;
    while ( curr ≠ nil and curr.key ≠ val ) {
        next = prev ∧ curr.np
        prev = curr
        curr = next
    }
    return curr
}

Insert ( L , x ){
    if (L.head = nil) L.head = x
    else {
        x.np = L.head ∧ nil
        L.head.np = x ∧ ( L.head.np ∧ 0 )
        L.head = x
    }
}

Delete ( L, val) {
    node * curr = L.head , *prev = nil , *next = nil ;
    while ( curr ≠ nil ) {
```

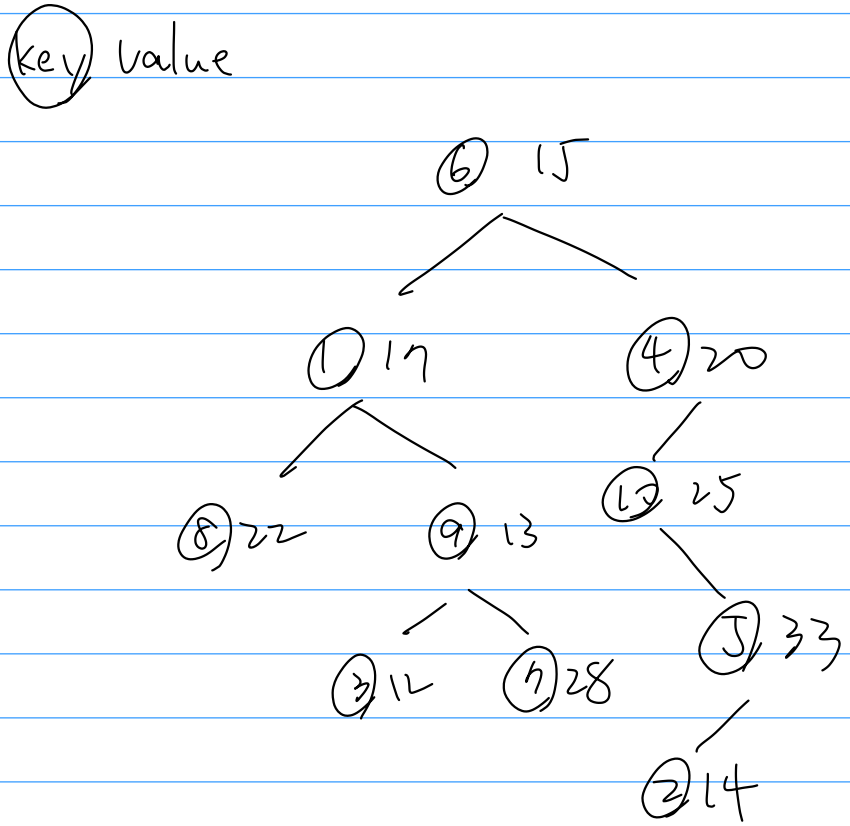
```
        next = curr.np ^ prev
        if (curr.key == val) {
            if (prev ≠ nil) prev.np = next ^ (prev.np ^ curr)
            if (next ≠ nil) next.np = prev ^ (next.np ^ curr)
            if (L.head == curr) L.head = next
            return
        }
        prev = curr
        curr = next
    }
}

reverse(L){
    L.head, L.tail = L.tail, L.head
}
```

10.3-1

Draw the binary tree rooted at index 6 that is represented by the following attributes:

index	*key*	*left*	*right*
1	17	8	9
2	14	NIL	NIL
3	12	NIL	NIL
4	20	10	NIL
5	33	2	NIL
6	15	1	4
7	28	NIL	NIL
8	22	NIL	NIL
9	13	3	7
10	25	NIL	5



10-1 Comparisons among lists

For each of the four types of lists in the following table, what is the asymptotic worst-case running time for each dynamic-set operation listed?

	unsorted, singly linked	sorted, singly linked	unsorted, doubly linked	sorted, doubly linked
SEARCH	n	n	n	n
INSERT	1	1	1	1
DELETE	n	n	1	1
SUCCESSOR	n	1	n	1
PREDECESSOR	n	n	n	1
MINIMUM	n	1	n	1
MAXIMUM	n	n	n	1

11.1-1

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?

```
max(T){
  for(key in m to 1){
    if(T[key] ≠ nil) return key
  }
  return nil
}
```

11.1-2

A ***bit vector*** is simply an array of bits (each either 0 or 1). A bit vector of length m takes much less space than an array of m pointers. Describe how to use a bit vector to represent a dynamic set of distinct elements drawn from the set $\{0, 1, \ldots, m-1\}$ and with no satellite data. Dictionary operations should run in $O(1)$ time.

let bit vector V

V[key] == 1 => key is in set

V[key] == 0 => key is not in set

11.2-1

You use a hash function h to hash n distinct keys into an array T of length m. Assuming independent uniform hashing, what is the expected number of collisions? More precisely, what is the expected cardinality of $\{\{k_1, k_2\} : k_1 \neq k_2 \text{ and } h(k_1) = h(k_2)\}$?

$$X_{ij} = \begin{cases} 1 & h(k_i) = h(k_j) \\ 0 & h(k_i) \neq h(k_j) \end{cases}$$

$\because$ uniform hash, $P(X_{ij}) = \frac{1}{m}$

$\Rightarrow E[X_{ij}] = \frac{1}{m}$

$$E = \sum_{i \neq j} E[X_{ij}]$$

$$= E\left[\sum_{i \neq j} \frac{1}{m}\right]$$

$$= \binom{n}{2} \frac{1}{m} = \frac{n(n-1)}{2} \frac{1}{m} \#$$

11.2-6

You have stored n keys in a hash table of size m, with collisions resolved by chaining, and you know the length of each chain, including the length L of the longest chain. Describe a procedure that selects a key uniformly at random from among the keys in the hash table and returns it in expected time $O(L \cdot (1 + 1/\alpha))$. $\alpha = \frac{n}{m}$

select j from 0 to $m-1$
let $n_j = \text{len}(T[j])$

select k from $0 \sim L-1$
if $(k < n_j)$ return $T[j][k]$
else repeat it again

each element has a probability of $\frac{1}{mL}$

let X = how many times we repeat

let p = successful select an element in one try

the process is like selecting one item in a box with mL slots and n items $\Rightarrow p = \frac{n}{mL} = \frac{\alpha}{L}$

successful / not successful / next try

$E[X] = p(1+\alpha) + (1-p)(1 + E[X])$

the expect of reaching the element on the chain / this try

$\therefore E[X] = \alpha + \frac{1}{p}$

$= \alpha + \frac{L}{\alpha} = L\left(\frac{\alpha}{L} + \frac{1}{\alpha}\right) = O\left(L\left(1 + \frac{1}{\alpha}\right)\right)$ #

$\because \alpha \le L$

11.3-1

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

store the hash of keys in element,

if element.hashed key = h(k), compare(element.key, k)

In this way, we can reduce the number of calling the expansive compare function.

11.3-2

You hash a string of r characters into m slots by treating it as a radix-128 number and then using the division method. You can represent the number m as a 32-bit computer word, but the string of r characters, treated as a radix-128 number, takes many words. How can you apply the division method to compute the hash value of the character string without using more than a constant number of words of storage outside the string itself?

```
k = 0
for c in str:
    k = (k + c) % m
```

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. Illustrate the result of inserting these keys using linear probing with $h(k, i) = (k + i) \bmod m$ and using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \bmod (m - 1))$.

linear :

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

double hashing $h_1(k) = k$, $h_2(k) = 1 + k \bmod 10$

$\Rightarrow h(k) = \left(k + j(1 + k \bmod 10)\right) \bmod 11$

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

★ ***11.4-5***

Show that, with double hashing, if m and $h_2(k)$ have greatest common divisor $d \geq 1$ for some key k, then an unsuccessful search for key k examines $(1/d)$th of the hash table before returning to slot $h_1(k)$. Thus, when $d = 1$, so that m and $h_2(k)$ are relatively prime, the search may examine the entire hash table. (*Hint:* See Chapter 31.)

12.1-2

What is the difference between the binary-search-tree property and the min-heap property on page 163? Can the min-heap property be used to print out the keys of an n-node tree in sorted order in $O(n)$ time? Show how, or explain why not.

BSTs $\Rightarrow$ all left nodes $<$ curr $<$ all right nodes

min-heap $\Rightarrow$ child $>$ parent

$\Rightarrow$ no, $\because$ we don't know which sub-tree is smaller in min-heap

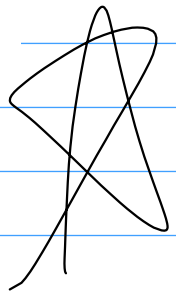
12.1-3

Give a nonrecursive algorithm that performs an inorder tree walk. (*Hint:* An easy solution uses a stack as an auxiliary data structure. A more complicated, but elegant, solution uses no stack but assumes that you can test two pointers for equality.)

```
walk (T){
    stack S
    curr = T.root
    while (curr ≠ null or S is not empty){
        while (curr ≠ null){
            S.push(curr)
            curr = curr.left
        }
        curr = S.pop()
        print (curr)
        current = current.right
    }
}
```

12.1-5

Argue that since sorting n elements takes $\Omega(n \lg n)$ time in the worst case in the comparison model, any comparison-based algorithm for constructing a binary search tree from an arbitrary list of n elements takes $\Omega(n \lg n)$ time in the worst case.



12.2-1

You are searching for the number 363 in a binary search tree containing numbers between 1 and 1000. Which of the following sequences *cannot* be the sequence of nodes examined?

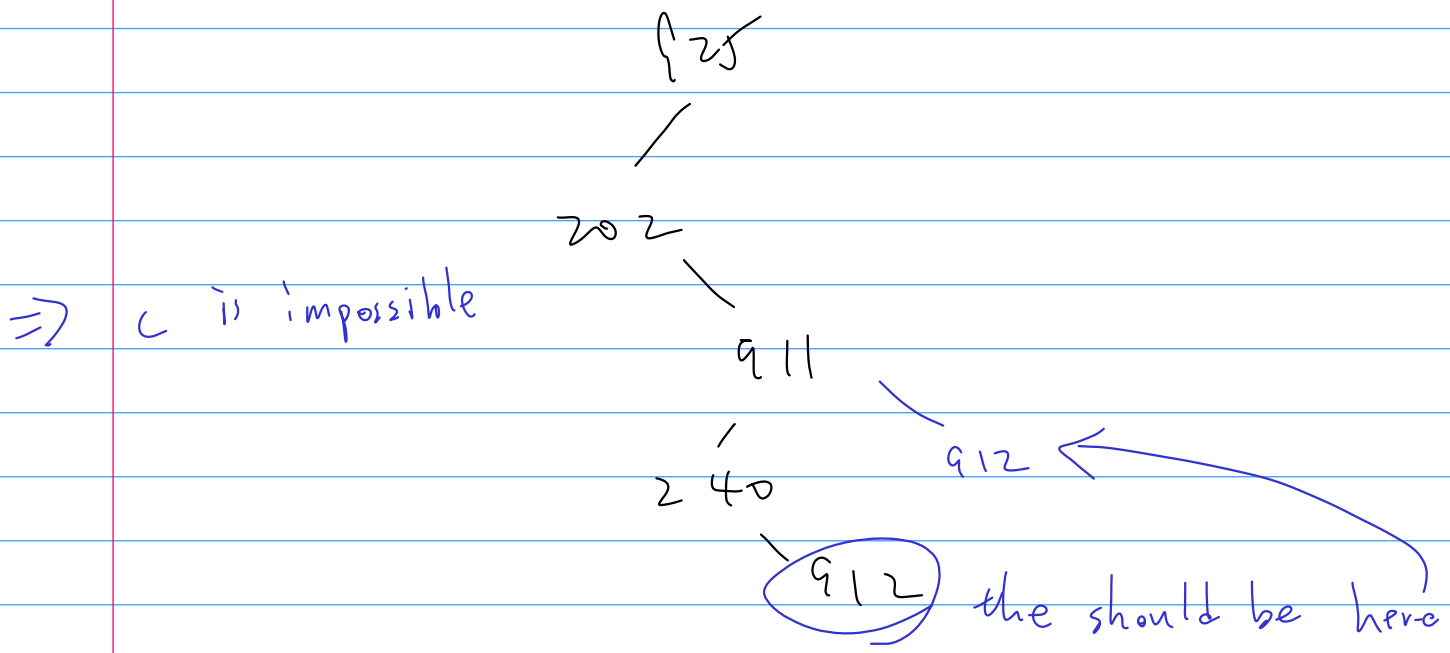
a. 2, 252, 401, 398, 330, 344, 397, 363.

b. 924, 220, 911, 244, 898, 258, 362, 363.

c. 925, 202, 911, 240, 912, 245, 363.

d. 2, 399, 387, 219, 266, 382, 381, 278, 363.

e. 935, 278, 347, 621, 299, 392, 358, 363.



⇒ c is impossible

12.2-5

Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

successor(node) = min(nodes in right subtree)
, which has no left child

predecessor(node) = max(nodes in left subtree)
, which has no right child

12.2-6

Consider a binary search tree T whose keys are distinct. Show that if the right subtree of a node x in T is empty and x has a successor y, then y is the lowest ancestor of x whose left child is also an ancestor of x. (Recall that every node is its own ancestor.)

① ∵ the right subtree of x is empty,

∴ y must be x's ancestor

if not, let z = common ancestor of x, y

⇒ $x < z < y$ ⇒ y is ≠ sucessor(x)

② if y.right is ancestor of x ⇒ $x > y$ ✗

⇒ y.left is ancestor of x

③ if y is not the lowest ancestor

and y.left is not an ancestor of x

let z be this lowest ancestor

⇒ $x < z < y$ ✗ $y = sucessor(x)$

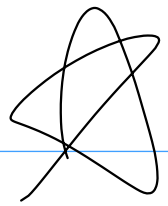
⇒ y must be the lowest ancestor of x

and y.left is an ancestor of x #

12.3-4

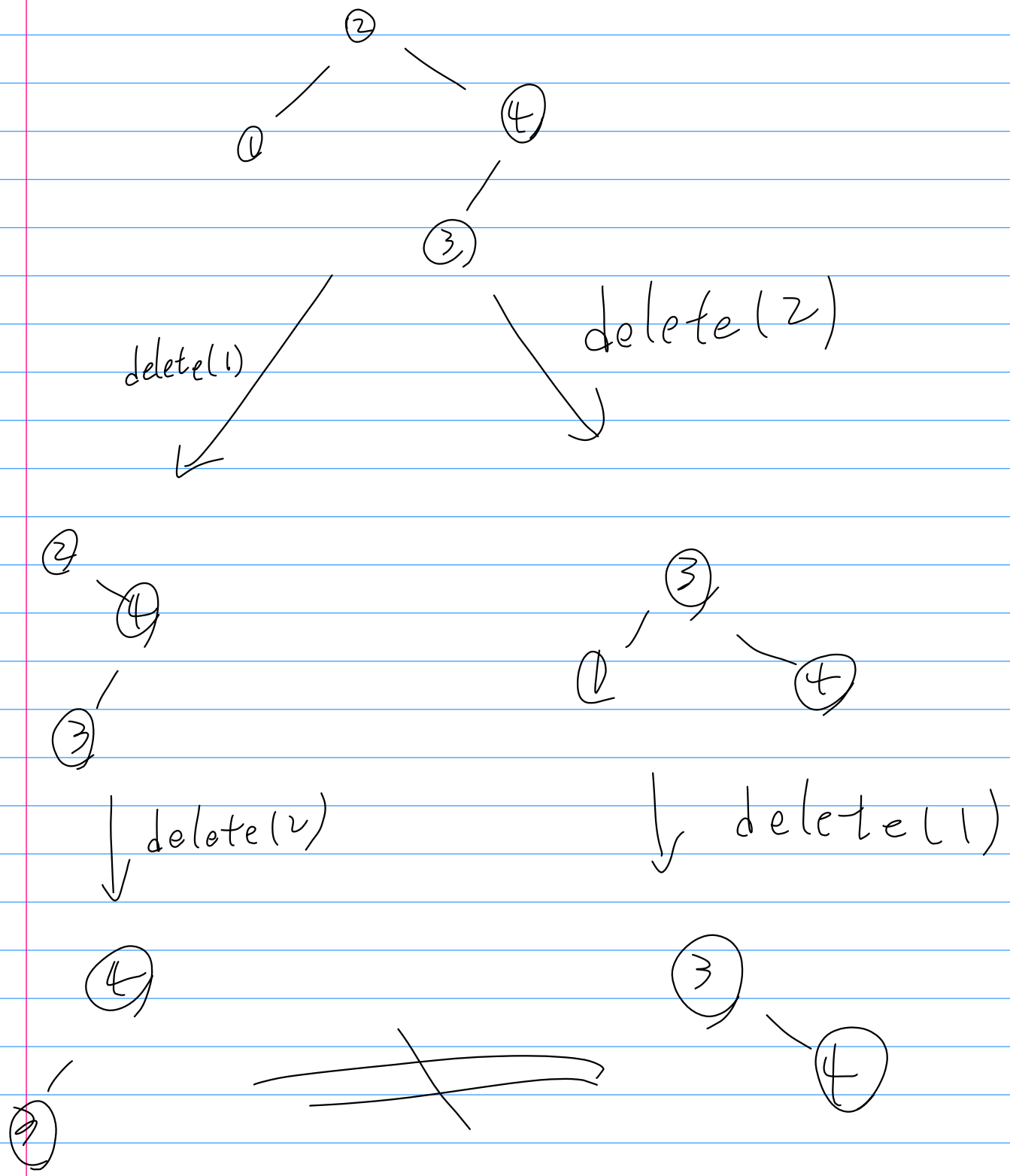
When TREE-DELETE calls TRANSPLANT, under what circumstances can the parameter v of TRANSPLANT be NIL?

z has no child



12.3-5

Is the operation of deletion "commutative" in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x? Argue why it is or give a counterexample.



$\Rightarrow$ deletion is not commute

12-1 Binary search trees with equal keys

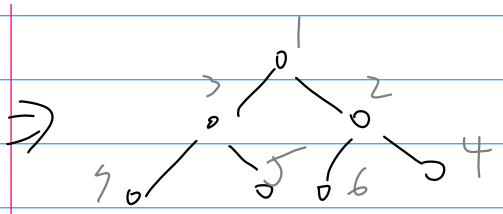
Equal keys pose a problem for the implementation of binary search trees.

a. What is the asymptotic performance of TREE-INSERT when used to insert n items with identical keys into an initially empty binary search tree?

$O(n^2)$: every item are place at the rightest end of right subtree

Consider changing TREE-INSERT to test whether $z.key = x.key$ before line 5 and to test whether $z.key = y.key$ before line 11. If equality holds, implement one of the following strategies. For each strategy, find the asymptotic performance of inserting n items with identical keys into an initially empty binary search tree. (The strategies are described for line 5, which compares the keys of z and x. Substitute y for x to arrive at the strategies for line 11.)

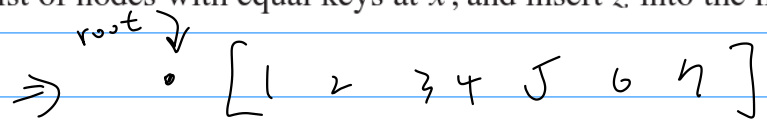
b. Keep a boolean flag $x.b$ at node x, and set x to either $x.left$ or $x.right$ based on the value of $x.b$, which alternates between FALSE and TRUE each time TREE-INSERT visits x while inserting a node with the same key as x.



$O(n \lg n)$

n items each take $\lg n$

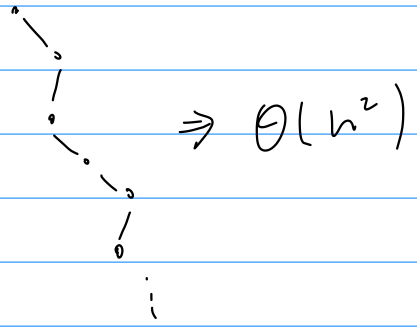
c. Keep a list of nodes with equal keys at x, and insert z into the list.



$\Rightarrow O(1)$

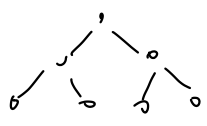
d. Randomly set x to either $x.left$ or $x.right$. (Give the worst-case performance and informally derive the expected running time.)

worst case



$\Rightarrow O(n^2)$

expect time



$\Rightarrow O(n \lg n)$