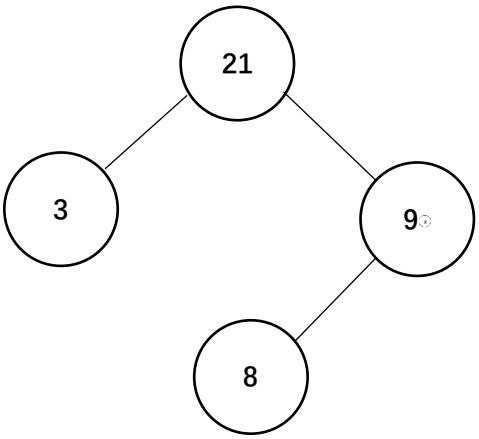
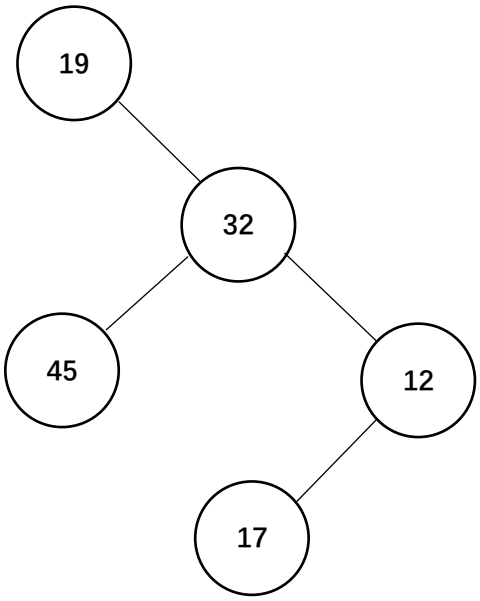
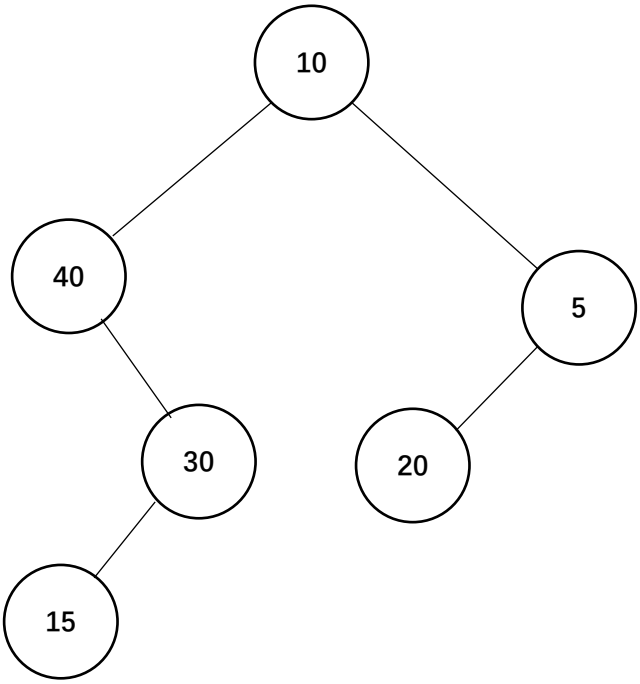
1. (i)



(ii)



2. (i)



2. (ii): depth of node 30 is 2; height of node 30 is 1.
2. (iii): 30
2. (iv): 15, 20

3.

Algorithm: isSingle(T) Requires: a binary tree T Returns: True if it has a single node; False otherwise
1: if isLeaf(T) 2: return False 3: elseif isLeaf(left(T)) && isLeaf(right(T)) 4: return True 5: else 6: return False 7: endif

4.

```
Algorithm: sum(T)
Requires: a binary tree T
Returns: sum of all node values
1:  if isLeaf(T)
2:      return 0
3:  else
4:      return root(T)+sum(left(T))+sum(right(T))
5:  endif
```

5.

```
Algorithm: min(T)
Requires: a non-empty binary tree
Returns: minimum value in the tree
1:  if isLeaf(left(T)) && isLeaf(right(T))
2:      return root(T)
3:  elseif isLeaf(left(T))
4:      return min2(root(T),min(right(T)))
5:  elseif isLeaf(right(T))
6:      return min2(root(T),min(left(T)))
7:  else
8:      return min2(root(T),min2(min(left(T)),min(right(T))))
9:  endif
```

The following sub-algorithm min2(x,y) is used:

```
Algorithm: min2(x,y)
Requires: two numbers x and y
Returns: smaller value of two numbers
1:  if x<y
2:      return x
3:  else
4:      return y
5:  endif
```

6.

```
Algorithm: add(x,T)
Requires: a non-empty binary tree and a number x
Returns: a new binary tree with updated values, each added by x
1:  if isLeaf(left(T)) && isLeaf(right(T))
2:      return node(leaf,x+root(T),leaf)
3:  elseif isLeaf(left(T))
4:      return node(leaf,x+root(T),add(x,right(T)))
5:  elseif isLeaf(right(T))
6:      return node(add(x,left(T)),x+root(T),leaf)
7:  else
8:      return node(add(x,left(T)),x+root(T),add(x,right(T)))
```

```
9:  endif
```

7. Refer to Seminar 7 slides.

8.

```
Algorithm: depth(x,T)
Requires: a number x and a binary tree T
Returns: depth of node with value x, or -1.
1:  if !search(x,T)
2:      return -1
3:  elseif x==root(T) || isLeaf(T)
4:      return 0
5:  else
6:      return 1+max2(depth(x,left(T)), depth(x,right(T)))
7:  endif
```

The following two sub-algorithms are used:

```
Algorithm: max2(x,y)
Requires: two numbers x and y
Returns: larger value of two numbers
1:  if x>y
2:      return x
3:  else
4:      return y
5:  endif
```

Algorithm: search(x,T). Refer to Lecture 7 slides.

9.

```
Algorithm: leafnodeNum(T)
Requires: a binary tree T
Returns: total number of leaf nodes
1:  if isLeaf(T)
2:      return 0
3:  elseif isLeaf(left(T)) && isLeaf(right(T))
4:      return 1
5:  else
6:      return leafnodeNum(left(T))+leafnodeNum(right(T))
7:  endif
```

10.

```
Algorithm: bisection(a,b,error)
Requires: three numbers a,b (a<b) and error
Returns: approximate root of f(x)
```

```
1:   if f(a)*f(b)>0
2:       return 'Choose a different interval'
3:   else
4:       let c=0.5*(a+b) // find middle point
5:       if abs(f(c))<error // approximate solution is accurate enough
6:           return c
7:       elseif f(a)*f(c)<0 // root exists on interval (a, c)
8:           return bisection(a,c,error) // replace b by c
9:       else // take the other half interval
10:          return bisection(c,b,error) // replace a by c
11:      endif
12:  endif
```