1 apply to right weighted tree 1 2 3

On first go apply append recurred left tree branch and then cons entry 1 and recur on right branch

Left branch is null so it returns null to append, right branch is not null so it calls again

On 2nd go apply appends recurred left tree branch and then cons entry 2 and recurs on the right branch

Left branch is null so it returns null to append, right branch is not null so it calls again

On 2nd go apply appends recurred left tree branch and then cons entry 3 and recurs on the right branch

Left branch is null so it returns null to append, right branch is null so it returns append

Append should flatten all this out to 1 2 3

O(2n) or O(n) in this case

Balanced tree 1 2 3

On first go apply append recurred left tree branch and then cons entry 2 and recur on right branch

Left branch is not null so append on recurred left branch and then cons entry 1 and recur on the right branch

Left branch is null so return null to append

Right branch is null so return null to append

Append null onto 1 and null

Return 1 to top level

Right branch is not null so append on recurred left branch and then cons entry 3 and recur on the right branch

Left branch so null is returned to append

Right branch is null so null is returned to append

Append null onto 3 and null

Return 3 to top level

Append and flatten all to 1 2 3

This appears to be O(n^2)

Not null so first time calls copy to list on left branch and “cons 7 on to copy to list of right branch1 with a null argument as left result list with a null result-list as right result-list

Second time is also not null so calls copy to list on left branch and “cons 3 on copy to list of right 2nd branch with: cons entry prime 7 on to copy to list of right branch prime with: null result” as result-list for both branches

Third time is not null so calls copy to list on left branch and “cons entry 1 on copy to list of right 3rd branch with: cons entry 3 on to copy list of right branch of right branch 2 with: cons entry prime 7 on to copy to list of right branch prime with: null result”

Left branch returns list result “cons entry 1 on copy to list of right branch 3 with: cons entry 3 on to copy list of right branch of right branch 2 with: cons entry prime 7 on to copy to list of right branch prime with: null result”

short version

Right prime branch is null so returns null to “cons entry 1 on copy to list of right branch 3 with: cons entry 1 on to copy list of right branch of right branch 2 with: cons 7 null” as the result-list

Right 2nd branch is not null so calls copy to list on left branch and “cons 5 on to copy to list of right 2nd  right branch branch with “cons entry 1 on copy to list of right branch 3 with: cons entry 3 on to copy list of right branch of right branch 2 with: cons 7 null” as the result-list

both children are null so returns “cons entry 1 on copy to list of right branch 3 with: cons entry 3 on to copy list of right branch of right branch 2 with: cons 5: cons 7 null

(define (tree->list-1 tree)

(if (null? tree)

'()

(append

(tree->list-1 ;append left branch

(left-branch tree))

(cons (entry tree) ;to cons of entry

(tree->list-1 ;to right branch

(right-branch tree))))))

(define (tree->list-2 tree)

(define (copy-to-list tree result-list) ;iterative

(if (null? tree) ;if tree/branch null

result-list ;return result list

(copy-to-list

(left-branch tree) ;left branch as tree and result-list returns

(cons (entry tree) ; from entry + right branch result list

(copy-to-list ;pass right branch and result list

(right-branch tree)

result-list)))))

(copy-to-list tree '()))

Exercise 2.64

Takes a list of elements and gets their length and passes them to partial-tree. If count is zero, partial tree returns ‘() consed onto elts. Otherwise left-size is calculated so that it is the number of elements left of the mid-point, or in the case of an even amount, one less than n/2. Left-result is calculated using leftsize. Left tree is stored as the car of left-result. Non-left elts are taken from the cdr of left result. This node’s entry is taken from the car of the non-left-etls. Right size is calculated so that it is the remainder of elements to the right of the midpoint or is n/2. Right-result is calculated using right-size and the cdr of non-left elts. Right tree is the car of right-result and right-results is the cdr. Remaining-elts is the cdr of right-results. Finally cons make tree of all the tree parts onto the remaining elts.

Calls partial-tree with 1 3 5 7 9 11 and count 6

-Calls left partial-tree with 1 3 5 7 9 11 and count 2

--Calls left partial-tree with 1 3 5 7 9 11 and count 0

---returns (‘() 1 3 5 7 9 11)

--entry 1

--Calls right partial-tree with 3 5 7 9 11 and count 1

---Calls left partial-tree with 3 5 9 7 11 and count 0

----returns (‘() 3 5 9 7 11)

---entry 3

---Calls right partial tree with 5 9 7 11 and count 0

----returns (‘() 5 9 7 11)

---(cons (make-tree (3’() ‘()) (5 9 7 11))

---(cons (make-tree(1 ‘() ‘(3()()) (5 9 7 11)

--cons(make-tree 1 ‘()’())

-entry = 5

-Calls right partial-tree with 7 9 11 and count 3

--Calls left partial-tree with 7 9 11 and count 1

---Calls left partial-tree with 7 9 11 and count 0

----returns (‘() 7 9 11)

---entry = 7

---calls right partial-tree with 9 11 and count 0

---- returns (‘() 9 11)

---cons (make-tree ‘() 7‘())

--- return TREE (7‘() ‘()) 9 11

--entry 9

--Calls Right partial-tree with 11 and 1

---receives TREE(11 ‘() ‘())

--(cons (make-tree (9 Tree 7()() Tree 11()())

-Cons (make-tree ( 5 (1 ‘() ‘(3()()))((9 Tree 7()() Tree 11()())

Or something like that