HW6

Q1. Although it still takes O(n) to delete the last item because we can't track back to the previous node of the last node, it reduces the time complexity of add operation to O(1) by pointing the tail to the new node you want to add and update the tail.

Q2. The basic idea is using priority queue. Add every head of k SLLs in priority queue, then keep polling the smallest one to the new SLL.

pseudocode:

```
for (list in lists[]) {
    if ( list != null)  pq.offer(list);
}
ListNode dummy = new ListNode(0);
ListNode tail = dummy;
while (! pq.isEmpty()) {
    ListNode head = pq.poll();
    tail.next ← head;
    tail ← head;
    if (head.next != null)  pq.offer(head.next);
}
return dummy.next;
```

Extra Credit.

The idea is to seperate SLL every k elements to a group, if the remaining numbers of elements is less than k, then return the head of this group; for other k-group, using two pointers reverse them one by one

pseudocode:

```
seperate ( head, k) {
```

```
while (node != null && count < k) {
    node ← node.next
    count++
}
if count < k    return head;
else {
    reverse(head, count)
}
```

Python Code:

HW6-EC.py

```python
import sys

# Definition for singly-linked list.
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

def seperate(self, head, k):
    count, node = 0, head
    while node and count < k:
        node = node.next
        count += 1
    if count < k: return head
    new_head, prev = reverse(head, head, count)
    head.next = seperate(new_head, new_head, k)
    return prev

def reverse(self, head, count):
    prev, curr, nxt = None, head, head
    while count > 0:
        nxt = curr.next
        curr.next = prev
        prev = curr
        curr = nxt
        count -= 1
    return (curr, prev)

def main(argv):
    head = ListNode(1)
    head.next = ListNode(2)
    head.next.next = ListNode(244)
    head.next.next.next = ListNode(7)
    head.next.next.next.next = ListNode(93)
    head.next.next.next.next.next = ListNode(105)
    k = int(argv[1])
    node = seperate(head, head, k)
    while (node != None):
        print('%d' % node.val, ' ', end = '')
        node = node.next
    print(' ')

if __name__ == '__main__':
    main(sys.argv)

```