CSI 2110 Tutorial (Section A)

Yiheng Zhao
yzhao137@uottawa.ca
Office Hour: Fri 13:00-14:00
Place: STE 5000G

2019-10-25

Mid-term Exercises:

1. The following code uses a stack implemented using a **singly linked list**. Give the **time complexity** of the following piece of code in terms of **big-Oh** as a function of **N**

```
Stack<Integer> myStack = new Stack<Integer>();
for (int i=0; i< N; i++) {
      myStack.push(i);
}
int tot=0;
for (int i=0; i< N; i++) {
      tot += myStack.pop();
}
```

A) O(1) B) O(log N) C) O(N) D) O(N log N) E) O(N^2)

2. The following code uses a **sorted array arr** with N elements and uses the well-known **binary search** method that returns the index of the array that contains the key or –1 if the key is not found. Give the **time complexity** of the following piece of code in terms of **big-Oh** as a function of N.

```
int j;
int count=0;
int N = arr.length;
for (int i=0; i< N/2; i++) {
      int key=2*i;
      j=binarySearch(arr,key,0,N-1);
      if (j!=-1) count++;
}
```

A) O(1) B) O(logN) C) O(N) D) O(NlogN) E) O(N^2)

Mid-term Exercises:

3. Give the **time complexity** of the following piece of code in terms of **big-Oh** as a function of N.

```
int i=1;
while (i< N){
        i = i*2;
        print(i);
}
```

A) O(1) B) O(logN) C) O(N) D) O(NlogN) E) O(N^2)

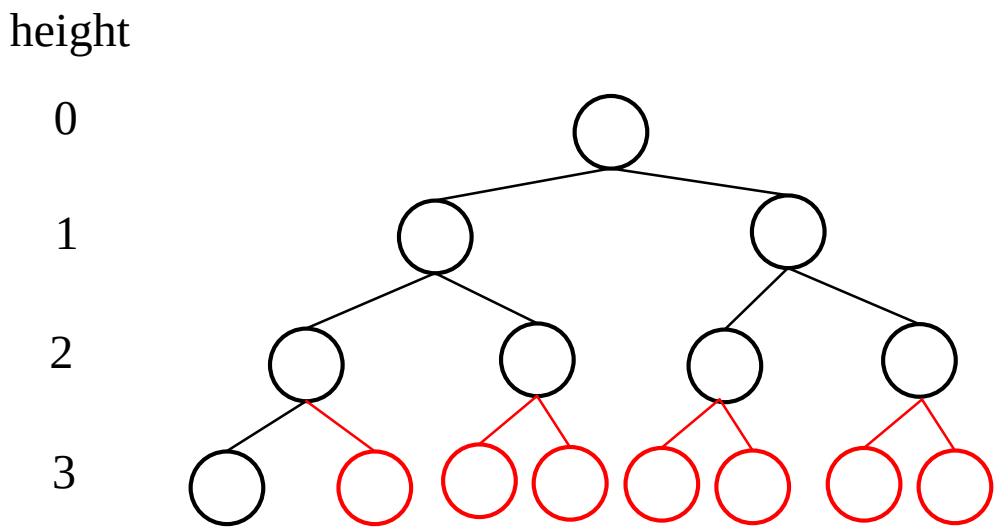
Print: 2, 4, 8, …2^k

$2^k < N$ -> $k < log_2 N$

Mid-term Exercises:

4. What is the minimum and maximum number of nodes in a **complete binary tree** with **height 3** ?

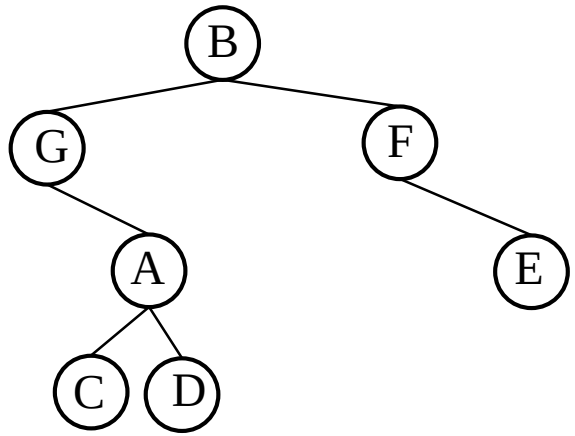
A) min= 4, max= 8
B) min= 4, max= 15
C) min= 8, max= 15
D) min= 8, max= 16
E) None of the above.



Mid-term Exercises:

5. Consider a binary tree containing letters, such that the **in-order** traversal prints **G,C,A,D,B,F,E** and its **pre-order** traversal prints **B, G, A, C, D, F, E**. What does a **post-order** traversal prints?

A) G, A, C, B, E, F, D
B) D, C, F, G, A, B, E
C) E, F, D, C, A, G, B
D) C, D, A, G, E, F, B
E) None of the above.



6. Consider a **priority queue** with n elements, implemented using a **min-heap**. Which of the following answers give the **time complexity** of the operations **insert** (insertion of an arbitrary element into the priority queue) **removeMin** (deletion of the smallest element in the priority queue) **min** (return the smallest element without modifying the priority queue), respectively:

A) Θ (1), Θ (l), Θ (1)
B) Θ (log n), Θ (log n), Θ (1)
C) Θ (log n), Θ (log n), Θ (log n)
D) Θ (n), Θ (n), Θ (n)
E) None of the above.

Insert: add to the end (O(1)) + upheap (O(log n))
RemoveMin: replace top (O(1)) + downheap(O(log n))
Min: return top (O(1))

Mid-term Exercises:

7. In which of the following data structures containing n elements, **deleting the element with minimum key** has worst-case complexity O(log n)?

A) a sorted array (sorted in increasing order) O(n): move one element forward for each element
B) an unsorted doubly linked list O(n): min key is the last element
C) a binary search tree O(n): when tree structure is like a link
D) a queue O(n): when the key is the last element
E) none of the above.

Mid-term Exercises:

8. Consider a **MAP** abstract data type and various different data structures that implement it. Assume we employ the most efficient known algorithm for each operation in the given data structure.
The MAP operations are detailed below:
get(k): searches for key k and returns the value v associated to key k, if such entry exists; otherwise returns null.
put(k, v): searches for key k, and if M does not have an entry with key k, then adds (k,v) and returns null; otherwise it replaces with v the value of the entry with key equal to k and returns the old value.
remove(k): searches for key k and removes from M the entry (v,k) and returns its value v; if M has no such entry, then it returns null.
For each data structure, indicate the **big-Oh (worst case)** complexity of each of the MAP operations as a function of *n*, the number of elements in the MAP:

	Get(k)	Put(k, v)	Remove(k)
Unsorted linked list	O(n)	O(n)	O(n)
Sorted list	O(log n)	O(n)	O(n)
Binary tree	O(n)	O(n)	O(n)

Mid-term Exercises:

9. Give an asymptotic upper bound using **big-Oh** for the following functions. Use the **tightest** and **simplest** upper bound. Briefly justify your answers.

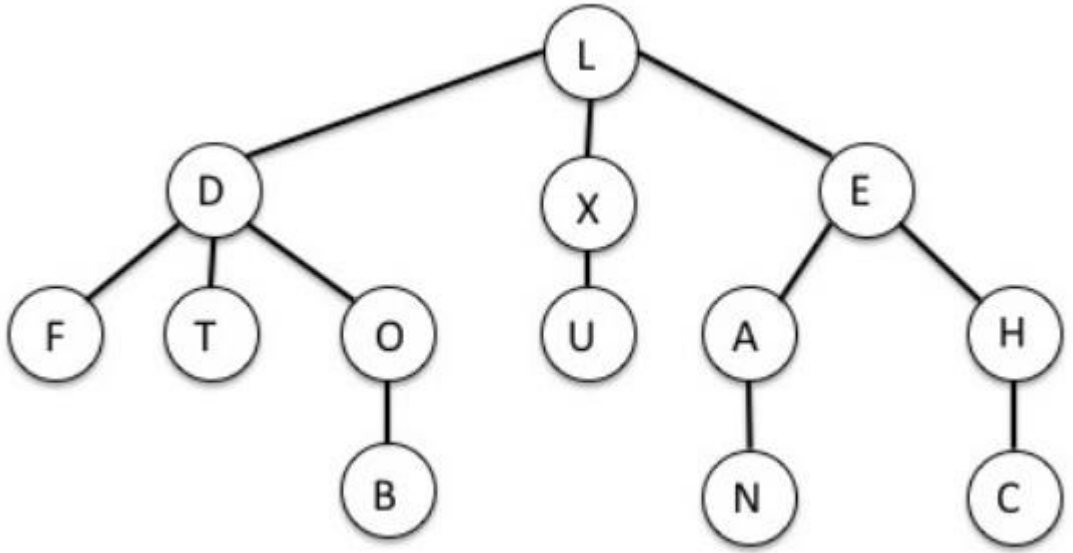
1) $f(n) = 1 + n + 3n^2 + 3n^3$ $\leq n^3 + n^3 + 3n^3 + 3n^3 \leq 8n^3$
for C=8, $n > n_0 > 0$. $O(n^3)$

2) $f(n) = log_2(2^n * n^3)$ $= log_2 2^n + log_2 n^3 = nlog_2 2 + 3log_2 n = n + 3log_2 n \leq n + 3n \leq 4n$
for C=4, n $> n_0 > 0$. $O(n)$

3) $f(n) = \sum_{i=0}^{n} 2^i = 1 + 2 + 4 + \ldots + 2^n$ $= 2^{n+1} - 1 \leq 2 * 2^n$
for C=2, n $> n_0 > 0, O(2^n)$

Mid-term Exercises:

10. Print values for the **pre-order** and **post-order** traversal for the tree:

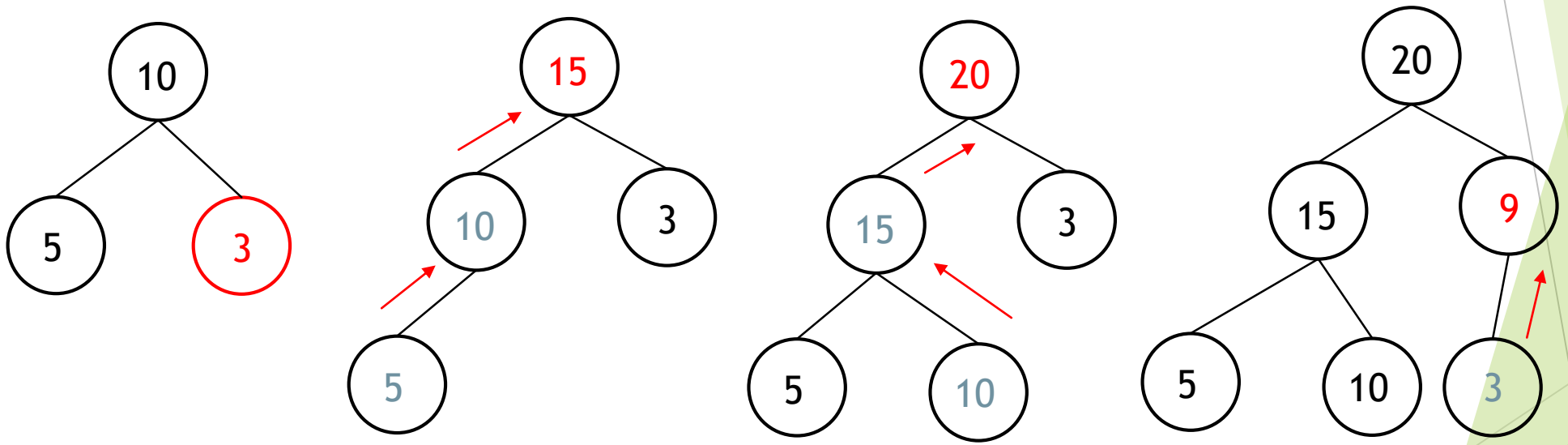


Pre-order: LDFTOBXUEANHC

Post-order: FTBODUXNACHEL

Mid-term Exercises:

11. Draw the **max-heap** obtained after **inserting** each of the following Keys (in this order) into an empty heap : 5, 10, **3, 15, 20, 9** **(**show the heap after each of the bold numbers are inserted)



12. Given the following array:

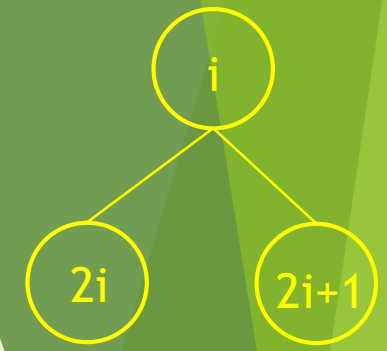
10	40	50	20	70	30	60

Sort this array **in-place** in increasing order using the **HeapSort algorithm**.
Phase 1: Bottom up maxheap contruction: **display array after each downheap operation.**

Phase 2: Phase that incrementally builds sorted array, reducing the heap part: **underline the sorted part at each step (the sorted part size must increase by 1 at each new array shown, while the heap part decreases by 1)**, do not show intermediate steps.

You may need more or less arrays than the ones provided below.
Show the array after each BIG STEP of each phase (not at each simple swap!).
Draw a line separating Phase 1 from Phase 2.

Mid-term Exercises:

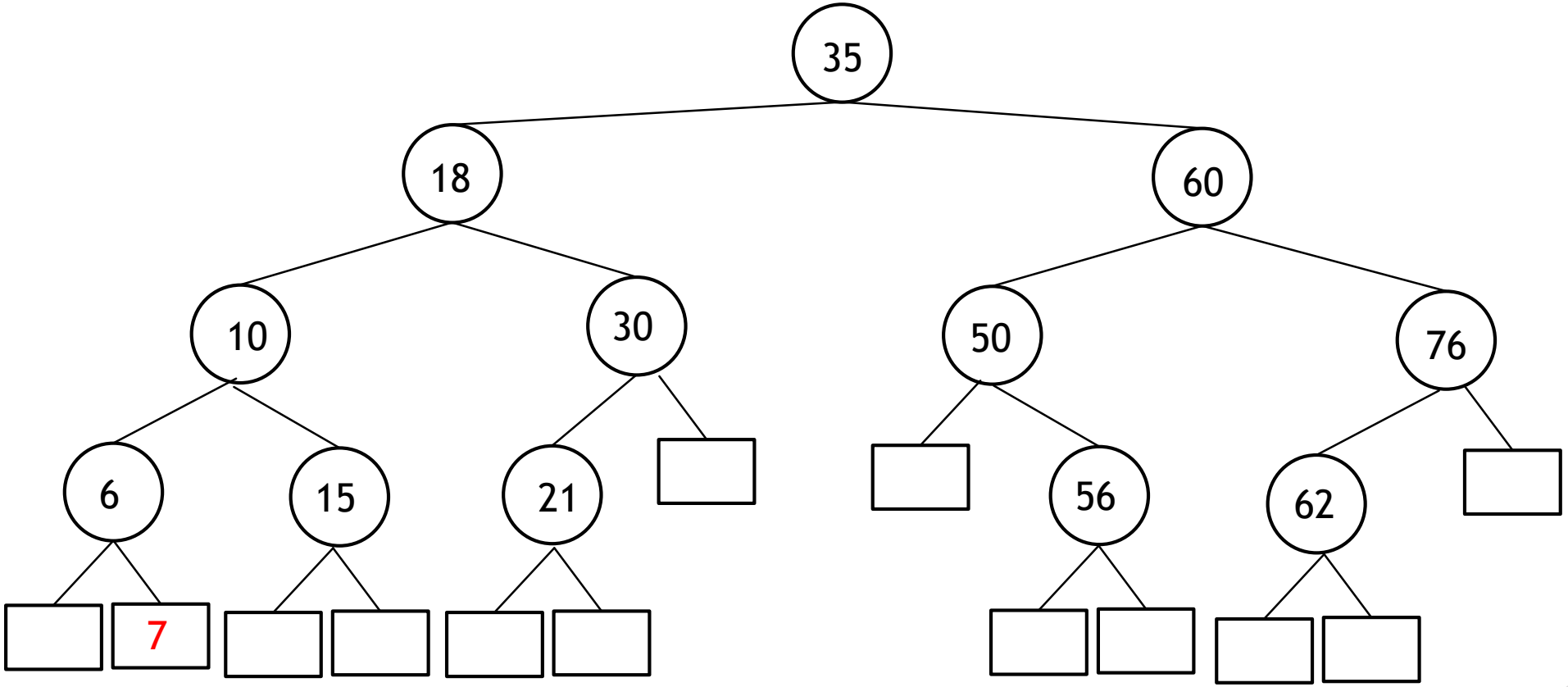


original	10	40	50	20	70	30	60
	10	40	60	20	70	30	50
Phase 1	10	70	60	20	40	30	50
	70	40	60	20	10	30	50

	60	40	50	20	10	30	70
	50	40	30	20	10	60	70
Phase 2	40	20	30	10	50	60	70
	30	20	0	40	50	60	70
	20	10	30	40	50	60	70
	10	20	30	40	50	60	70

Mid-term Exercises:

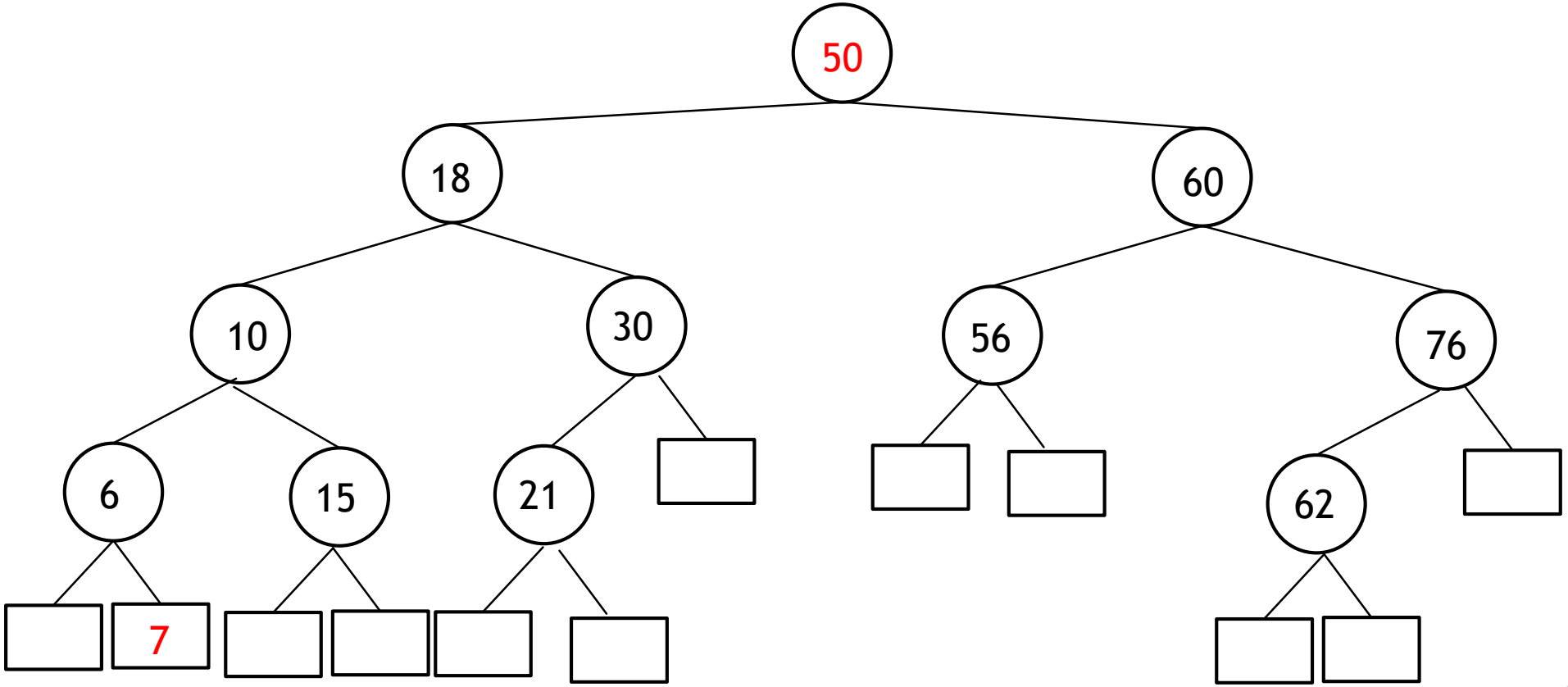
13. Consider the **binary search tree** below :



1) Show the tree after inserting 7
2) Show the tree after deleting key 35 from the original tree.

Mid-term Exercises:

13. Consider the **binary search tree** below :



1) Show the tree after inserting 7
2) Show the tree after deleting key 35 from the original tree.

14. Consider a **binary search tree** that provides the following methods :

root()	returns the root node of the tree or null if the tree is empty
parent(p)	returns the parent node of node p or null if p is the root
leftChild(p)	returns the left child node of node p or null if p has no left child.
rightChild(p)	returns the right child node of node p or null if p has no right child
key(p)	returns the key (integer) stored in node p
isInternal(p)	returns true if and only if p is an internal node
isExternal(p)	returns true if and only if p is an external node
size()	returns the number of nodes stored in the tree

You may give the required algorithm in pseudocode or Java program excerpt.
You may use the methods listed above and design any auxiliary method you wish.

Mid-term Exercises:

1) Give a pseudocode or Java code for a method that computes the maximum value of a key stored in the binary search tree.
Example : for the tree in page 8 (question 13) this method would return 76.

```
int maxKey(){
      p=root();
      if(p==null)
            return -maxint
      while(right(p)!=null)
            p=right(p);
      return key(p);
}
```

Mid-term Exercises:

2) If the tree has n nodes and height $h=2 \log n$, what is the **big-Oh** running time of your **maxKey()** algorithm as a function of n ?

$O(\log(n))$ (worst case)