Note about the midterm

This midterm was hard, so congratulations on getting through it. No matter you did, you are incredible just for making it this far in the course. Treat yourself - you deserve it :)

Last time...

Solving recurrences:

- Substitution method
- Recursion trees
- Master Theorem

CSC 236 Lecture 7: Correctness

Harry Sha

July 5, 2023

Today

Correctness - Merge Sort

Multiplication

Correctness - Binary Search

Algorithm Correctness

Today, we will see how to prove algorithms are “correct”.

What does it mean for an algorithm to be correct?

Correctness (formally)

For any algorithm/function/program, define a precondition and a postcondition.

- The precondition is an assertion about the inputs to a program.
- The postcondition is an assertion about the end of a program.

An algorithm is correct if the **precondition implies the postcondition**.

I.e. "If I gave you valid inputs, your algorithm should give me the expected outputs."

This is essentially a design specification.

Documentation Analogy

```
def transpose(a, axes=None):
    """
    Reverse or permute the axes of an array; returns the modified array.

    For an array a with two axes, transpose(a) gives the matrix transpose.

    Refer to `numpy.ndarray.transpose` for full documentation.

    Parameters
    ----------
    a : array_like
        Input array.
    axes : tuple or list of ints, optional
        If specified, it must be a tuple or list which contains a permutation of
        [0,1,..,N-1] where N is the number of axes of a.  The i'th axis of the
        returned array will correspond to the axis numbered ``axes[i]`` of the
        input.  If not specified, defaults to ``range(a.ndim)[::-1]``, which
        reverses the order of the axes.

    Returns
    -------
    p : ndarray
        `a` with its axes permuted.  A view is returned whenever
        possible.
```

PRE.

POST

What is the pre/post conditions for `mergesort`(l)?

PRE: - input is an array.

- values in the array must be of $\underbrace{\text{"similar type"}}_{\text{comparable.}}$

POST: the sorted array.

What is the pre/post condition for `binsearch`(l, t, a, b)?

PRE: - l : is a <u>sorted</u> array.

- a, b are valid indices in l.
- $b \geq a$.

POST: - index of t in l iff t is in l.

- None if t is not in l.

How do you prove correctness for recursive functions?

- Must terminate
- Induction → Base case
  → recursive case.
  (Inductive step.).

How do you prove correctness for recursive functions?

By induction on the size of the inputs!

Notation

Let's use CS/Python notation. I.e., the elements of a list of length n in order are $l[0], l[1], ..., l[n-1]$.

Slicing:

$$l[i:j] = [l[i], l[i+1], ..., l[j-1]]$$

By convention, if $j \leq i$, then $l[i:j] = []$.

Conventions

Today, let's think of all lists as being lists of natural numbers.

Correctness - Merge Sort

Multiplication

Correctness - Binary Search

Merge Sort

```
def merge_sort(l):
    n = len(l)
    if n <= 1:
        return l
    else:
        left = merge_sort(l[:n//2])
        right = merge_sort(l[n//2:])
        return merge(left, right)
```

Merge Sort - Correctness

As usual, we break this down and show for all $n \in \mathbb{N}$, if $l \in \texttt{List}[\mathbb{N}]$ is a list of length n, then `mergesort` works on l.

Correctness

PRE.

$P(n)$: Let $l \in \texttt{List}[\mathbb{N}]$ be a list of natural numbers of length n, then $\texttt{mergesort}(l)$ returns the sorted list.

POST

Claim: $\forall n \in \mathbb{N}.(P(n))$.

Correctness of Merge

For now, let's assume `merge` is correct. I.e. that on sorted lists `left` and `right`, `merge(left, right)` returns a sorted list containing all the elements in either list.

We'll come back and prove that later!

Base case

Base case: $n=0$: the only list of length 0 is the empty list which is sorted.

$n=1$: $l = [a]$ where $a \in \mathbb{N}$, l is sorted.

furthermore in both cases, the function terminates.

Inductive step

Let $k \in \mathbb{N}$, $k \geq 2$. and assume merge sort is correct on lists of size $0, 1, \ldots, \lfloor k/2 \rfloor, \ldots, k-1$. We'll show merge sort works on lists of length k. Let l be an arbitrary list of length k.

1. Show $l[:k//2]$, $l[k//2:]$ are shorter lists.

$$k//2 = \lfloor k/2 \rfloor \qquad 1 \leq \lfloor k/2 \rfloor < k \quad \forall k \geq 2.$$

$$\text{len}(l[:k//2]) = \lfloor k/2 \rfloor < k \implies$$ mergesort$(l[:k//2])$ returns the sorted version of $l[:k//2]$.

$$\text{len}[l[k//2:]] = k - \lfloor k/2 \rfloor \leq k-1.$$

$\implies$ mergesort$(l[k//2:])$ returns sorted . . . .

Inductive Step

$\Rightarrow$ left contains sorted $l[:k/\!/2]$

right contains sorted $l[k/\!/2:]$.

$\Rightarrow$ Precondition for merge is met

$\Rightarrow$ merge(left, right) returns a sorted list of elements from left and right which is a sorted version of l.

Notes

The fact that the algorithm terminates (i.e. doesn't get stuck in an infinite loop) is implied by the statement of the claim in the word **returns**.

Correctness - Merge Sort

Multiplication

Correctness - Binary Search

Recursive Algorithms

This next example will put together what we have studied so far on recursive runtime and correctness.

Multiplication

Let's study multiplication!

If I gave you two 10 digit numbers, how would you multiply them?

Grade School Multiplcation

1   1 1

2 3 6

6 3 2

---

4 7 2

0 8 0

Lower bound for the Grade School Multiplcation Algorithm

Suppose I gave you two *n*-digit numbers. What is a lower bound for the runtime of the Grade School Multiplication Algorithm?

$a_n \quad a_{n-1} \quad \ldots \quad a_2 \quad a_1$

$b_n \quad b_{n-1} \quad \ldots \quad b_1$

$\Theta(n^2)$

Can we do better?

Karatsuba's Algorithm

$x = 1\,2\,3\,456$ $\quad x_h = 123$ $\quad x_l = 456$

```
def karatsuba(x, y):
    if x < 10 and y < 10:
        return x * y

    n = max(len(str(x)), len(str(y)))
    m = n // 2
    x_h, x_l =  x // 10**m, x % 10**m
    y_h, y_l =  y // 10**m, y % 10**m

    z0 = karatsuba(x_l, y_l)
    z1 = karatsuba((x_l + x_h), (y_l + y_h))
    z2 = karatsuba(x_h, y_h)

    return (z2 * 10**(2*m)) + ((z1 - z2 - z0) * 10**m) + z0
```

What are `x // 10**m`, and `x % 10**m`?

Karatsuba's Algorithm

```
def karatsuba(x, y):
    if x < 10 and y < 10:
        return x * y

    n = max(len(str(x)), len(str(y)))
    m = n // 2
    x_h, x_l =  x // 10**m, x % 10**m
    y_h, y_l =  y // 10**m, y % 10**m

    z0 = karatsuba(x_l, y_l)
    z1 = karatsuba((x_l + x_h), (y_l + y_h))
    z2 = karatsuba(x_h, y_h)

    return (z2 * 10**(2*m)) + ((z1 - z2 - z0) * 10**m) + z0
```

Trace the algorithm by hand on inputs $a = 31$ and $b = 79$, report the values of each of the variables $m, a_l, a_u, b_l, b_u, z_0, z_1, z_2$ as well as the result.

$\boxed{x=31}$ $\boxed{y=79}$ $m=1$

$x_h = 3$ $x_\ell = 1$ $y_h = 7$ $y_\ell = 9$.

$z_0 = k(1, 9) = 9$

$z_1 = k(4, 16) = 64$

$z_2 = k(3, 7) = 21$

return: $z_2 \cdot 10^{2m} + (z_1 - z_0 - z_2) \cdot 10^m + z_0$.

$2100 + (64 - 21 - 9) \cdot 10 + 9$

$2100 + 340 + 9$

$= \cancel{224}\ 2449$

$\boxed{x=4, y=16}$

$x_h = 0$ $x_\ell = 4$

$y_h = 1$ $y_\ell = 6$

$z_0 = 24$

$z_1 = 28$

$z_2 = 0$

$(z_1 - z_2 - z_0) \cdot 10 + z_0$

$= 40 + 24$

$= \underline{\underline{64}}$

Karatsuba's Algorithm

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

```
def karatsuba(x, y):
    if x < 10 and y < 10:
        return x * y

    n = max(len(str(x)), len(str(y)))
    m = n // 2
    x_h, x_l =  x // 10**m, x % 10**m
    y_h, y_l =  y // 10**m, y % 10**m

    z0 = karatsuba(x_l, y_l)
    z1 = karatsuba((x_l + x_h), (y_l + y_h))
    z2 = karatsuba(x_h, y_h)

    return (z2 * 10**(2*m)) + ((z1 - z2 - z0) * 10**m) + z0
```

leaf: $n^{\log_2 3}$

root: n

$\Rightarrow$ leaf heavy

$\Rightarrow \Theta(n^{\log_2 3})$.

$\boxed{n^{\log_2 3}}$

$< n^{1.6} < n^2$

Write a recurrence for the runtime of Karatsuba's algorithm. Solve the recurrence.

Correctness

Precondition. $x, y \in \mathbb{N}$, **Postcondition.** Return xy

- $m = \lfloor n/2 \rfloor$, $x_h = \lfloor x/10^m \rfloor$, $x_l = x\%10^m$
- $z_0 = x_l y_l$, $z_1 = (x_l + x_h)(y_l + y_h)$, $z_2 = x_h y_h$
- `return` $(z_2 \cdot 10^{2m}) + ((z_1 - z_2 - z_0) \cdot 10^m) + z_0$

$P(n)$: if $\max(x,y) = n$, then $k(x,y) = xy$.

Base case(s).

Let $n < 10$, and let x, y be such that $\max(x,y) = n$. the $k(x,y)$ reaches the base case and we return xy. $\blacksquare$

Correctness

Precondition. $x, y \in \mathbb{N}$, **Postcondition.** Return xy

- $m = \lfloor n/2 \rfloor$, $x_h = \lfloor x/10^m \rfloor$, $x_l = x \% 10^m$
- $z_0 = x_l y_l$, $z_1 = (x_l + x_h)(y_l + y_h)$, $z_2 = x_h y_h$
- `return` $(z_2 \cdot 10^{2m}) + ((z_1 - z_2 - z_0) \cdot 10^m) + z_0$

Inductive step. let $k \in \mathbb{N}$, $k \geq 10$. suppose $\max(x, y) = k$.

IH: suppose $P(0), \dots, P(k-1)$.

WTS: Recursive calls are handled by IH. i.e. WTS $x_l, x_h, y_l, y_h, x_l + x_h, y_l + y_h < k$.

$\boxed{x = x_h \cdot 10^m + x_l}$ $m \geq 1$ b/c $k \geq 10$

$x > x_h + x_l \implies x > x_h,\ x > x_l$.

$y > y_h + y_l \implies y > y_h,\ y > y_l$.

IH, $\implies$ recursive calls work.

i.e. $z_0 = x_l y_l$, $z_1 = (x_l + x_h)(y_l + y_h)$, $z_2 = x_h y_h$.

Correctness

Precondition. $x, y \in \mathbb{N}$, **Postcondition.** Return xy

- $m = \lfloor n/2 \rfloor$, $x_h = \lfloor x/10^m \rfloor$, $x_l = x\%10^m$
- $z_0 = x_l y_l$, $z_1 = (x_l + x_h)(y_l + y_h)$, $z_2 = x_h y_h$
- `return` $(z_2 \cdot 10^{2m}) + ((z_1 - z_2 - z_0) \cdot 10^m) + z_0$

Inductive step cont...

return $z_2 10^{2m} + (z_1 - z_2 - z_0) 10^m + z_0$

$= x_h y_h 10^{2m} + ((x_l + x_h)(y_l + y_h) - x_h y_h - x_l y_l) 10^m + x_l y_l$.

$= x_h y_h 10^{2m} + (x_l y_l + x_h y_h + x_h y_l + y_h x_l - x_h y_h - x_l y_l) 10^m + x_l y_l$.

$= x_h y_h 10^{2m} + (x_h y_l + y_h x_l) \cdot 10^m + x_l y_l$.

$= (x_h \cdot 10^m + x_l)(y_h \cdot 10^m + y_l)$

$= xy$

An alternate $P(n)$

$\boxed{99}$ $x_h = 9$

$x_\ell = 9$

$x_h + x_\ell = \boxed{18}$.

Instead of letting n be the maximum value of x and y, could we have set n to be the maximum length of x and y?

Summary: Correctness for Recursive Algorithms

Prove the correctness of recursive algorithms by *induction*. The link between recursive algorithms and inductive proofs is strong.

- The base case of the recursive algorithm corresponds to the inductive proof's base case(s).
- The recursive case of the recursive algorithm corresponds to the inductive step.
- The 'leap of faith' in believing that the recursive calls works correspond to the inductive hypothesis.

Correctness - Merge Sort

Multiplication

Correctness - Binary Search

Binary Search

```
def bin_search(l, t, a, b):
    if b == a:
        return None
    else:
        m = (a + b)//2
        if l[m] == t:
            return m
        elif l[m] < t:
            return bin_search(l, t, m+1, b)
        elif l[m] > t:
            return bin_search(l, t, a, m)
```

Correctness Claim: First attempt

> $P(n)$: If $l \in \texttt{List}[\mathbb{N}]$ is a list of length n, $\texttt{binsearch}(l, t, a = 0, b = n)$ returns the index of t if t is in l and `None` otherwise.

Claim: for all $n \in \mathbb{N}.(P(n))$.

Base case

Consider the $n = 0$ case. let l be any list of length 0, t be any object, and consider `binsearch`$(l, t, 0, 0)$.

The check `a==b` is true since both variables are 0 so we return `None`. This is the expected result since an empty list surely does not contain t.

Inductive Step

Let $k \in \mathbb{N}$ and assume for all $i \in \mathbb{N}, i \leq k$, $\texttt{binsearch}(l, t, 0, i)$ returns the desired result for all lists of length i.

Let $l \in \texttt{List}[\mathbb{N}]$ be a list of length $k+1$, and let $t \in \mathbb{N}$. Consider the execution of $\texttt{binsearch}\boxed{(l, t, 0, k+1)}$. Since $k+1 \geq 1$, the if condition fails. Let $m = (k+1)//2 = \lfloor (k+1)/2 \rfloor$ There are then 3 cases.

- Case 1. `l[m]` $==$ `t`.
- Case 2. `l[m]` $<$ `t`.
- Case 3. `l[m]` $>$ `t`.

Case 1. `l[m]` $==$ `t`

In this case `binsearch` returns m, which is indeed the index of t in l.

Case 2. `l[m]` $<$ `t`

$a \quad \underbrace{m \quad t \quad k+1}$

In this case, we return $\texttt{binsearch}(l, t, \underline{m+1}, \underline{k+1})$.

Case 2. `l[m]` $<$ `t`

In this case, we return `binsearch`$(l, t, m+1, k+1)$.

...

Case 2. `l[m]` $<$ `t`

In this case, we return `binsearch`$(l, t, m+1, k+1)$.

...

Here was our inductive hypothesis.

> Let $k \in \mathbb{N}$ and assume for all $i \in \mathbb{N}, i \leq k$, `binsearch`$(l, t, 0, i)$ returns the desired result for all lists of length i.

The inductive hypothesis doesn't apply here! Since $a \neq 0$!

Case 2. `l[m]` $<$ `t`

In this case, we return $\texttt{binsearch}(l, t, m+1, k+1)$.

...

Here was our inductive hypothesis.

Let $k \in \mathbb{N}$ and assume for all $i \in \mathbb{N}, i \leq k$, $\texttt{binsearch}(l, t, 0, i)$ returns the desired result for all lists of length i.

The inductive hypothesis doesn't apply here! Since $a \neq 0$!

How can we fix it?

A fix that doesn't quite work

Instead of calling $\texttt{binsearch}(l, t, m+1, k+1)$ make the recursive call

$$\texttt{binsearch}(l[m+1:k+1], t, 0, k+1)$$

Why doesn't this work?

A fix that doesn't quite work

Instead of calling $\texttt{binsearch}(l, t, m+1, k+1)$ make the recursive call

$$\texttt{binsearch}(l[m+1:k+1], t, 0, k+1)$$

Why doesn't this work?

The index of t in $l[m+1:k+1]$ is different from the index of t in l!

Correctness Claim, Corrected

Instead of doing induction on the length of the list, do induction on the length of the search window!

$P(n)$: For all lists $l \in \texttt{List}[\mathbb{N}]$ and $t \in \mathbb{N}$, if $\boxed{b - a = n}$, then $\texttt{binsearch}(l, t, a, b)$ returns `None` if t is not in $l[a : b]$ and the index of t in l otherwise.

Claim: $\boxed{\forall n \in \mathbb{N}. P(n).}$

Base Case

$P(0)$: $b - a = 0 \Rightarrow a = b \Rightarrow$ base case.

$\Rightarrow$ None. which is to be expected since

$l[a:b] = [\,]$, and f & $[\,]$.

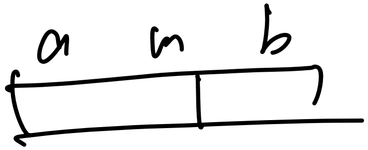
Inductive Step

$P(n)$: For all sorted lists $l \in \texttt{List}[\mathbb{N}]$ and $t \in \mathbb{N}$, if $b - a = n$, then $\texttt{binsearch}(l, t, a, b)$ returns $\texttt{None}$ if t is not in $l[a : b]$ and the index of t in l otherwise.

IH: assume $P(0), \ldots, P(k-1)$.

WTS $P(k)$. $\quad m = \left\lfloor \frac{a+b}{2} \right\rfloor$

`l[m] < t`



$P(n)$: For all sorted lists $l \in \texttt{List}[\mathbb{N}]$ and $t \in \mathbb{N}$, if $b - a = n$, then $\texttt{binsearch}(l, t, a, b)$ returns `None` if t is not in $l[a:b]$ and the index of t in l otherwise.

recursive call is $\texttt{binsearch}(l, t, m+1, b)$.

WTS the IH applies, i.e. whenever $b-(m+1) < k$.

suffices to show: $m+1 > a$.

$$m+1 = \left\lfloor \frac{a+b}{2} \right\rfloor + 1 > \left\lfloor \frac{a+a}{2} \right\rfloor + 1 = a+1 > a.$$

$\Rightarrow$ $\texttt{binsearch}(l, t, m+1, b)$ is correct.

`l[m] = t`

$P(n)$: For all sorted lists $l \in \texttt{List}[\mathbb{N}]$ and $t \in \mathbb{N}$, if $b - a = n$, then $\texttt{binsearch}(l, t, a, b)$ returns $\texttt{None}$ if t is not in $l[a : b]$ and the index of t in l otherwise.

✓

`l[m] > t`

> $P(n)$: For all sorted lists $l \in$ `List`$[\mathbb{N}]$ and $t \in \mathbb{N}$, if $b - a = n$, then `binsearch`(l, t, a, b) returns `None` if t is not in $l[a:b]$ and the index of t in l otherwise.

binsearch(l, t, a, m). to show that $m - a < k$,

i'll show $m < b$

$$m = \left\lfloor \frac{a+b}{2} \right\rfloor < \left\lfloor \frac{b+b}{2} \right\rfloor \text{ since } a < b.$$

$$= \left\lfloor \frac{2b}{2} \right\rfloor = \lfloor b \rfloor$$

$\Rightarrow m < b$ ✓.