CoP#2 Activity Guide

As you work through today’s activities, use this worksheet to record your answer to various questions as they are assigned.

Activity 1a – Introducing Linear Search

Before beginning, please download activity1.py from the class website.

1. How many checks are made for each of the two examples already provided in the code?
2. How many checks are made if *target* is the last item in the list? [Modify the code to confirm].
3. How many checks are made if *target* is a number **not** in the list? [Modify the code to confirm].
4. How is the efficiency of this algorithm impacted when applied to larger lists?

To test this, add the following line of code at the bottom of the program: A black background with white text

Description automatically generated

Run the program several times to observe what happens both when 25 is in the random list and when it is absent.

Activity 1b– Exploring Linear Search

Use the following code segment to complete the following tables.

A screen shot of a computer code

Description automatically generated

Complete the table using the name from the first column to fill in the blank in the call to **getLocation()** using the list declared below.

friends = ["Tad", "Harume", "Joy", "Aurora", "Gloria"]

getLocation(friends, \_\_\_\_\_\_\_)

|  |  |  |
| --- | --- | --- |
| **Name to search for in friends** | **Number of executions before getLocation() returns an answer** | **Best, Average, or Worst Case?** |
| "Tad" |  |  |
| "Joy" |  |  |
| "Gloria" |  |  |
| "Brandon" |  |  |

Complete the table using the name from the first column to fill in the blank in the call to **getLocation()** using the array declared below.

friends2 = ["Lisa", "Angela", "Pamela", "Renee", "Agnes", "Agatha", "Germaine","Jack", "Jolene"]

getLocation(friends, \_\_\_\_\_\_\_)

|  |  |  |
| --- | --- | --- |
| **Name to search for in friends** | **Number of executions before getLocation() returns an answer** | **Best, Average, or Worst Case?** |
|  |  | Best Case |
|  |  | Average Case |
|  |  | Worst Case |

## Challenge Questions

Consider a list of ***n*** items.

For any linear search, the best case will take \_\_\_\_\_\_\_\_\_\_ executions.

For a linear search over a list of length ***n***, the worst case will take \_\_\_\_\_\_\_\_\_\_ executions.

For a linear search over a list of length ***n***, the average case would take \_\_\_\_\_\_\_\_\_\_ executions.

Activity 2a – Introducing Binary Search

Before beginning, please download activity2.py from the class website.

1. Take a look at the code in activity2.py. Write down what you think this program will do. There are no wrong answers.
2. Experiment with the program by making the following modifications, then run the program to observe the results. How many checks are used?
   1. Search for 10
   2. Search for 60
   3. Search for 25
   4. Switch so you are searching the secondSet and search for 81
   5. Switch so you are searching the secondSet and search for 45

Activity 2b – Exploring Binary Search

Set out the following cards

A full deck of cards

Description automatically generated

1. Using the linear search on these 5 cards (activity1a.py), what is the execution count to find:
   1. 2
   2. 3
   3. 4
   4. 5
   5. 8
   6. J (Let’s treat this like an 11)
2. What would the worst-case execution count be if we used a linear search to find a specific card?
3. Using the binary search on these 5 cards (activity2a.py), what is the execution count to find:
   1. 2
   2. 3
   3. 4
   4. 5
   5. 8
   6. J (Let’s treat this like an 11)
4. What would the worst-case execution count be if we used a binary search to find a specific card?

Now double the number of cards in your layout by putting out the cards from A-10 in order (let’s treat A as the #1)

1. Using the binary search on these 10 cards, what is the execution count to find:
   1. 2
   2. 3
   3. 4
   4. 5
   5. 8
   6. J (Let’s treat this like an 11)
2. Given the worst case linear search execution counts, complete the table with your prediction of the worst case execution counts of your new algorithm given the length of the list.

|  |  |  |
| --- | --- | --- |
| **List length** | **Worst case linear search execution count** | **Worst case execution count for new algorithm** |
| 1 | 1 |  |
| 3 | 3 |  |
| 5 | 5 |  |
| 9 | 9 |  |

1. How does increasing the length of the list impact the execution count?
2. How does the execution count for the binary search compare to the execution count for the linear search?
3. How is a binary search more efficient than a linear search algorithm?
4. What is one benefit and one limitation of the binary search algorithm?

Activity 3a – Introducing Recursion through unplugged activities

You will complete two activities to explore the idea of recursion

* Wall walking – Model a program that navigates a robot to a wall and stops before they crash.
  + Supplies:
    - Method Cards
* Coloring – Model a program that colors specific shapes in an image.
  + Supplies:
    - Method Cards
    - Coloring page

Half the class will do wall walking first, the other half will do coloring first. When you are done with the first activity, find a group that was doing the other activity and exchange the “method cards”

**Directions**

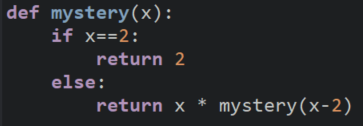
1. Retrieve all necessary materials listed above.
2. One student should be the Computer, and one student will be the Counter.
3. The student acting as the Computer starts with the stack of cards. All cards should begin Side A up.
4. The Computer will read Side A and do the action indicated by the method.
5. Once the Computer has completed the action, they will hand the card to the Counter.
6. The Computer will repeat steps 4 and 5 until the card indicates they should stop.
7. The Counter will return the requested information once indicated.

Complete this table only when directed by your facilitator to do so:

|  |  |  |
| --- | --- | --- |
|  | **Recursive Call** | **Base Case** |
| **Wall Walking** |  |  |
| **Coloring** |  |  |

Activity 3b – Exploring Recursion in Code

**Problem 1**



What is the base case?

What is the recursive call?

Trace the code segment in the space below to determine the output of the call **mystery(10)**.

**Problem 2**

A black background with white text

Description automatically generated

What is the base case?

What is the recursive call?

Trace the code segment in the space below to determine the output of the call **division(14)**.

Activity 3c

Re-examine the Scratch code listed below and identify the recursive call and the base case in each.

|  |  |  |
| --- | --- | --- |
|  | **Recursive Call** | **Base Case** |
| **Fractal Fern** |  |  |
| **Sierpinski Triangle** |  |  |

Activity 3d

Consider two different recursive implementations of the Binary Search algorithm

* activity3\_version1.py models the code your textbook authors use in section 5.4
* activity3\_version2.py mimics the code that Code.org uses in the CS-A curriculum

1. In version 1, what is the base case?
2. In version 1, what is the recursive call?
3. Describe how version 1 works.
4. In version 2, what is the base case?
5. In version 2, what is the recursive call?
6. Describe how version 2 works.