Foundations of Algorithms

Homework 0

Divesh Badod

1. Calculate iSort([4, 1, 3, 2]) as in the notes showing every step.

* i(4, iSort([1, 3, 2])

i(4, i(1, iSort([3,2]))

i(4, i(1, i(3, iSort([2])))

i(4, i(1, i(3, i(2, iSort([]))))

i(4, i(1, i(3, [2])))

i(4, i(1, 2 :: i(3,[])))

i(4, i(1 :: 2 :: [3]))

i(4, [1, 2, 3])

1 :: i(4, [2, 3])

1 :: 2 :: i(4[3]))

1 :: 2 :: 3 :: i(4,[])

1 :: 2 :: 3 :: 4 :: []

[1, 2, 3, 4]

2. Look up the selection-sort algorithm. Translate the algorithm into functional pseudo-code. (Note that selection does not require swapping. You may find it helpful to test your code using ALTO.)

* minimum([d]) = d

minimum([d :: ds]) = d if d < minimum([a :: as])

= min(ds) else

delete(d, [d]) = []

delete(d, [a :: as]) = [as] if x == a

= a :: remove(d, [as]) else

selectionSort(d :: ds) = minimum(d :: ds) :: selectionSort(remove(min(d :: ds), a :: as))

9. Consider the following pseudo-code.

b0 = 1

b(n+1) = bn\*b

a. Transform the pseudo-code by adding an accumulation parameter and making it tail- recursive. It should continue to have the form of functional pseudo-code.

* powIt(b, 0, a) = a

powIt(b, n, a) = powIt(b, n -1, b\*a)

b. Transform the tail-recursive functional pseudo-code into imperative pseudo-code. Then transform this imperative pseudo-code into iterative pseudo-code that has no recursive calls.

* Imperative code:

def powIt(b, n, a)

if n = 0

return a

else:

return powIt(b, n – 1, a\*b)

Iterative code:

def powIt(b, n)

a ← 1

while n > 0

n ← n -1

a ← b\*a

return a