CS2104 Problem Solving in Computer Science

Margaret Ellis,
Naren Ramakrishnan,
Sehrish Basir Nizamani

VIRGINIA TECH

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

VIRGINIA TECH

Python and Algorithms

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Day 1 Objectives

- Explain algorithms and efficiency
- Interpret flow charts
- Implement algorithms in python
- Experiment using LLMS to code and test

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Algorithms

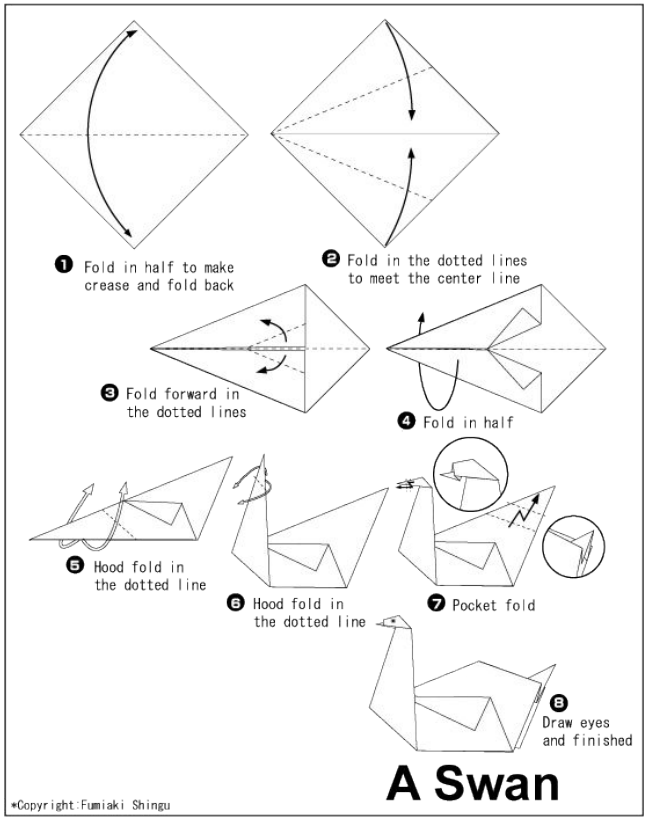
Algorithms are the threads that tie together most of the subfields of computer science.

Something magically beautiful happens when a sequence of commands and decisions is able to marshal a collection of data into organized patterns or to discover hidden structure.

Donald Knuth

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Example: Diagram for making an origami swan



© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Examples of Algorithms

- Recipe for Chicken and Macaroni Casserole
- Knitting pattern for a blanket
- Calculating a tip at a restaurant
- Finding the best deal on EXPO markers
- Finding the fastest route to Virginia Beach

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Definition of Algorithm

- **effective method (or procedure)**
  - a procedure that reduces the solution of some class of problems to a series of rote steps which, if followed to the letter, and as far as may be necessary, is bound to:
    - always give some answer rather than ever give no answer;
    - always give the right answer and never give a wrong answer;
    - always be completed in a finite number of steps, rather than in an infinite number;
    - work for all instances of problems of the class.
- **Algorithm**
  - an effective method expressed as a finite list of well-defined instructions for calculating a function

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Properties of an Algorithm

An algorithm must possess the following properties:

- ***finiteness***: The algorithm must always terminate after a finite number of steps.
- ***definiteness*****:** Each step must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case.
- ***input*****:** An algorithm has zero or more inputs, taken from a specified set of objects.
- ***output*****:** An algorithm has one or more outputs, which have a specified relation to the inputs.
- ***effectiveness*****:** All operations to be performed must be sufficiently basic that they can be done exactly and in finite length.

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

*Common **Elements** of Algorithms*

Acquire data (input): Some means of reading values from an external source; most algorithms require data values to define the specific problem (e.g., coefficients of a polynomial)

Computation: Some means of performing arithmetic computations, comparisons, testing logical conditions, and so forth...

Selection: Some means of choosing among two or more possible courses of action, based upon initial data, user input and/or computed results

Iteration: Some means of repeatedly executing a collection of instructions, for a fixed number of times or until some logical condition holds

Report results (output): Some means of reporting computed results to the user, or requesting additional data from the user

Simple and list variables: Name and store data values

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Modern CS Algorithms

- **Matching users to servers**, using **Gayle-Shapely Algorithm** for matching medical students to their residency placements
  - This is a beautiful algorithm for fair matching. Simple, elegant and effective. In its core form, it's also straightforward to implement. Has numerous applications. See: Stable marriage problem – Wikipedia
- **Music Search** using **Fast Fourier Transforms (FFT)**
  - Music recognition is done by converting it into frequency domain using FFT. FFT has implementations in number of languages. See this article for a great start: Shazam It! Music Recognition Algorithms, Fingerprinting, and Processing.

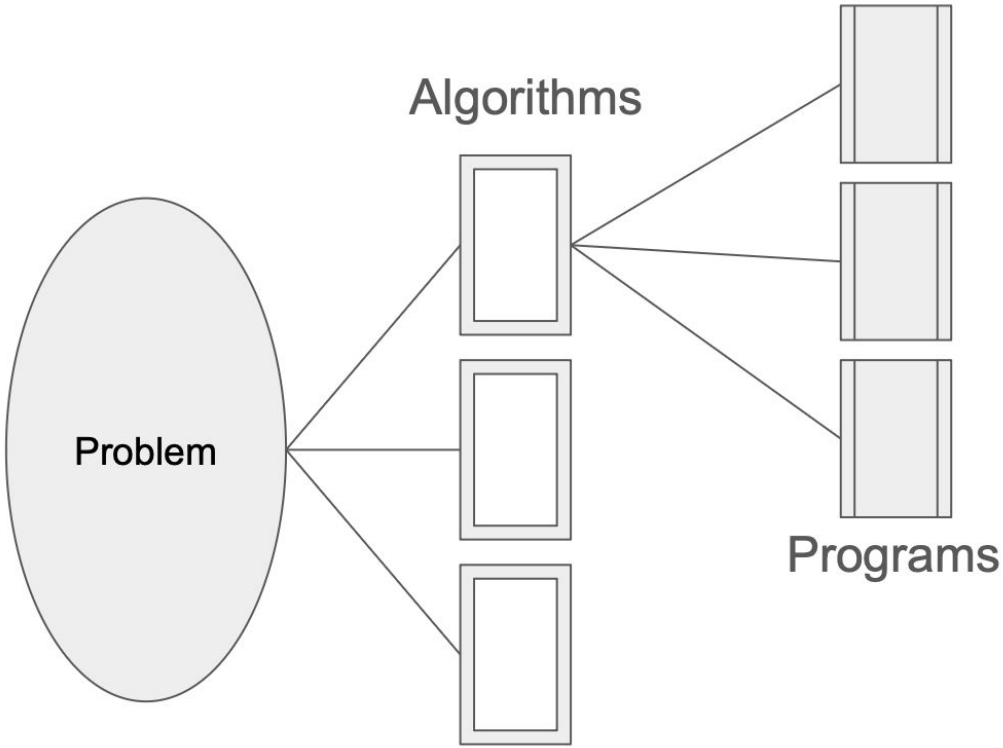
© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Problems, Algorithms and Programs

- For each problem or class of problems, there may be many different algorithms.
- For each algorithm, there may be many different implementations (programs).

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Problems vs Algorithms vs Programs



© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

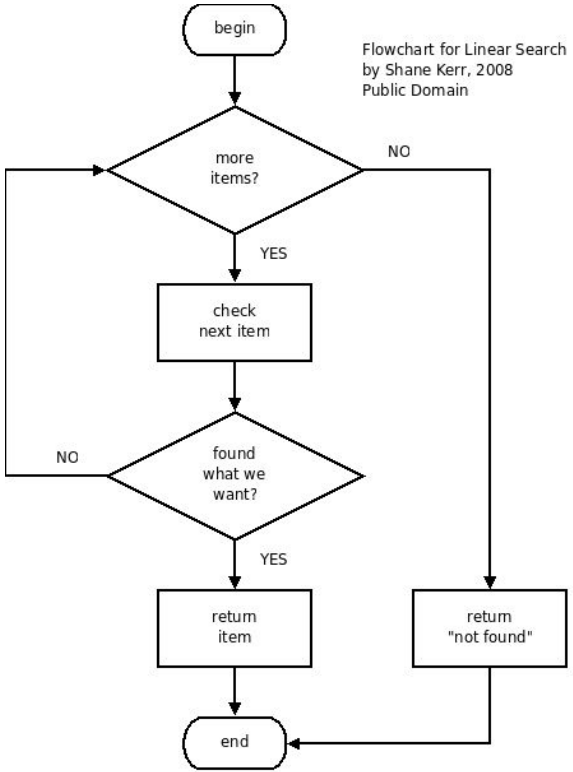
Expressing Algorithms

An algorithm may be expressed in a number of ways:

- *natural language*: usually verbose and ambiguous
- *flow charts*: avoid most (if not all) issues of ambiguity; difficult to modify w/o specialized tools; largely standardized
- *pseudo-code*: also avoids most issues of ambiguity; vaguely resembles common elements of programming languages; no particular agreement on syntax
- *programming language*: tend to require expressing low-level details that are not necessary for a high-level understanding

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Flowchart Example of Linear Search



© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Testing Correctness

- How do we know whether an algorithm is actually correct?
- First, perform logical analysis
- Second, testing
- **testing can never prove that the algorithm produces correct results in all cases**.

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Recap on Unit Test

1. Set up the initial conditions for your test (create any objects needed, place them in the correct state, put everything necessary where it needs to be, etc.).
2. Call the method you are testing.
3. Check that the behavior you expected has occurred. This could involve checking the return value of the method, or checking the state of the objects involved in the test. Be sure to check everything you expect to happen, not just the most obvious item.

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Polya's Problem Solving Steps

Understand your problem solving process then decide how and when an LLM can and cannot assist.

- Step 1: Understand the problem.
  - *Have concrete examples (these become test cases)*
- Step 2*:* Devise a plan (translate).
  - *Break the problem down into manageable pieces/step*
- Step 3: Carry out the plan (solve).
  - *Iterate on 3 and 4 piece by piece*
- Step 4: Look back (check and interpret).
  - *Need to really understand the problem!*

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

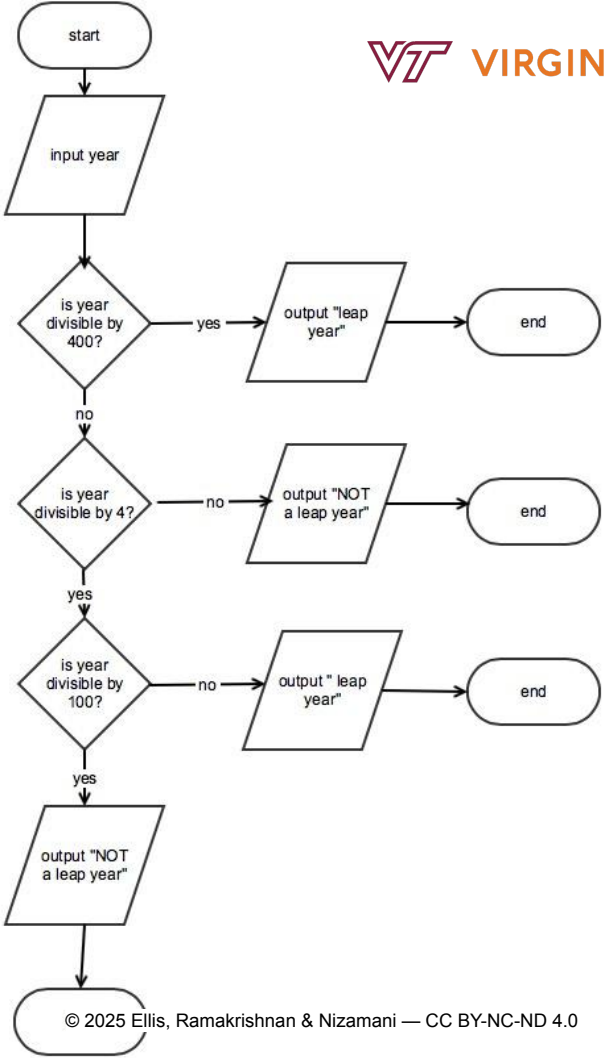
VIRGINIA TECH

Class Activity -Turn Flowcharts into Python Code New Algorithm:

1. Turn Leap Year Flowchart into python code
2. Write test cases for each branch in the flowchart
3. Come up with alternate ways to code it

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Leap Year Flowchart



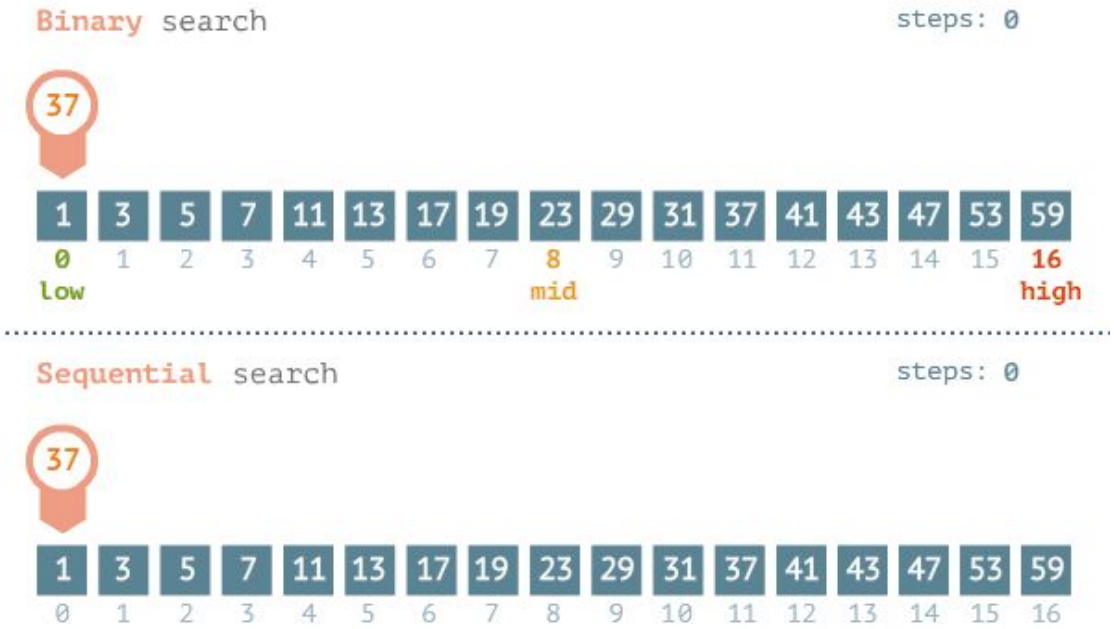
© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Class Activity

Do python first steps in class (use vscode in Rlogin)

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Animation of Binary Search vs Sequential



© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Efficiency of Linear Search

- In many applications, it is easy to come up with a numeric value that specifies the problem size, which is generally denoted by the letter *N*.
- For most array applications, the problem size is simply the size of the array.
- In the worst case—which occurs when the value you're searching for comes at the end of the array or does not appear at all—linear search requires *N* steps.
- On average, it takes approximately half that time.
    - A linear search of 10 items takes an average of 5 looks
    - A linear search of 800 items takes an average of 400 looks

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Efficiency of Binary Search

- The running time of binary search also depends on the number of elements, but in a profoundly different way.
- **On each step in the process, the binary search algorithm rules out half of the remaining possibilities.**
- In the worst case, the number of steps required is equal to the number of times you can divide the original size of the array in half until there is only one element remaining.

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Efficiency of Binary Search (Cont.)

- In other words, what you need to find is the value of *k* that satisfies the following equation:

$$1 = \frac{N}{\underbrace{2 \times 2 \times 2 \times \cdots \times 2}_{k \text{ times}}}$$

$$1 = \frac{N}{2^k}$$

$$2^k = N$$

$$k = \log_2 N$$

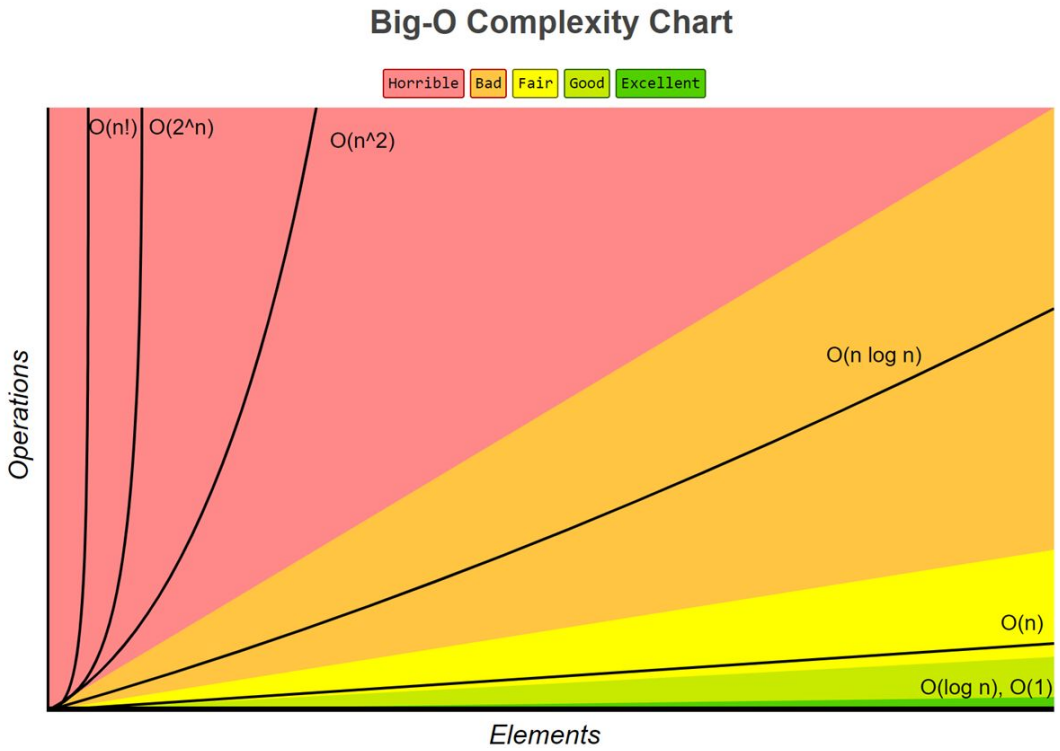
© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Assessing Algorithmic Efficiency

- The discussion of the efficiency of the various searching and sorting algorithms illustrates a fundamental computer science technique called **algorithmic analysis**.
- One of the most important problems in algorithmic analysis is deducing the **computational complexity** of an algorithm, which is the relationship between the size of the problem and the expected running time.

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Examples of Algorithmic Efficiency



© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Big-O Notation

- The most common way to express computational complexity is to use **big-O notation**, which was introduced by the German mathematician Paul Bachmann in 1892.
- Big-O notation consists of the letter O followed by a formula that offers a qualitative assessment of running time as a function of the problem size, traditionally denoted as *N*.
- For example, the computational complexity of linear search is O(*N*) and the computational complexity of radix sort is *O(N* log *N*)
- If you read these formulas aloud, you would pronounce them as “big-O of *N* ” and “big-O of *N* log *N* ” respectively.

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Exercise: Computational Complexity

Assuming that none of the steps in the body of the following **`for`** loops depend on the problem size stored in the variable **`n`**, what is the computational complexity of each of the following examples:

a)

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        . . . loop body . . .
    }
}
```

b)

```
for (int k = 1; k <= n; k *= 2) {
    . . . loop body . . .
}
```

c)

```
for (int i = 0; i < 100; i++) {
    for (int j = 0; j < i; j++) {
        . . . loop body . . .
    }
}
```


© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Algorithm's Time Complexity

Big O Notation	Name	Example(s)
$O(1)$	Constant	# Odd or Even number, # Look-up table (on average)
$O(\log n)$	Logarithmic	# Finding element on sorted array with **binary search**
$O(n)$	Linear	# Find max element in unsorted array, # Duplicate elements in array with Hash Map
$O(n \log n)$	Linearithmic	# Sorting elements in array with **merge sort**
$O(n^2)$	Quadratic	# Duplicate elements in array **(naïve)**, # Sorting array with **bubble sort**
$O(n^3)$	Cubic	# 3 variables equation solver
$O(2^n)$	Exponential	# Find all subsets
$O(n!)$	Factorial	# Find all permutations of a given set/string

https://adrianmejia.com/most-popular-algorithms-time-complexity-every-programmer-should-know-free-online-tutorial-course/

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Binary Search

```
def binarySearch (key,values):
    lh = 0
    rh = len(values)  - 1
    while (lh <= rh):
        mid =  (lh + rh) // 2
        if (key == values[mid]):
            return mid
        if (key < values[mid]):
            rh = mid - 1
        else:
            lh = mid + 1
    return - 1
```

Given totals = {23, 44, 55, 146, 255, 300, 440, 575}

Trace binarySearch(440, totals)

List all the values of lh, rh and mid.

0	1	2	3	4	5	6	7
23	44	55	146	255	300	440	575

lh	rh	mid
0	7	3
4	7	5
6	7	6 ✅

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

VIRGINIA TECH

LLMs to help (as allowed by assignments)

Sample Response: This unit test covers various scenarios, including cases where the key is present in the list, not present in the list, an empty list, and a single-element list. You can run this test to verify that your `binarySearch` function works correctly. If you have any other questions or need further assistance, feel free to ask!

import

Sample Response Code:

```
import unittest

def binarySearch(key, values):
    lh = 0
    rh = len(values) - 1

    while lh <= rh:
        mid = (lh + rh) // 2
        if key == values[mid]:
            return mid
        if key < values[mid]:
            rh = mid - 1
        else:
            lh = mid + 1
    return -1….
```

```
…
#co-pilot 12/29/24
class TestBinarySearch(unittest.TestCase):
    def test_binary_search(self):
        # Test cases where the key is present in the list
        self.assertEqual(binarySearch(3, [1, 2, 3, 4, 5]), 2)
        self.assertEqual(binarySearch(1, [1, 2, 3, 4, 5]), 0)
        self.assertEqual(binarySearch(5, [1, 2, 3, 4, 5]), 4)

        # Test cases where the key is not present in the list
        self.assertEqual(binarySearch(0, [1, 2, 3, 4, 5]), -1)
        self.assertEqual(binarySearch(6, [1, 2, 3, 4, 5]), -1)

        # Test cases with an empty list
        self.assertEqual(binarySearch(3, []), -1)

        # Test cases with a single element list
        self.assertEqual(binarySearch(1, [1]), 0)
        self.assertEqual(binarySearch(0, [1]), -1)

if __name__ == '__main__':
    unittest.main(argv=[''], exit=False)
```

Unit test class

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Find the Longest Run

- Given a list of values, finds the length of the longest sequence of values that are in strictly increasing order.
- Concrete example?
  - longest_run([5,4,8,1,2,3,1]) should output 3 as the longest run
  - What variables would you use?
  - What procedures would you take?
  - How do you test the correctness of your algorithm?

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Find the Longest Run (Cont.)

```
# Given a list of values, finds the length of the longest sequence
# of values that are in strictly increasing order.
def longest_run(arr):
    size = len(arr)
    if size <= 0:                       # if list is empty, no runs...
        print("The list is empty")
    else:
        current_position = 0        # start with first element in list
        max_run_length = 1          # it forms a run of length 1
        this_run_length = 1
…
```


© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Find the Longest Run (Continue)

```
…
# QUESTION TO PONDER:  is this Algorithm Correct?
        while  current_position < size - 1:
            if ( arr[current_position] < arr[current_position + 1] ):
                this_run_length = this_run_length + 1
            else:
                if ( this_run_length > max_run_length ):
                    max_run_length = this_run_length
                this_run_length = 1
            current_position = current_position + 1
        print("The max run length is: ", max_run_length)
```

```
[0, 1, 2, 3]
```

```
[1, 3, 0, 5, -1, 1, 2, 3]
```


© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Unit Testing…

Demo Class Activity: Unit Testing,

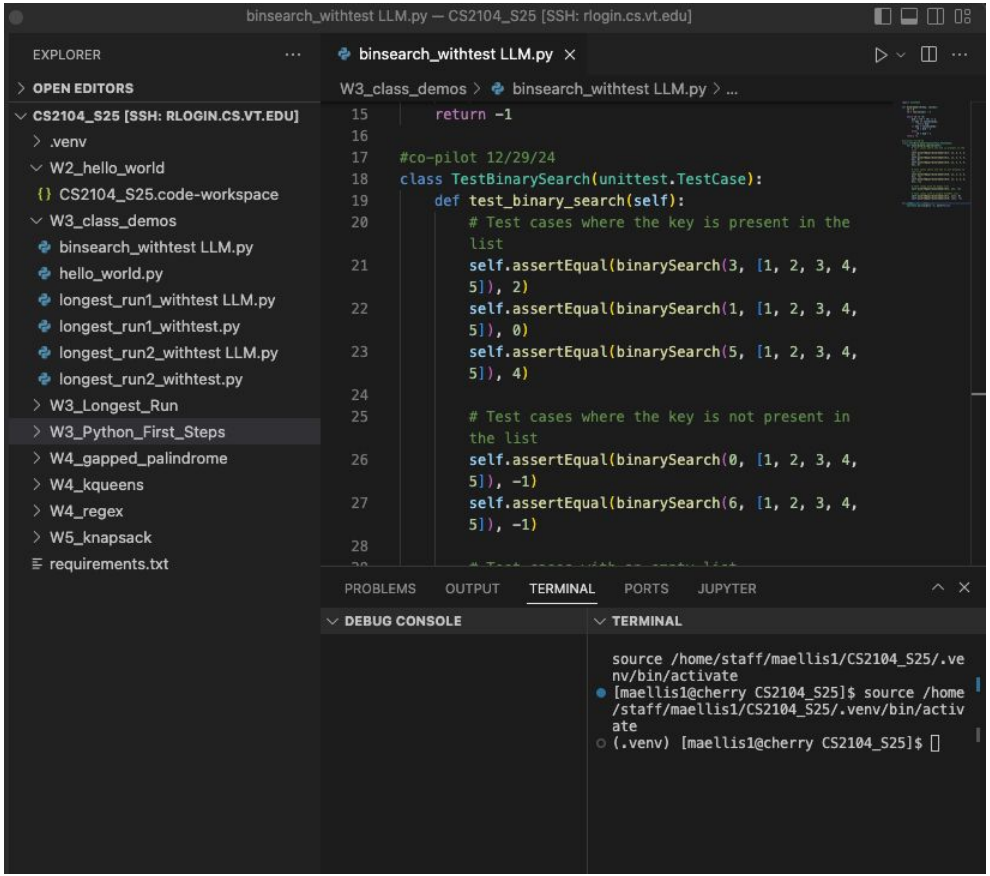
- students follow along
- create W3_demo folder in vscode
- click on the download arrow in canvas week 5 to get the py version of each file
  - binsearch_withtest_LLM.py
  - longest_run1_withtest.py
  - longest_run2_withtest.py
  - longest_run1_withtest LLM.py
  - longest_run2_withtest LLM.py
- Drag files into the directory (they get uploaded to rlogin)

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Unit Testing (Cont.)

Demo Class Activity: Unit Testing,

- binsearch_withtest_LLM.py should pass



© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

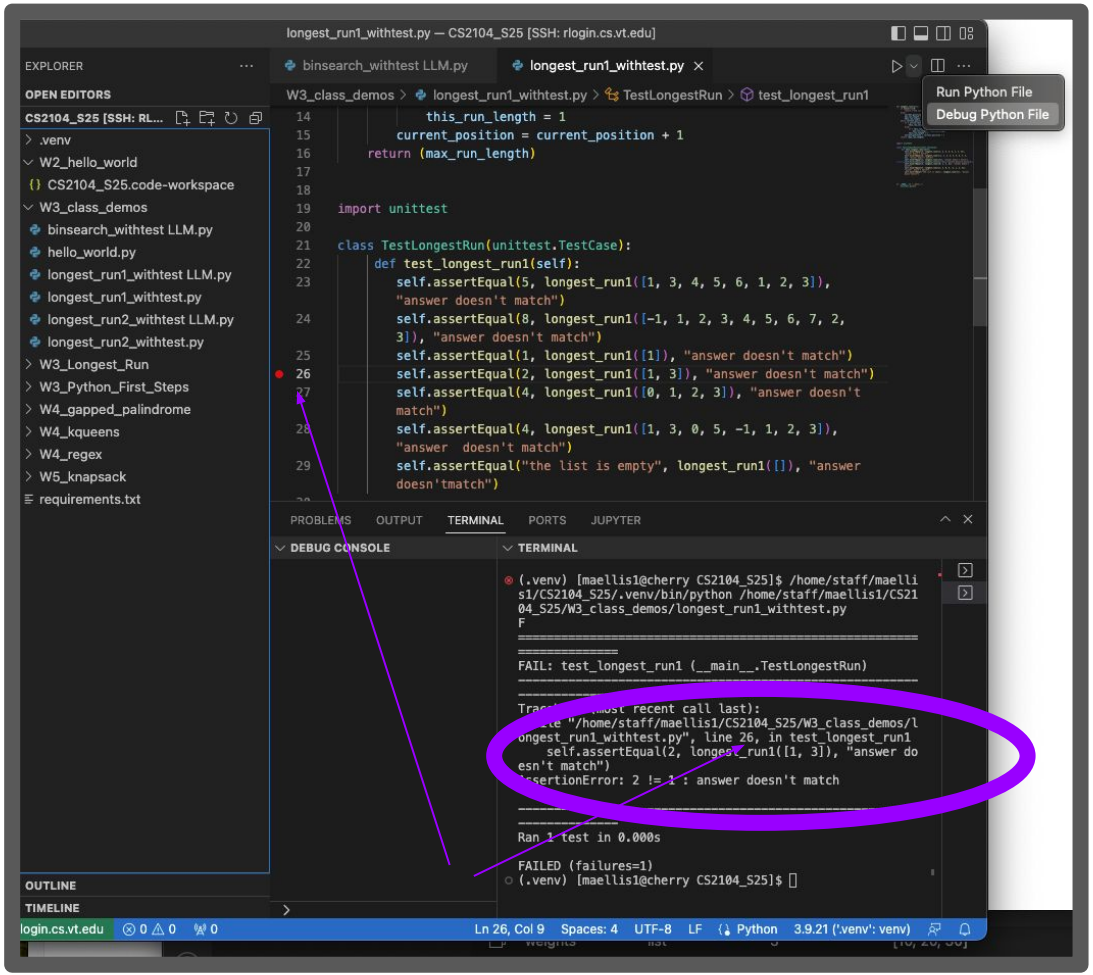
VIRGINIA TECH

Unit Testing cont.

Demo Class Activity: Unit Testing

- Longest_run1_withtest.py
  - should not pass
  - (investigate with debugger)
    - Set breakpoint in the gutter of failing test
    - Use buttons to trace code

    - Watch values in variables
  - Fixed version (can look side-by-side) Longest_run2_withtest.py should pass



© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

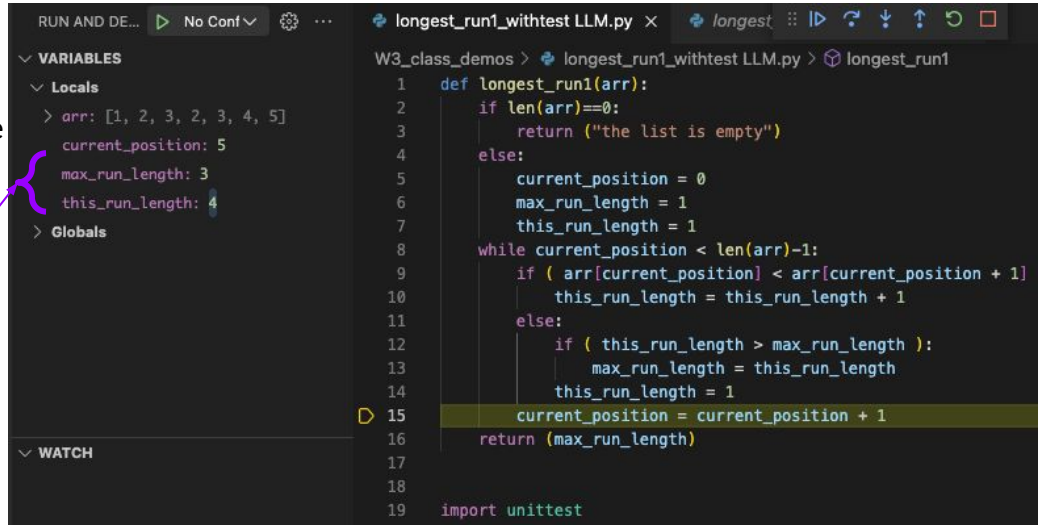
VIRGINIA TECH

Unit Testing (Continue)

Demo Class Activity: Unit Testing (same code, different tests)

- longest_run1_withtest LLM.py
  - Should not pass
  - Investigate with debugger
  - Use buttons to trace code(hover to see functionality)

  - Watch values in variables
- longest_run2_withtest LLM.py
  - Should pass
  - Compare side-by-side



© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Maximum Subsequence Sum Problem

- Given (possibly negative) integers A_1, A_2, … , A_N, find the maximum value of:

$$\sum_{k=1}^{j} \left(A_k \right)$$

- The maximum subsequence is defined as zero if all the integers are negative, (i..e., the subsequence of the empty set is zero).
- Consider the sequence: 4 -3 5 -2 -1 2 6 -2

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Maximum subsequence sum solution 1

```
# finds the max subsequence sum in the list values

def max_subsequence_sum_1(values):

    size = len(values)
    max_sum = 0
    i = 0
    while i < size:
        j = i
        while  j < size:
            this_sum = 0
            k = i
            while  k <= j:
                this_sum = this_sum + values[k]
                k = k + 1
            if (this_sum > max_sum ):
                max_sum = this_sum
            j = j + 1
        i= i + 1
    return(max_sum)
```

If size = 1000, how many times are the statements in the innermost loop executed?

Can you/we do better?

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Maximum subsequence sum solution 2

- Reuse the subsum, so just adds the newest value in the subsequence to what was previously calculated

4 -3 5 -2 -1 2 6 -2

[4]
[4 -3]
[4 -3 5]
[4 -3 5 -2]
[4 -3 5 -2 -1]
[4 -3 5 -2 -1 2]
[4 -3 5 -2 -1 2 6]
[4 -3 5 -2 -1 2 6 -2]

[-3]
[-3 5]
[-3 5 -2]
[-3 5 -2 -1]
[-3 5 -2 -1 2]
[-3 5 -2 -1 2 6]
[-3 5 -2 -1 2 6 -2]

• • •

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Maximum subsequence sum solution 2 (Continue)

```
# finds the max subsequence sum in the list values
def max_subsequence_sum_2(values):

    size = len(values)
    max_sum = 0
    i = 0

    while i < size:
        j = i
        this_sum = 0
        while  j < size:
            this_sum = this_sum + values[j]
            if (this_sum > max_sum ):
                max_sum = this_sum
            j = j + 1
        i= i + 1
    return(max_sum)
```

If size = 1000, how many times are the statements in the innermost loop executed?

Can you/we do even better?

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Maximum subsequence sum solution 3

```
# finds the max subsequence sum in the list values
def max_subsequence_sum_3(values):
    size = len(values)
    max_sum = 0
    this_sum = 0
    j = 0
    while (j <= size -1):
        this_sum = this_sum + values[j]
        if ( this_sum > max_sum ):
            max_sum = this_sum
        elif ( this_sum < 0 ):
            this_sum = 0
        j = j + 1
    return max_sum
```

Try it:

4 -3 5 -2 -1 2 6 -2

One must observe that if any list[i] is negative then it cannot be the beginning of the optimal sequence since any sequence starting with it would be improved by omitting it and starting with list[i+1].

Also by the same logic any negative subsequence cannot be the start of the optimal subsequence.

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Maximum subsequence sum solution LLM

```
def max_subsequence_sum(arr):
    max_sum = 0
    current_sum = 0

    for num in arr:
        current_sum += num
        if current_sum > max_sum:
            max_sum = current_sum
        if current_sum < 0:
            current_sum = 0

    return max_sum
```

```
# Given sequence
sequence = [4, -3, 5, -2, -1, 2, 6, -2]

# Calculate the maximum subsequence sum
result = max_subsequence_sum(sequence)

print(f"The maximum subsequence sum is:
{result}")
```


© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Research on LLMS in CS Education

- Students using LLMS had reduced ability to write code from scratch but similar ability to trace and read code
  - source:CS1-LLM: Integrating LLMs into CS1 Instruction
- Students who used AI less or later in problem solving process had higher grades in course (don't know cause-effect)
  - source:Interactions with How Novices Use LLMs to Solve Programming Problems
- Students with strong programming foundations benefited more from LLMs, while those without a solid base may be hindered by them
  - Sources: Insights from Social Shaping Theory: The Appropriation of Large Language Models in an Undergraduate Programming Course and The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0

Leetcode Assignment

1. Looks over assignment together
2. Open up Leetcode together(set to Python)
3. VT approved LLM (bing.com/chat)
4. Considerations
   - *Can also experiment in vscode*
   - *Can also create your own test cases*
   - *Keep trying to get more efficient solutions*

© 2025 Ellis, Ramakrishnan & Nizamani — CC BY-NC-ND 4.0