CSE 250 Recitation

Mar 6 - 7: Recursion

Binary Search

The binary search algorithm let's us effectively search a List

To work correctly and efficiently the List must:

- Be sorted
- Allow constant time random access (ie an Array)

It works by comparing our target to the midpoint, then searching **only** the left half or the right half

Binary Search Code

```
int binarySearch(ArrayList<T> list, T target) {
  return binarySearch(list, target, 0, list.size() - 1);
}
// Searches the array from [start, end], returns -1 if target not found
int binarySearch(ArrayList<T> list, T target, int start, int end) {
  if (start > end) { return -1; }
  int mid = (start + end) / 2;
  T guess = list.get(mid);
  if(guess.equals(target)){ return mid; } // We found our target!
  else if(target.compareTo(guess) < 0) {  // Target is in the left half
    return binarySearch(list, target, start, mid - 1);
  } else {                                // Target is in the right half
    return binarySearch(list, target, mid + 1, end);
  }
```

Exercise: Determine the growth function for the runtime of binarySearch

Runtime Growth Function

$$T(N) = \begin{cases} T\left(\frac{N}{2}\right) + \Theta(1) & \text{if target is not found} \\ \Theta(1) & \text{otherwise} \end{cases}$$

Runtime Growth Function

$$T(N) = \begin{cases} T\left(\frac{N}{2}\right) + c_1 & \text{if target is not found} \\ c_0 & \text{otherwise} \end{cases}$$

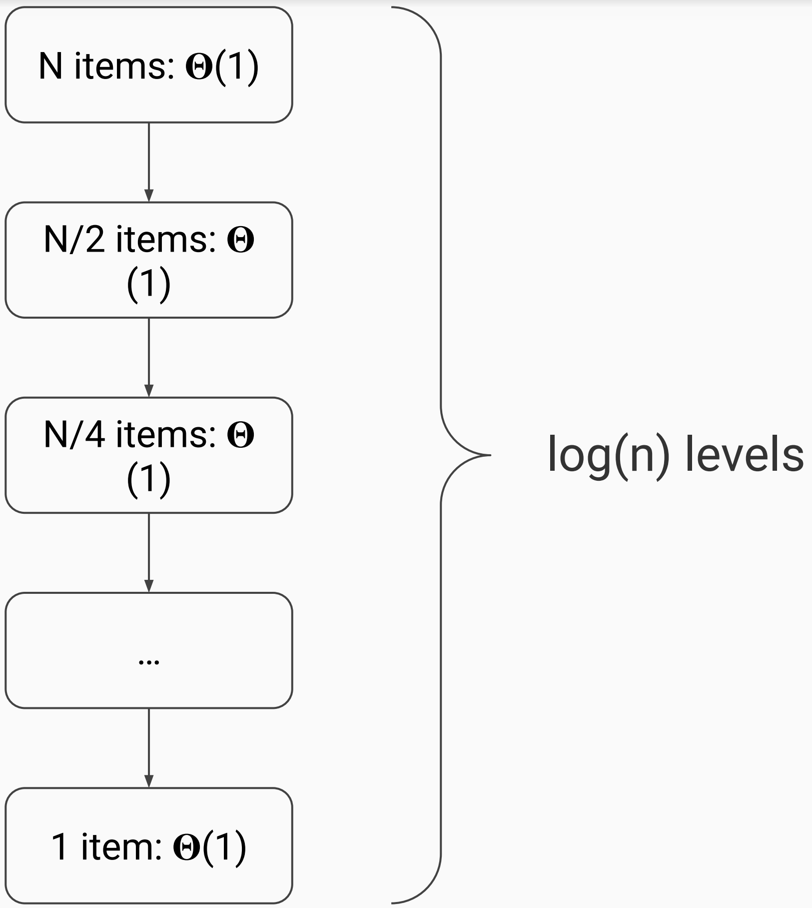
Exercise: Draw the recursion tree for this growth function.
Label the height in terms of ***n*** and each box with it's cost excluding recursive calls.

Recursion Tree for binarySearch

Exercise: Write the summation that represents the total amount of work shown in this recursion tree.

Consider how many levels are in the tree, and the cost of each level

Once you have a summation, simplify it to come up with a hypothesis for the runtime bound



Hypothesis

Summation:

$$\sum_{i=1}^{O(\log_2(N))} \theta(1)$$

Hyptothesis:

$$T(N) \in O(\log_2(N))$$

Exercise: Write the hypothesis as an inequality and prove a Base Case

Base Case

$$T(1) \leq c \cdot \log_2(1)$$

Base Case

$$T(\cancel{1}^{\,2}) \leq c \cdot \log_2(\cancel{1}^{\,2})$$

Base Case

$$T(2) \leq c \cdot \log_2(2)$$

$$T(1) + c_1 \leq c$$

$$c_0 + c_1 \leq c$$

Exercise: Come up with an inductive hypothesis and prove the inductive step

Inductive Case

Assume: $T\left(\frac{N}{2}\right) \leq c \cdot \log_2\left(\frac{N}{2}\right)$

Show: $T(N) \leq c \cdot \log_2(N)$

Inductive Proof

$$T(N) \overset{?}{\leq} c \cdot \log_2(N)$$

Use definition of our growth function

$$T\left(\frac{N}{2}\right) + c_1 \overset{?}{\leq} c \cdot \log_2(N)$$

This is from our assumption

$$\boxed{T\left(\frac{N}{2}\right) + c_1 \leq c \cdot \log_2\left(\frac{N}{2}\right) + c_1} \overset{?}{\leq} c \cdot \log_2(N)$$

$$c(\log_2(N) - \log_2(2)) \overset{?}{\leq} c \cdot \log_2(N)$$

$$c_1 \overset{?}{\leq} c$$

Conclusion

The inequality is true for a base case ($n = 2$) as long as $c \geq c_1 + c_0$

The inductive step showed that:

- If the inequality is true for $n/2$, then it is true for n as long as $c \geq c_1$

Therefore: If $c \geq c_1 + c_0$ the inequality is true for all $n \geq 2$

Therefore $T(N) \in O(\log(n))$