COMP 2119A: Solution for Assignment 3

1

Answer:
a)see Figure1(Next page)

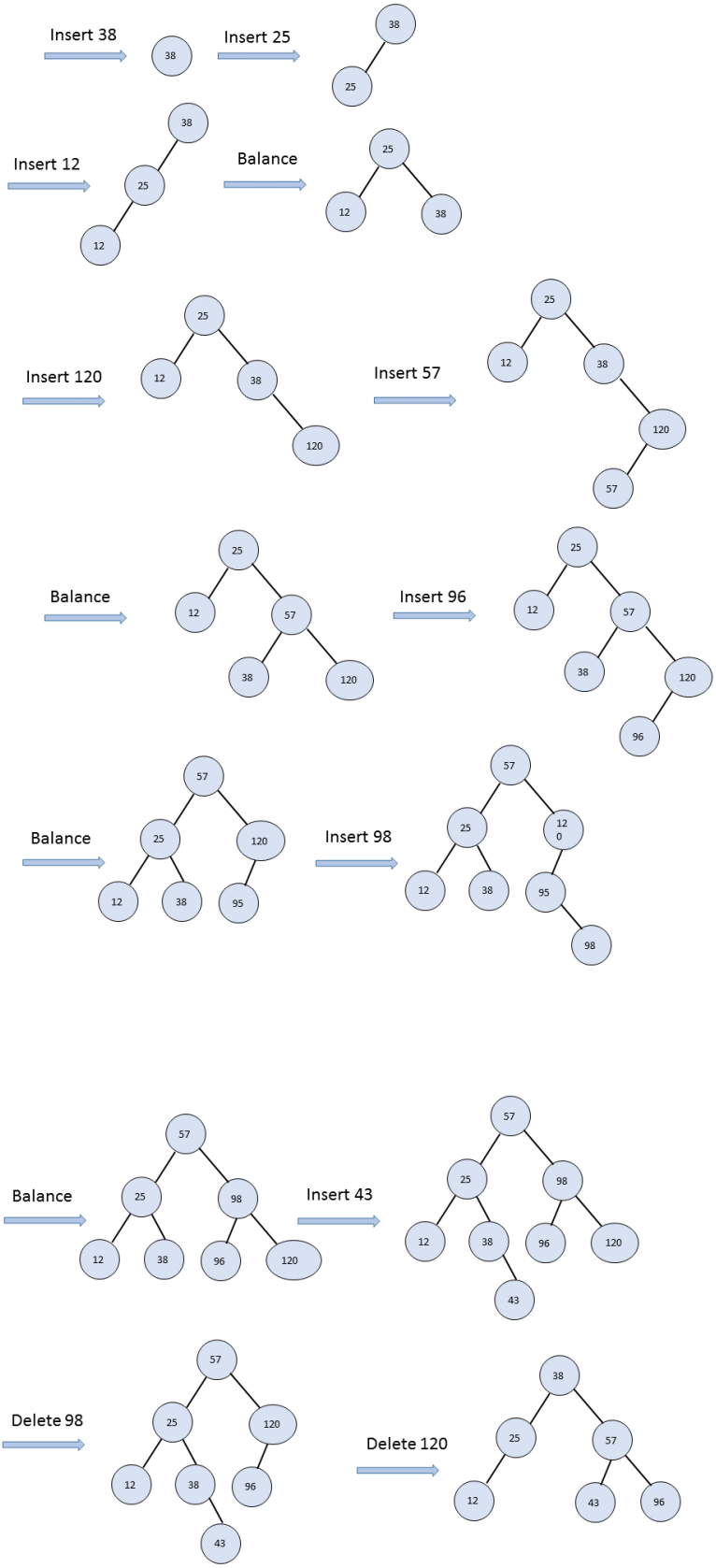


Figure 1: question 1

b) 1. First we give out an algorithm with time complexity $O(n \log k)$. To begin with, we can build a binary search tree that has size at most k given n numbers array A[n]. Specifically, we create an AVL tree with tree node:

```
struct node {
element e;
int b; // b is the balance factor
int count; // number of the element e
node * left;
node * right;
node * p;
}
```

For each insertion A[i], we first search the tree for A[i]. If the return node is not NULL, then update count to count+1 in the node. Otherwise, insert a new node with element A[i] and count 0 to the tree.

```
buildAVL( A[n]){
root = new AVLTree();
for ( i = 1; i ≤ n; ++ i )
node * nPtr = root.search(A[i]);
if(nPtr != NULL){
nPtr → count += 1;
}else{
root.insert(A[i]);}
return root;
}
```

After we build the AVL tree with the given n numbers, we only have to traverse the tree and search the element that has the largest count, which is the highest frequency that required.

```
getHighFren(Tree T){
int hf = 0; //highest frequency is initialized as 0;
element val; //value of the current highest frequency element;
for each node in T{
if(node.count > hf){
hf = node.count;
val = node.element;
}
}
return (hf, val);
}
    main(A[n]){
AVLTree = buildAVL(A[n]);
(hf, val) = getHighFren(AVLTree);
}
```

Complexity Analysis: It is easy to see that the size of the AVL tree is at most k since there are only k distinct values. In buildAVL, the cost is $O(n \log k)$(at most $O(\log k)$ for each element insertion), and the second step will run in $O(k)$ time. Therefore, the total time complexity is $O(n \log k) + O(k) = O(n \log k)$.

2. You can also design a O(n) time complexity with O(k) space complexity algorithm assume that there is cryptography hash function h. The idea is to record the count of each distinct number by setting

map(A[i]) = $count_i$.

main(A[n]){
Initialize each count as 0
for each number A[i]:
map(A[i]) = $count_i$ +1
hf=0
for each number A[i]:
if map(A[i]) > hf :
hf = map(A[i])
return hf
}

2

a)One case for one time comparison; Two cases for two comparisons.
b)2^{i-1}
c)$\sum_{i=1}^{k} \frac{i*2^{i-1}}{n} = O(\log n)$

3

(a) bubble sort
157 122 148 188 136 167 31 138 192 126 202
122 148 157 136 167 31 138 188 126 192 202
122 148 136 157 31 138 167 126 188 192 202
122 136 148 31 138 157 126 167 188 192 202
122 136 31 138 148 126 157 167 188 192 202
122 31 136 138 126 148 157 167 188 192 202
31 122 136 126 138 148 157 167 188 192 202
31 122 126 136 138 148 157 167 188 192 202
31 122 126 136 138 148 157 167 188 192 202
31 122 126 136 138 148 157 167 188 192 202

(b) insertion sort
157 188 122 148 192 136 167 31 138 202 126
122 157 188 148 192 136 167 31 138 202 126
122 148 157 188 192 136 167 31 138 202 126
122 148 157 188 192 136 167 31 138 202 126
122 136 148 157 188 192 167 31 138 202 126
122 136 148 157 167 188 192 31 138 202 126
31 122 136 148 157 167 188 192 138 202 126
31 122 136 138 148 157 167 188 192 202 126
31 122 136 138 148 157 167 188 192 202 126
31 122 126 136 138 148 157 167 188 192 202

(c) selection sort

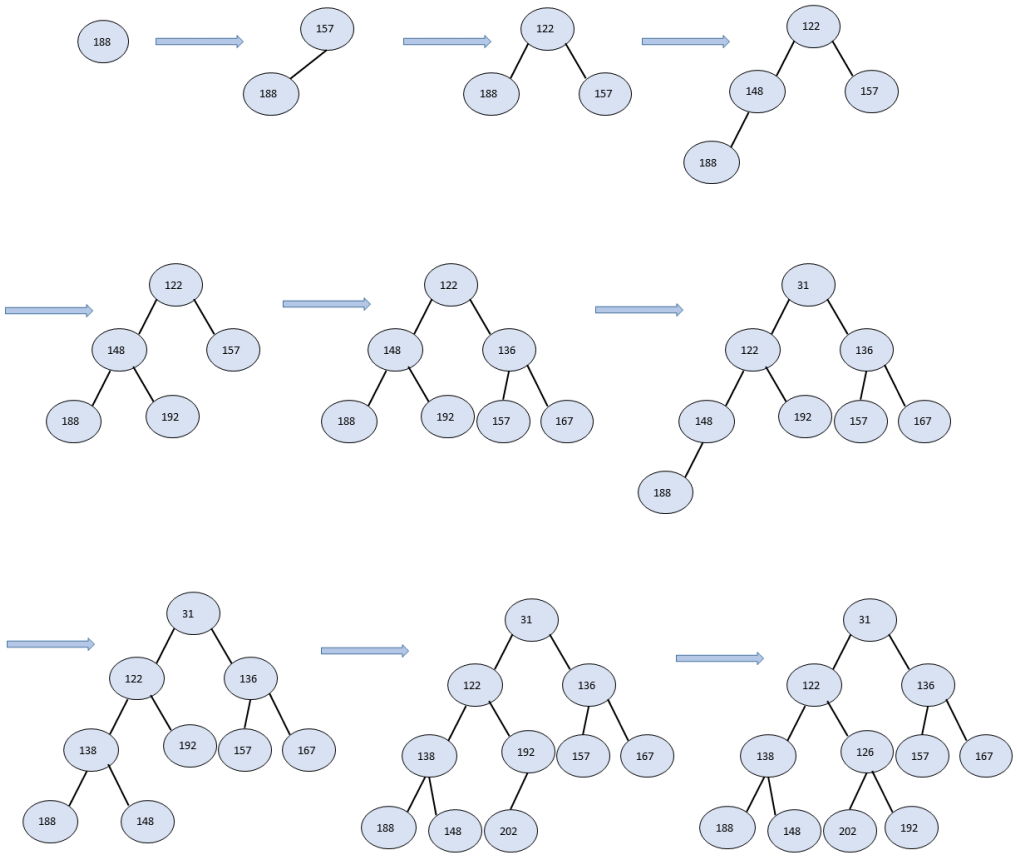
31 157 122 148 192 136 167 188 138 202 126
31 122 157 148 192 136 167 188 138 202 126
31 122 126 148 192 136 167 188 138 202 157
31 122 126 136 192 148 167 188 138 202 157
31 122 126 136 138 148 167 188 192 202 157
31 122 126 136 138 148 167 188 192 202 157
31 122 126 136 138 148 157 188 192 202 167
31 122 126 136 138 148 157 167 192 202 188
31 122 126 136 138 148 157 167 188 202 192
31 122 126 136 138 148 157 167 188 192 202

(d) quick sort

122 31 126 148 192 136 167 157 138 202 188
122 31 126 148 136 167 157 138 188 202 192
122 31 126 148 136 167 157 138 188 192 202
122 31 126 136 138 167 157 148 188 192 202
122 31 126 136 138 148 157 167 188 192 202
122 31 126 136 138 148 157 167 188 192 202
31 122 126 136 138 148 157 167 188 192 202

4

a)see Figure2



Extract 31

Extract 122

Figure 2: question 4

b)Answer: Given n distinct numbers, we can firstly build a min-heap bottom-up with complexity $O(n)$, then execute k times of Extract-Min which run in $O(\log n)$ time for each Extract-Min including the Min-heapify operation. To sum up, the total time to retrieve the k smallest numbers if $O(n + k \log n)$ time.

5

a)False
see Figure3

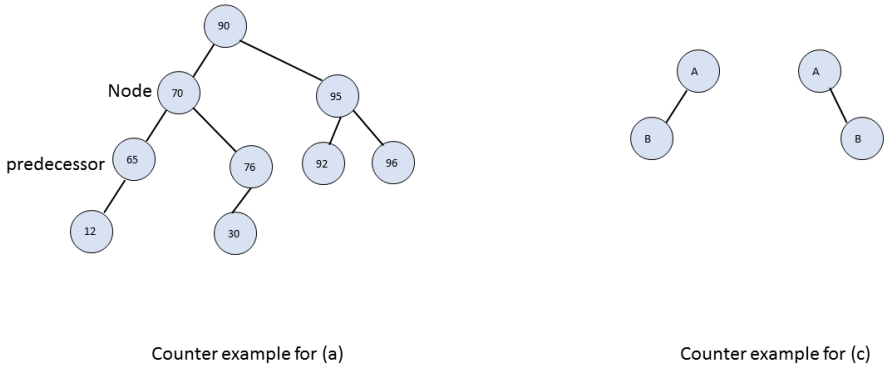


Figure 3: question 5

b)True
If node u has two children, its predecessor would be the right most of its left subtree. Suppose v is the predecessor of node u, and v_r is the right child of v. So v_r must have the property that $v < v_r < u$. Then v_r should be the predecessor of u, which contradicts to the statement that v is the predecessor of u.

c)False
see figure3