Agenda

We've done

- Growth of functions
- Asymptotic Notations ($O, o, \Omega, \omega, \Theta$)

Now

- Binary search as an example
- Recurrence relations, solving them

Searching in an array/vector

Example (The most basic search problem)

Given `vector<int> myvec` of n distinct integers, and an integer k, is k one of the n integers in `myvec`?

If `myvec` is not sorted

```
bool linear_search(vector<int> vec, int k) {
    for (size_t i=0; i<vec.size(); i++) }
        if (k == vec[i]) return true;
    }
    return false;
}
```

Takes $O(n)$-time. Too slow, especially when there are many searches into the same vector/array.

```
bool binary_search(vector<int> vec, int k) {
    for (size_t i=0; i<vec.size(); i++) }
        if (k == vec[i]) return true;
    }
    return false;
}
```

Takes $O(n)$-time. Too slow, especially when there are many searches into the same vector/array.

If `myvec` is sorted

```
// not correct yet, but intuitively OK;
// assume left/right are in range
bool binary_search(vector<int> sorted_vec, int key,
                   size_t left, size_t right) {
while (left <= right) {
        size_t mid = (left + right)/2; // problematic!
        if (key > sorted_vec[mid])
            left = mid+1;
        else if (key < sorted_vec[mid])
            right = mid-1;
        else return true;
    }
    return false;
}
```

Takes $O(\log n)$-time (we'll see later why). Extremely fast!

Fixing the code for binary search

```
binary_search(vector<int> sorted_vec, int key,
              size_t left, size_t right) {
while (left <= right) {
        // correct! doesn't overflow
        size_t mid = left + (right-left)/2;
        if (key > sorted_vec[mid])
            left = mid+1;
        else if (key < sorted_vec[mid])
            right = mid-1;
        else return true;
    }
    return false;
}
```

Examples of recurrence relations

`fib1()`

$$T(n) = T(n-1) + T(n-2) + \Theta(1)$$

Binary search

$$T(n) \leq T(\lceil n/2 \rceil) + \Theta(1)$$

Merge sort (we'll discuss next lecture)

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) \tag{1}$$

and many others

$$\begin{aligned} T(n) &= 4T(n/2) + n^2 \lg n \\ T(n) &= 3T(n/4) + \lg n \\ T(n) &= T(n/a) + T(a) \end{aligned}$$

Recall the way to interpret (1): "$T(n)$ is $T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$ plus some function $f(n)$ which is $\Theta(n)$"

Methods of solving recurrent relations

- Guess and induct (we only discuss this method!)
- Master Theorem (take CSE331 or CSE431)
- Generating functions (read Enumerative Combinatorics books)

Guess and induct

- Guess a solution
  - Guess by substitution
  - Guess by drawing a recurrence tree
- Use induction to show that the guess is correct

Guess by substitution - Example 1

Example (The `fib1()` algorithm)

$$T(n) = \begin{cases} d & \text{if } n \le 1 \\ T(n-1) + T(n-2) + c & \text{if } n \ge 2 \end{cases}$$

Guess by iterating the recurrence a few times:

- $T(0) = d$, $T(1) = c$
- $T(2) = 2d + 1c$
- $T(3) = 3d + 2c$
- $T(4) = 5d + 4c$
- $T(5) = 8d + 7c$
- ...

So, what's $T(n)$?

Guess by substitution - Example 1

The guess

$$T(n) = (c+d)F_{n+1} - c \tag{2}$$

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n = \Theta(\phi^n), \tag{3}$$

where F_n is the nth Fibonacci number, ϕ is the golden ratio

Conclude with

$$T(n) = \Theta(\phi^n) \tag{4}$$

We have shown (2), (3) & (4) by induction.

Guess by substitution – Example 2

Example (Merge Sort)

$$\begin{aligned} T(1) &= \Theta(1) \\ T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) \end{aligned}$$

Clean up the recurrence before guessing

It is often safe to ignore the issue of integrality:

$$T(n) \approx T(n/2) + T(n/2) + cn = 2T(n/2) + cn.$$

Guess by substitution – Example 2

$$
\begin{aligned}
T(n) &= 2T(n/2) + cn \\
&= 2\left(2T(n/4) + c\frac{n}{2}\right) + cn \\
&= 4T(n/4) + 2cn \\
&= 4\left(2T(n/8) + c\frac{n}{4}\right) + 2cn \\
&= 8T(n/8) + 3cn \\
&= \dots \\
&= 2^k T(n/2^k) + kcn \\
&= ... \\
&= 2^{\lg n} T(n/2^{\lg n}) + cn \lg n \\
&= \Theta(n \lg n)
\end{aligned}
$$

Guess by substitution – Example 2

- Rigorously, we have

$$
\begin{aligned}
T(1) &= c_0 \\
T(n) &\geq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_1 n \\
T(n) &\leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n
\end{aligned}
$$

- Guess: $T(n) = \Theta(n \lg n)$.
- By induction, show that there are constants $a, b > 0$ such that

$$a n \lg n \leq T(n) \leq b n \lg n.$$

Now try

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

Notes

- To (sort of) see why integrality isn't important, consider

$$T(n) = 2T(\lfloor n/2 \rfloor + 17) + cn.$$

- Approximate this by ignoring both the integrality issue and the annoying constant 17

$$T(n) = 2T(n/2) + cn.$$

- The guess is then $T(n) = O(n \lg n)$. (You should prove it.)

Common mistake

$$T(n) \leq 2c\lfloor n/2 \rfloor + n \leq cn + n = O(n)$$

Another commonly used trick: change of variable

Solve

$$T(n) = 2T(\sqrt{n}) + 1$$

Let $m = \lg n$, then

$$T(2^m) = 2T(2^{m/2}) + 1$$

Let $S(m) = T(2^m)$, then

$$S(m) = 2S(m/2) + 1.$$

Hence,

$$S(m) = O(m).$$

Thus,

$$T(n) = S(\lg n) = O(\lg n).$$

Guess by recurrence tree

Example (Binary search)

$$T(n) \leq T(\lceil n/2 \rceil) + \Theta(1).$$

Recursion tree suggests $T(n) = O(\log n)$. Prove rigorously by induction.

Example

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2).$$

Recursion tree suggests $T(n) = O(n^2)$. Prove rigorously by induction.

Example (Now try this)

$$T(n) = T(n/3) + T(2n/3) + O(n)$$

Other methods of solving recurrences

- Masters Theorem
- Generating functions
- Hypergeometric series
- Finite calculus, finite differences
- ...

Further readings

- "$A = B$," by M. Petkovsek, H. Wilf, D. Zeilberger
- "Concrete mathematics," R. Graham, D. Knuth, O. Patashnik
- "Enumerative combinatorics," R. Stanley (two volumes)
- "Theory of partitions," G. Andrews