**Time Complexity**

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

(a):

(b):

(c):

(d):

(e):

**Fibonacci 1**

Prove: .

By trial and error: It appears that for all

To prove: For all positive integers

By induction on . Base case:

Step, assume: Indeed, true that for all

To prove:

LHS:

Suffices to prove:

(by dividing the above by )

It is indeed true that

**Multiplication**

Graphical user interface, text, application

Description automatically generated

Suppose instead of both and being n-bit, is n-bit and is m-bit. What is the worst-case time efficiency of ?

Proposed:

Time Efficiency:

So, final answer:

**Fibonacci 2**

Let be the nth Fibonacci number, Prove .

* Somewhere, we have shown:
* But here, seek to show: There exists positive real , for all in
* Natural proof strategy for “there exists” – construction (i.e., propose some concrete , and show that it works)
* Try some small values for , and see what would work
* Appears that works. Adopt it and check if proof goes through. Now, proof by induction with
* Base case,
* Step: Seek to show given that for all
* by induction assumption

**Fibonacci 3**

Let be the nth Fibonacci number, Prove .

* Recall from logic: not (there exists an egg-laying mammal) for all mammals , is not egg-laying
* Here, : There exists positive real , for all natural ,
* So here, need to prove: Given any positive real , it is true that there exists such that
* By contradiction: Suppose that there exists positive real , such that, for all natural ,
* Then:
* This is true only if is “large” compared to
* What is large? We need
* Try :
* Try :
* Try :
* Try :
* Try :
* Try :
* Prove by induction: for all natural
* Base case : See above
* Step:
* So far: We have shown that indeed, for ,

**Selection Sort**

What is a meaningful characterization of the time efficiency of ?

* Suppose we invoke . In . Suppose now, min is at index in . This index of a min in is at index
* Suppose on input: . Then A evolves in as follows:
* For time efficiency: Need to make meaningful assumption(s)
* Customary Assumptions: (1) is unbounded, (ii) each is bounded
* What should we count? Suppose we all agree that counting # swaps is a meaningful measure for time efficiency
* Then:
* Now, let’s say we want to get a bit more fine-grained. Incorporate (worst case) time for each swap # swaps
* So now, time efficiency:

**Modular Simplification**

1. Is ?

So:

So:

Now:

So:

So:

1. Is divisible by ?

Trick: Keep exponentiating until numbers start to repeat.

Suppose we repeatedly exponentiate :

So: . And . So

Now check whether is divisible by . Indeed:

Repeat with . Repeated exponentiation of :

So:

Now: .

.

1. Is a multiple of ?

is prime. And .

Compare with :

.

Show that if has a multiplicative inverse modulo , then this inverse is unique (modulo ).

Let’s assume .

Suppose are both multiplicative inverses of . Then:

: Substitution Rule:

Then:

(2): Commutativity

Suppose . Show that is an integer.

So: , which is divisible by .

We say that is a square root of modulo a prime if . Show that if and has a square root modulo , then is such a square root.

Let be the square root of . Then: .

Write . Then,

Keep in mind: .

Try plugging in in the last expression:

Is ?

So, we’re asking: Is ?

So at least one of: or must be .

We know: There exists such that .

We seek to prove: . Sufficient condition for that to be true:

is okay, because is invertible modulo

**Proving Recurrence 1**

Suppose . Prove recurrence correctness.

Case Analysis:

1. If , then . So, the recurrence is correct for the case where
2. If : then . So
3. If : then . So now:

**Proving Recurrence 2**

Let be the quotient and remainder of and be the quotient and remainder of . Prove recurrence correctness.

To be absolutely clear, what are the quotient and remainder of ?

We call the quotient, and the remainder if and only if and are non-negative integers that satisfy:

Proof by case analysis:

1. If , then . So, recurrence is correct for this case.
2. If is even and : then . So:

Where we infer the last line from the facts that: equation is of the form from definition for quotient and remainder, , and we are given .

1. If is odd and :
2. is even, : . So:

This is of the form of the definition of quotient and remainder, except that we need to confirm that indeed lies between and . Which it does not necessarily. Actually, we are given that and therefore not between and . Now we observe:

Now only question that remains: is it the case that ?

* Is ? Yes, because
* Is ? Yes, because:

1. odd, :

Now:

* because .
* because:

**Proving Recurrence 3**

Prove that is correct.

Above is recursive version of binary search. Iterative version:

Typically, for iterative algorithms, towards correctness, we articulate a *loop invariant*:

Let and be the values of and respectively on input. Just before we successfully enter an iteration of the loop of Line (1), it is true that:

Going back to the recursive version, what is a correctness property?

Given an array that is sorted, non-decreasing, are each on input, returns:

Proof by case analysis:

Case 1: on input: then condition of Line (1) evaluates to **false**, and we correctly return **false** in Line (6). Then, this is either from Line without making any recursive calls, or as the return value from a recursive call from one of Lines or .

For , we first observe that because the only recursive calls are within the block of Line . So, all that remains to be proven is that indeed: .

We prove that by induction on . Base case: . We claim we return within the first recursive invocation. That is, we claim: and , , and .

easy to prove:

is , because then we would have returned in Line .

To prove : we simply exploit:

So, the algorithm is correct if it returns , and .

For the step, we know that on input . So, we returned in some recursive call. So, all we have to prove to appeal to induction assumption: and .

**Proving Master Theorem**

Give a closed form solution for the following recurrence. Assume: is non-decreasing, .

Proposed approach: Inductive “rewriting” of the function . But first: adopt concrete functions wherever we have , or . In this case: adopt for , and for . Now onto the rewriting:

To figure out the power of in that last term:

Power of is the same as the power of inside the . In other words, we’re asking: what is the power of , i.e., for which ? Answer: .

Our next step: Simplify/figure out:

Suppose:

Now subtract one from the other:

When , how do we figure out what is? Answer: then, is:

So, going back to our :

And:

When is ? Answer: .

So, going back to our : first, the case that .

But even before that: rewrite . Because:

So, when . So, in this case:

Onto the other two cases: .

Before we continue: a closer look at :

So: when

So, going back to :

So, if :

And if :