

Data Structures and Algorithms  
INFO 6205, Wed  
Homework 6  
Due: October 19, 2019

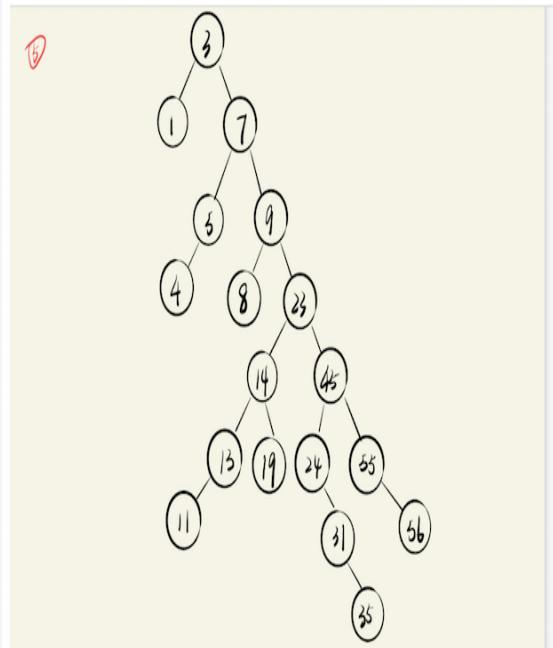
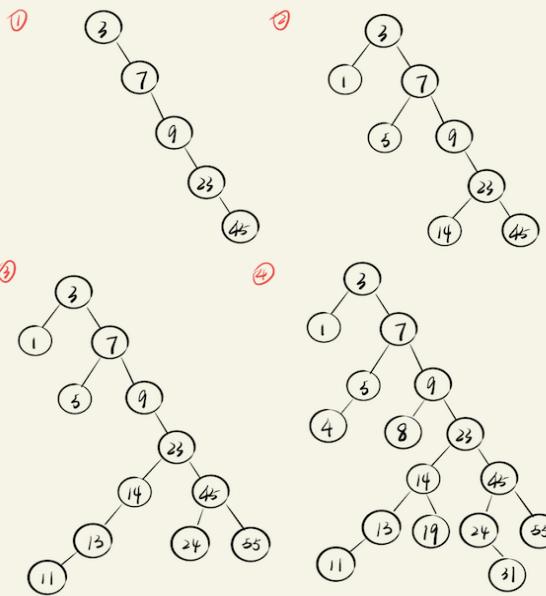
**Name: Qian Cai**  
**NU ID:001389278**

Put all your java, compiled class files and documentation files into a zip file named Homework6.zip and submit it via the drop box on the blackboard before the END of due date. Put your name on all .java files. There will be a short quiz on this homework.

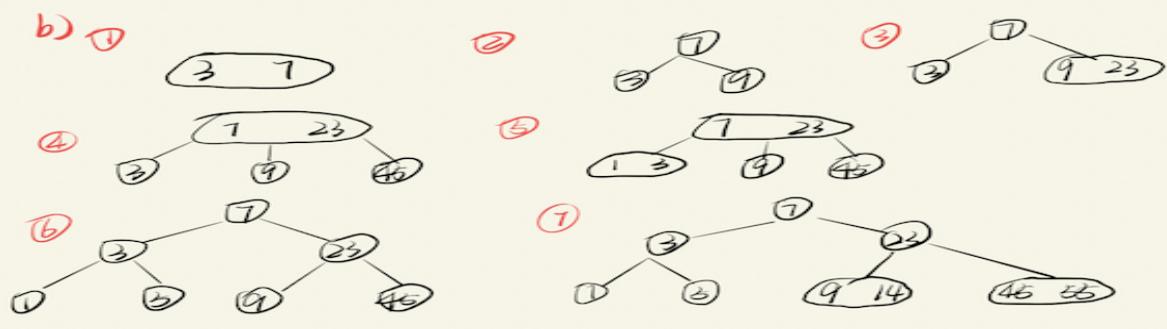
1. What is the Balanced Tree, Complete Tree and Non-Complete Tree?  
**A balanced tree is a tree where every leaf is “not more than a certain distance” away from the root than any other leaf.**  
**A complete tree is a tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.**  
**Non-complete tree is tree in which every level is not completely filled and all nodes are not as far left as possible.**
2. Consider following data: {3,7,9,23,45,1,5,14,55,24,13,11,8,19,4,31,35,56}
  - a) Construct Binary Tree
  - b) Construct 2-3 Tree
  - c) Construct 2-3-4 Tree
  - d) Construct Binary Heap Tree
  - e) What is Time complexity of each case, Why would you use one versus the other?
  - f) Insert 17, 22, 32, 6, 33 in (b)
  - g) Delete 13 in (a) and (b)
  - h) What is the Height of (a), (b), (c)?
  - i) Write Java Search and Insert code for (a) and (b)
  - j) Write Java code for DeleteMin() and DeleteMax Algorithms for (a), provide example

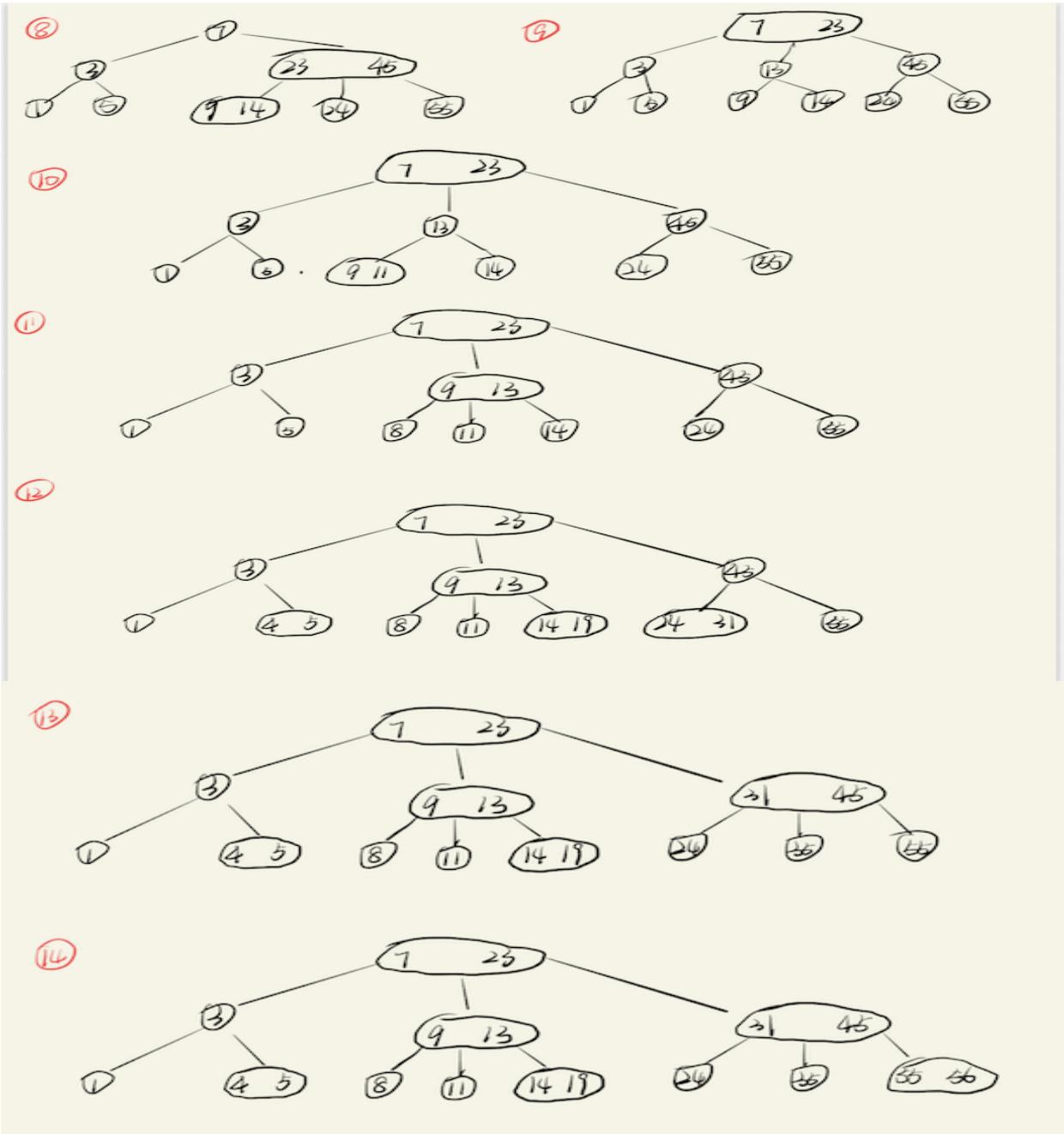
**a)**

a)  $\{3, 7, 9, 23, 45, 1, 5, 14, 65, 24, 13, 11, 8, 19, 4, 31, 35, 56\}$

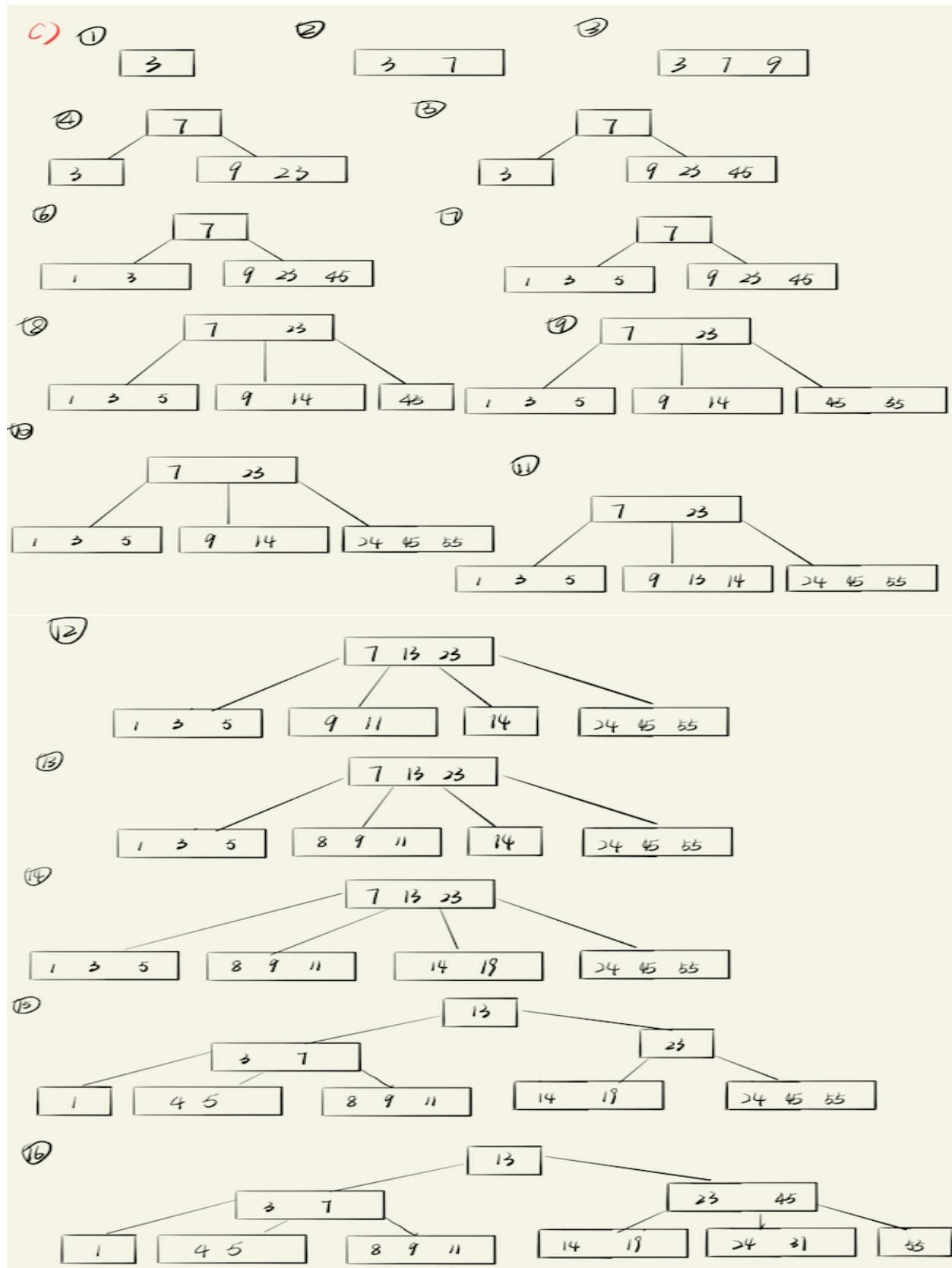


b)

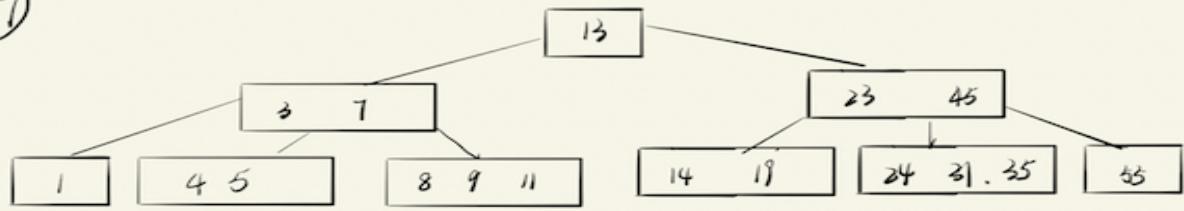




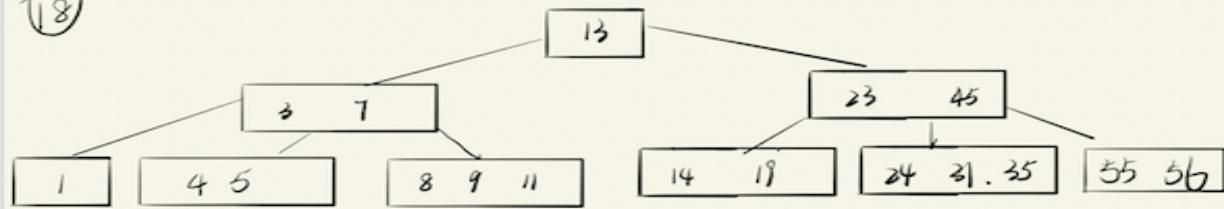
c)



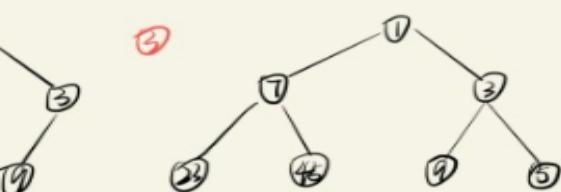
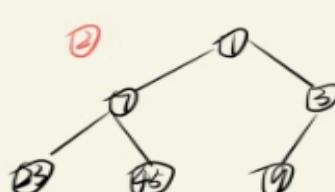
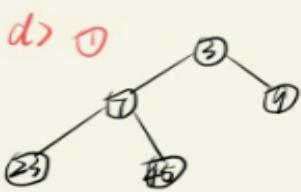
17



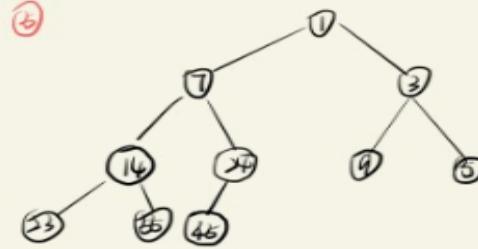
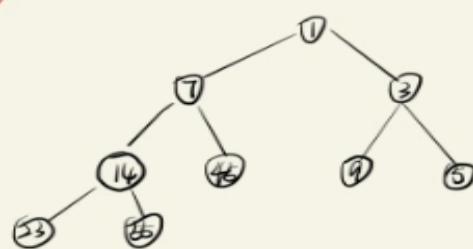
18



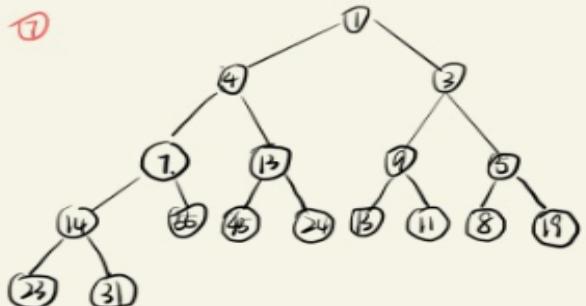
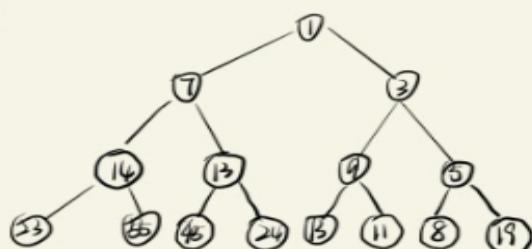
d)



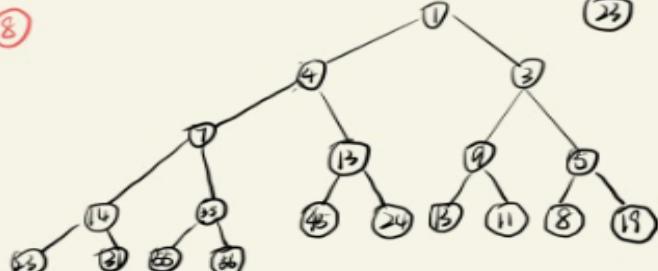
④



⑥



⑧



e)

Time complexity of Binary Tree in worst case is  $O(n)$ , if it is balanced, the time complexity in average case is  $O(\log n)$  when it is inserted, search, delete.

Time complexity of 2-3 Tree and 2-3-4 Tree in worst case is  $O(\log n)$ , the time complexity in average case is  $O(\log n)$  when it is inserted, search, delete.

Time complexity of Binary Heap Tree in worst case is  $O(\log n)$ , the time complexity in average case is  $O(1)$  when it is inserted.

Search: worst case and average case are both  $O(n)$ .

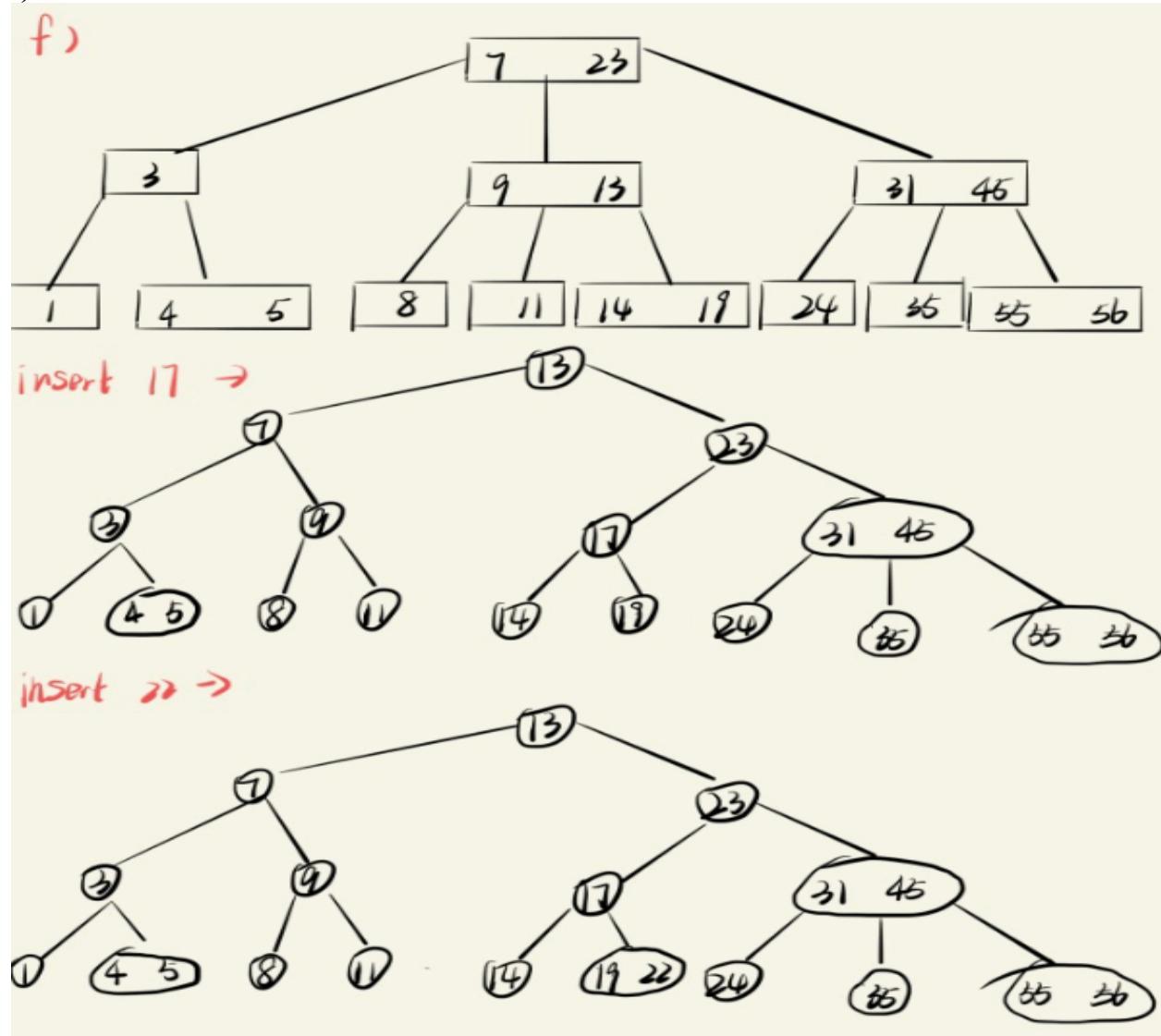
delete: worst case and average case are both  $O(\log n)$ .

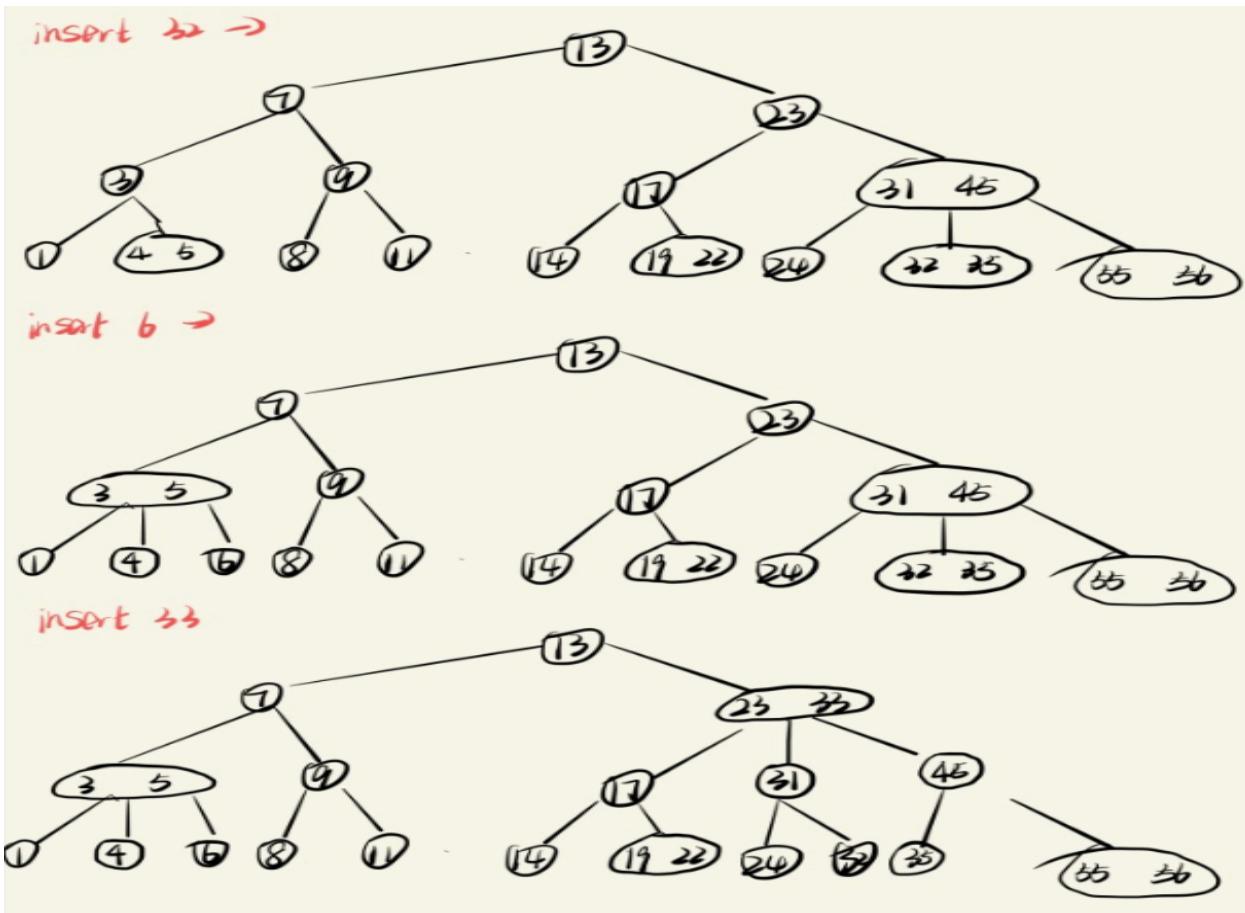
2-3 Tree and 2-3-4 tree is stable and suitable for the large amount of data because the time complexity in worst case or in average case are always  $O(\log n)$  when insertion, delete, search.

Binary Tree is unstable and lowest.

Binary Heap Tree is fastest when we insert data, it is easy to find max data or min value.

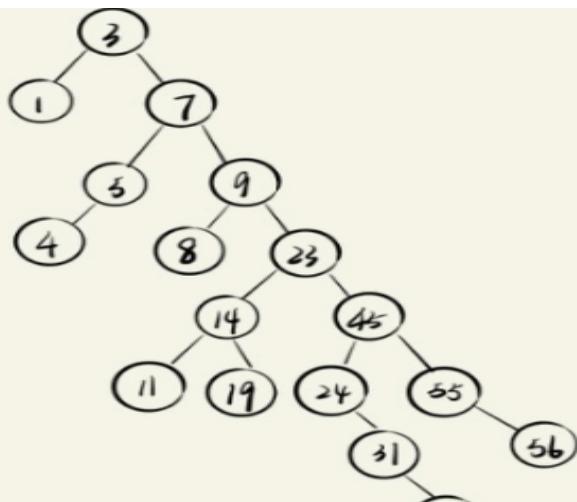
f)



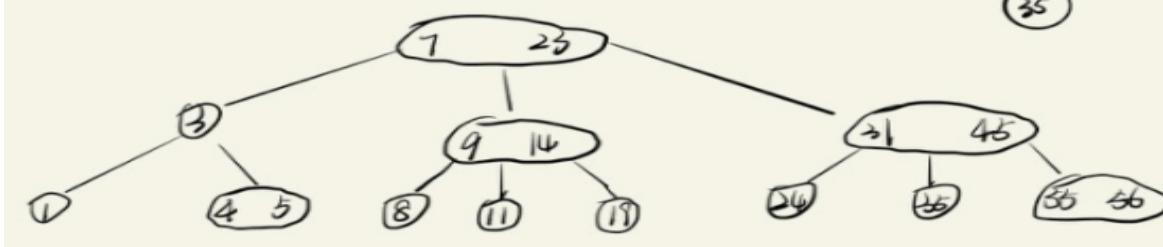


g)

g) delete 13 in (a)



delete 13 in (b)



**h) The Height of (a):7**

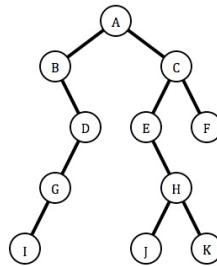
**The Height of (b):2**

**The Height of (c):2**

**i) See the code in H6\_2\_i.java**

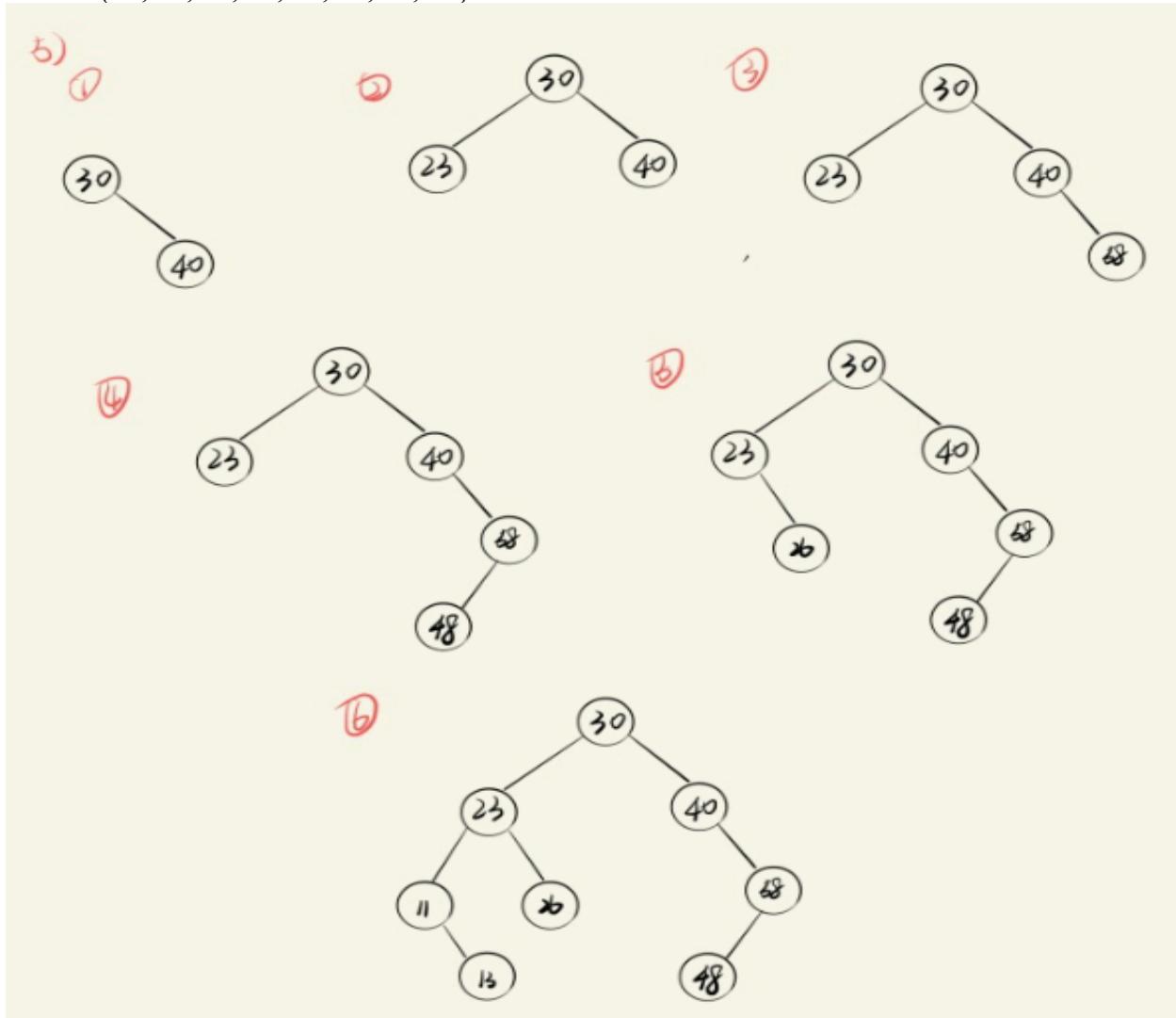
**j) See the code in H6\_2\_j.java**

4. Consider the following Binary tree, write Java code to find **minimum** element in binary search tree. You may write either a recursive or iterative implementation.



5. Insert the following items into an empty binary search tree in order:

{30, 40, 23, 58, 48, 26, 11, 13}



6. What is the maximum height of a binary search tree in problem-5? Why?

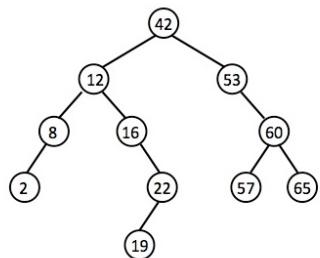
What is the time complexity of the Tree you built in problem-5? Why?

N is the number of elements of binary search tree.

The maximum height of a binary search tree is N-1, the maximum height of a binary search tree in problem 5 is 7. Because if the inserted element is in an ascending order, the elements will be only inserted in the right.

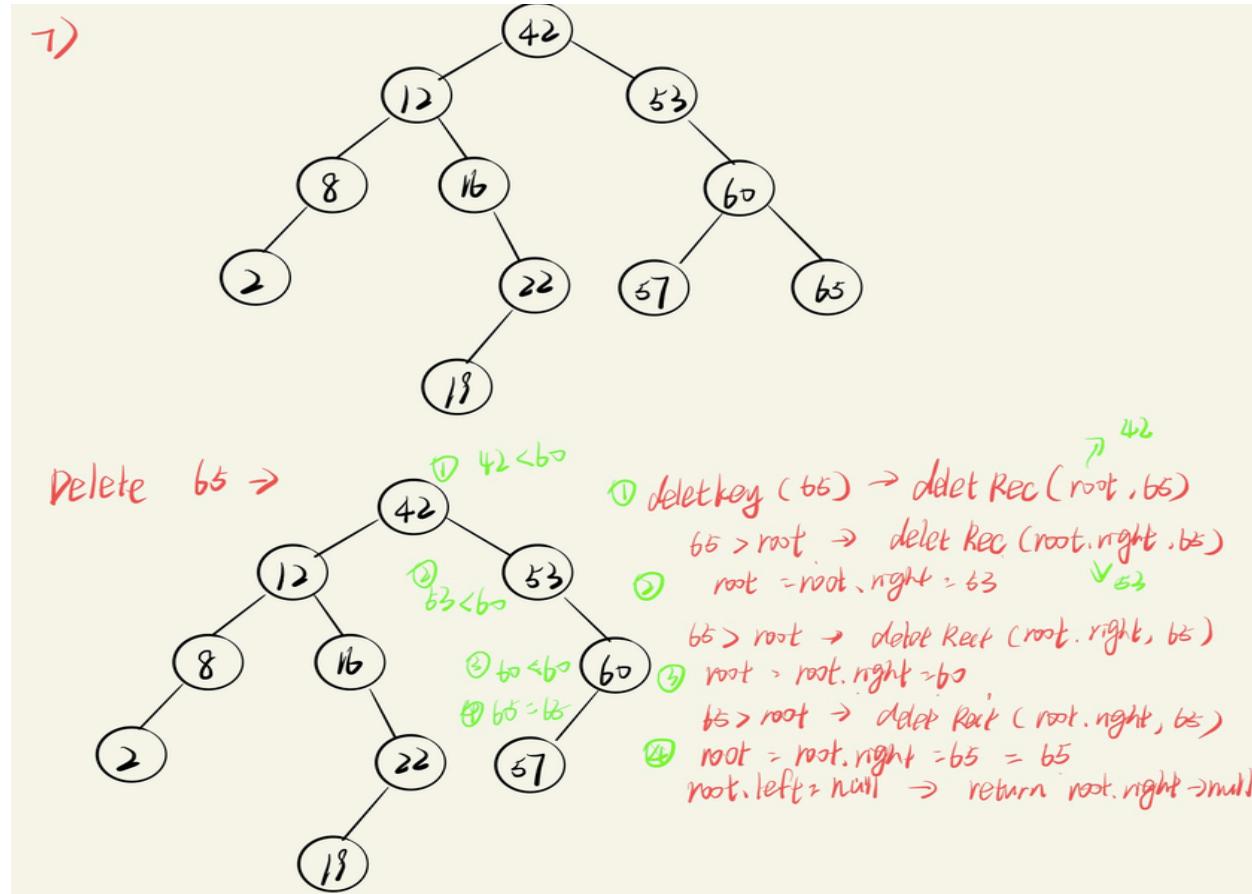
Time complexity of Binary Tree in worst case is O(n), if it is balanced, the time complexity in average case is O (log n) when it is inserted, search, delete. Because the tree in problem-5 is not balanced, the time complexity of the Tree in problem-5 is O(n), n=8.

7. Consider the following binary tree:

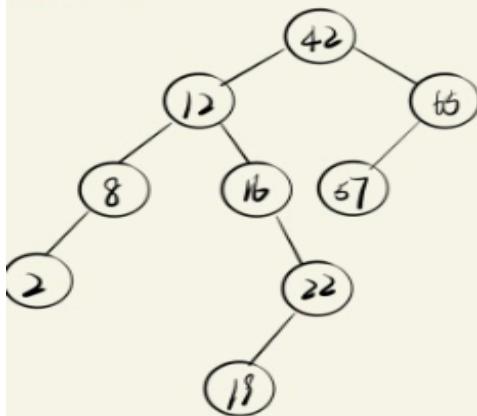


Use BST functions deleteKey, deleteRec, and minValue discussed in class: Show step-by-step code logic to delete the following nodes and redraw the binary tree for each deletion:

Delete 65 Delete 60 Delete 42



Delete 60 →



$\text{deleteKey}(60) \rightarrow \text{deleteRect}(\text{root}, 60)$

①  $42 < 60 \rightarrow \text{deleteRect}(\text{root.right}, 60)$

$\text{root} = \text{root.right} = 65$

②  $57 < 60 \rightarrow \text{deleteRect}(\text{root}, 60)$

$\text{root} = \text{root.right} = 60$

③  $60 = 60 \rightarrow \text{root.key} = \minValue(\text{root.right})$

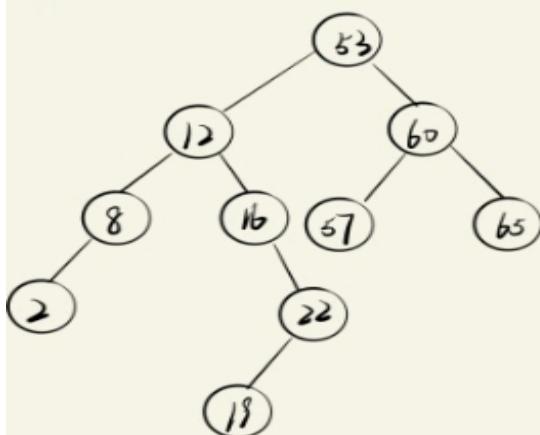
$\downarrow 65$

$\text{root.right} = \text{deleteRect}(\text{root.right}, \text{root.key})$

④  $65 = 65 \rightarrow \text{root.right} = \text{null}$

$\text{root.left} = 67$

Delete 42 →



$\text{deleteKey}(42) \rightarrow \text{deleteRect}(\text{root}, 42)$

$42 = 42 \rightarrow \text{root.key} = \minValue(\text{root.right})$

$\hookrightarrow \text{root} = \text{root.right} \quad \text{root.left} = \text{null}$

$\text{return root} = 53$

$\text{root.right} = \text{deleteRect}(\text{root.right}, \text{root.key})$

$65 = 65 \rightarrow \text{root.left} = \text{null} \quad \text{return root.right}$

$\text{root.right} = 60 :$

8. Consider  $\text{arr[]} = \{15, 80, 20, 90, 40, 60, 70\}$ . Walk through InsertionSort, SelectionSort, TimSort, and QuickSort algorithms, show step by step how the sort works.

#### InsertionSort:

Insertion Sort { 15, 80, 20, 90, 40, 60, 70 }

①

15	80	20	90	40	60	70
----	----	----	----	----	----	----

②

15	20	80	90	40	60	70
----	----	----	----	----	----	----

③

15	20	40	80	90	60	70
----	----	----	----	----	----	----

④

15	20	40	60	80	90	70
----	----	----	----	----	----	----

⑤

15	20	40	60	70	80	90
----	----	----	----	----	----	----

QuickSort:

Quick Sort : { 15, 80, 20, 90, 40, 60, 70 }

①

15	80	20	90	40	60	70
$\downarrow$ pivot value	$\downarrow$ leftmark $\rightarrow$			$\leftarrow$ rightmark		

②

15	80	20	90	40	60	70
$\downarrow$ pivot value	$\downarrow$ leftmark $\rightarrow$			$\leftarrow$ rightmark		

$15 < 80$     $15 < 70$     $15 < 90$   
 $15 < 40$     $15 < 60$     $15 < 70$   
leftmark = rightmark   skip  
then quicksort(1,6)

③

15	80	20	70	40	60	90
		$\downarrow$ leftmark			$\uparrow$ rightmark	

$80 > 20$     $80 < 90$    skip  
 $80 > 70$    stop   exchange 70 and 90

④

15	60	20	70	40	80	90
$\downarrow$ pivot value				$\uparrow$ rightmark		

$40 < 80$     $60 < 80$    leftmark > rightmark  
skip  
exchange 80 and 60  
quicksort(1,4)

⑤

15	60	20	40	70	80	90
----	----	----	----	----	----	----

$60 > 20$     $60 < 70$    skip  
 $60 < 40$    skip   exchange 40 and 70

⑥

15	40	20	60	70	80	90
$\downarrow$ pivot value	$\downarrow$ left				$\uparrow$ right	

exchange 60 and 40  
quicksort(1,2)  
exchange 20 and 40

⑦

15	20	40	60	70	80	90
----	----	----	----	----	----	----

### SelectionSort:

Selection Sort : { 15, 80, 20, 90, 40, 60, 70 }

①	<table border="1"> <tr> <td>15</td><td>80</td><td>20</td><td>90</td><td>40</td><td>60</td><td>70</td></tr> </table>	15	80	20	90	40	60	70
15	80	20	90	40	60	70		

find min (0, b)  
put it at beginning

②	<table border="1"> <tr> <td>15</td><td>20</td><td>80</td><td>90</td><td>40</td><td>60</td><td>70</td></tr> </table>	15	20	80	90	40	60	70
15	20	80	90	40	60	70		

find min (1, b)  
put it at (1, b) beginning

③	<table border="1"> <tr> <td>15</td><td>20</td><td>40</td><td>90</td><td>80</td><td>60</td><td>70</td></tr> </table>	15	20	40	90	80	60	70
15	20	40	90	80	60	70		

find min (2, b)  
put it at (2, b) beginning

④	<table border="1"> <tr> <td>15</td><td>20</td><td>40</td><td>60</td><td>80</td><td>90</td><td>70</td></tr> </table>	15	20	40	60	80	90	70
15	20	40	60	80	90	70		

find min (3, b)  
put it at (3, b) beginning

⑤	<table border="1"> <tr> <td>15</td><td>20</td><td>40</td><td>60</td><td>70</td><td>90</td><td>80</td></tr> </table>	15	20	40	60	70	90	80
15	20	40	60	70	90	80		

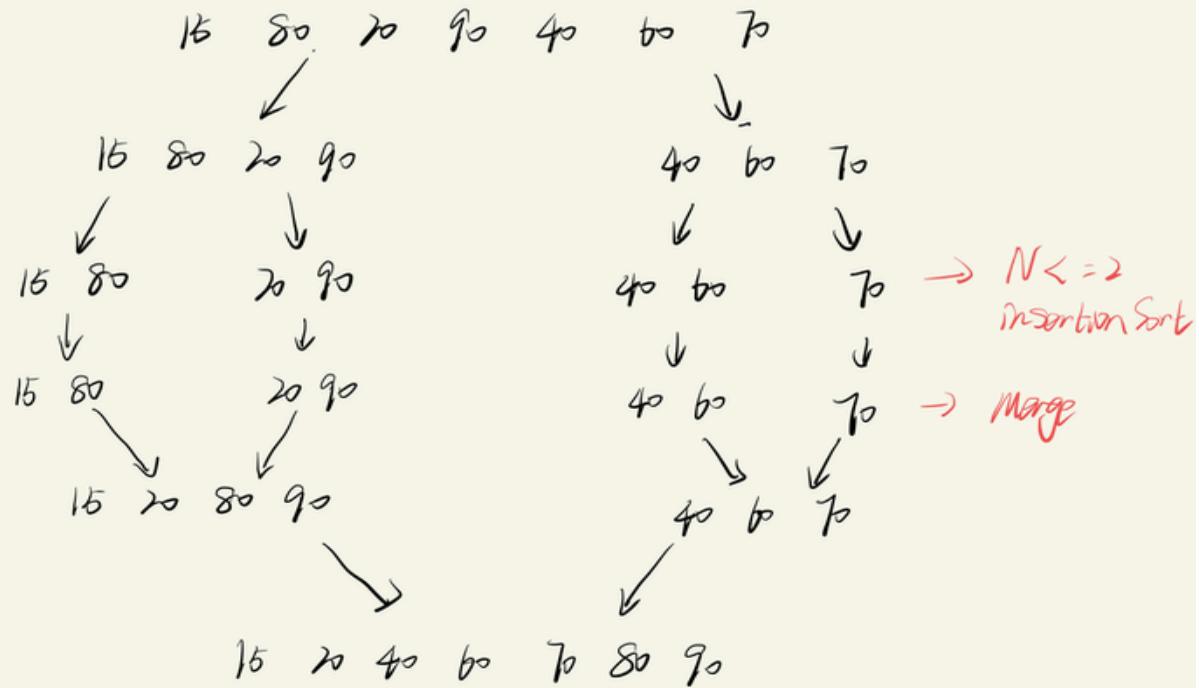
find min (4, b)  
put it at (4, b) beginning

⑥	<table border="1"> <tr> <td>15</td><td>20</td><td>40</td><td>60</td><td>70</td><td>80</td><td>90</td></tr> </table>	15	20	40	60	70	80	90
15	20	40	60	70	80	90		

find min (5, b)  
put it at (5, b) beginning

### Timsort:

TimSort { 15, 80, 20, 90, 40, 60, 70 } Set RUN = 2.



9. Consider attached image Boston.jpg. Write a program to sort the image Pixels by “brightness”. You program for four sorting algorithms: InsertionSort, SelectionSort, TimSort, and QuickSort, MergeSort. You need to sort the Pixel array size of the image in Ascending order and compare and show the runtime time complexity of each Sorting algorithm.

Notes:

You may NOT use any Java library function for sorting. You should use ONLY the Sorting Java code I provided in class. The Pixel sorting should start from (0,0) to (high,high) for Brightness. For each Pixel, you need to convert RGB color to appropriate intensity. Use intensity formula:  $I = 0.2989R + 0.5870G + 0.1140B$ . If the current pixel Intensity is larger than the next pixel intensity, you need to swap, going in descending order.

	InsertionSort	SelectionSort	TimSort	QuickSort	MergeSort
Worst case	$O(n^2)$	$O(n^2)$	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Best case	$O(n)$	$O(n^2)$	$O(n)$	$O(n \log n)$	$O(n \log n)$
Average case	$O(n^2)$	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

The screenshot shows five separate Java application windows, each displaying the execution time of a specific sorting algorithm. The windows are arranged vertically, each with a green play button icon and an upward arrow icon to its right. The text output is as follows:

- /Library/Java/JavaVirtualMachines/  
The time of quickSort is 69
- /Library/Java/JavaVirtualMachines/  
The time of mergeSort is 65
- /Library/Java/JavaVirtualMachines/1  
The time of Insertion Sort is 795
- /Library/Java/JavaVirtualMachines/1.  
The time of Selection Sort is 2466
- /Library/Java/JavaVirtualMachines/  
The time of TimSort is 57