

FIT2004

Algorithms and Data Structures

Ian Wern Han Lim
lim.wern.han@monash.edu

Referencing materials by
Nathan Compane, Aamir Cheema, Arun Konagurthu and Lloyd Allison



Ready?

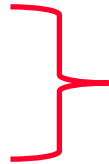
Agenda

- Proof of Correctness
 - Loop invariants
 - Termination

Agenda

- Proof of Correctness

- Loop invariants
- Termination



Covered in Lecture 02
using sorting algorithms
for case study

Let us begin...

Proof of Correctness for Algorithms

- Why?

Proof of Correctness for Algorithms

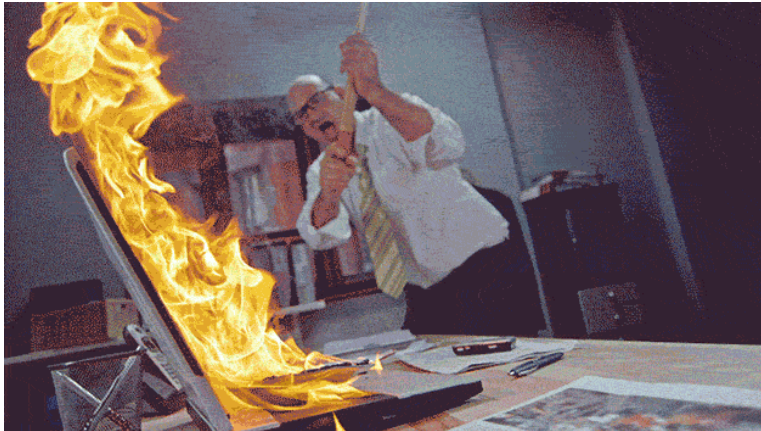
- Why?
- Why not just program it out and run it?

Proof of Correctness for Algorithms

- Why?
- Why not just program it out and run it?
 - Development cost can be costly
 - Resources can be limited

Proof of Correctness for Algorithms

- Why?
- Why not just program it out and run it?
 - Development cost can be costly
 - Resources can be limited
 - Damage can happen!



Consequences of errors

Explosion of unmanned Ariane 5 rocket in 1996

- Exploded within 40 seconds after launch
- Horizontal velocity incorrectly computed
- Loss ~\$7 Billion dollars



Consequences of errors

American Patriot Missile battery in Saudi Arabia failed to intercept an incoming Iraqi Scud Missile

- Killed 28 US soldiers
- Incorrect computation of the time since boot



Consequences of errors

Incorrect maps almost started a war between Costa Rica and Nicaragua



Proof of Correctness for Algorithms

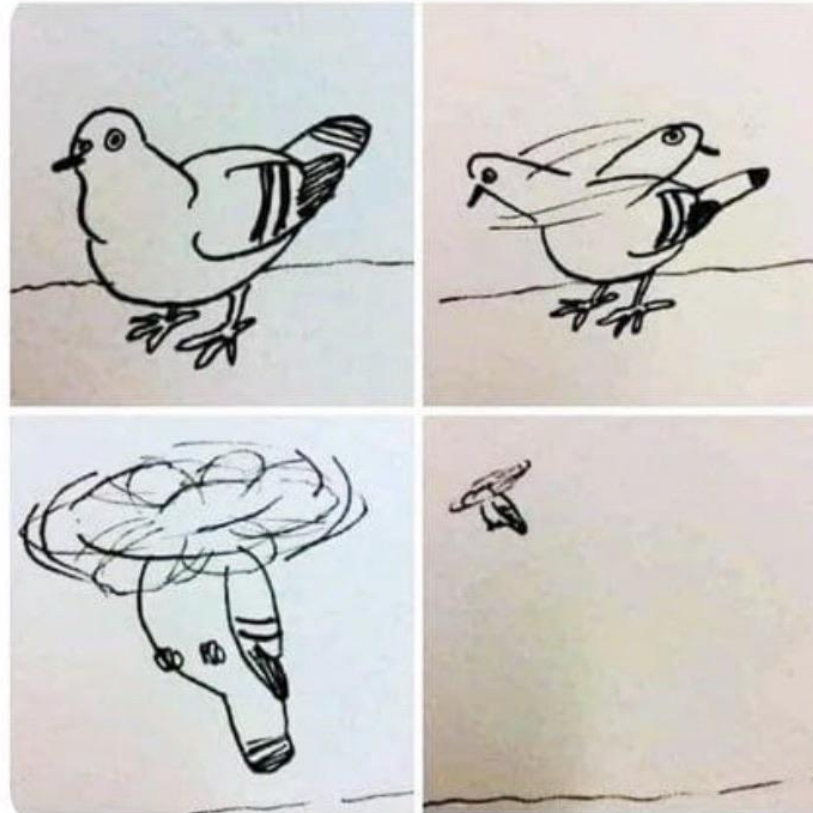


APPS & SOFTWARE

People are still driving into lakes because their
GPS tells them to

Proof of Correctness for Algorithms

When your program
is a complete mess,
but it does its job



Proof of Correctness for Algorithms

- Why?
- Why not just program it out and run it?
 - Development cost can be costly
 - Resources can be limited
 - Things can go really wrong
- But you can't argue against testing right?

Proof of Correctness for Algorithms

- Why?
- Why not just program it out and run it?
 - Development cost can be costly
 - Resources can be limited
 - Things can go really wrong
- But you can't argue against testing right?
 - How often you think your solution work...

Proof of Correctness for Algorithms

- Why?
- Why not just program it out and run it?
 - Development cost can be costly
 - Resources can be limited
 - Things can go really wrong
- But you can't argue against testing right?
 - How often you think your solution work...
THEN SCUMBAG IAN MINUS YOUR MARKS???

Proof of Correctness for Algorithms

- Why?
- Why not just program it out and run it?
 - Development cost can be costly
 - Resources can be limited
 - Things can go really wrong
- But you can't argue against testing right?
 - How often you think your solution work...
 - You can't test everything...
 - But you can **apply it when you design** algorithms!

Questions?

Proof of Correctness for Algorithms

- So how do we prove?
 - Termination
 - Loop invariant

Proof of Correctness for Algorithms

- Termination
 - Program needs to end to return the result
 - If it doesn't end, then you don't have your result

Proof of Correctness for Algorithms

- Termination
 - Program needs to end to return the result
 - If it doesn't end, then you don't have your result

- Loop invariant = constants in a loop
 - What keeps happening over and over
 - Will lead to the solution/ rightness/ result

Proof of Correctness for Algorithms

- Let us have a relatable example
- Getting a Degree from Monash

Proof of Correctness for Algorithms

- Let us have a relatable example
- Getting a Degree from Monash
 - You have a loop go through semester after semester...

Proof of Correctness for Algorithms

- Let us have a relatable example
- Getting a Degree from Monash
 - You have a loop go through semester after semester...
 - Only when this loop terminate, you get your degree

Proof of Correctness for Algorithms

- Let us have a relatable example
- Getting a Degree from Monash
 - You have a loop go through semester after semester...
 - Only when this loop terminate, you get your degree
 - What must not change for you to get your degree in every semester?

Proof of Correctness for Algorithms

- Let us have a relatable example

- Getting a Degree from Monash
 - You have a loop go through semester after semester...
 - Only when this loop terminate, you get your degree
 - What must not change for you to get your degree in every semester?
 - Number of sleep per night?
 - Number of anime you watch?
 - Number of games you play?

- Let us have a relatable example

- Getting a Degree from Monash
 - You have a loop go through semester after semester...
 - Only when this loop terminate, you get your degree
 - What must not change for you to get your degree in every semester?
 - Number of sleep per night?
 - Number of anime you watch?
 - Number of games you play?
 - Passing at least 1 unit per semester!

Proof of Correctness for Algorithms

- Let us have a relatable example

- Getting a Degree from Monash
 - You have a loop go through semester after semester...
 - Only when this **loop terminate**, you get your degree
 - What must not change for you to get your degree in every semester?
 - Number of sleep per night?
 - Number of anime you watch?
 - Number of games you play?
 - **Passing at least 1 unit per semester!**

Proof of Correctness for Algorithms

- Let us have a relatable example
- Getting a Degree from Monash
 - You have a loop go through semester after semester...
 - Only when this **loop terminate**, you get your degree
 - What must not change for you to get your degree in every semester?
 - Number of sleep per night?
 - Number of anime you watch?
 - Number of games you play?
 - **Passing at least 1 unit per semester!**
 - Eventually, you run out of units...

Questions?

Proof of Correctness for Algorithms

- Now let us look at the actual algorithm examples

Proof of Correctness for Finding Minimum

- We'll use our code one (sent via Slack)

```
1  def find_min(array):
2      """
3      Find the minimum...
4      Does this work?
5      """
6      my_min = array[0]
7      index = 1
8      while index < len(array):
9          if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```

Proof of Correctness for Finding Minimum

- Does it terminate?
- What is the loop invariant?

```
1  def find_min(array):
2      """
3      Find the minimum...
4      Does this work?
5      """
6      my_min = array[0]
7      index = 1
8      while index < len(array):
9          if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```

Proof of Correctness for Finding Minimum

- Does it terminate?
 - Yes
- What is the loop invariant?

```
1  def find_min(array):  
2      """  
3      Find the minimum...  
4      Does this work?  
5      """  
6      my_min = array[0]  
7      index = 1  
8      while index < len(array):  
9          if array[index] < my_min:  
10             my_min = array[index]  
11             index = index + 1  
12     return my_min
```

Proof of Correctness for Finding Minimum

- Does it terminate?
 - Yes
 - But how do you explain it?
- What is the loop invariant?

```
1  def find_min(array):
2      """
3      Find the minimum...
4      Does this work?
5      """
6      my_min = array[0]
7      index = 1
8      while index < len(array):
9          if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```

Proof of Correctness for Finding Minimum

- Does it terminate?
 - Yes
 - But how do you explain it?
 - array is finite
 - index starts from 1
 - At the end of each loop, increment by 1 and we will eventually reach the end of the array to terminate
- What is the loop invariant?

```
1 def find_min(array):
2     """
3     Find the minimum...
4     Does this work?
5     """
6     my_min = array[0]
7     index = 1
8     while index < len(array):
9         if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```

Proof of Correctness for Finding Minimum

- Does it terminate?
 - Yes
 - But how do you explain it?
 - array is finite
 - index starts from 1
 - At the end of each loop, increment by 1 and we will **eventually reach the end of the array** to terminate
- What is the loop invariant?

```
1 def find_min(array):
2     """
3     Find the minimum...
4     Does this work?
5     """
6     my_min = array[0]
7     index = 1
8     while index < len(array):
9         if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```



Proof of Correctness for Finding Minimum

- Does it terminate?
 - Yes
 - But how do you explain it?
 - array is finite
 - index starts from 1
 - At the end of each loop, increment by 1 and we will eventually reach the end of the array to terminate
- What is the loop invariant?

```
1 def find_min(array):
2     """
3     Find the minimum...
4     Does this work?
5     """
6     my_min = array[0]
7     index = 1
8     while index < len(array):
9         if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```

Proof of Correctness for Finding Minimum

- Does it terminate?
 - But how do you explain it?
 - array is finite
 - index starts from 1
 - At the end of each loop, increment by 1 and we will eventually reach the end of the array to terminate
- What is the loop invariant?
 - How would you write it?
 - my_min initialized to the first element

```
1 def find_min(array):
2     """
3     Find the minimum...
4     Does this work?
5     """
6     my_min = array[0]
7     index = 1
8     while index < len(array):
9         if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```


Proof of Correctness for Finding Minimum

- Does it terminate?
 - But how do you explain it?
 - array is finite
 - index starts from 1
 - At the end of each loop, increment by 1 and we will eventually reach the end of the array to terminate
- What is the loop invariant?
 - How would you write it?
 - my_min initialized to the first element
 - my_min has the minimum value in array[0...index]

```
1 def find_min(array):
2     """
3     Find the minimum...
4     Does this work?
5     """
6     my_min = array[0]
7     index = 1
8     while index < len(array):
9         if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```

Proof of Correctness for Finding Minimum

- Does it terminate?
 - But how do you explain it?
 - array is finite
 - index starts from 1
 - At the end of each loop, increment by 1 and we will eventually reach the end of the array to terminate
- What is the loop invariant?
 - How would you write it?
 - my_min initialized to the first element
 - my_min has the minimum value in array[0...index]
- Combination of the above explains why the algorithm works!

```
1 def find_min(array):
2     """
3     Find the minimum...
4     Does this work?
5     """
6     my_min = array[0]
7     index = 1
8     while index < len(array):
9         if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```

Proof of Correctness for Finding Minimum

- Does it terminate?
 - But how do you explain it?
 - array is finite
 - index starts from 1
 - At the end of each loop, increment by 1 and we will eventually reach the end of the array to terminate
- What is the loop invariant?
 - How would you write it?
 - my_min initialized to the first element
 - my_min has the minimum value in array[0...index] ←
- Combination of the above explains why the algorithm works!

```
1 def find_min(array):
2     """
3     Find the minimum...
4     Does this work?
5     """
6     my_min = array[0]
7     index = 1
8     while index < len(array):
9         if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```

Index eventually reach the end of array; ie we have the minimum value of the entire array

Proof of Correctness for Finding Minimum

- Does it terminate?
 - But how do you explain it?
 - array is finite
 - index starts from 1
 - At the end of each loop, increment by 1 and we will eventually reach the end of the array to terminate
- What is the loop invariant?
 - How would you write it?
 - my_min initialized to the first element
 - my_min has the minimum value in array[0...index]
- Combination of the above explains why the algorithm works!

```
1 def find_min(array):
2     """
3     Find the minimum...
4     Does this work?
5     """
6     my_min = array[0]
7     index = 1
8     while index < len(array):
9         if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```

} Initialization

Proof of Correctness for Finding Minimum

- Does it terminate?
 - But how do you explain it?
 - array is finite
 - index starts from 1
 - At the end of each loop, increment by 1 and we will eventually reach the end of the array to terminate
- What is the loop invariant?
 - How would you write it?
 - my_min initialized to the first element
 - my_min has the minimum value in array[0...index]
- Combination of the above explains why the algorithm works!

```
1 def find_min(array):
2     """
3     Find the minimum...
4     Does this work?
5     """
6     my_min = array[0]
7     index = 1
8     while index < len(array):
9         if array[index] < my_min:
10             my_min = array[index]
11             index = index + 1
12     return my_min
```

} Initialization
} Maintenance

Questions?

Proof of Correctness for Algorithms

- This is commonly asked in the exam

Proof of Correctness for Algorithms

- This is commonly asked in the exam
 - We will show you an algorithm (not code usually)
 - Then ask you to explain why it is correct
 - <past year.jpg>

Proof of Correctness for Algorithms

- (b) (2 marks) Write a useful loop invariant for the following function which computes the sum of all even numbers in a list. You must write the loop invariant that holds at the end of each iteration of the for loop (write next to `#INVARIANT`). Using the loop invariant, prove that the function correctly computes the sum of all even numbers in the list.

```
def sumOfEvens(aList):  
    total = 0  
    n = len(aList)  
    for i in range(n):  
        if aList[i]%2 == 0:  
            total += aList[i]  
  
    #INVARIANT:  
  
    return total
```

`#INVARIANT`: total is the sum of all even numbers in `aList[i+1]`

At the end of the last iteration, $i = n - 1$. Thus, total is the sum of all even numbers in `aList[:n]` when the for loop terminates, i.e., total is the sum of all even numbers in the list.

Proof of Correctness for Algorithms

- (b) (1 mark) Write a loop invariant for the Floyd-Warshall algorithm that can be used to show that the algorithm correctly computes all-pairs shortest distances.

Questions?

Proof of Correctness for Binary Search

- Now let us try to binary search
 - Something we are all familiar with...
 - OR DO WE?

Proof of Correctness for Binary Search

- Now let us try to binary search
 - Something we are all familiar with...
 - OR DO WE?

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find the key because we use lo for the index of key
6      """
7      lo = 0
8      hi = len(array)
9      while lo < hi:
10         mid = (lo+hi) // 2
11         if key >= array[mid]:
12             lo = mid
13         else:
14             hi = mid
15     if len(array) > 0 and array[lo] == key:
16         print("key found at index " + str(lo))
17     else:
18         print("key not found")
```

Proof of Correctness for Binary Search

- Same questions
 - Do it terminate?
 - What is the loop invariant?

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find the key because we use lo for the index of key
6      """
7      lo = 0
8      hi = len(array)
9      while lo < hi:
10         mid = (lo+hi) // 2
11         if key >= array[mid]:
12             lo = mid
13         else:
14             hi = mid
15     if len(array) > 0 and array[lo] == key:
16         print("key found at index " + str(lo))
17     else:
18         print("key not found")
```

Proof of Correctness for Binary Search

- It doesn't terminate
 - Terminates when $lo \geq hi$
 - In the loop, $lo < mid < hi$ and each iteration move lo or hi to mid
 - So eventually lo and hi will meet it right?

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find
6      the key because we use lo for the index of key
7      """
8      lo = 0
9      hi = len(array)
10     while lo < hi:
11         mid = (lo+hi) // 2
12         if key >= array[mid]:
13             lo = mid
14         else:
15             hi = mid
16     if len(array) > 0 and array[lo] == key:
17         print("key found at index " + str(lo))
18     else:
19         print("key not found")
```

Proof of Correctness for Binary Search

- It doesn't terminate
 - Terminates when $lo \geq hi$
 - In the loop, $lo < mid < hi$ and each iteration move lo or hi to mid
 - So eventually lo and hi will meet it right?
 - No, we have a situation...
 - $Lo = 5$
 - $Hi = 6$
 - So $mid = 5$
 - Then what is $lo = mid$?

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find
6      the key because we use lo for the index of key
7      """
8      lo = 0
9      hi = len(array)
10     while lo < hi:
11         mid = (lo+hi) // 2
12         if key >= array[mid]:
13             lo = mid
14         else:
15             hi = mid
16     if len(array) > 0 and array[lo] == key:
17         print("key found at index " + str(lo))
18     else:
19         print("key not found")
```


Proof of Correctness for Binary Search

- It doesn't terminate
 - Terminates when $lo \geq hi$
 - In the loop, $lo < mid < hi$ and each iteration move lo or hi to mid
 - So eventually lo and hi will meet it right?
 - No, we have a situation...
 - $Lo = 5$
 - $Hi = 6$
 - So $mid = 5$
 - Then what is $lo = mid$?
 - How would you fix this?

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find
6      the key because we use lo for the index of key
7      """
8      lo = 0
9      hi = len(array)
10     while lo < hi:
11         mid = (lo+hi) // 2
12         if key >= array[mid]:
13             lo = mid
14         else:
15             hi = mid
16     if len(array) > 0 and array[lo] == key:
17         print("key found at index " + str(lo))
18     else:
19         print("key not found")
```

Proof of Correctness for Binary Search

- It doesn't terminate
 - Terminates when $lo \geq hi$
 - In the loop, $lo < mid < hi$ and each iteration move lo or hi to mid
 - So eventually lo and hi will meet it right?
 - No, we have a situation...
 - $Lo = 5$
 - $Hi = 6$
 - So $mid = 5$
 - Then what is $lo = mid$?
 - How would you fix this?
 - Change to while $lo < hi - 1$?

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find
6      the key because we use lo for the index of key
7      """
8      lo = 0
9      hi = len(array)
10     while lo < hi:
11         mid = (lo+hi) // 2
12         if key >= array[mid]:
13             lo = mid
14         else:
15             hi = mid
16     if len(array) > 0 and array[lo] == key:
17         print("key found at index " + str(lo))
18     else:
19         print("key not found")
```

Proof of Correctness for Binary Search

- It doesn't terminate
 - Terminates when $lo \geq hi$
 - In the loop, $lo < mid < hi$ and each iteration move lo or hi to mid
 - So eventually lo and hi will meet it right?
 - No, we have a situation...
 - $Lo = 5$
 - $Hi = 6$
 - So $mid = 5$
 - Then what is $lo = mid$?
 - How would you fix this?
 - Change to while $lo < hi - 1$?
 - Yes because hi is exclusive!
Initialized to $len(array)$!
 - So **search space shrink till size of 1**

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find
6      the key because we use lo for the index of key
7      """
8      lo = 0
9      hi = len(array)
10     while lo < hi:
11         mid = (lo+hi) // 2
12         if key >= array[mid]:
13             lo = mid
14         else:
15             hi = mid
16     if len(array) > 0 and array[lo] == key:
17         print("key found at index " + str(lo))
18     else:
19         print("key not found")
```

Proof of Correctness for Binary Search

- It doesn't terminate
- So please be careful, certain test cases can cause your program to run forever...
 - In your assignment, I set a **time limit** before I kill off your processing thread

Questions?

Proof of Correctness for Binary Search

- What is the loop invariant?

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find the key because we use lo for the index of key
6      """
7      lo = 0
8      hi = len(array)
9      while lo < hi:
10         mid = (lo+hi) // 2
11         if key >= array[mid]:
12             lo = mid
13         else:
14             hi = mid
15     if len(array) > 0 and array[lo] == key:
16         print("key found at index " + str(lo))
17     else:
18         print("key not found")
```

Proof of Correctness for Binary Search

- What is the loop invariant?
 - If the key exist in array[0...N] then the key exist in array[lo...hi]

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find the key because we use lo for the index of key
6      """
7      lo = 0
8      hi = len(array)
9      while lo < hi:
10         mid = (lo+hi) // 2
11         if key >= array[mid]:
12             lo = mid
13         else:
14             hi = mid
15     if len(array) > 0 and array[lo] == key:
16         print("key found at index " + str(lo))
17     else:
18         print("key not found")
```

Proof of Correctness for Binary Search

- What is the loop invariant?
 - If the key exist in array[0...N] then the key exist in array[lo...hi]
 - Kinda make sense from the if-else

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find the key because we use lo for the index of key
6      """
7      lo = 0
8      hi = len(array)
9      while lo < hi:
10         mid = (lo+hi) // 2
11         if key >= array[mid]:
12             lo = mid
13         else:
14             hi = mid
15     if len(array) > 0 and array[lo] == key:
16         print("key found at index " + str(lo))
17     else:
18         print("key not found")
```


Proof of Correctness for Binary Search

- What is the loop invariant?
 - If the key exist in array[0...N] then the key exist in array[lo...hi]
 - Kinda make sense from the if-else
 - And there isn't a need to be so complex...

```
1  def binary_search(array, key):
2      """
3      Binary search for key
4      Does this work?
5      Note: We don't terminate it earlier when we find the key because we use lo for the index of key
6      """
7      lo = 0
8      hi = len(array)
9      while lo < hi:
10         mid = (lo+hi) // 2
11         if key >= array[mid]:
12             lo = mid
13         else:
14             hi = mid
15     if len(array) > 0 and array[lo] == key:
16         print("key found at index " + str(lo))
17     else:
18         print("key not found")
```

Proof of Correctness for Binary Search

- What is the loop invariant?
 - If the key exist in array[0...N] then the key exist in array[lo...hi]
 - Kinda make sense from the if-else
 - And there isn't a need to be so complex...

- But you can do the following:
 1. Define the invariant
 2. Code based on the invariant

Questions?

Proof of Correctness

TL;DR

- Termination
- Loop invariant

- Termination
 - What updates/ step to ensure the loop will be exited
- Loop invariant

- Termination
 - What updates/ step to ensure the loop will be exited
 - Or function will reach base case?
- Loop invariant

- Termination
 - What updates/ step to ensure the loop will be exited
 - Or function will reach base case?
- Loop invariant
 - What doesn't change?
 - But what doesn't change but help you reach the output?
Or closer towards the answer.

Questions?

Thank you