Written assignment 2

1. (15') Divide and Conquer

Suppose you are given an array A[1..n] of sorted integers that has been circularly shifted k positions to the right. For example, [31,43,3,17,26,28] is a sorted array that has been circularly shifted k = 2 positions, while [26,28,31,43,3,17] has been shifted k = 4 positions. We can obviously find the largest element in A in O(n) time. Describe an O(logn) algorithm by using a divide and conquer solution.

We can use Binary Search.

```
Function FindLargest(A, n):
   low ← 1
   high ← n

   While low ≤ high:
      mid ← (low + high) // 2

      If mid < n AND A[mid] > A[mid + 1]:
         Return A[mid]

      If A[mid] ≥ A[low]:
         low ← mid + 1
      Else:
         high ← mid - 1

   Return -1
```

2. (15') Heap Sort

Sort the array {1, 3, 7, 4, 5, 6} with a maximum heap without using extra memory. Write down the content of the array every time an insert() or a deleteMax() operation completes. The initial state and the array content after the first two insertions are already written for you:

```
0: initial state
| 1 | 3 | 7 | 4 | 5 | 6 |
1: insert (1)
| 1 | 3 | 7 | 4 | 5 | 6 |
2: insert (3)
| 3 | 1 | 7 | 4 | 5 | 6 |

3: insert (7)
```

```
| 7 | 1 | 3 | 4 | 5 | 6 |
4: insert (4)
| 7 | 4 | 3 | 1 | 5 | 6 |
5: insert (5)
| 7 | 5 | 3 | 1 | 4 | 6 |
6: insert (6)
| 7 | 5 | 6 | 1 | 4 | 3 |
7: deleteMax()
| 6 | 5 | 3 | 1 | 4 | 7 |
8: deleteMax()
| 5 | 4 | 3 | 1 | 6 | 7 |
9: deleteMax()
| 4 | 1 | 3 | 5 | 6 | 7 |
10: deleteMax()
| 3 | 1 | 4 | 5 | 6 | 7 |
11: deleteMax()
| 1 | 3 | 4 | 5 | 6 | 7 |
12: deleteMax()
| 1 | 3 | 4 | 5 | 6 | 7 |
```

3. (10') Binary Search Trees

Verify the binary search tree. Given a binary tree `root` node, determine whether it is a valid binary search tree by writing the pseudo code of the function

`Boolean isBST(root, lower, upper).`

A valid binary search tree is defined as follows:

- The `left` subtree of a node only contains numbers less than the current node.
- The `right` subtree of a node only contains numbers greater than the current node.
- All left and right subtrees must also be binary search trees.

Note: The number of nodes in the tree is in the range [MIN_VALUE, MAX_VALUE], i.e., MIN_VALUE <= `Node.key` <= MAX_VALUE. In the function `isBST(root, lower, upper)`, `lower` and `upper` are the two boundaries of the `keys` of a valid BST subtree.

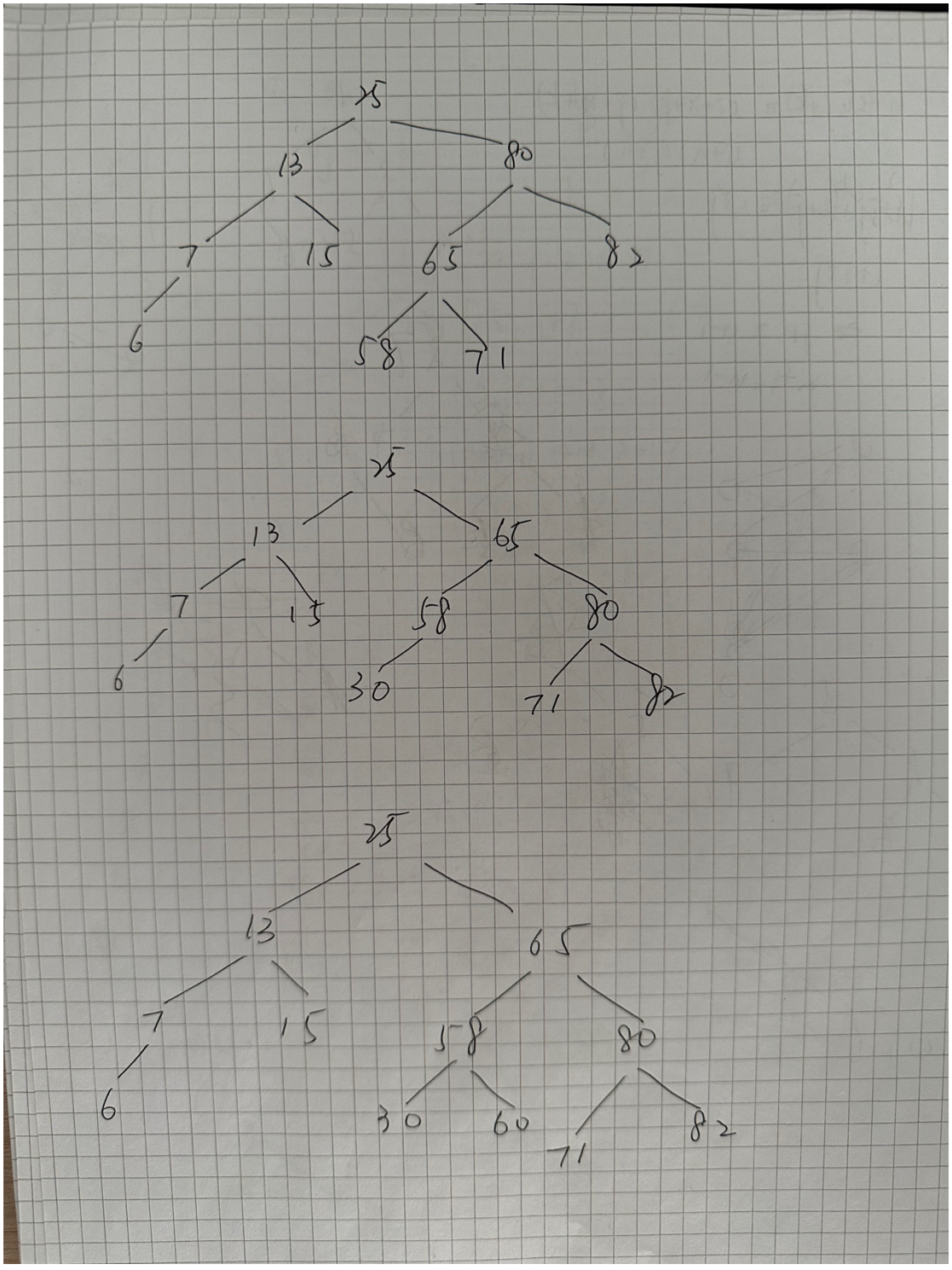
```
Function isBST(root, lower, upper):
   If root is NULL:
      Return True

   If root.key ≤ lower OR root.key ≥ upper:
      Return False
```

```
Return isBST(root.left, lower, root.key) AND
       isBST(root.right, root.key, upper)
```

4. (15') AVL Trees

Construct an AVL tree by inserting the input array {13, 7, 25, 58, 80, 15, 82, 6, 65, 71, 30, 60}. Draw the three trees after inserting the last three elements: 71, 30, and 60, respectively.



5. (30') B+ Trees

5.1. *(10')* Suppose you are managing employee records on a computer with the following setting:

- Computer hard disk access is block-based and the size of one block is *2048 bytes*.
- The size of each employee record is *256 bytes* (including the primary key).
- The primary key for an employee record is of type *long* `long` *(16 bytes).*

- The size of a pointer is 8 bytes.

You decide to store the data using a B+ tree. Propose the best setting for *M* and *L*. Show the steps that lead to your proposal.

Note: The definition of *M* and *L is* as described in the lecture slides.

$$MP + (M-1)K \leq BlockSize$$

$P = 8, K = 16, M \leq \frac{2064}{24} = 86$

$$LR + P \leq BlockSize$$

$R = 256, P = 8, L \leq \frac{2040}{256} = 7.96875$

Thus $M = 86, L = 7$

5.2. *(20')* In this question, we use a B+ tree with M=L=4. The initial B+ tree is shown in Figure 1.

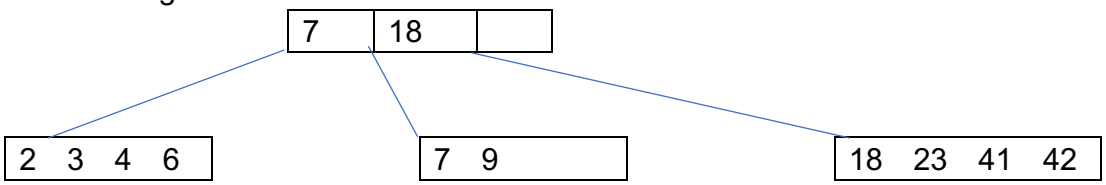
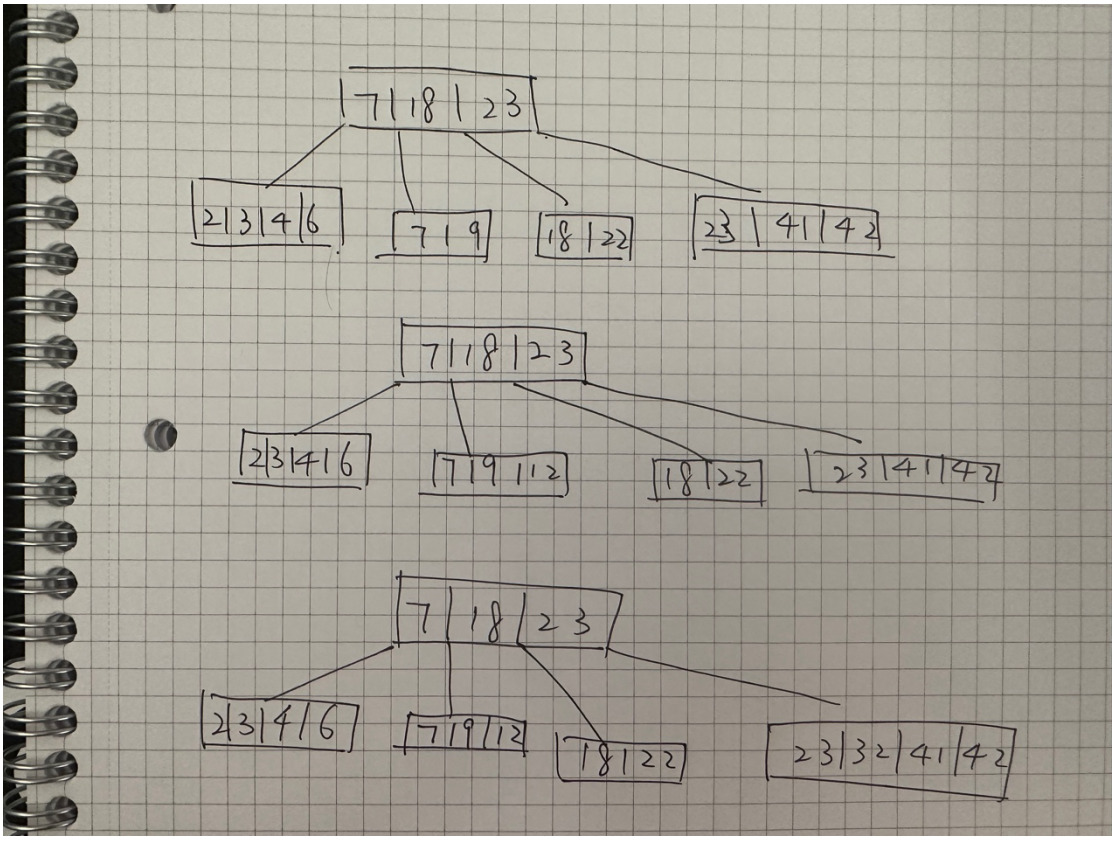


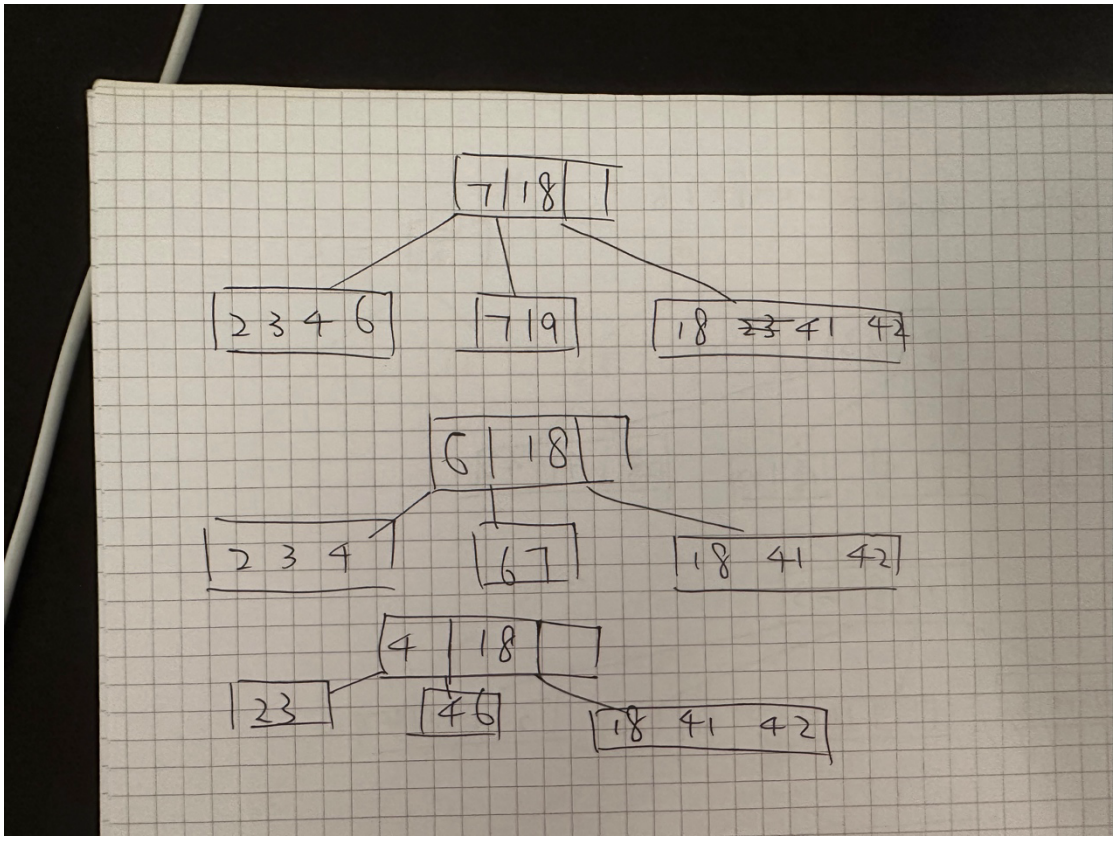
Figure 1: The initial B+ Tree

a) *(10')* Given an insertion sequence *{22, 12, 32}*, draw the three B+ trees after each insertion, respectively. Please start from the initial B+ tree shown in Figure 1.



b) *(10')* Given a deletion sequence *{ 23, 9, 7}*, draw the three B+ trees after each deletion. Please start from the initial B+ tree shown in Figure 1.

Note: please strictly follow the lecture notes when you do the operations.


7 18
2 3 4 6
7 9
18 41 42
6 18
2 3 4
6 7
18 41 42
4 18
23
4 6
18 41 42

6. (15') Graph

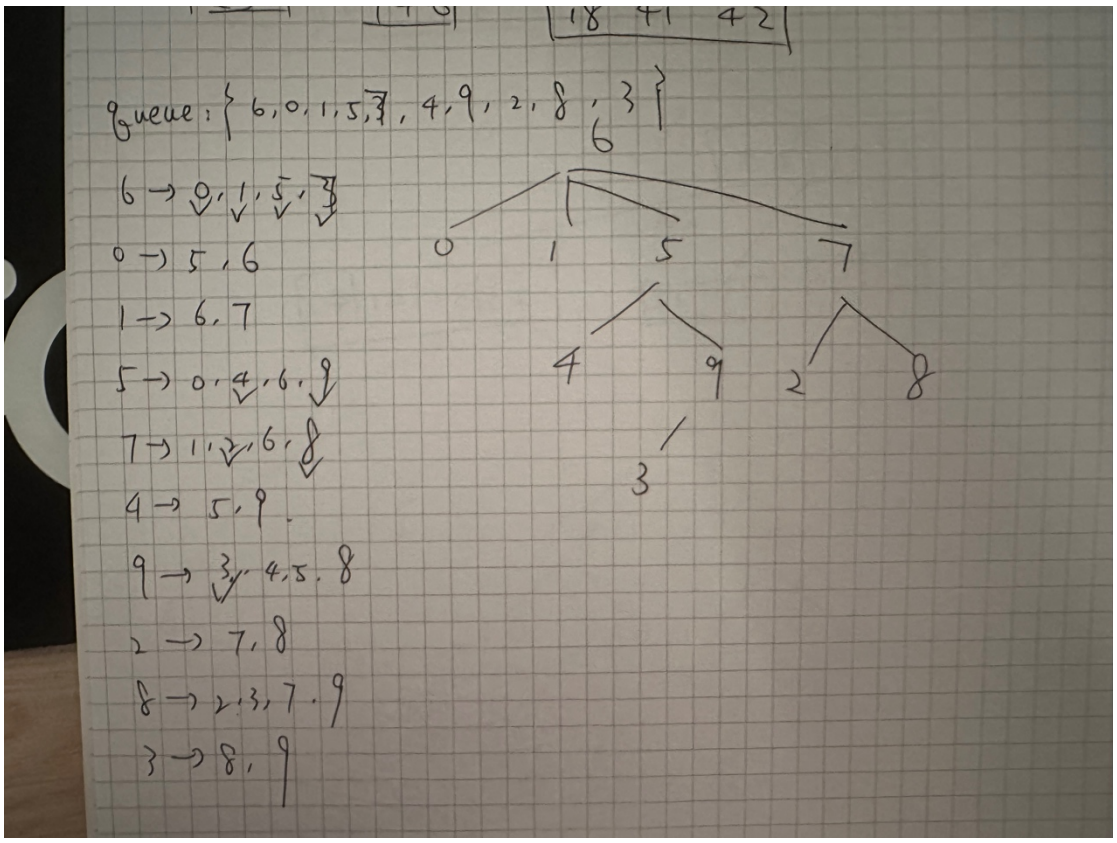
Given a graph as shown in Figure 2. Imagine that node 6 is the vertex:

a) *(2')* Write out its adjacency matrix.
b) *(3')* Write out its adjacency List.
c) *(10')* Use the algorithm of "BFS + Path Finding" (as shown in Lecture 15, page 3) to work out the BFS tree step by step.

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	0	0	1	1	0	0
2	0	0	0	0	0	0	0	1	1	0
3	0	0	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0	0	1
5	1	0	0	0	1	0	1	0	0	1
6	1	1	0	0	0	1	0	1	0	0
7	0	1	1	0	0	0	1	0	1	0
8	0	0	1	1	0	0	0	1	0	1
9	0	0	0	1	1	1	0	0	1	0

0: [5, 6]
1: [2, 6, 7]
2: [1, 7, 8]
3: [4, 8, 9]
4: [3, 5, 9]
5: [0, 4, 6, 9]
6: [0, 1, 5, 7, 9]
7: [1, 2, 6, 8]
8: [2, 3, 7, 9]
9: [3, 4, 5, 6, 8]

(c)


queue: {6, 0, 1, 5, 7, 4, 9, 2, 8, 3}
6 → 0, 1, 5, 7
0 → 5, 6
1 → 6, 7
5 → 0, 4, 6, 9
7 → 1, 2, 6, 8
4 → 5, 9
9 → 3, 4, 5, 8
2 → 7, 8
8 → 2, 3, 7, 9
3 → 8, 9
6
0
1
5
7
4
9
2
8
3

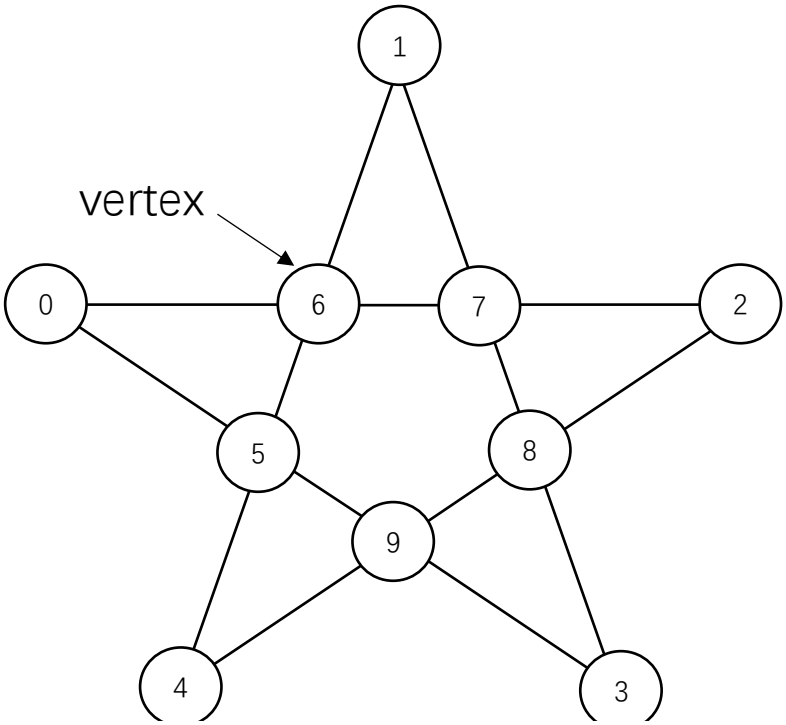


Figure 2 The initial graph