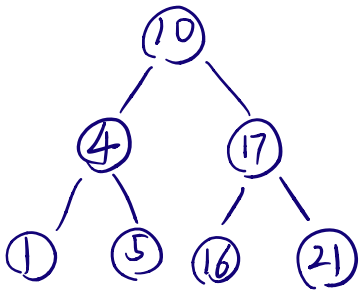
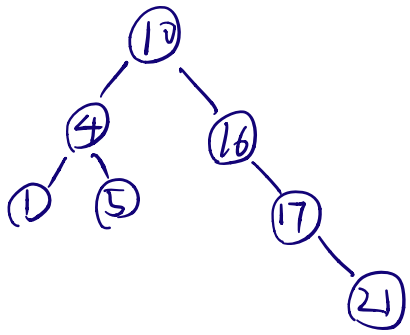
Chapter 5.

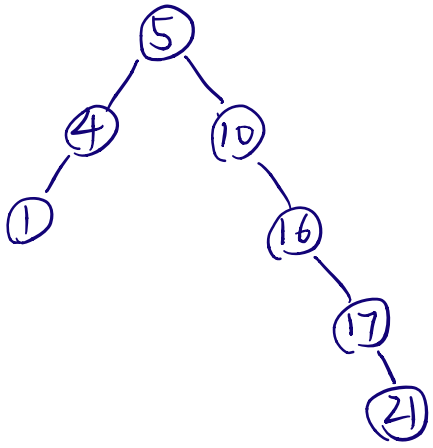
Q 1 :



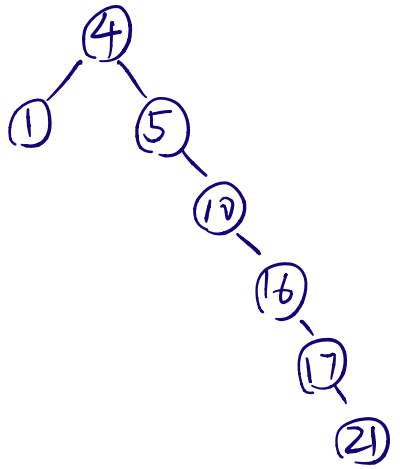
h = 2



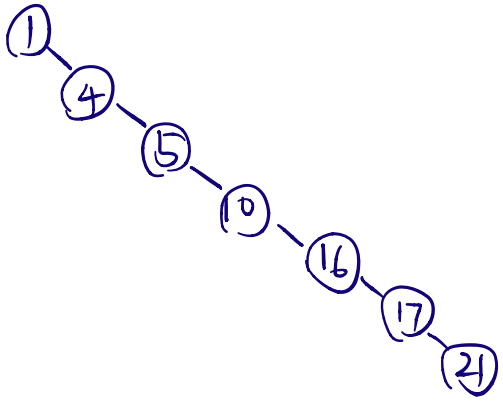
h = 3



h = 4



h = 5



h = 6

Q2.

Proof: suppose that a_{i-1}, a_i, a_{i+1} are 3 adjacent elements in an ordered array. So, a_{i-1} is the predecessor and a_{i+1} is the successor. Let a_l and a_r be the left and right children of a_i, so a_{i-1} should be the rightmost node of the subtree with root a_l, and a_{i+1} should

be the left-most node of the subtree with root a_r.

So, if a_{i-1} still has a right child, it means there exits a value between a_{i-1} and a_i, which contradicts the assumption. a_{i+1} is similar.

Hence, the successor has no left child and the predecessor has no right child.

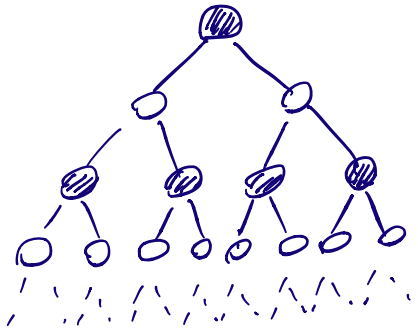
Q3.

Worst case: $O(n^2)$. Occurs when inserting results in a single chain path in the tree. Each time an element is inserted, the entire tree is iterated, and there are n elements, so the time cost is $O(n^2)$

Best case: $O(n \log n)$. Occurs when the height of BST is $\theta(\lg n)$.

Chapter 6.

Q 1.



The largest number of interval nodes occurs when black and red nodes alternates as the graph above.

The black-height is k, so that the height is 2k at most.

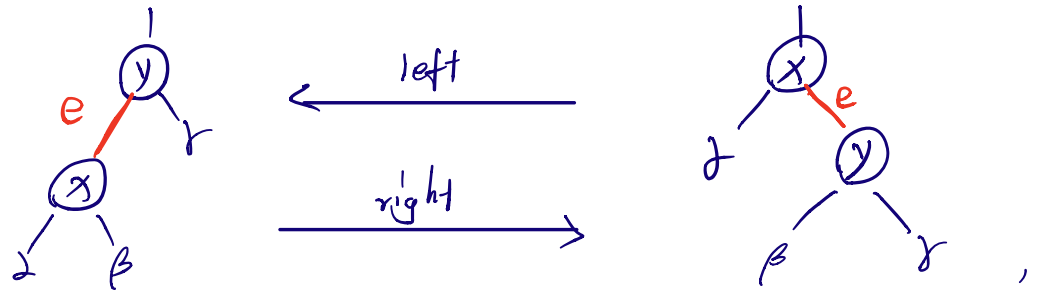
so, the number of internal nodes in total is :

$$1 + 2 + 2^2 + \cdots + 2^{2k} = \frac{1 \times (2^{2k} - 1)}{2 - 1} = 2^{2k} - 1.$$

Hence, the largest possible number is $2^{2k} - 1$, and the smallest possible number is $2^k - 1$ (according to the lemma 13.1), because it occurs when all nodes are black, and in order to meet the condition that all paths have the same black height, the tree must be a complete binary tree.

Q2. proof:

For any two node, the rotating operation can be abstracted as following:



here, α, β, γ can be null.

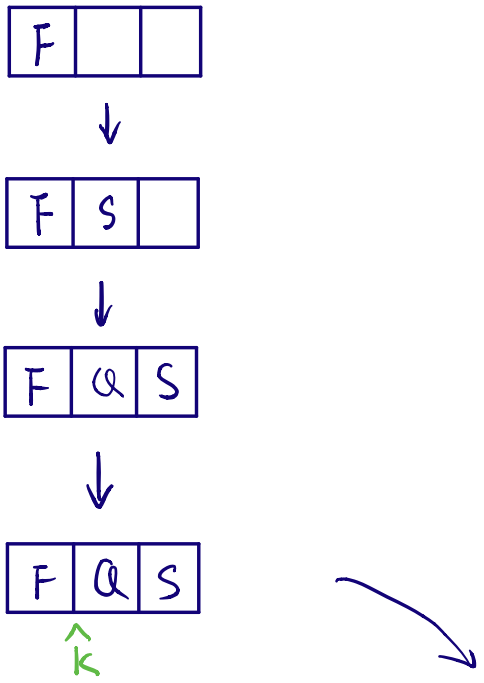
We notice that such structure can be identified uniquely by a unique edge e.

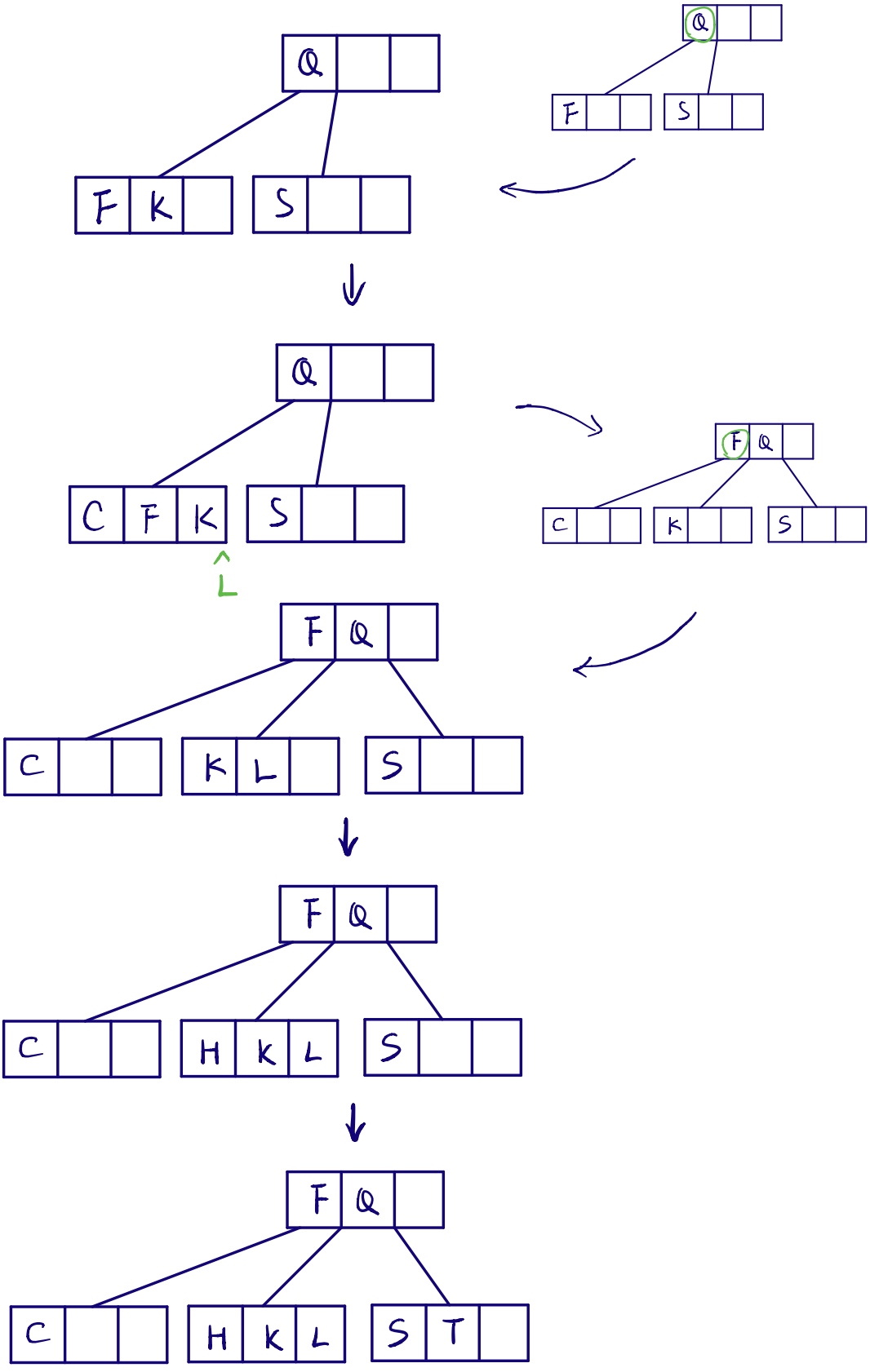
In other word, every edge can be rotated left or right, and there are n-1 edges, so there are n-1 possible rotations exactly.

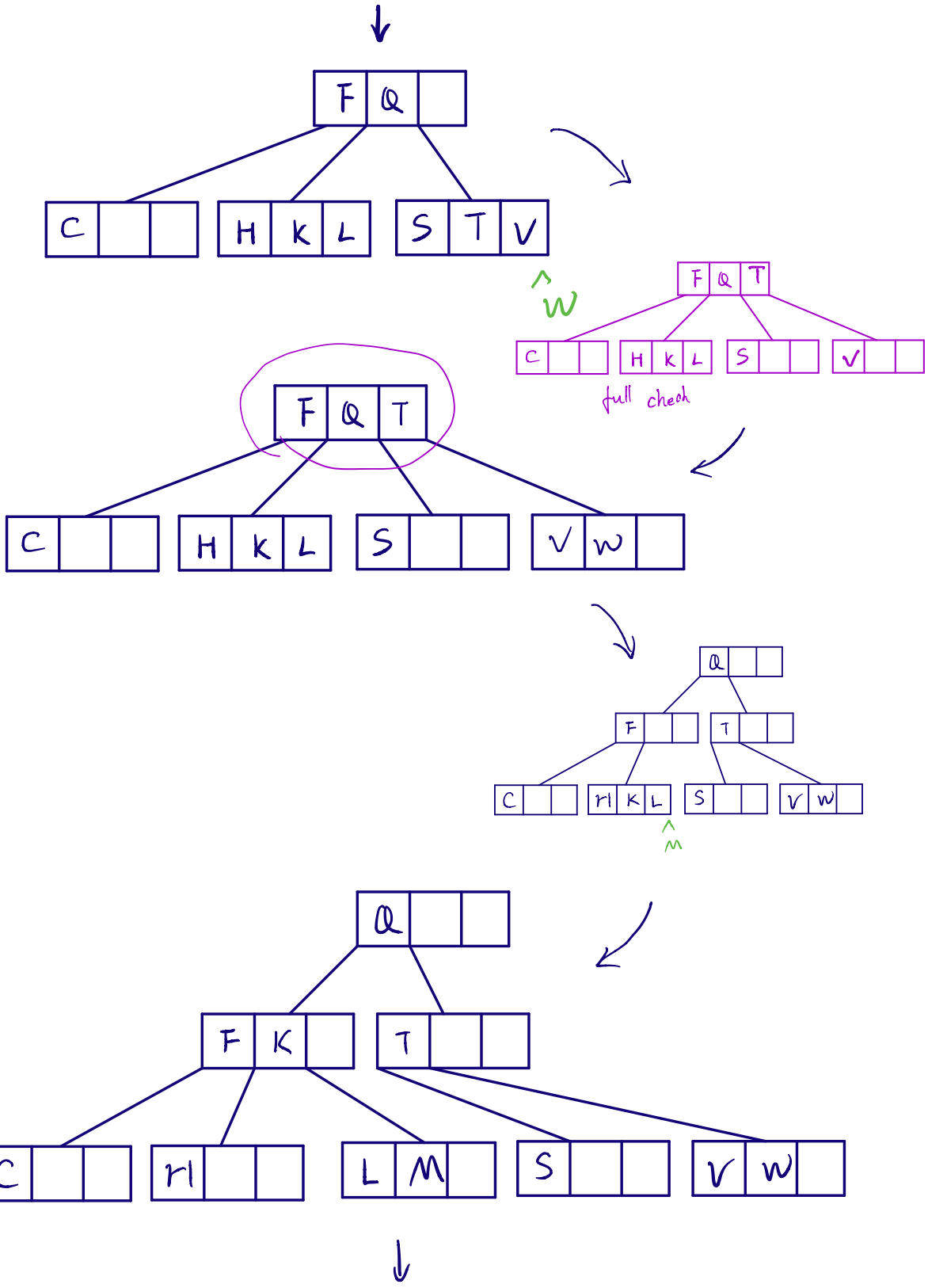
Chapter 7.

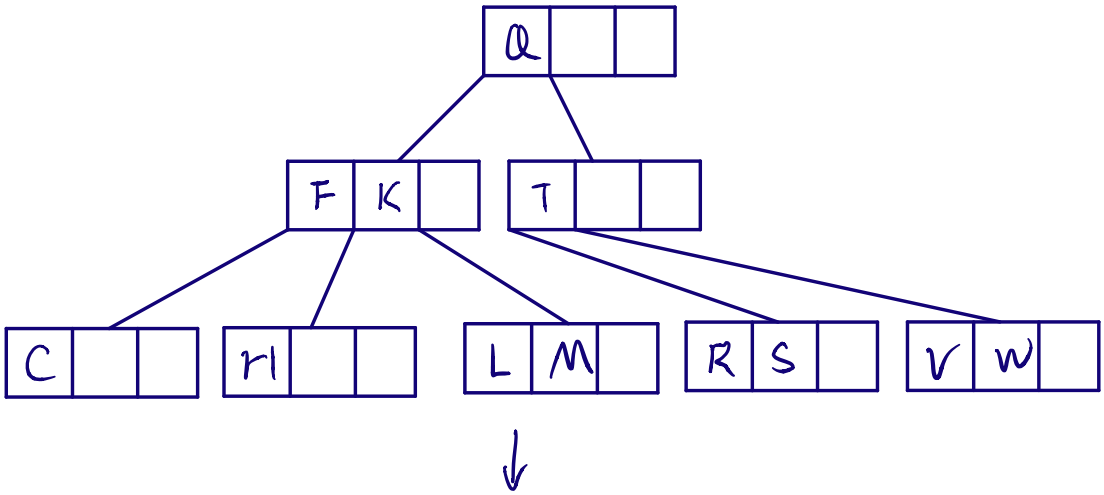
Q1 According to the definition of B tree, there are lower and upper bounds on the number of keys, which represented the minimum degree t, where $t \geq 2$. It is required that all nodes should have minimum $t-1$ keys except the root. So, if $t = 1$, the number of keys could be zero, and the B-tree would be an empty tree. Hence, let t be 2 at least to guarantee a non-empty B-tree should contains at least one key.

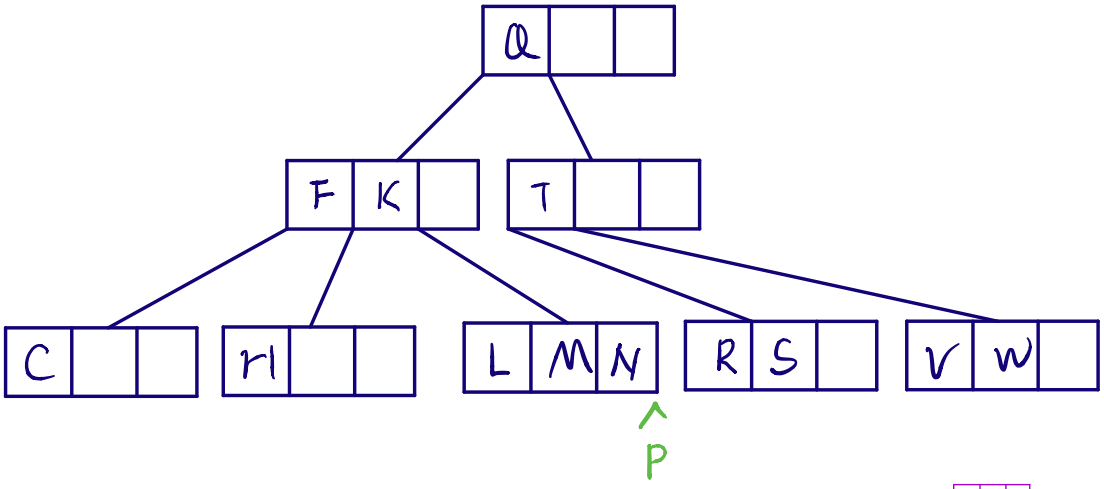
Q2.

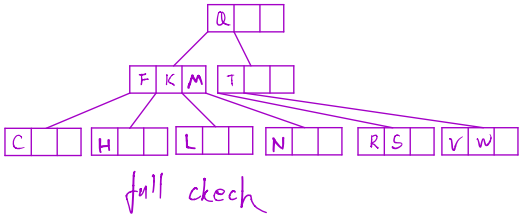


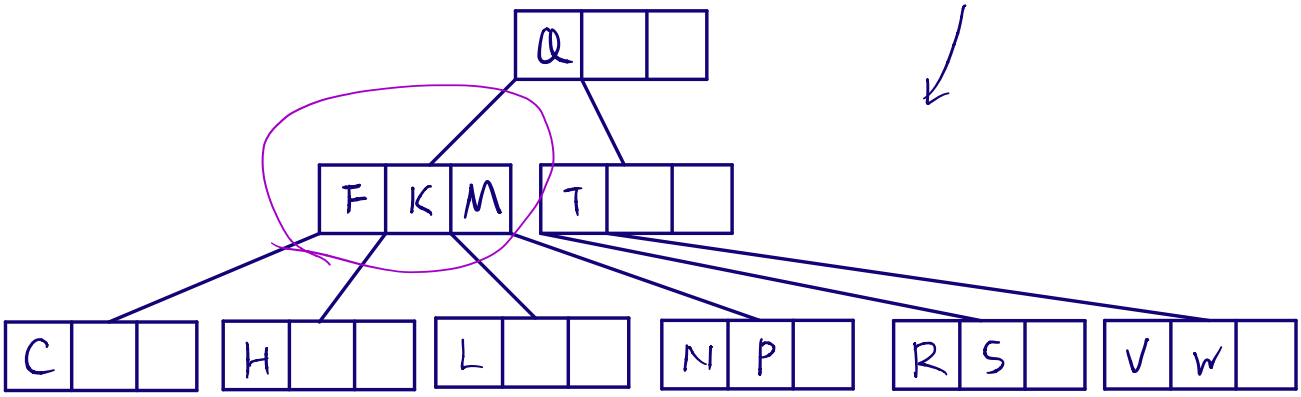

Q
F K
S
Q
F
S
Q
C F K
S
L
F Q
C
K
S
F Q
C
K L
S
F Q
C
H K L
S
F Q
C
H K L
S T

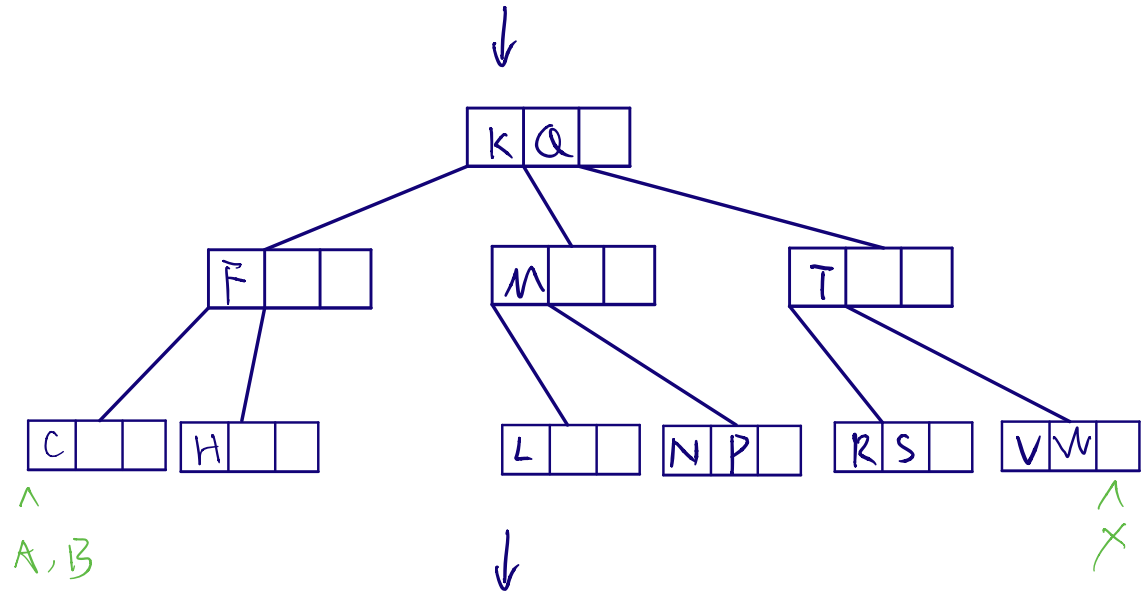

F Q
C
H K L
S T V
W
F Q T
C
H K L
S
V
full check
F Q T
C
H K L
S
V W
Q
F
T
C
M L K L
S
V W
M
Q
F K
T
C
M L
L M
S
V W

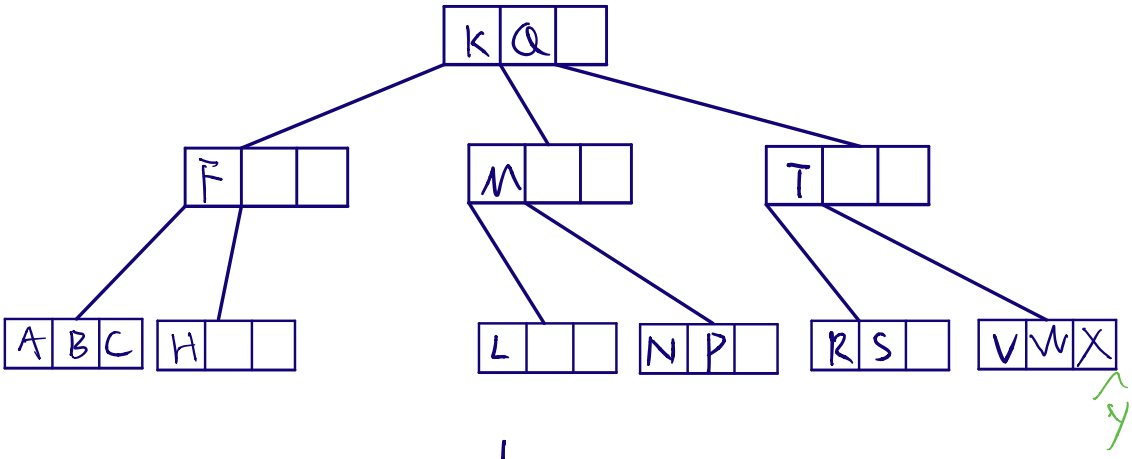

Q
F K
T
C
H
L M
R S
V W

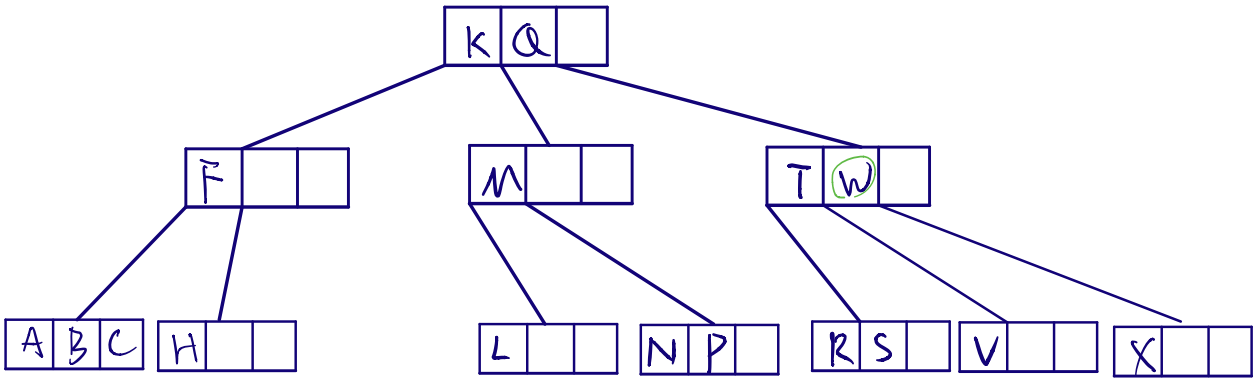

Q
F K
T
C
H
L M N
R S
V W
P

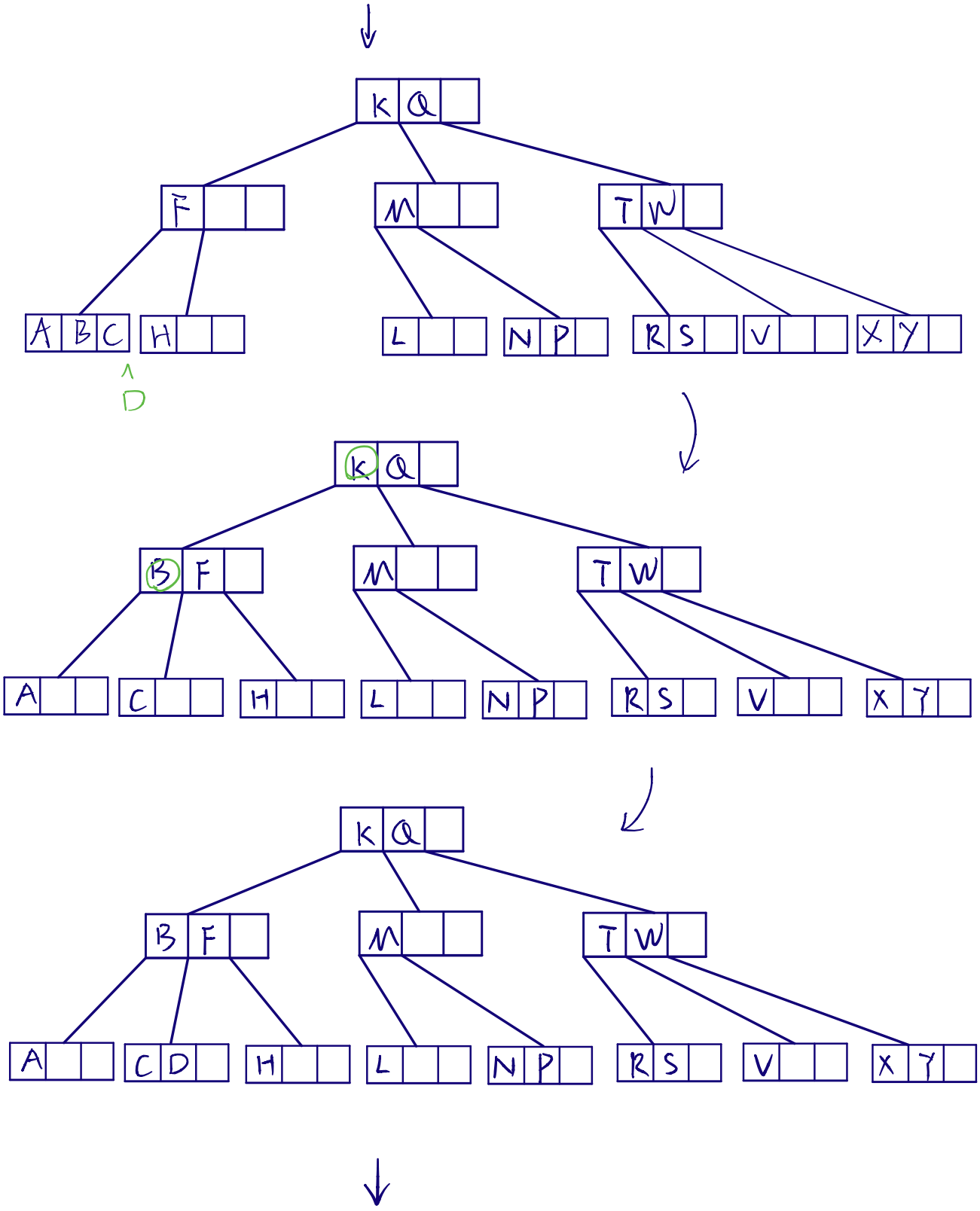

Q
F K M
T
C
H
L
N
R S
V W
full check

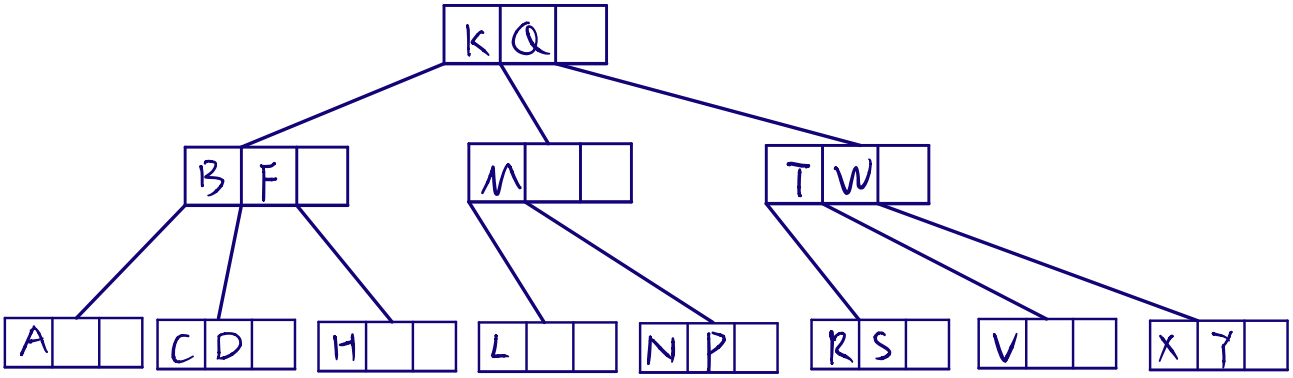

Q
F K M
T
C
H
L
N P
R S
V W

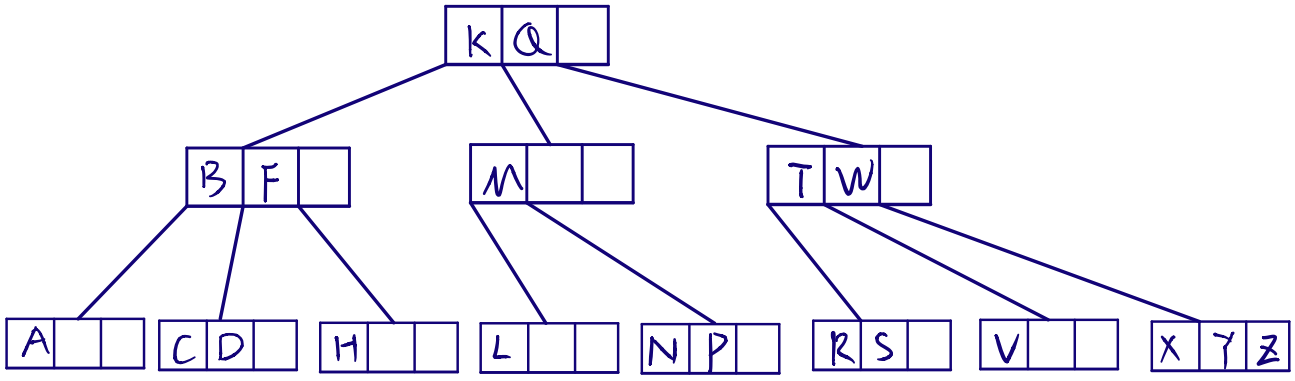

K Q
F
M
T
C
H
L
N P
R S
V W
A, B
X

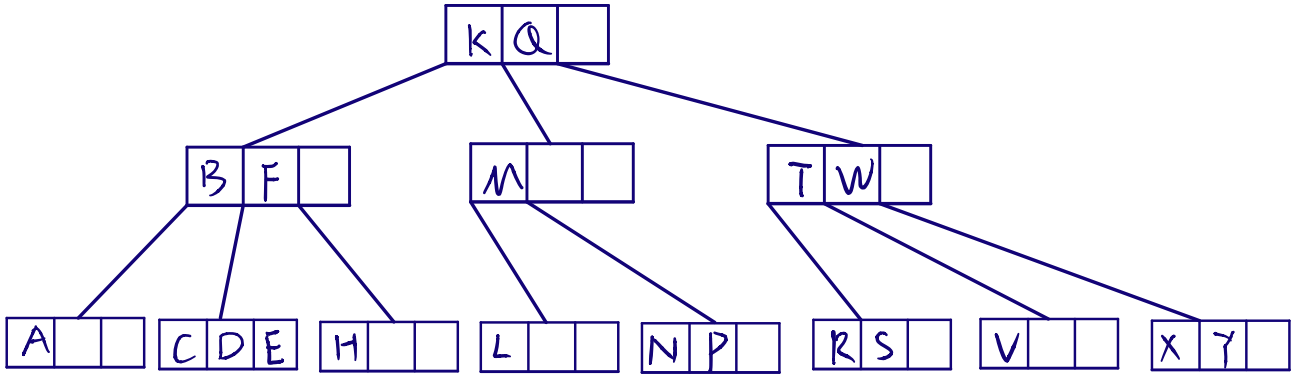

K Q
F
M
T
A B C
H
L
N P
R S
V W X
Y


K Q
F
M
T W
A B C
H
L
N P
R S
V
X


K Q
F
M
T W
A B C
H
L
N P
R S
V
X Y
D
K Q
B F
M
T W
A
C
H
L
N P
R S
V
X Y
K Q
B F
M
T W
A
C D
H
L
N P
R S
V
X Y


K Q
B F
M
T W
A
C D
H
L
N P
R S
V
X Y


K Q
B F
M
T W
A
C D
H
L
N P
R S
V
X Y Z


K Q
B F
M
T W
A
C D E
H
L
N P
R S
V
X Y

Chapter 8.

Q1

Let x be initialized to T.root, then execute the program defined as following.

```
OS-KEY-RANK(T, k)
1   x = T.root
2.  return  OS-KEY-RANK-SEARCH(x, k)
```

where the OS-KEY-RANK-SEARCH(x, k) is difined as following.

```
OS-KEY-RANK-SEARCH(x, k)
1   if x == NIL
2       return 0
3   if x.key == k
4       return x.left.size + 1
5   else if x.key < k
6       return x.left.size + 1 + OS-KEY-RANK-SEARCH(x.right, k)
7   else
8       return OS-KEY-RANK-SEARCH(x.left, k)
```

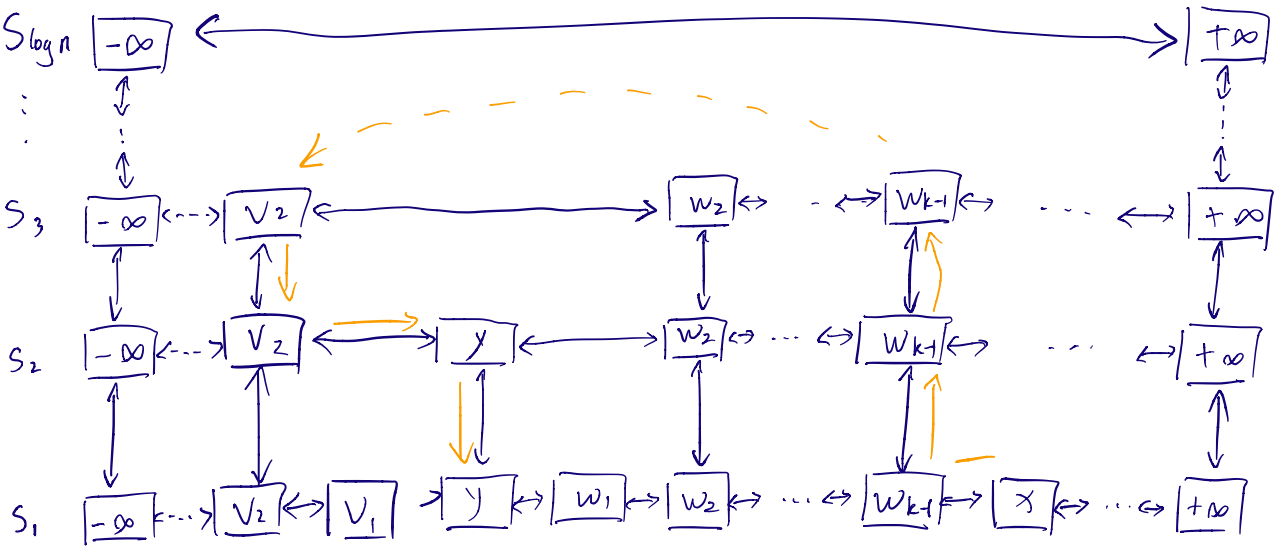
if k is not in the tree, the program will return 0, otherwise, it will return the rank of k in order.

Q2 Let x be initialized to T.root, and A is the output initialized to empty array. The function is defined as following

```
SEARCH-ALL(x, i, A)
    if x ≠ NIL and i does not overlap x.int
        A.add(x)
    if x.left ≠ NIL and x.left.max >= i.low
        SEARCH-ALL(x.left, i, A)
    if x.right ≠ NIL and x.key <= i.high
        SEARCH-ALL(x.right, i, A)
```

Analisis: the worst case is all intervals overlapping i, so the time cost of recursive search is $O(n)$. And the height of interval tree is $\log n$. when searching, there are $n-k$ nodes are pruned, so in fact, there are only k nodes being searched, so the cost is $O(k \cdot \log n)$, so, the time cost less than n and less than $k \cdot \log n$, Hence, the cost if $O(\min(n, k\log n))$.

Chapter 9.



start with the given pointer to x in level S_1, use the up pointer in order to move to a higher level until the key of current element is not larger than j, meanwhile, the next element key is larger than j.
Then, start with this node and search the element j in a smallest segment and check the next key, if the key is larger than j, down to the low level until find j.

Analisis: the part of skip list between x and y is k, so the time cost is $O(\log k)$.