CSC 225

Algorithms and Data Structures I

Rich Little
rlittle@uvic.ca
ECS 516

Recursion

- **Recursion** in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem.
- Most computer programming languages support recursion by allowing a function to call itself within the program text.
- There are 3 important rules of thumb:
    1. The recursion has a *base case*.
    2. The recursive calls must *converge* to the base case.
    3. The subproblems must be *contained* in the bigger problem.

Iterative Algorithm for Factorial

Basic units: A & C – ignore 1st A and last C in loop

```
Algorithm factorialIterative(n)
   Input: Integer n ≥ 1.
   Output: n!

1. result ← 1        1A
2. for i ← 2 to n do
3.     result ← i * result   1A
                i = i+1
4. end
5. return result     1A
```

$$T(n) = 1 + \sum_{i=2}^{n} (3) + 1$$

$$= 2 + 3 \sum_{i=2}^{n} 1$$

$$= 2 + 3(n-2+1)$$

$$= 3n-1$$

$$\boxed{\begin{array}{l} \mathbf{T(n) = 2 + 3(n-1)} \\ \mathbf{T(n) = 3n - 1} \end{array}}$$

Recursive Algorithm for Factorial

Basic units: A & C

Let $T(n)$ runtime for n

```
Algorithm factorialRecursive(n)
   Input: Integer n ≥ 1.
   Output: n!

   if n = 1 then
      return 1
   else
      return n * factorialRecursive(n-1)
   end
```

1C

1A

$n = 1$

1C

1A

$T(n-1)$

Recurrence Equation

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 2 & n \geq 2 \end{cases}$$

Solving Recurrence Equations by Repeated Substitution (bottom-up)

$$T(n) = T(n-1) + 2$$
$$T(n-1) = T(n-2) + 2$$
$$T(n-2) = T(n-3) + 2$$
$$\ldots$$
$$T(2) = T(1) + 2$$
$$T(1) = 2$$

$T(1) = 2$

$T(2) = T(1) + 2 = 2 + 2 = 4 = 2(2)$

$T(3) = T(2) + 2 = 4 + 2 = 6 = 2(3)$

$T(4) = T(3) + 2 = 6 + 2 = 8 = 2(4)$

$\vdots$

$T(i) = T(i-1) + 2 = 2(i)$

$\vdots$

$T(n) = T(n-1) + 2 = 2(n) = 2n$

$\boxed{2n}$

Solving Recurrence Equations by Repeated Substitution (top-down)

$$T(n) = T(n-1) + 2$$
$$T(n-1) = T(n-2) + 2$$
$$T(n-2) = T(n-3) + 2$$
$$\ldots$$
$$T(2) = T(1) + 2$$
$$T(1) = 2$$

$T(n) \overset{1^{st}}{=} T(n-1) + 2$

$2^{nd} = [T(n-2) + 2] + 2$

$3^{rd} = [T(n-3) + 2] + 2 + 2$

$\vdots$

$i^{th} = \boxed{T(n-i) + 2i}$

when does $n - i = 1$? $\boxed{i = n-1}$

$T(n) = T(n-(n-1)) + 2(n-1)$

$= T(1) + 2(n-1)$

$= 2 + 2(n-1) = 2 + 2n - 2$ $\boxed{T(n) = 2n}$

Structure of a Recursive Algorithm

```
Algorithm recursiveAlgorithm(n)
   if n = 1 then
      base-case
   else
      induction-step
      recursiveAlgorithm(n-1)
   end
```

- Let the worst case running time of `recursiveAlgorithm` be T(n)

- Then $$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ T(n-1) + c_2 & \text{otherwise} \end{cases}$$

Recall Iterative arraryMax Algorithm

```
Algorithm arrayMax(A,n):
  Input: An array A storing n ≥ 1
         integers
  Output: The maximum element in A

  currentMax ← A[0]
  for k ← 1 to n-1 do
     if currentMax < A[k] then
         currentMax ← A[k]
  return currentMax
```

$\boxed{T(n) = 7n - 2}$

Recursive arrayMax Algorithm

$T(max()) = 1$

```
Algorithm recursiveMax(A,n)
   Input: An array A storing n ≥ 1 integers.
   Output: The maximum element in A.

   if n = 1 then
       return A[0]
   else
       return max(recursiveMax(A,n−1),A[n−1])
   end
```

1 3

4? 6? 5?

1 1 1 1 1 1

$$T(n) = \begin{cases} 3, & n = 1 \\ T(n-1) + 7, & n \geq 2 \end{cases}$$

Counting a Recursive Algorithm

- **Base case:** 3 operations (n=1, A[0], return)
- **Induction step:** T(n-1)+7 ops (n=1, n-1, n-1,A[n-1], call, max, ret)

Recurrence Equation

$$\boldsymbol{T(n) = \begin{cases} 3 & n = 1 \\ T(n-1) + 7 & n \geq 2 \end{cases}}$$

Solving Recurrence Equation by Repeated Substitution

$$T(n) = T(n-1) + 7$$
$$T(n-1) = T(n-2) + 7$$
$$T(n-2) = T(n-3) + 7$$
$$\ldots$$
$$T(2) = T(1) + 7$$
$$T(1) = 3$$

$$T(n) = \underbrace{T(n-1)} + 7$$
$$= (T(n-2) + 7) + 7$$
$$= T(n-3) + 3(7)$$
$$\vdots$$
$$= T(n-i) + 7i$$
$$\vdots$$

$n - i = 1?$

$i = n-1$

$$T(n) = T(1) + 7(n-1)$$
$$= 3 + 7(n-1)$$
$$= \boxed{7n - 4}$$

Towers of Hanoi - Recursive Algorithm

```
Algorithm tohRecursive(n,A,B,C):
   Input: Integer n ≥ 1 (disks) pegs A, B, C
   Output: n disks from A to C in min moves

   if n=1 then
       move(A,C)
   else
       tohRecursive(n-1,A,C,B)
       move(A,C)
       tohRecursive(n-1,B,A,C)
   end
```

Recurrence Equation

$$\boxed{T(n) = \begin{cases} 1 & n = 1 \\ 2T(n-1) + 1 & n \geq 2 \end{cases}}$$

Solving Recurrence Equation by Repeated Substitution

$T(n) = 2T(n-1) + 1$

$T(n-1) = 2T(n-2) + 1$

$T(n-2) = 2T(n-3) + 1$

...

$T(2) = 2T(1) + 1$

$T(1) = 1$

$$T(n) = 2T(n-1) + 1$$

$$= 2[2T(n-2) + 1] + 1$$

$$= 2^2 T(n-2) + 2 + 1$$

$$= 2^2[2T(n-3) + 1] + 2 + 1$$

$$= 2^3 T(n-3) + 2^2 + 2^1 + 1 \leftarrow 2^0$$

$$\vdots$$

$$T(n) = 2^i T(n-i) + 2^{i-1} + \cdots + 2^2 + 2^1 + 2^0$$

$$\vdots$$

$$T(n) = 2^{n-1} + 2^{n-2} + \cdots + 2^0$$

$$= \sum_{i=0}^{n-1} 2^i = \frac{1 - 2^{(n-1)+1}}{1-2}$$

$n - i = 1$?

$\boxed{i = n-1}$

$$\boxed{T(n) = 2^n - 1}$$

The Principle of Induction

- Let $S_1, S_2, S_3, \ldots$ be statements such that
  1. S_1 is true; and
  2. Whenever S_k is true, where $k \in \mathbb{N}$, then S_{k+1} is true.

  Then all of the statements $S_1, S_2, S_3, \ldots$ are true.

The Principle of Induction

Ex 5: Show that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ is true for all integers $n \geq 1$.

S_j

B.C.: S_1 w.t.s. $\sum_{i=1}^{1} i = \frac{1(1+1)}{2}$?

$\sum_{i=1}^{1} i = 1 \qquad \frac{1(1+1)}{2} = 1$ ✓

I.H.: Let $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$ be true. $(k \geq 2)$ $\therefore$ $k \geq$

I.S.: $\sum_{i=1}^{k+1} i = \underbrace{\sum_{i=1}^{k} i}_{\text{by I.H.}} + (k+1) = \frac{k(k+1)}{2} + (k+1)$

$= \frac{k(k+1) + 2(k+1)}{2} = \frac{(k+1)(k+2)}{2}$ ✓

The Principle of Induction

Ex 6: Prove that the closed form of the Towers of Hanoi equation is $T(n) = 2^n - 1$.

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T(n-1)+1, & n \geq 2 \end{cases}$$

B.C. $n = 1$

$T(1) = 1$, $T(1) = 2^1 - 1 = 1$

I.H.: Assume true for $n = k$, $k \geq 1$. i.e. $T(k) = 2^k - 1$

I.S.: Let $n = k+1$. $T(k+1) = 2T(k) + 1$

$= 2(2^k - 1) + 1 = 2^{k+1} - 2 + 1$

$= 2^{k+1} - 1$

$\therefore$ by induction $T(n) = 2^n - 1$ for $n \geq 1$

The Strong Form of Induction

- Let $S_1, S_2, S_3, \ldots$ be statements such that
  1. S_1 is true; and (sometimes more)
  2. Whenever S_i is true for all i such that $1 \leq i \leq k$, where $k \in \mathbb{N}$, then S_{k+1} is true.

  Then all of the statements $S_1, S_2, S_3, \ldots$ are true.

The Strong Form of Induction

Ex 7: Consider the Fibonacci sequence 1,1,2,3,5,8,13,…, which can be given by the recurrence equation

$$T(n) = \begin{cases} 1, & \text{if } n = 0 \text{ or } n = 1 \\ T(n-1) + T(n-2), & \text{if } n \geq 2 \end{cases}$$

Prove that $T(n) \leq 2^n$ for $n \geq 0$.

B.C.: $T(0) = T(1) = 1 \leq 1$

$T(0) = 2^0 = 1$ $T(1) = 2^1, \; 1 \leq 2$

I.H.: Suppose $T(i) \leq 2^i$ for $0 \leq i \leq k$

I.S.: $n = k+1$: $T(k+1) = T(k) + T(k-1)$

$\leq 2^k + 2^{k-1}$

$\leq 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$

Loop Invariants

- What is a loop invariant?
- An **invariant** is a property that is always true at particular points in a program.
- A **loop invariant** is a property that is true before (and after) each iteration of a loop

Loop Invariants

- To prove some statement S, about a loop, is correct, define S in terms of smaller statements $S_0, S_1, \ldots, S_k$ where
  1. S_0 is true before the loop.
  2. S_{i-1} is true before iteration i, then show that S_i is true after iteration i.
  3. Thus, S_k implies S is true by induction.

Loop Invariants

- In general, a loop invariant consists of the following cases:
    1. **Base case (initialization):** prove the invariant holds (is true) before the loop starts
    2. **Inductive step (maintenance):** prove that ***if*** the invariant holds right before beginning iteration i (inductive hypothesis), ***then*** it must also hold at the end of that iteration (right before the next iteration, $i + 1$)
    3. **Termination:** make sure the loop will eventually end (with the invariant holding)

Loop Invariants

Ex 8: Prove `arrayMax(A,n)` is correct.

S: arrayMax() returns the maximum value in array A.

S_j: currentMax is the maximum value from A[0] to A[j]

B.C.: S_0: currentmax is the max. value from A[0] to A[0].

true because currentMax ← A[0]

I.H.: For some $n-1 \geq i-1 \geq 0$

S_{i-1} is true.

Algorithm arrayMax(A, n)

   Input: An array A storing $n \geq 1$ integers

   Output : The maximum element in A

   $currentMax \leftarrow A[0]$ ←

   for k ← 1 to n − 1 **do**

      if $currentMax < A[k]$ **then**

         $currentMax \leftarrow A[k]$

      end

   end

   return $currentMax$

Loop Invariants

Loop terminates when $k = n$ & S_{n-1} is true

i.e. currentMax is the max value of $A[0]$ to $A[n-1]$. ∴

That is, currentMax is the max value from $A[0]$ to $A[i-1]$.

∴ currentMax $\geq A[0]$ to $A[i$

after ite

i.

I.S.: Show that S_i is true.

currentMax $< A[i]$ → 2 cases

True: set currentMax $\leftarrow A[i]$

∵ $A[i-1]$ by I.H. (S_i is

$A[i] >$ currentMax $\geq A[0]$,

∴ currentMax $\geq A[0], \ldots, A[i]$

False: currentMax $\geq A[i]$

by I.H. currentMax $\geq A[0]$ to $A[i-1]$

Algorithm arrayMax(A, n)

Input: An array A storing $n \geq 1$ integers

Output : The maximum element in A

$currentMax \leftarrow A[0]$

for k ← 1 to n − 1 **do**

  if $currentMax < A[k]$ **then**

    $currentMax \leftarrow A[k]$

  end

end

return $currentMax$