Binary Search Algorithm

Analysis of Binary Search

Base Case

\[ T(1) = c_0 \]

Recursive Case

\[ T(n) = T(n/2) + c_1 \]

“A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.” (CLRS)
Recurrence relations

- By itself, a recurrence does not describe the running time of an algorithm
  - need a closed-form solution (non-recursive description)
  - exact closed-form solution may not exist, or may be too difficult to find
- For most recurrences, an asymptotic solution of the form $\Theta()$ is acceptable
  - … in the context of analysis of algorithms

How to solve recurrences?

- By unrolling (expanding) the recurrence
  - a.k.a. iteration method or repeated substitution
- By guessing the answer and proving it correct by induction
- By using a Recursion Tree
- By applying the Master Theorem

Unrolling a Recurrence

- Keep unrolling the recurrence until you identify a general case
  - then use the base case
- Not trivial in all cases but it is helpful to build an intuition
  - may need induction to prove correctness

Unrolling a Recurrence

$T(1) = c_0$
$T(n) = T(n/2) + c_1$

$= T(n/2^k) + kc_1$
Applying the base case

We already know $T(1)$ is equal to a constant $c_0$:

$$T(n) = T(n/2^k) + kc_1$$

$$= c_0 + c_1 \log n$$

$$= O(\log n)$$

Example

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

Example

```cpp
int power(int b, int n) {
    // base case
    if (n == 0) {
        return 1;
    }
    // recursive call
    return b * power(b, n-1);
}
```

Example

$T(1) = a$
$T(n) = 2T(n/2) + n$
### Analysis of Merge Sort

\[ T(1) = 1 \quad T(n) = 2T(n/2) + n \]

*two recursive calls + one merge*

\[ = \Theta(n \log n) \]

### Example

\[ T(0) = 1 \]
\[ T(n) = 2T(n - 1) + 1 \]

### Recursion Trees

*nodes in a recursion tree account for how much work each recursive call makes*

```c
int binsearch(int *A, int lo, int hi, int k) {
  // base case
  if (hi < lo) {
    return NOT_FOUND;
  }
  // calculate midpoint index
  int mid = lo + ((hi-lo)/2);
  // key found?
  if (A[mid] == k) {
    return mid;
  }
  // key in upper subarray?
  if (A[mid] < k) {
    return binsearch(A, mid+1, hi, k);
  }
  // key is in lower subarray?
  return binsearch(A, lo, mid-1, k);
}
```
Recursion Tree (binary search)

\[ \begin{align*}
  &= c_0 + c_1 \log n \\
  &= O(\log n)
\end{align*} \]

Recursion Tree (mergesort)

\[
\text{if } (hi \leq lo) \text{ return;}
\]
\[
\text{int } \text{mid} = lo + (hi - lo) / 2;
\]
\[
\text{mergesort}(A, lo, \text{mid});
\]
\[
\text{mergesort}(A, \text{mid} + 1, hi);
\]
\[
\text{merge}(A, lo, \text{mid}, hi);
\]

Recursion Tree (mergesort)

\[
\begin{align*}
  & cn \\
  & \downarrow \\
  & c(n/2) \\
  & \downarrow \\
  & c(n/4) \\
  & \downarrow \\
  & T(1)
\end{align*} \]

Additional Examples
Unimodal arrays

• An array is (**strongly**) unimodal if it can be split into an increasing part followed by a decreasing part

• An array is (**weakly**) unimodal if it can be split into a nondecreasing part followed by a nonincreasing part

How to efficiently find the max?

Find the max (strongly unimodal)

• Algorithm?

• Running Time?
### Find the **max** (weakly unimodal)

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>22</td>
<td>35</td>
<td>38</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>22</td>
<td>35</td>
<td>38</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Two recursive calls**

### Find the **max** (strongly unimodal)

- Algorithm?
- Running Time?
- Recursion Tree?