Recurrence Relations

Get out pen and paper!

Announcements

- Boggle
  - due November 11

- Exam 2
  - November 13

- Review – Monday in class
- Review – Monday night (Jimmy)
Running time

• How “fast” is my algorithm?
  • in terms of \( n \) – the length of input

• Big–Oh – the growth rate as a function of \( n \)
  • \( O(N^2) \)
  • \( O(N \log N) \)

<table>
<thead>
<tr>
<th>Sorting Algorithm</th>
<th>Best</th>
<th>Average</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heapsort</td>
<td>( O(n\log n) )</td>
<td>( O(n\log n) )</td>
<td>( O(n\log n) )</td>
</tr>
<tr>
<td>Mergesort</td>
<td>( O(n\log n) )</td>
<td>( O(n\log n) )</td>
<td>( O(n\log n) )</td>
</tr>
<tr>
<td>Quicksort</td>
<td>( O(n\log n) )</td>
<td>( O(n\log n) )</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>Bubblesort</td>
<td>( O(n) )</td>
<td>( O(n^2) )</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>Insertionsort</td>
<td>( O(n) )</td>
<td>( O(n^2) )</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>Bogosort</td>
<td>( O(n) )</td>
<td>( n \ast n! )</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>
Warm Up

// Assume strings has length n.
// Assume "robot" appears in strings.

public int findRobot(String[] strings) {

    int current = 0;
    while (!strings[current].equals("robot")) {
        current++;
    }
    return current;
}

Warm Up

// Return index of v in sorted array of positive numbers,
// or -1 if not there.
// Search between the indices low and high.

public int findInSorted(int[] sorted, int v, int low, int high) {

    while (low < high) {
        int midpoint = (low + high) / 2;
        if (v < sorted[midpoint]) {
            high = midpoint; // Search in the lower half.
        } else if (v > sorted[midpoint]) {
            low = midpoint + 1; // Search in the upper half.
        } else {
            return midpoint;
        }
    }
    return -1;
}
// Is the value v contained in the binary search tree rooted at node?

public boolean BSTcontainsValue(int v, TreeNode node) {
    if (node == null) {
        return false;
    }

    if (node.value == v) {
        return true;
    }

    if (v < node.value) {
        return BSTcontainsValue(v, node.left);
    } else {
        return BSTcontainsValue(v, node.right);
    }
}
Recurrence Relations

- How to calculate the Big-Oh of a recursive function
  - Write the recurrence relation
  - Solve the recurrence relation
  - Compute the Big-Oh

Problem 1

//your tree is balanced

```java
public int height(TreeNode node) {
    if (node == null) {
        return 0;
    }
    int leftHeight = height(node.left);
    int rightHeight = height(node.right);
    return Math.max(leftHeight, rightHeight) + 1;
}
```
Problem 2

//your tree is not balanced

public int height(TreeNode node) {
    if (node == null) {
        return 0;
    }
    int leftHeight = height(node.left);
    int rightHeight = height(node.right);
    return Math.max(leftHeight, rightHeight) + 1;
}

Problem 3

//your tree is balanced

public boolean isBalanced(TreeNode node) {
    int left = height(node.left);
    int right = height(node.right);
    if (Math.abs(left - right) > 1) {
        return false;
    }
    return (isBalanced(node.left) &&
            isBalanced(node.right));
}
Problem 4

// your tree is not balanced

public boolean isBalanced(TreeNode node) {
    int left = height(node.left);
    int right = height(node.right);
    if (Math.abs(left - right) > 1) {
        return false;
    }

    return (isBalanced(node.left) &&
            isBalanced(node.right));
}

Problem 5

public int maximum(int[] values, int low, int high) {
    if (low == high) {
        return values[low];
    }

    int mid = (low + high) / 2;
    return Math.max(maximum(values, low, mid),
                    maximum(values, mid+1, high));
}
Problem 6

// Reverse the array values, between the indices low and high.

public static void reverse(int[] values, int low, int high) {
    if (low >= high) {
        return;
    }
    int temp = values[low];
    values[low] = values[high];
    values[high] = temp;
    reverse(values, low+1, high-1);
}

Recurrence Relations

<table>
<thead>
<tr>
<th>Recurrence</th>
<th>Example</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>T(n) = T(n/2) + O(1)</td>
<td>Binary Search</td>
<td>O(log n)</td>
</tr>
<tr>
<td>T(n) = T(n-1) + O(1)</td>
<td>Linear Search</td>
<td>O(n)</td>
</tr>
<tr>
<td>T(n) = 2T(n/2) + O(1)</td>
<td>Tree traversal</td>
<td>O(n)</td>
</tr>
<tr>
<td>T(n) = 2T(n/2) + O(n)</td>
<td>QuickSort</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>T(n) = T(n-1) + O(n)</td>
<td>BubbleSort</td>
<td>O(n^2)</td>
</tr>
</tbody>
</table>
Recurrence Relations

• How to calculate the Big-Oh of a recursive function
  • Write the recurrence relation
  • Solve the recurrence relation
  • Compute the Big-Oh
    • Use look-up table