Name: Pheakdey Luk
ID: 986591

# Assignment 3

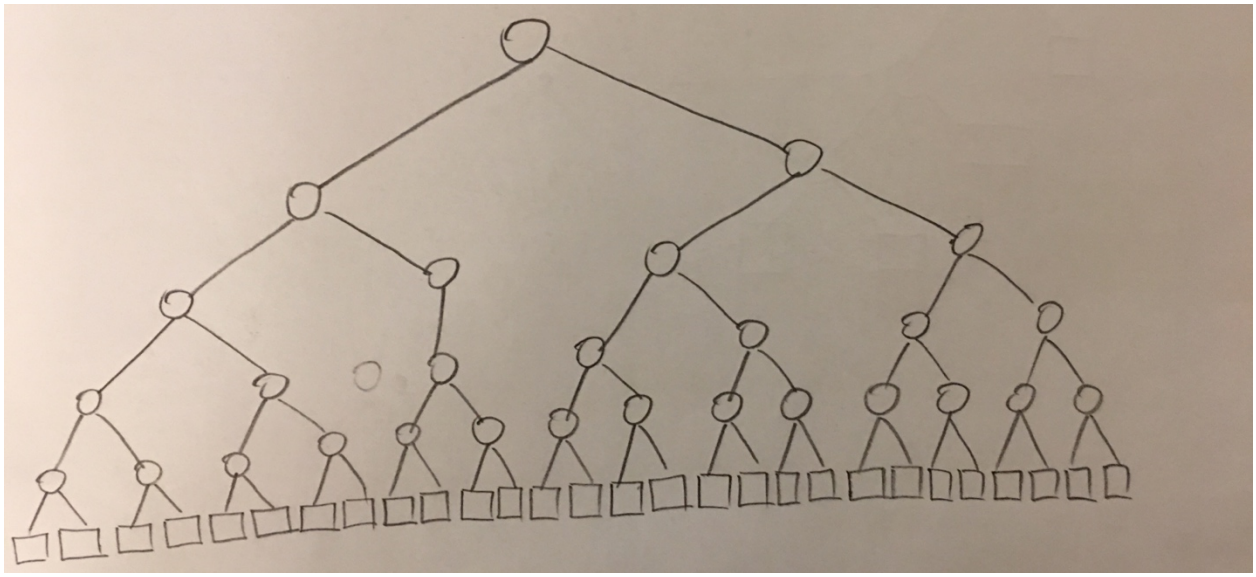
R-2.7 Let T be a binary tree with n nodes that is implemented with a vector, S, and let p be the level numbering of the nodes in T, as given in Section 2.3.4. Give pseudo-code descriptions of each of the methods root, parent, leftChild, rightChild, isInternal, isExternal, and isRoot.

**⇨ Answer**

```
Algorithm root()
      Output: position of the root of the
      return 1


Algorithm parent(p)
    Input : a position p of a node in the vector S
    Output: position of the root of the tree T in S
    return p / 2


Algorithm leftChild(p)
    Input : a position p of a node in the vector S
    Output:position of the leftChild of the tree T in S
    return 2 * p


Algorithm rightChild(p)
    Input : a position p of a node in the vector S
    Output:position of the leftChild of the tree T in S
    return 2 * p + 1


Algorithm isInternal(p)
    Input : a position p of a node in the vector S
    Output:position of the leftChild of the tree T in S
    if 2 * p <= S.size() or (2 * p + 1) <= S.size()
        return true
    return false


Algorithm isExternal(p)
    Input : a position p of a node in the vector S
    Output:position of the leftChild of the tree T in S
    if 2 * p > S.size() ^ (2 * p + 1) > S.size()
        return true
    return false
```

R-2.8 Answer the following questions so as to justify Theorem 2.8.

a. Draw a binary tree with height 5 and with the maximum number of external nodes.



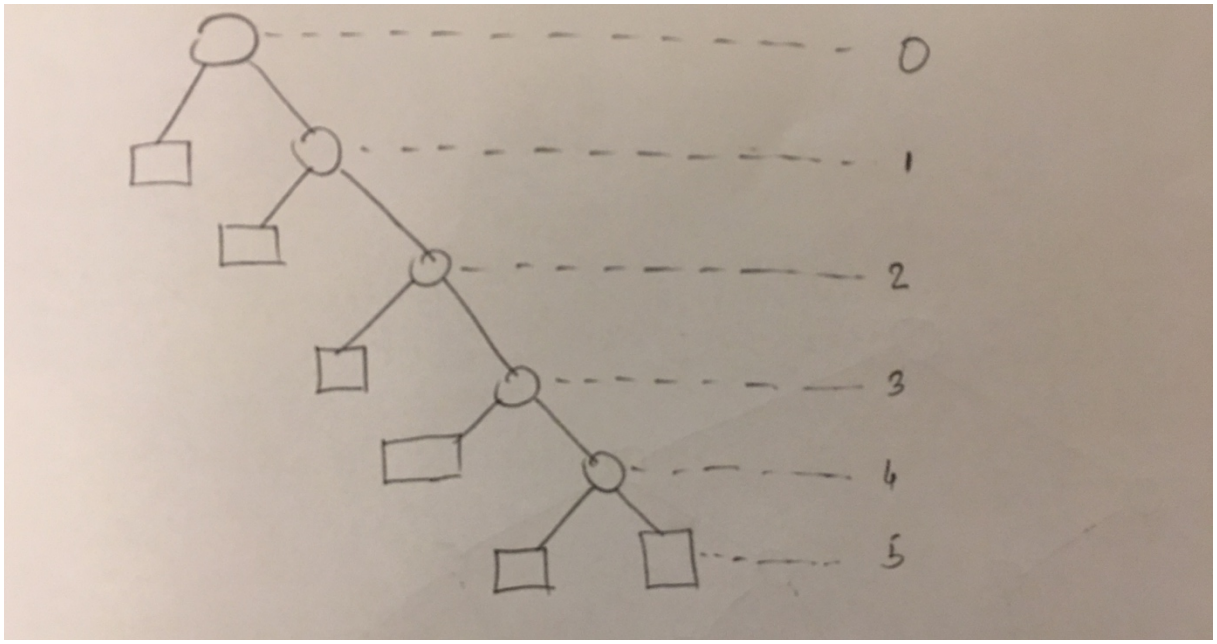
b. What is the minimum number of external nodes for a binary tree with height h? Justify your answer.

**⇨ Answer**

Minimum number of external nodes for binary tree with height = h + 1.

Justification: A binary tree can have minimum number of external nodes if the tree is drawn in a way that in each level at least one child is internal node.

In below tree, no of external nodes = 6,height = 5 which proofs min external node = h + 1




c. What is the maximum number of external nodes for a binary tree with height h? Justify your answer.

**⇨ Answer**

Maximum number of external node = $2^h$

We know that external nodes don't have child nodes. So, the maximum number of externals node we can get when all the nodes in the same depth

d. Let T be a binary tree with height h and n nodes. Show that $\log(n+1) - 1 < h < (n-1)/2$

**⇨ Answer**

External node, e = i – 1, i=internal node

Total node, n = e + i

=>n = 2e – 1

=>e = (n + 1) / 2

Since, $e <= 2^h$

=> $(n+1)/2 \leq 2^h$

=> $\log (n+1)/2 \leq \log 2^h$

=> $\log (n+1) - \log 2 \leq h \log 2$

=> $\log (n+1) - \log 2 \leq h \log 2$

=> $\log (n+1) - 1 \leq h$ [log 2 = 1]

Again, $h+1 \leq e$

=> $h+1 \leq (n+1)/2$

=> $h \leq (n+1)/2 - 1$

=> $h \leq (n-1)/2$

Combining both equation we get, $\log (n+1) -1 \leq h \leq (n-1)/2$

e. For which values of n and h can the above lower and upper bounds on h be attained with equality?

**⇨ Answer**

For n=1 and h=0, lower and upper bounds on h be attained with equality.

$\log(1+1) -1 \leq h \leq (1-1)/2$

=>$0 \leq h \leq 0$

C-2.2 Analyze your implementation of the queue ADT that used two stacks (from assignment 2). What is the amortized running time for dequeue and enqueue, assuming that the stacks support constant time push, pop, and size methods?

**⇨ Answer**

As push, pop and size methods support constant time,

Running time of the dequeue and enqueue = n + n * k.

T(n) = O(n).

Amortized running time = T(n) / n = 1

C-2.7 Using the Sequence ADT, describe an efficient way of putting a sequence representing a deck of *n* cards into random order. Use the function randomInt(n), which returns a random number between 0 and n-1, inclusive. Your method should guarantee that every possible ordering is equally likely. What is the running time of your method, if the sequence is implemented with an array? What if it is implemented with a linked list?

**⇨ Answer**

```
Algorithm putSequenceInRandomOrder(S)
    Input:Sequence S with n elements
    Output: S in random order
    r <- n
    while r > 0 do
        rand <- randomInt(r)
        S.swapElements(S.atRank(r),S.atRank(rand))
        r <- r - 1
    return S


Running Time, for array based implementation
L1: O(1)
L2: O(n)
L3: O(n)
L4: O(n)
L5: O(n)
L6: O(1)
Total = O(n)


Running time, for inked list based implementation
```

L1: O(1)
L2: O(n)
L3: O(n)
L4: $O(n^2)$=>for finding rank, it takes O(n) time.
L4: O(n)
L5: O(1)
Total = $O(n^2)$