Runtime Analysis

Computer Science - Week 9  
Jul 29, 2022 - Version 1.3.0

Please make sure that all members of the group place their UD **email** AND **name** below.

Choose roles following the [instructions here](https://blockpy.cis.udel.edu/assignments/reading/bakery_appendix_pogil).

You should work in groups of 3. If you cannot find 3 group members, then work in groups of 2.

| **Role** | **Name** | **Email** |
| --- | --- | --- |
| **Manager** | Zach Phillips | phillizr@udel.edu |
| **Speaker** | Zach Phillips | phillizr@udel.edu |
| **Recorder** | Dakota | dkelley@udel.eddu |

# 1) Warmup

For each of the algorithms below, identify how many lines of actual code there are and how many steps it will take to execute.

Remember that a line of actual code does not include blank lines and hashtag (#) comments.

When counting steps, do not include lines that you return to (but do include duplicate lines).

| **Algorithm** | **Lines** | **Steps** |
| --- | --- | --- |
| bunnies = 7 ears = bunnies \* 2 print(ears) | 3 | 3 |
| grades = [99, 87]  count = 0  **for** grade **in** grades:  count = count + 1  print(count) | 5 | 8 |
| **def** add(a: int, b: int) -> int:  **return** a + b  total = add(1, 2) + add(3, 4)  print(total) | 4 | 9 |
| rainy = True  temperature = 19  **if** rainy **and** temperature <= 32:  print("Snowy")  **else**:  print("Not snowy") | 6 | 4 |

# 2) Introducing the Runtime Case Builder

In this lab, we are going to use a tool called the Runtime Case Builder that makes it easier to measure the steps that a Python program takes, and relate it back to the input by making a graph.

As an example, let us look at a specific algorithm for finding the length of the list:

def summate(values: [int]) -> int:

total = 0

for value in values:

total = total + value

return total

print(summate([1, 2, 3]))

print(summate([]))  
print(summate([4, 5, 6, 7]))

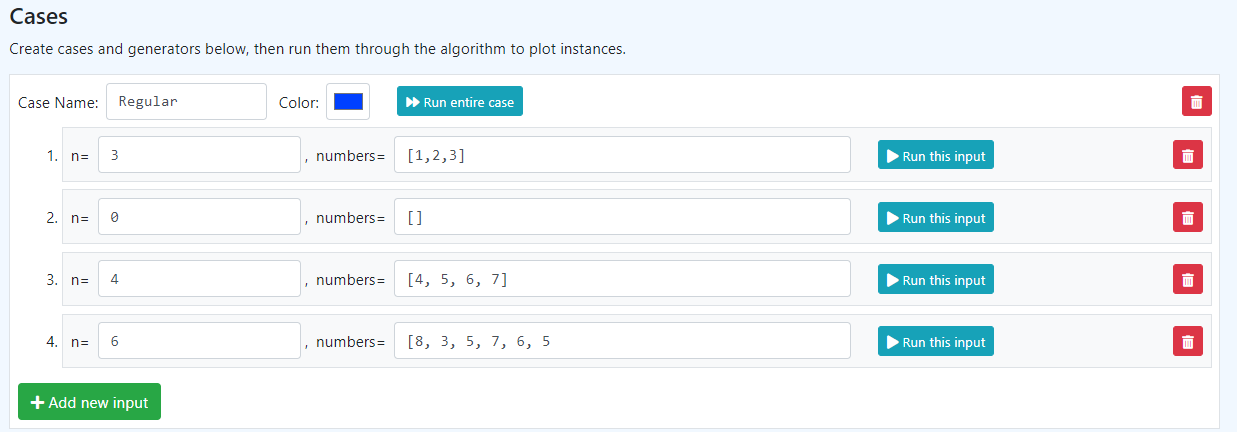
print(summate([8, 3, 5, 7, 6, 5]))

* The first call to summate has three elements (n=3), and requires 3 loop iterations.
* The second call to summate has zero elements (n=0), and requires 0 loop iterations.
* The third call to summate has four elements (n=4), and requires 4 loop iterations.
* The fourth call to summate has six elements (n=6), and requires 6 loop iterations.

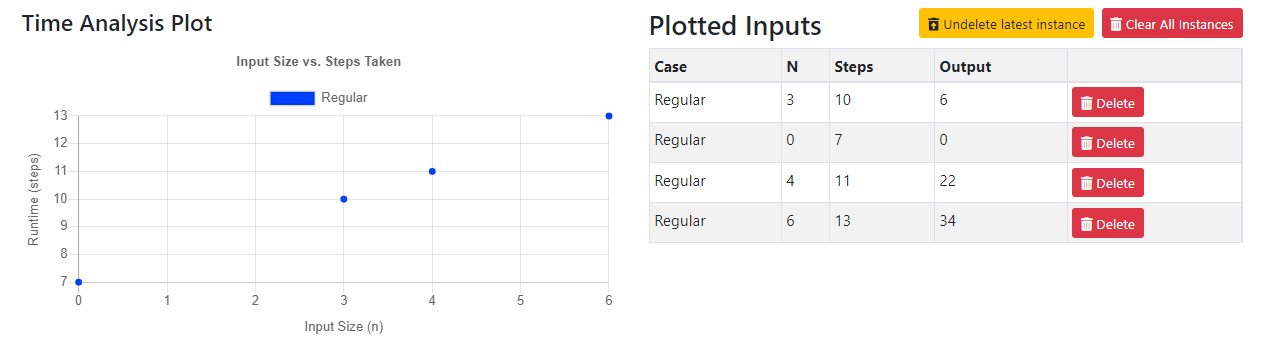
Now, in addition to the loop steps, there is additional overhead to define the function, initialize the variable, return the value, call the function, and print the result. In practice, these actually end up taking 7 additional steps. So the precise formula that our tool finds is as follows:

* steps(n) = n \* 3 + 7

Using the Runtime Case Builder, you can set up all these values as a *Case* of *Inputs*.

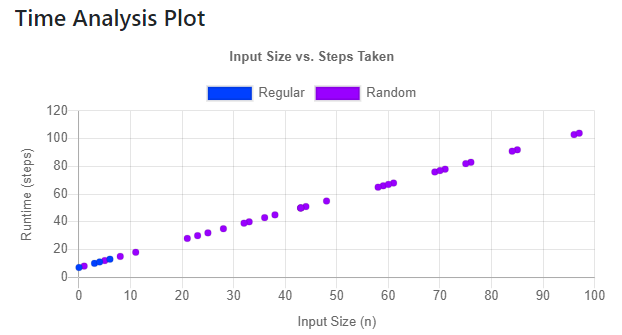


Then, when you click the [Run entire case] button, you will generate a plot like this:



The N, Steps, and Printed Output is shown on the right hand side, matching the values we predicted from the formula before. Usually, however, Computer Scientists don’t care about the exact steps - instead, they care about the shape of the graph. In this case, we can see that the plot forms a diagonal line, indicating a **linear** relationship between the N and the steps. We could make this relationship more clear by introducing more test cases with randomized data.

<https://acbart.github.io/runtime-case-builder/?preload=RCB_summate_function.json>



As another example, consider the algorithm below:

area = N \* N

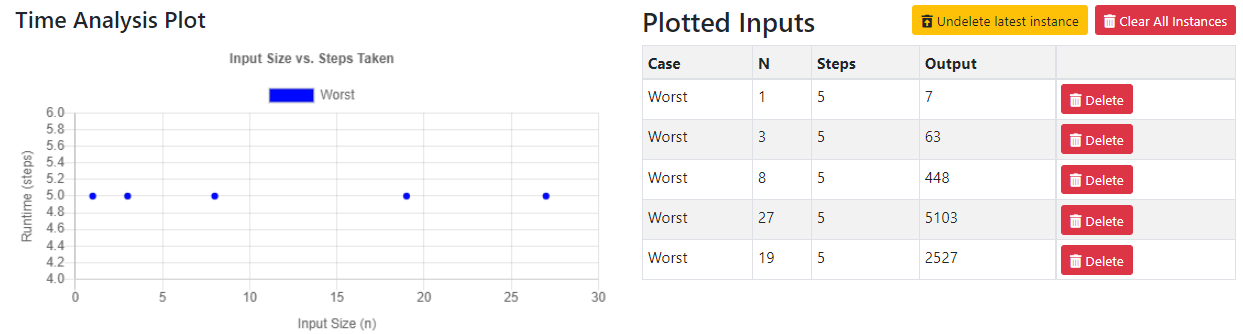
depth = 7

volume = area \* depth

print(volume)

It doesn’t matter what value you pass in for N, the number of steps taken will always be the same (5, since calling a function takes an extra step in Runtime Case Builder). The resulting graph is a horizontal line, with the same number of steps for each possible N that you use as input.

<https://acbart.github.io/runtime-case-builder/?preload=RCB_volume_calculation.json>



In the rest of this lab, you will work together to determine the relationship between N and Steps for different algorithms. You will create graphs like the one that we have here to make your “case” about the runtime that you believe an algorithm has.

# 3) Using the Runtime Case Builder

For each of the following problems, click the URL and read over the algorithm. Write a sentence describing what you think the algorithm does, in terms of its inputs and outputs.

Then, predict what kind of relationship (linear, constant, or quadratic) you expect to see for this algorithm.

Next, create and run a set of inputs to generate a graph that demonstrates the time function that you previously predicted. You should have **at least** **6** plotted points per algorithm.

Finally, use the “Create Report” button to generate a page that you can copy/paste into this Google Doc.

## Question 1

URL: <https://acbart.github.io/runtime-case-builder/?preload=RCB_add_ends.json>

**BEFORE YOU RUN**: Delete the 4th, 5th, and 6th cases’ values and replace them with your own.

A) Brief description of algorithm:

| Add the first number of the list to the last number of the list. |
| --- |

B) Predicted relationship:

| Constant |
| --- |

C) Copy and paste your created Report below, including the graph you made.

### **Add Ends**

#### **Algorithm**

n = ???

numbers = ???

def add\_ends(values: [int]) -> int:

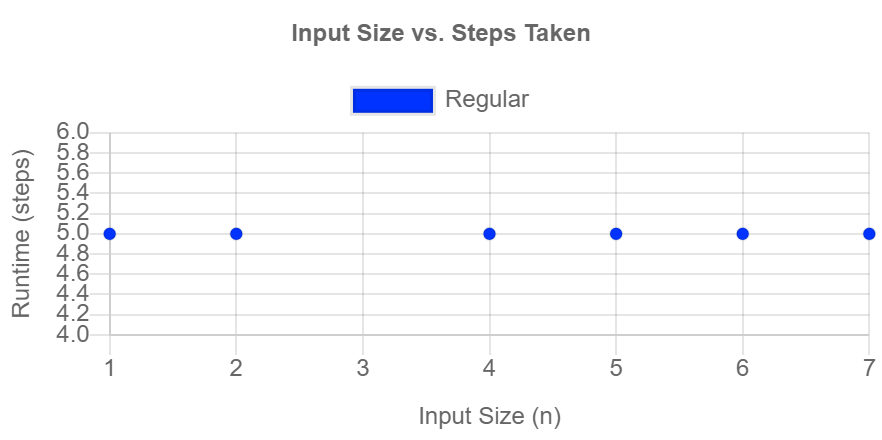
return values[0] + values[-1]

print(add\_ends(numbers))

#### **Cases**

| **Regular:** |
| --- |
| n = 1  numbers = [7] |
| n = 2  numbers = [1, 8] |
| n = 4  numbers = [1, 2, 3, 4] |
| n = 5  numbers = [1, 2, 3, 4, 5] |
| n = 6  numbers = [1, 2, 3, 4, 5, 6] |
| n = 7  numbers = [1, 2, 3, 4, 5, 6, 7] |

#### **Time Analysis Plot**

****

#### **Plotted Inputs**

| **Case** | **N** | **Steps** | **Output** |
| --- | --- | --- | --- |
| Regular | 1 | 5 | 14 |
| Regular | 2 | 5 | 9 |
| Regular | 4 | 5 | 5 |
| Regular | 5 | 5 | 6 |
| Regular | 6 | 5 | 7 |
| Regular | 7 | 5 | 8 |

## Question 2

URL: <https://acbart.github.io/runtime-case-builder/?preload=RCB_max_list.json>

A) Brief description of algorithm:

| Goes through a list of numbers comparing each number to a preset variable with an int. And if the number in the list is bigger than the variable then that number replaces the variable number and returns the biggest number. |
| --- |

B) Predicted relationship:

| Linear |
| --- |

C) Copy and paste your created Report below, including the graph you made.

| **Max List****Algorithm** n = ???  numbers = ???  def maximum(values: [int]) -> int:  max\_so\_far = values[0]  for value in values:  if value > max\_so\_far:  max\_so\_far = value  return max\_so\_far  print(maximum(numbers)) **Cases**  | **Regular:** | | --- | | n = 1  numbers = [7] | | n = 5  numbers = [1, 3, 5, 2, 4] | | n = 3  numbers = [1, 2, 3] | | n = 2  numbers = [1, 2] | | n = 7  numbers = [3, 4, 1, 6, 2, 9, 8] | | n = 6  numbers = [3, 4, 1, 6, 2, 9] |  **Time Analysis Plot**  **Plotted Inputs**  | **Case** | **N** | **Steps** | **Output** | | --- | --- | --- | --- | | Regular | 1 | 8 | 7 | | Regular | 5 | 14 | 5 | | Regular | 3 | 12 | 3 | | Regular | 2 | 10 | 2 | | Regular | 7 | 17 | 9 | | Regular | 6 | 16 | 9 | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

## Question 3

URL: <https://acbart.github.io/runtime-case-builder/?preload=RCB_find_in_list.json>

A) Brief description of algorithm:

| Goes through a list of integers in an attempt to find the target integer. The iteration continues even after the target is found. Then returns the target integer. |
| --- |

B) Predicted relationship:

| Linear |
| --- |

C) Copy and paste your created Report below, including the graph you made.

### **Find in List**

#### **Algorithm**

n = ???

numbers = ???

k = ???

def find(values: [int], target: int):

result = None

for value in values:

if value == target:

result = value

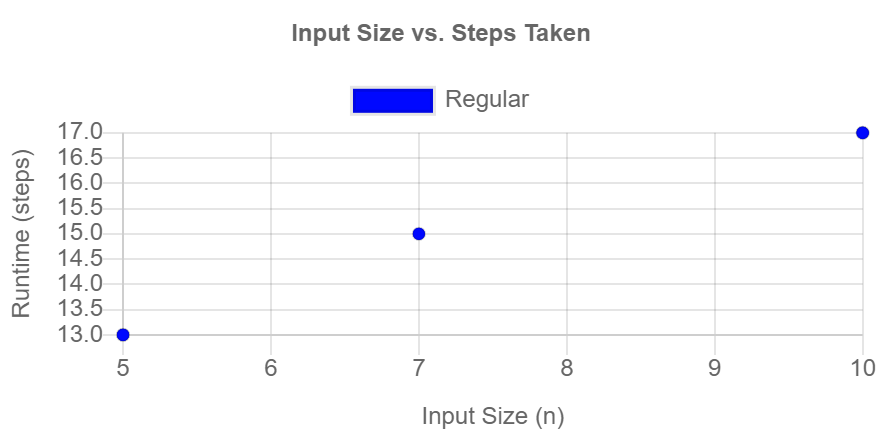
return result

print(find(numbers, k))

#### **Cases**

| **Regular:** |
| --- |
| n = 5  numbers = [0, 1, 2, 3, 4]  k = 2 |
| n = 5  numbers = [0, 1, 2, 3, 4]  k = 4 |
| n = 5  numbers = [0, 1, 2, 3, 4]  k = 0 |
| n = 7  numbers = [1, 2, 3, 4, 5, 6, 7]  k = 7 |
| n = 7  numbers = [1, 2, 3, 4, 5, 6, 7]  k = 6 |
| n = 10  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  k = 1 |
| n = 10  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  k = 7 |
| n = 10  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  k = 4 |

#### **Time Analysis Plot**

****

#### **Plotted Inputs**

| **Case** | **N** | **Steps** | **Output** |
| --- | --- | --- | --- |
| Regular | 5 | 13 | 2 |
| Regular | 5 | 13 | 4 |
| Regular | 5 | 13 | 0 |
| Regular | 7 | 15 | 7 |
| Regular | 7 | 15 | 6 |
| Regular | 10 | 17 | 1 |
| Regular | 10 | 17 | 7 |
| Regular | 10 | 17 | 4 |

## Question 4

URL: <https://acbart.github.io/runtime-case-builder/?preload=RCB_find_in_list__alternate_.json>

*Note that for this algorithm, you will need to create TWO different cases, because the algorithm performs differently depending on what values you choose for the list and K. You will need to have one set of inputs that maximizes the steps, and another set of input that minimizes the steps. Think critically about what the “worst” and “best” cases of inputs are!*

A) Brief description of algorithm:

| Iterates through a list of integers until it finds the target integer, then breaks the loops to return the target integer. |
| --- |

B) Predicted relationship:

| Random |
| --- |

C) Copy and paste your created Report below, including the graph you made.

| **Find in List (Alternate)** **Algorithm** n = ???  numbers = ???  k = ???  def find(values: [int], target: int):  for value in values:  if value == target:  return value  return None  print(find(numbers, k)) **Cases**  | **Best:** | | --- | | n = 5  numbers = [0, 1, 2, 3, 4]  k = 0 | | n = 5  numbers = [0, 1, 2, 3, 4]  k = 0 | | n = 7  numbers = [1, 2, 3, 4, 5, 6]  k = 1 | | n = 7  numbers = [1, 2, 3, 4, 5, 6]  k = 1 | | n = 10  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  k = 1 | | n = 10  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  k = 1 | | n = 10  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  k = 1 | | **Worst:** | | n = 5  numbers = [0, 1, 2, 3, 4]  k = 4 | | n = 5  numbers = [0, 1, 2, 3, 4]  k = 4 | | n = 7  numbers = [0, 1, 2, 3, 4, 5, 6]  k = 6 | | n = 7  numbers = [0, 1, 2, 3, 4, 5, 6]  k = 6 | | n = 10  numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  k = 9 | | n = 10  numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  k = 9 | | n = 10  numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  k = 9 |  **Time Analysis Plot**  **Plotted Inputs**  | **Case** | **N** | **Steps** | **Output** | | --- | --- | --- | --- | | Best | 5 | 7 | 0 | | Best | 5 | 11 | 4 | | Best | 7 | 7 | 1 | | Best | 7 | 12 | 6 | | Best | 10 | 7 | 1 | | Best | 10 | 15 | 9 | | Best | 10 | 11 | 5 | | Worst | 5 | 11 | None | | Worst | 5 | 11 | 4 | | Worst | 7 | 13 | 6 | | Worst | 7 | 13 | 6 | | Worst | 10 | 16 | 9 | | Worst | 10 | 16 | 9 | | Worst | 10 | 16 | 9 | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

## Question 5

URL: <https://acbart.github.io/runtime-case-builder/?preload=RCB_nested_loop.json>

This algorithm may be less intuitive for you, since we are iterating over lists created with range.

Make sure you choose numbers that are VERY different from each other! (Think of BIG numbers for your n).

A) Brief description of algorithm:

| This algorithm iterates from 1..N, and for each number along the way it iterates from 1 until that number. For each of those pairs, it multiplies them and then adds them to a running sum. |
| --- |

B) Predicted relationship:

| Quadratic |
| --- |

C) Copy and paste your created Report below, including the graph you made.

| **Nested Loop****Algorithm** n = ???  result = 0  for i in range(n):  for j in range(i):  result += i \* j  print(result) **Cases**  | **Regular:** | | --- | | n = 4 | | n = 8 | | n = 16 | | n = 12 | | n = 10 | | n = 9 |  **Time Analysis Plot**  **Plotted Inputs**  | **Case** | **N** | **Steps** | **Output** | | --- | --- | --- | --- | | Regular | 4 | 19 | 11 | | Regular | 8 | 49 | 322 | | Regular | 16 | 157 | 6580 | | Regular | 12 | 95 | 1925 | | Regular | 10 | 70 | 870 | | Regular | 9 | 59 | 546 | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

# 4) Reflect and Review

Discuss among yourselves: what did you learn from this activity? What was surprising or interesting? If you didn’t learn anything, what do you think we were trying to teach you? How could this activity be improved?

| This activity taught me about how codes can be shown in graphs and that depending on the code you can see if it is a linear or quadratic, and others. Good to know |
| --- |
| We learned more about how runtime is important and the different types of cases for runtime lengths. |
| We learned that there are tools out there to help us evaluate our code and how to use them. |

# Final Submission

When your team is happy with your answers for all the questions, download this file as a Word Document (docx) and upload the file to the appropriate assignment on Canvas.

Only one member of your team needs to submit.