Solutions

1 Assignment 1

1.1 Solving Recurrences

1. $C_N = C_{\frac{N}{2}} + N$ **for** $N \geq 2$ **with** $C_1 = 0$
   The recurrence is defined when $N = 2^n$,i.e, when N is a power of 2 In fact:

$$\begin{aligned} C_N = C_{\frac{N}{2}} + N &= C_{\frac{N}{4}} + \frac{N}{2} + N \\ &= C_{\frac{N}{2^n}} + \frac{N}{2^{n-1}} + \cdots + \frac{N}{2} + N \\ &= C_1 + N(2(1 - \frac{1}{2^n})) \\ &= 2N \end{aligned}$$

   If the sum $N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \cdots$ is infinite, it evaluates to exactely $2N$. Since we stop at 1, this value is an approximation to the exact answer. The precise solution involves properties of the binary representation of N.

2. $C_N = 2C_{\frac{N}{2}} + N$ **for** $N \geq 2$ **with** $C_1 = 0$
   Again, the recurrence is precisely defined only when N is a power of 2:

$$\begin{aligned} C_{2^n} &= 2C_{2^{n-1}} + 2^n \\ \frac{C_{2^n}}{2^n} &= \frac{C_{2^{n-1}}}{2^{n-1}} + 1 \\ &= \frac{C_{2^{n-2}}}{2^{n-2}} + 1 + 1 \\ &\vdots \\ &= n \end{aligned}$$

   So the recurrence is about $N \log N$

3. $C_N = 2C_{\frac{N}{2}} + 1$ **for** $N \geq 2$ **with** $C_1 = 1$

$$\begin{aligned} C_N &= 2C_{\frac{N}{2}} + 1 \\ &= 2(2C_{\frac{N}{2}} + 1) + 1 \\ &\vdots \\ &= 2^n C_{\frac{N}{2^n}} + 2^{n-1} + 2^{n-2} + \cdots + 2 + 1 \\ &= 2^n C_1 + 2^{n-1} + 2^{n-2} + \cdots + 2 + 1 \\ &= 2^n + 2^{n-1} + 2^{n-2} + \cdots + 2 + 1 \\ &= 2^{n+1} - 1 \\ &= 2N - 1 \end{aligned}$$

The recurrence evaluates then to $2N$ (when N is a power of 2).

1.2 Some recursive algorithms

1. • **factorial function (iterative version)**

   Algorithm 1. ***factI(n)***

   (a) *int fact = 1;*

   (b) *int i = 2;*

   (c) *while (i $\leq$ n)*
   fact = fact $\times$ i++;

   (d) *return fact;*

   • **factorial function (recursive version)**

   Algorithm 2. ***factR(n)***

   (a) *(if n == 0) then return* 1

   (b) *else return factR(n-1)$\times$ n;*

   • **Cost**: solving the recurrence $C_N = C_{N-1}$ results in $C_N = N$.

2. • **gcd function (iterative version)**

   Algorithm 3. ***gcdI(a,b)***

   (a) *int r ;*

   (b) *while (b != 0)*
   – *r = a % b;*
   – *a = b ;*
   – *b = r;*

   (c) *return (a);*

   • **gcd function (recursive version)**

Algorithm 4. ***gcdR(a,b)***

(a) (if b == 0) then return a

(b) else return gcd(b,a % b);

2 Assignment 2

2.1 Recursive lists

1. **Concatenation of lists**
   We define concat: list $\times$ list $\rightarrow$ list as the concatenation of two lists.

   - length(concat(l, m)) = length(l) + length(m)
   - i^{th}(concat(l, m), j) = i^{th}(l,j) if $1 \leq$ j $\leq$ length(l), i^{th}(m,j-length(l)) otherwise.
   - concat(empty_list, l) = l
   - concat(cons(e, l), m) = cons(e,concat(l,m))

2. **Search of an element (present) in a list**
   We define search: list $\times$ element $\rightarrow$ position. Complete the following axioms:

   - content(search(l, e))= e
   - search(cons(e,l), e) = head(cons(e,l))
   - e $\neq$ f, search(cons(e, l), f)= succ(search(l,f))

2.2 Mathematical properties of binary trees

Property 1. *A binary tree with N internal nodes has 2N links: N-1 links to internal nodes and N+1 links to external nodes.*

Proof. In any tree, each node except the root, has a unique parent, and every edge connects a node to its parent, so there are N-1 links connecting internal nodes. Similarly, each of the N +1 external nodes has one link, to its unique parent. $\square$

2.3 Assignment 3

sectionMathematical properties of binary trees

Property 2. *The internal path length of a binary tree with N internal nodes is at least $N \log(\frac{N}{4})$ and at most $N(N-1)/2$*

Proof. The worst case and best case are achieved for the same trees referred to in the discussion of a binary tree's height's bounds, namely, the degenerate tree and the balanced tree.
The internal path length of the worst case tree is $0 + 1 + \cdots + N - 1 = N(N-1)/2$.
The best case tree has $N + 1$ external nodes at height no more that $\log N$. Multiplying these and applying the property that relates the external path of tree with its internal path we get the bound $(N+1)\log N - 2N < Nlog(\frac{N}{4})$ $\square$

2.4 Tree traversal

- **Preorder: node left right**
  $n_0 n_1 n_3 n_6 n_7 n_4 n_8 n_{10} n_{12} n_2 n_5 n_9 n_{11} n_{13}$
- **Inorder: left node right**
  $n_6 n_3 n_7 n_1 n_{12} n_{10} n_8 n_4 n_0 n_5 n_9 n_{13} n_{11} n_2$
- **Postorder: left right node**
  $n_6 n_7 n_3 n_{12} n_{10} n_8 n_4 n_1 n_{13} n_{11} n_9 n_5 n_2 n_0$

3 Assignment 4

3.1 Sorting using a BST

- The tree traversal suitable in this case is the inorder one: it will give the following sequence: a a e e g i l m n o p r s t x.
- A sorting method consists on building a BST using successive insertions at the leaf and then using the inorder traversal. The first operation takes O(height of the tree * n), where n is the size of the array to be sorted. The traversal takes O(n).

3.2 Remove operation in a BST

Algorithm 5. ***remove: tree*** $\times$ ***item*** $\rightarrow$ ***tree***

1. $remove(< x, emptyTree(), emptyTree() >, x) = emptyTree()$
2. $remove(< x, emptyTree(), d\ , x) = d$
3. $remove(< x, g, emptyTree()\ , x) = g$
4. *if* g *and* d *are different from the empty tree then* $remove(< x, g, d >, x) =< max(g), m\bar{a}x(g), d >$

Algorithm 6. ***max: tree*** $\rightarrow$ ***node***

1. $max\ (< r, g, emptyTree() >) = r.$
2. *if* $d \neq emptyTree()$, *then* $max(< r, g, d >) = max(d)$

Algorithm 7. $m\bar{a}x$***: tree*** $\rightarrow$ ***tree***

1. $m\bar{a}x(< r, g, emptyTree() >) = g$
2. *if* $d \neq emptyTree()$ *then* $m\bar{a}x(< r, g, d >) =< r, g, m\bar{a}x(d) >$

3.3 Quicksort: example

A S O R T I N G E X A M P L E
A A O R T I N G E X S M P L E
A A M E L I N G E O S X P T R
A A G E L I E M N O P R S T X
A A E E G I L M N O P R S T X