Introduction to Functional Programming
Lent 2005
Suggested Exercises 2

1. **Binary Trees**
Write a function `reverse` which creates the mirror image of a binary tree. That is, if T is a binary tree, then `reverse(`T`)` is a binary tree in which, at every node, left and right branches are interchanged.

Write a function `totree` of type `'a list -> 'a tree` which takes a sorted list and creates a balanced binary tree with the same elements. Use a simple binary tree (not a red-black tree); choose the order in which you insert the elements to ensure that the tree is balanced.

2. **Functional Queues**
The functional queue implementation we discussed can be extended to support a *double-ended queue*, or *deque*, which allows reads and writes to both ends of the queue. The invariant is updated to be symmetric in its treatment of f and r: both are required to be non-empty whenever the deque contains two or more elements. When one list becomes empty, we split the other list in half and reverse one of the halves.

(a) Implement this version of deques, including `cons`, `head`, and `tail`, which insert, inspect, and remove the front element, and `snoc`, `last`, and `init`, which insert, inspect, and remove the rear element.

(b) Prove that each operation takes $O(1)$ amortized run time using the potential function $\Phi(f, r) = abs(|f| - |r|)$. The amortized cost of each operation is the actual cost of that operation plus the change it causes in Φ. For regular queues, the potential function is $\Phi(r) = |r|$.

(Based on Exercise 5.1 from *Purely Functional Data Structures*)

3. **Merge Sort**
Write a generic version of `mergesort`, which takes a comparison function as argument.

4. **Minimum**
Write a functional to compute the minimum value $\min_{i=0}^{m-1} f(i)$ of a function f. Use the functional to express the two dimensional minimum $\min_{i=0}^{m-1} \min_{j=0}^{n-1} g(i, j)$ of a function g of *two* arguments.