The Josephus Game

Given n player sitting in a circle, and a number m.

A hot potato starts at player 1, and is passed around m times. The player holding the potato then is eliminated, the next player gets the potato, and the game continues until only one player is left.

$n = 5,\ m = 2$

What data structure to use?

We need a data structure to store the circle of people.

Required methods:

- Pass the potato to the next person.
- Delete the person holding the potato.

A linked list does it all. A circular linked list would be even better, but Java doesn't provide one, and we can simulate it easily.

```
public static int josephus(int people, int passes)
{
  List<Integer> list = new LinkedList<Integer>();
  for (int i = 1; i <= people; i++)
    list.add(i);
  Iterator<Integer> itr = list.iterator();
  while (people > 1) {
    for (int i = 0; i <= passes; i++) {
      if (!itr.hasNext())
        itr = list.iterator();
      itr.next();
    }
    itr.remove();
    --people;
  }
  itr = list.iterator();
  return itr.next();
}
```

Simulate circular list

Find the winner

Can we do it in less than quadratic time?

Our Josephus program needs $(n-1)m$ link transversals. Can we do it more efficiently?

First observation: If we are currently at position p, then after m passes we will be at position $p + m \bmod n$ (positions numbered from 0 to $n-1$).

Difficulty: How can we maintain the names of the people remaining in the game?

We need a data structure that stores a sequence of n elements, and supports one main operation: Remove the kth element.

No standard Java data structure supports this operation efficiently. We need to implement it ourselves...

The Rank Tree

Supports the following operations:

- Construct from an array with n elements;
- `find(int k)` returns the item at rank (index) k;
- `remove(int k)` removes the element at rank k;
- `size()` returns the current size.

Idea: Store the elements in a binary tree in in-order sequence. Store in each node t the size of the subtree whose root is t.

To find the node with rank k, we just have to follow a path from the root.

How to remove the node t with rank k?

- Easy if t has zero or one subtree;
- If t has two subtrees, then delete the leftmost node in its right subtree instead, and move the element stored there to t.

Rank tree analysis

`find` and `remove` take time $O(h)$, where h is the height of the tree.

When we construct the tree, we can make a perfectly balanced tree.
Its height is $\lceil \log(n+1) \rceil - 1$.

Therefore `find` and `remove` take time $O(\log n)$, and the total running time for the Josephus problem is $O(n \log n)$.