1.
To get the parent of a node at index i in a complete binary tree represented by a singly linked list, we can use the following formula:
parent(i) = (i-1) // 2
To get the left child of a node at index i, we can use the following formula:
left_child(i) = 2i + 1
To get the right child of a node at index i, we can use the following formula:
right_child(i) = 2i + 2

```
class BinHeap:
  # 二叉堆初始化
  def __init__(self):
     self.heapList = [0]
     self.currentSize = 0
# 通过无序表创建二叉堆
  def buildHeap(self,alist):
    # 从最后节点的父节点开始，因为叶节点无需下沉
    i = len(alist) // 2
    self.currentSize = len(alist)
    self.heapList = [0] + alist[:]
    while (i > 0):
       self.percDown(i)
       i = i - 1
```

2.
To implement the minimum priority queue using a linked list representation of a complete binary tree, we can use the following methods:

insert(key):
插入元素

```
  def insert(self,k):
    self.heapList.append(k)
    self.currentSize = self.currentSize + 1
    self.percUp(self.currentSize)
```

Create a new node with the given key
If the linked list is empty, set the new node as the root and return
Otherwise, find the next available position in the linked list to insert the new node
If the new node is to be inserted as a left child, insert it at the next available position and set its parent's left child to the new node
If the new node is to be inserted as a right child, insert it at the next available position and set its parent's right child to the new node
Compare the key of the new node with its parent's key. If the key of the new node is less than its parent's key, swap the keys of the new node and its parent
Repeat the above step until the key of the new node is greater than or equal to its parent's key, or until the new node becomes the root
delMin():
删除最小值

```
  def delMin(self):
    retval = self.heapList[1]
    self.heapList[1] = self.heapList[self.currentSize]
    self.currentSize = self.currentSize - 1
    self.heapList.pop()
    self.percDown(1)
```

return retval
If the linked list is empty, return
Save the root node's key in a temporary variable
If the linked list has only one node, set the root to None and return the saved key
Find the next available position in the linked list
If the next available position is a left child, set the root's left child to the next available node
If the next available position is a right child, set the root's right child to the next available node
Compare the key of the root with its children's keys. If the key of the root is greater than its children's keys, swap the keys of the root and the child with the smallest key
Repeat the above step until the key of the root is less than or equal to its children's keys, or until the root has no children
3.
The time complexity of the insert() method is O(log n), since in the worst case the new node may have to be compared with its parent all the way up to the root. The time complexity of the delMin() method is also O(log n), since in the worst case the root may have to be compared with its children all the way down to the bottom of the tree.

4.
To conduct a performance benchmark for the linked list based heap, we can measure the time it takes to perform a given number of insert and delMin operations. We can then visualize the results by plotting the time taken on the y-axis and the number of operations on the x-axis.

5.
(Bonus)
To draw the tree structure of the linked list based heap using graphviz, we can use the following steps:
Traverse the linked list and for each node, create a corresponding node in the graph with the node's key as the label
For each node in the linked list, draw an edge from the node to its left child and right child, if they exist
Render the graph using graphviz