1 Josephus Problem

1.1 Linked list implementation

Circularize the linked list, move k times (including the starting node hence $k-1$), delete from list in $O(1)$. Will perform $n-1$ deletion and for each skip over $k-1$ node.

JOSEPHUS$(A, k) \in O(kn)$

```
1  cur ← A.head
2  A.head.prev ← A.tail
3  A.tail.next ← A.head
4  while A.head.next ≠ A.head
5        do
6           for i ← 1 to k − 1
7                 do
8                    cur ← cur.next
9
10          if cur = A.head
11             then
12                   A.head ← cur.prev
13
14          cur.prev.next ← cur.next
15          cur.next.prev ← cur.prev
16          cur ← cur.next
17
18 return cur
```

1.2 Sequence Implementation

Naive $O(n^2)$ implementation as baseline.

JOSEPHUS$(A, k) \in O(n^2)$

```
1  i ← 0
2  while SizeOf(A) ≠ 1
3        do
4           i ← (i + k) % sizeOf(A)
5           A.delete(i)
6
7  return A[0]
```

1.3 A linear algorithm for doing this

Thanks to wikipedia. Assumes that all individuals are in a 0 indexed sequence of some kind. This function returns the index of the surviving person.

```
F(n, k)
1  if n = 1
2     then return 1
3  return (F(n − 1, k) + k) mod n
```

```
JOSEPHUS(A, k) ∈ O(n)
1  i ← F(SIZEOF(A), k)
2  return A[i]
```

1.4 Roman Nodes

This recursive function will work in linear time. It assumes that all nodes have a rightchild and leftchild pointer as attributes. If called with LABELNODE($T.root, 0$), it will print all non-Roman nodes which are parent to two Roman nodes.

```
LABELNODE(n, s)
 1  if ISLEAF(n)
 2     then
 3          return (s, False)
 4
 5  (r, a) ← LABELNODE(n.rightchild, s + 1)
 6  (l, b) ← LABELNODE(n.leftchild, r + 1)
 7  if ABS((l − r) − (r − s)) ≤ 5
 8     then
 9          return (l,True)
10  if a ∧ b
11     then print n
12  return (l, False)
```

2 EulerTour

Non-recursive function to perform a Euler tour. Assumes a vector implementation of a tree with node access in $O(1)$.

```
EulerTour(T)
 1  m ← 0
 2  n ← 1
 3  while not (n = 1 and m = 3)
 4        do
 5           print T[n]
 6           if n ≥ sizeOf(T) or T[n] = NULL or m = 2n + 1
 7              then
 8                   m ← n
 9                   if n%2
10                      then n ← (n − 1)/2
11                      else  n ← n/2
12
13           elseif m < n
14              then
15                   m ← n
16                   n ← 2n
17           else
18                   m ← n
19                   n ← 2n + 1
20
21
```