Recurrence Relations – Running Time for Recursive Functions

Gnome Sort - *trivia*

http://www.portlandoctopus.com/top-5-garden-gnomes/

Divide and Conquer Algorithms

- Analysis of divide and conquer algorithms requires knowledge of:
  - Mathematical Induction
  - Substitution Method
  - Recurrences

```
class Tree
{
public:
   // returns true if t represents a binary
   // search tree containing no duplicate values;
   bool IsBST();

   // return true if & only if all values in the tree are
   // less than val. Running time for next 2 functions is n
   bool isLessThan(int val);
   // see above
   bool isGreaterThan(int val);

private:
   int mInfo;
   Tree * mpsLeft;
   Tree * mpsRight;
};
```

Thank you Owen Astrachan

```
   // returns true if t represents a binary
   // search tree containing no duplicate values;

bool IsBST()

{
   bool bLeftIsTree = true, bRightIsTree = true;
   bool bLessThan = true, bGreaterThan = true;
   if( t->left )
   {
     bLeftIsTree = t->left->IsBST();
     bLessThan = t->left->isLessThan(t->info);

   }
   if( t->right )
   {
     bRightIsTree = t->right->IsBST();
     bGreaterThan = t->right->isGreaterThan(t->info);
   }
   return bLessThan &&

          bGreaterThan &&

          bLeftIsTree &&

          bRightIsTree;

}             // Complexity with n nodes in the tree?
```

What is the complexity of IsBST()?

Another Example

- What is the asymptotic complexity of the function below? Assume Combine is O(n)

```
// postcondition: a[left] <= ... <= a[right]
void DoStuff(vector<int> & a, int left, int right)
{
  int mid = (left + right)/2;
  if (left < right)
  {
    DoStuff(a, left, mid);
    DoStuff(a, mid + 1, right);
    Combine(a, left, mid, right);
  }
}
```

Recurrence Relation

- A ***recurrence relation*** contains two equations
  - One for the general case
  - One for the base case

Merge Sort

```
MERGE-SORT(A, p, r) // A:Array; p,r: ints
 // p & r are indices into the array (p < r)
 if p < r                        //Check for base case
   q = ⌊(p + r) / 2⌋             //Divide
   MERGE-SORT(A, p, q)           //Conquer
   MERGE-SORT(A, q + 1, r)       //Conquer
   MERGE(A, p, q, r)             //Combine
```

Recurrence Relation

- Let T(n) be the time for Merge-Sort to execute on an n element array.
- The time to execute on a one element array is O(1)
- Then we have the following relationship:

Merge Sort – Substitution Method

- To solve the recurrence relation we'll write n instead of O(n) as it makes the algebra simpler:

  $T(n) = 2\,T(n/2) + n$

  $T(1) = 1$

- Derive a solution using the iteration method
  - Hope that you find a pattern
- Prove the solution using induction

Substitution Method

- Derive a solution to a recurrence using iteration of back substitution
- Prove

Your Turn

- Use the iteration method to derive a solution to the recurrence below:

  $T(n) = T(n/2) + c$

  $T(1) = 1$

Recurrence Relations to Remember

$T(n) = T(n/2) + O(1)$		
$T(n) = T(n-1) + O(1)$		
$T(n) = 2\ T(n/2) + O(1)$		
$T(n) = T(n-1) + O(n)$		
$T(n) = 2\ T(n/2) + O(n)$		

Approaches to Algorithm Design

- Incremental
  - Job is partly done – do a little more, repeat until done.
- Divide-and-Conquer (recursive)
  - Divide problem into sub-problems of the same kind.
  - For small subproblems, solve, else, solve them recursively.
  - Combine subproblem solutions to solve the whole thing.

For Next Time

- So far we've covered chapters 1, 2, 3, and part of 4.