


INEFFECTIVE SORTS

<pre> DEFINE HALHEARTEDMERGESORT(LIST): IF LENGTH(LIST) < 2: RETURN LIST PIVOT = INT(LENGTH(LIST) / 2) A = HALHEARTEDMERGESORT(LIST[:PIVOT]) B = HALHEARTEDMERGESORT(LIST[PIVOT:]) // UMMMMMM RETURN [A, B] // HERE. SORRY. </pre>	<pre> DEFINE FASTBOGOSORT(LIST): // AN OPTIMIZED BOGOSORT // RUNS IN O(N LOG N) FOR N FROM 1 TO LOG(LENGTH(LIST)): SHUFFLE(LIST): IF ISSORTED(LIST): RETURN LIST RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)" </pre>
<pre> DEFINE JOBININTERVIEWQUICKSORT(LIST): OK SO YOU CHOOSE A PIVOT THEN DIVIDE THE LIST IN HALF FOR EACH HALF: CHECK TO SEE IF IT'S SORTED NO, WAIT, IT DOESN'T MATTER COMPARE EACH ELEMENT TO THE PIVOT THE BIGGER ONES GO IN A NEW LIST THE EQUAL ONES GO INTO, UH THE SECOND LIST FROM BEFORE HANG ON, LET ME NAME THE LISTS THIS IS LIST A THE NEW ONE IS LIST B PUT THE BIG ONES INTO LIST B NOW TAKE THE SECOND LIST CALL IT LIST, UH, A2 WHICH ONE WAS THE PIVOT IN? SCRATCH ALL THAT IT JUST RECURSIVELY CALLS ITSELF UNTIL BOTH LISTS ARE EMPTY RIGHT? NOT EMPTY, BUT YOU KNOW WHAT I MEAN AM I ALLOWED TO USE THE STANDARD LIBRARIES? </pre>	<pre> DEFINE PANICSORT(LIST): IF ISSORTED(LIST): RETURN LIST FOR N FROM 1 TO 10000: PIVOT = RANDOM(0, LENGTH(LIST)) LIST = LIST[:PIVOT] + LIST[PIVOT:] IF ISSORTED(LIST): RETURN LIST IF ISSORTED(LIST): // THIS CAN'T BE HAPPENING RETURN LIST IF ISSORTED(LIST): // COME ON COME ON RETURN LIST // OH JEEZ // I'M GONNA BE IN SO MUCH TROUBLE LIST = [] SYSTEM("SHUTDOWN -H +5") SYSTEM("RM -RF /*") SYSTEM("RM -RF /*") SYSTEM("RM -RF /*") SYSTEM("RM /S /Q C:*") // PORTABILITY RETURN [1, 2, 3, 4, 5] </pre>


Before Class: Get out pen and paper



Announcements

- Exams
 - will be returned on Wednesday
- Boggle
 - due April 4
- Apt Set 6 – Posted tomorrow
 - due April 9


2



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
 - $O(n^2)$
 - $O(n \log n)$

3




Running time

- A metric for comparison

Sorting Algorithm	Best	Average	Worst
Heapsort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Mergesort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quicksort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Bubblesort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertionsort	$O(n)$	$O(n^2)$	$O(n^2)$
Bogosort	$O(n)$	$n * n!$	∞


4



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;  
  
}
```


5



Warm Up

```
// Return index of v in sorted array, 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;  
  
}
```

6



???


```
// 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);
    }
}
}
```

7



???

```
// 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);
    }
}
}
```

What if the tree is
balanced?

What if the tree is
unbalanced?


8



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

9



Recurrence Relations

Recurrence	Example	Running Time
$T(n) = T(n/2) + O(1)$	Binary Search	$O(\log n)$
$T(n) = T(n-1) + O(1)$	Linear Search	$O(n)$
$T(n) = 2T(n/2) + O(1)$	Tree traversal	$O(n)$
$T(n) = 2T(n/2) + O(n)$	QuickSort	$O(n \log n)$
$T(n) = T(n-1) + O(n)$	BubbleSort	$O(n^2)$

10



Problem 1

```
//your tree is 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;
}
```

Submit answers here: <http://goo.gl/ltZJZ>

11



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;
}
```

Submit answers here: <http://goo.gl/ltZJZ>

12



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));

}

Submit answers here: http://goo.gl/ltZJZ
```

13



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));

}

Submit answers here: http://goo.gl/ltZJZ
```

14



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));  
  
}
```

Submit answers here: <http://goo.gl/ltZJZ>

15



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);  
  
}
```

Submit answers here: <http://goo.gl/ltZJZ>

16



Today

- You learned a new tool
 - Recurrence relations - how to calculate Big-Oh for recursive functions
- Exams
 - will be returned on Wednesday
- Boggle
 - due April 4
- Apt Set 6
 - due April 9