

Algorithms and Data Structures for Data Science

File I/O and Efficiency

CS 277

January 31, 2024

Brad Solomon



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Exam 1

Exams will be proctored by the CBTF: <https://cbtf.engr.illinois.edu/>

(That link will have a link to **Prairietest**, where you can sign up for exam 0)

Reservations open on February 1st

You must take the exam sometime between 2/13 and 2/15!

See website for expected content:

<https://courses.grainger.illinois.edu/cs277/sp2024/exams/>

Learning Objectives

Review file I/O in Python

Introduce the concept of asymptotic efficiency

Introduce list implementation strategies

Compare list implementations using big O

Python List



There are many implementations of lists in Python. Here are three*:

```
1 myList = [1, 2, 3, 4, 5]
2
3 print(myList)
4
5 print(len(myList))
6
7 print(myList[2])
8
9
```

```
1 myTuple = (1, 2, 3, 4,
2 5)
3
4 print(myTuple)
5
6 print(len(myTuple))
7
8 print(myTuple[2])
9
```

```
1 import numpy as np
2 myNP =
3 np.array([1,2,3,4,5])
4
5 print(myNP)
6
7 print(len(myNP))
8
9 print(myNP[2])
```

Why are there so many different implementations?

Lets find out together in our first class mini-project!

P1) Generate random datasets

P2) Code various analysis functions on datasets using lists

P3) Measure efficiency of **P2** using varying size datasets from **P1**

Mini-Project 0: Random Data Generation

Learning Objectives:

Practice string and list manipulations in the context of **file I/O**

Introduce seeded **random data generation**

Introduce how to measure **runtime efficiency** in Python

Investigate the efficiency of different Python implementations of lists

Programming Toolbox: File I/O

```
1 readableFile = open('inputFile.txt', 'r')
2
3
4
5
6 writableFile = open('outputFile.txt', 'w')
7
8
9
10
11 carefulWriteFile = open('outputFile.txt', 'x')
12
13
14
15
16 appendableFile = open('outputFile.txt', 'a')
17
18
19
20
21
22
23
```

Programming Toolbox: File I/O

Which approach do you prefer?

```
1 # Approach 1
2
3 readableFile = open('inputFile.txt', 'r')
4
5
6 fileData = readableFile.read()
7
8
9 readableFile.close()
10
11
12 # Approach 2
13
14 with open('inputFile.txt', 'r') as myFile:
15     fileData = myFile.read()
16
17
18
```


Programming Toolbox: File I/O

```
1 with open('data/temp1.txt', 'x') as myFile:
2     for i in range(10):
3         myFile.write(str(i))
4     myFile.write("\n")
5     myFile.write("Line 2")
6
7
8
9 myFile = open('data/temp2.txt', 'w')
10 for i in range(5):
11     myFile.write(str(i) + "\n")
12 myFile.close()
13
14
15
16 with open('data/temp2.txt', 'a') as myFile:
17     myFile.write("Hello World!\n")
18
19 with open('data/temp2.txt', 'a') as myFile:
20     myFile.write("Hello World!\n")
21
22
23
```

Programming Toolbox: File I/O

```
1 with open('data/temp1.txt', 'x') as myFile:
2     for i in range(10):
3         myFile.write(str(i))
4     myFile.write("\n")
5     myFile.write("Line 2")
6
7
8
9 myFile = open('data/temp2.txt', 'w')
10 for i in range(5):
11     myFile.write(str(i) + "\n")
12 myFile.close()
13
14
15
16 with open('data/temp2.txt', 'a') as myFile:
17     myFile.write("Hello World!\n")
18
19 with open('data/temp2.txt', 'a') as myFile:
20     myFile.write("Hello World!\n")
21
22
23
```

1	0123456789
2	Line 2

temp1.txt

1	0
2	1
3	2
4	3
5	4
6	

temp2.txt

Programming Toolbox: File I/O

```
1 with open('data/temp1.txt', 'x') as myFile:
2     for i in range(10):
3         myFile.write(str(i))
4     myFile.write("\n")
5     myFile.write("Line 2")
6
7
8
9 myFile = open('data/temp2.txt', 'w')
10 for i in range(5):
11     myFile.write(str(i) + "\n")
12 myFile.close()
13
14
15
16 with open('data/temp2.txt', 'a') as myFile:
17     myFile.write("Hello World!\n")
18
19 with open('data/temp2.txt', 'a') as myFile:
20     myFile.write("Hello World!\n")
21
22
23
```

```
1 0123456789
2 Line 2
```

temp1.txt

```
1 0
2 1
3 2
4 3
5 4
6 Hello World!
7
```

temp2.txt

Programming Toolbox: File I/O

```
1 with open('data/temp1.txt', 'x') as myFile:
2     for i in range(10):
3         myFile.write(str(i))
4     myFile.write("\n")
5     myFile.write("Line 2")
6
7
8
9 myFile = open('data/temp2.txt', 'w')
10 for i in range(5):
11     myFile.write(str(i) + "\n")
12 myFile.close()
13
14
15
16 with open('data/temp2.txt', 'a') as myFile:
17     myFile.write("Hello World!\n")
18
19 with open('data/temp2.txt', 'a') as myFile:
20     myFile.write("Hello World!\n")
21
22
23
```

```
1 0123456789
2 Line 2
```

temp1.txt

```
1 0
2 1
3 2
4 3
5 4
6 Hello World!
7 Hello World!
8
```

temp2.txt

Programming Toolbox: File I/O

```
1 with open('data/temp1.txt') as myFile:
2     inList = myFile.readlines()
3 print(inList)
4
5
6
7
8
9 myFile = open('data/temp2.txt')
10 for i in range(10):
11     print("Line Content: {}".format(myFile.readline()))
12 myFile.close()
13
14
15
16
17
18 with open('data/temp1.txt') as myFile:
19     print(myFile.read())
20
21
22
23
```

1	0123456789
2	Line 2
3	
4	
5	
6	

temp1.txt

1	0
2	1
3	2
4	3
5	4
6	

temp2.txt

Programming Toolbox: File I/O

1	0123456789
2	Line 2
3	
4	
5	
6	

temp1.txt

1	0
2	1
3	2
4	3
5	4
6	

temp2.txt

```
1 with open('data/temp1.txt') as myFile:
2     inList = myFile.readlines()
3 print(inList)
4
5
6
7
8
9 myFile = open('data/temp2.txt')
10 for i in range(10):
11     print("Line Content: {}".format(myFile.readline()))
12 myFile.close()
13
14
15
16
17
18 with open('data/temp1.txt') as myFile:
19     print(myFile.read())
20
21
22
23
```

`['0123456789\n', 'Line 2']`

Line Content: 0

Line Content: 1

Line Content: 2

Line Content: 3

Line Content: 4

Line Content:

Line Content:

Line Content:

Line Content:

Line Content:

0123456789

Line 2

Programming Toolbox: File I/O

`str.strip()` will remove whitespace from the string.

1	0
2	1
3	2
4	3
5	4
6	

temp2.txt

```
1 myFile = open('data/temp2.txt')
2 for i in range(6):
3     print("Line Content: {}".format(myFile.readline().strip()))
4 myFile.close()
5
6 with open('data/temp2.txt') as myFile:
7     for line in myFile:
8         print(line.strip())
```

```
Line Content: 0
Line Content: 1
Line Content: 2
Line Content: 3
Line Content: 4
Line Content:
0
1
2
3
4
```

Programming Toolbox: File I/O

`str.strip()` will remove whitespace from the string.

```
1 tmp = "1, 2, 3"
2 tmp2 = "1,2,3"
3
4 x = tmp.split(",")
5 y = tmp2.split(",")
6 for i in range(len(x)):
7     if x[i]!=y[i]:
8         print("No match!", x[i], y[i])
9     else:
10        print("Match!")
11
12
```


Programming Toolbox: File I/O



Four ways to open a file: read, write, carefulWrite, append

Three ways to read a file: all lines, line-by-line, text as string

Writing to file: String formatting is key, file writes are literal copies

Python File I/O

Whats wrong here:

```
1  
2 with open('dataFile2.txt','w') as myFile:  
3     data = myFile.readlines()  
4  
5 for line in data:  
6     print(line)  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18
```

Programming Practice: File I/O

Given an input filename, write a new file that is the same file with reversed lines.

Mini-Project 0: Random Data Generation

Learning Objectives:

Practice string and list manipulations in the context of **file I/O**

Introduce seeded **random data generation**

Introduce how to measure **runtime efficiency** in Python

Investigate the efficiency of different Python implementations of lists

What do we care about when we write code?

Time Efficiency (What is the execution speed of our code?)

Memory Efficiency (How much memory does our code use?)

Coding Efficiency (How much time does it take us to write? Others to read?)

Coding Efficiency: Hard to measure

Student self-report total assessment time (surveys)

Prairielearn records total time per assignment

Exams test your ability to complete assessments in capped time

Python Toolbox: Tracemalloc (Memory Efficiency)

```
tracemalloc.start()
```

```
tracemalloc.get_traced_memory()
```

```
tracemalloc.reset_peak()
```

Python Toolbox: Tracemalloc (Memory Efficiency)

```
1 import tracemalloc
2
3 tracemalloc.start()
4
5
6
7 #23978913 24000645
8 #import pandas as pd
9
10
11
12 # 605 10974
13 current, peak = tracemalloc.get_traced_memory()
14
15 print(current, peak)
16
17
18
19
20
21
22
23
```


How do we measure time efficiency?

```
1 for i in range(n):  
2     time.sleep(30)  
3     doStuff()  
4
```

```
1 for i in range(n):  
2     for j in range(n):  
3         doStuff()  
4
```

How do we measure time efficiency?

```
1 for i in range(n):  
2     for j in range(n/2):  
3         doStuff()  
4
```

```
1 for i in range(n):  
2     for j in range(n):  
3         doStuff()  
4
```



Idea 1: Measure time using physical time

Python Toolbox: Timeit

```
Timeit.repeat(setup, stmt, repeat, number))
```

setup

stmt

repeat

number

Python Toolbox: Timeit

```
1 def doStuff_A(n):
2     total = 0
3     for i in range(n):
4         for j in range(n):
5             total+=j
6
7 def doStuff_B(n):
8     total = 0
9     for i in range(n):
10        for j in range(int(n/2)):
11            total+=j
12
13 SETUP_CODE = '''
14 from __main__ import doStuff_A, doStuff_B'''
15
16 TEST_CODE= '''
17 doStuff_A(1000)
18 '''
19
20 myTime = timeit.repeat(setup=SETUP_CODE,
21 stmt=TEST_CODE, repeat=5, number=10)
22 print(myTime)
23
```

Python Toolbox: Timeit

```
Timeit.default_timer()
```

Returns the current time as fractional seconds.

Includes time elapsed during sleep or system-wide processes

```
Timeit.perf_counter_ns()
```

Returns the current time as nanoseconds.














Includes time elapsed during sleep or system-wide processes

Python Toolbox: Timeit



```
1 import random
2 import time
3
4 def timeWaste():
5     stall_time = random.randint(1, 5)
6     print("Wasting {} time".format(stall_time))
7     time.sleep(stall_time)
8
9 start = timeit.default_timer()
10 for i in range(5):
11     timeWaste()
12 end = timeit.default_timer()
13
14 print("I wasted {} time total.".format(end-start))
15
16
17
18
19
20
21
22
23
```

Problem 1: Hardware complexity

	Time x1 billion	Like
L1 cache reference	0.5 seconds	Heartbeat 
Branch mispredict	5 seconds	Yawn 
L2 cache reference	7 seconds	Long yawn   
Mutex lock/unlock	25 seconds	Make coffee 
Main memory reference	100 seconds	Brush teeth
Compress 1K bytes	50 minutes	TV show 
Send 2K bytes over 1 Gbps network	5.5 hours	(Brief) Night's sleep 
SSD random read	1.7 days	Weekend
Read 1 MB sequentially from memory	2.9 days	Long weekend
Read 1 MB sequentially from SSD	11.6 days	2 weeks for delivery 
Disk seek	16.5 weeks	Semester
Read 1 MB sequentially from disk	7.8 months	Human gestation 
Above two together	1 year	 
Send packet CA->Netherlands->CA	4.8 years	Ph.D. 

(Care of <https://gist.github.com/hellerbarde/2843375>)

Problem 2: Algorithm Complexity

P : word

T : There would have been a time for such a word



Problem 2: Algorithm Complexity

P : word

T : There would have been a time for such a word
word word word word word word word word word **word**
word word word word word word word word
word word word word word word word word
word word word word word word word word
word word word word word word word word

Problem 2: Algorithm Complexity

P: aaa

T: bbbbbbb

aaa

aaa

aaa

aaa

aaa

P: bbb

T: bbbbbbb

bbb

bbb

bbb

bbb

bbb

Problem 3: Scaling Performance Measures



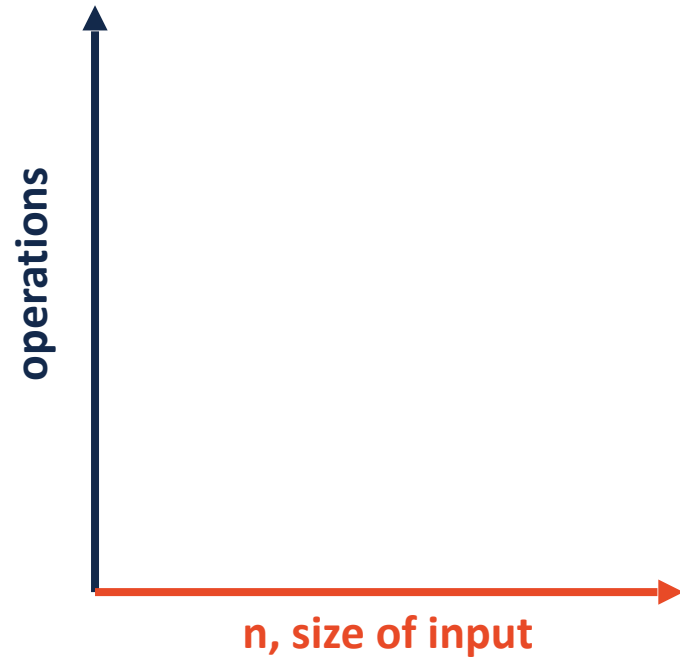
The problem with measuring time...



- 1) The difference between our **best case** and **worst case** can be significant
- 2) Measuring actual time can be messy — computers handling multiple processes at the same time. Hardware differences exist between machines
- 3) We are most interested in the performance on large datasets, which are significantly more difficult to measure due to (1) and (2)

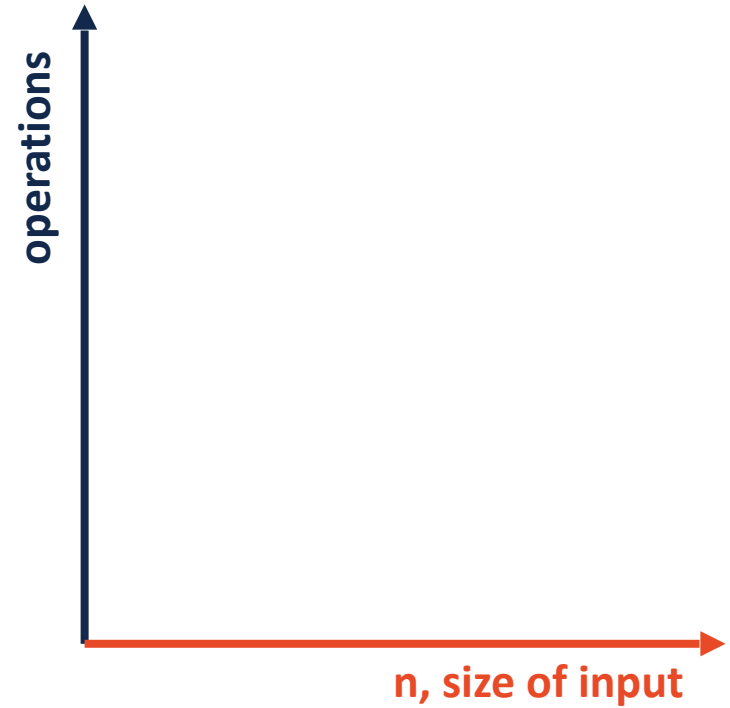
Big-O notation

$f(n)$ is $O(g(n))$ iff $\exists c, k$ such that $f(n) \leq cg(n) \forall n > k$



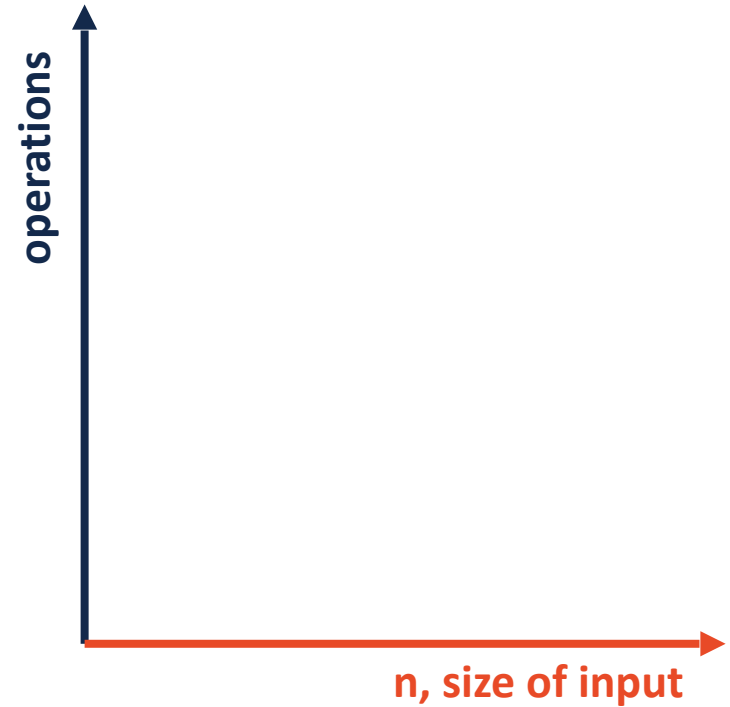
Constant Time, $O(1)$

```
1 def constant(n):  
2     ops = 0  
3     for i in range(10):  
4         ops+=1  
5     return ops  
6  
7 print(constant(5))  
8 print(constant(9001))
```



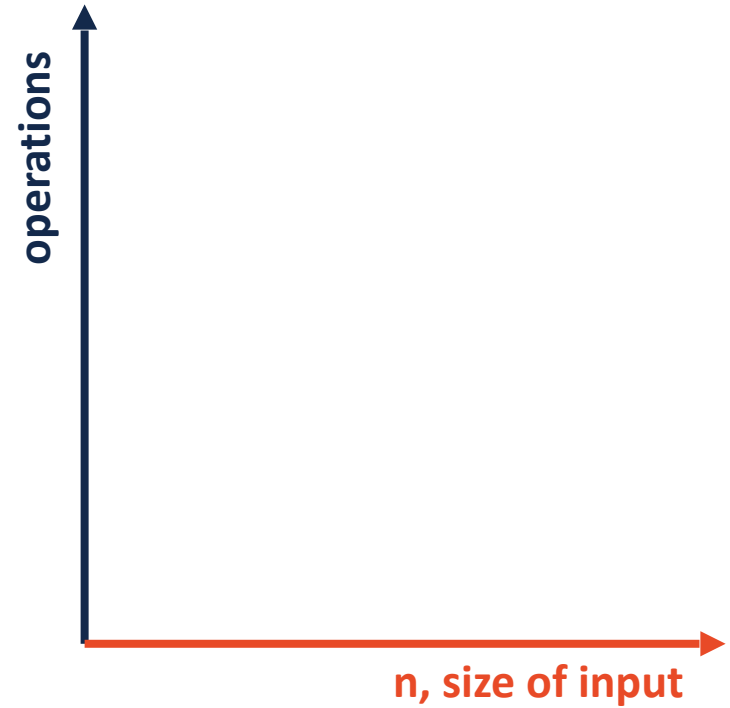
Logarithmic Time, $O(\log n)$

```
1 import math
2 def logarithmic(n):
3     ops = 0
4     for i in range(int(math.log2(n))):
5         ops+=1
6     return ops
7
8 print(logarithmic(5))
9 print(logarithmic(9001))
```



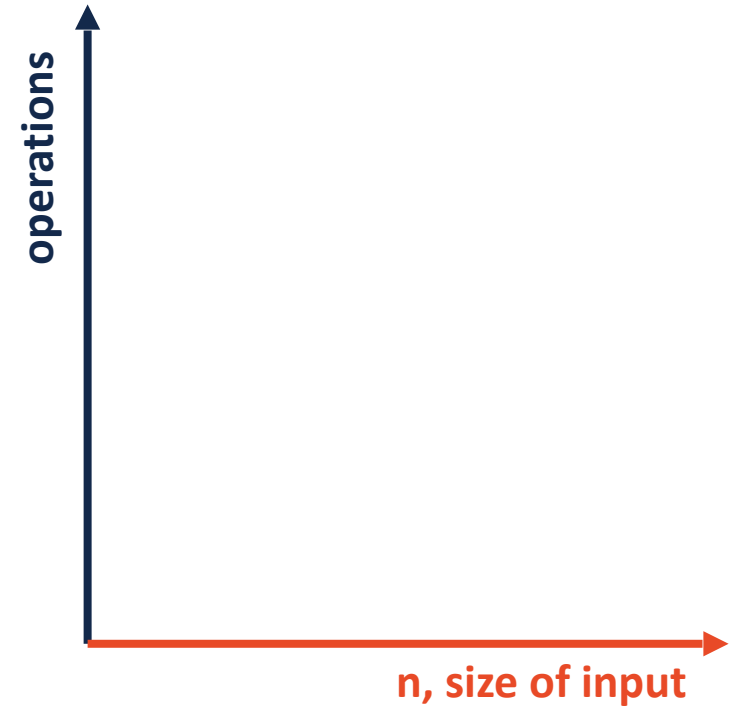
Linear Time, $O(n)$

```
1 def linear(n):  
2     ops = 0  
3     for i in range(n):  
4         ops+=1  
5     return ops  
6  
7 print(linear(5))  
8 print(linear(9001))
```

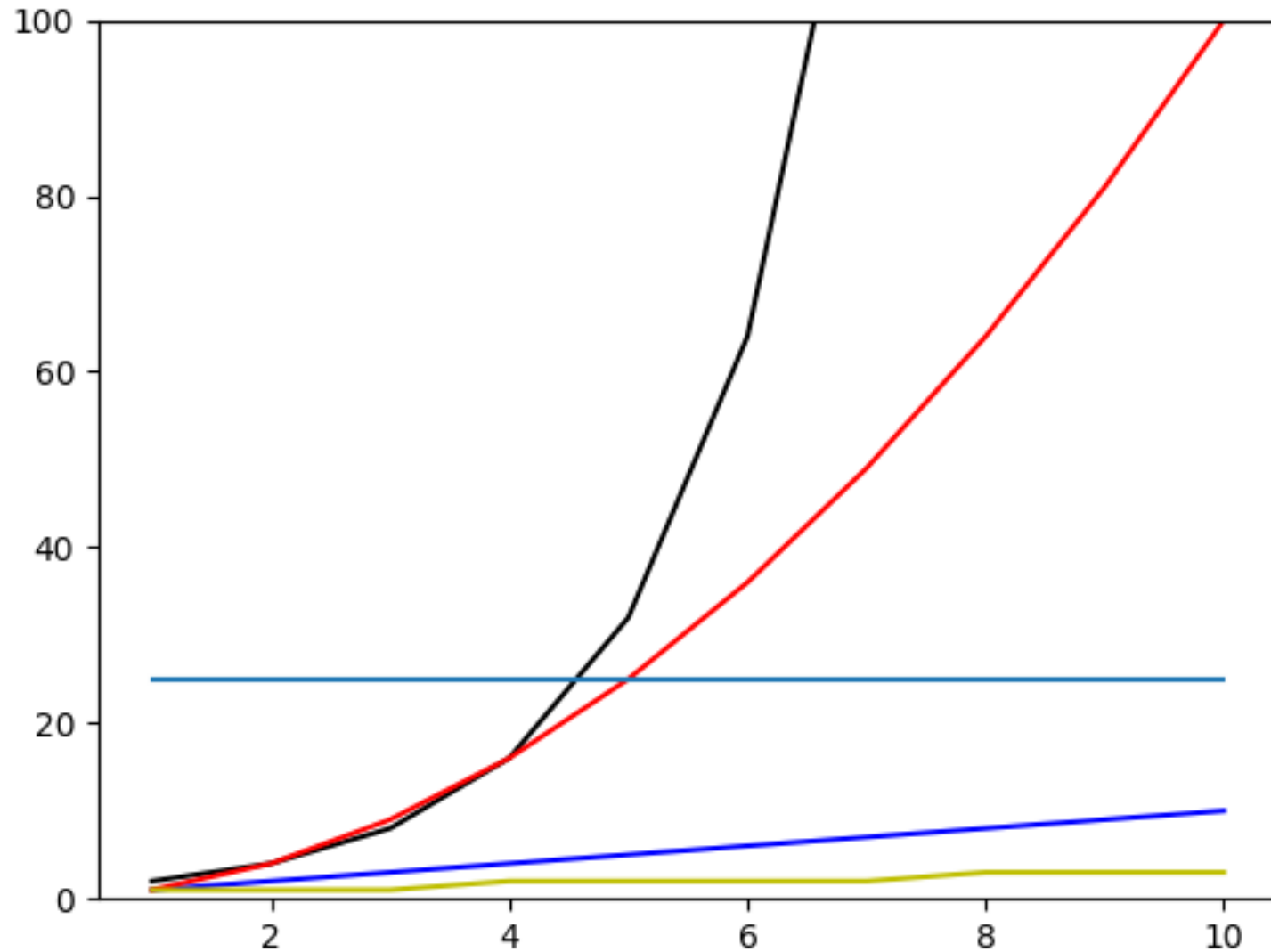


Quadratic Time, $O(n^2)$

```
1 # Quadratic Time
2 def quadratic(n):
3     ops = 0
4     for i in range(n):
5         for j in range(n):
6             ops+=1
7     return ops
8
9 print(quadratic(5))
10 print(quadratic(9001))
```

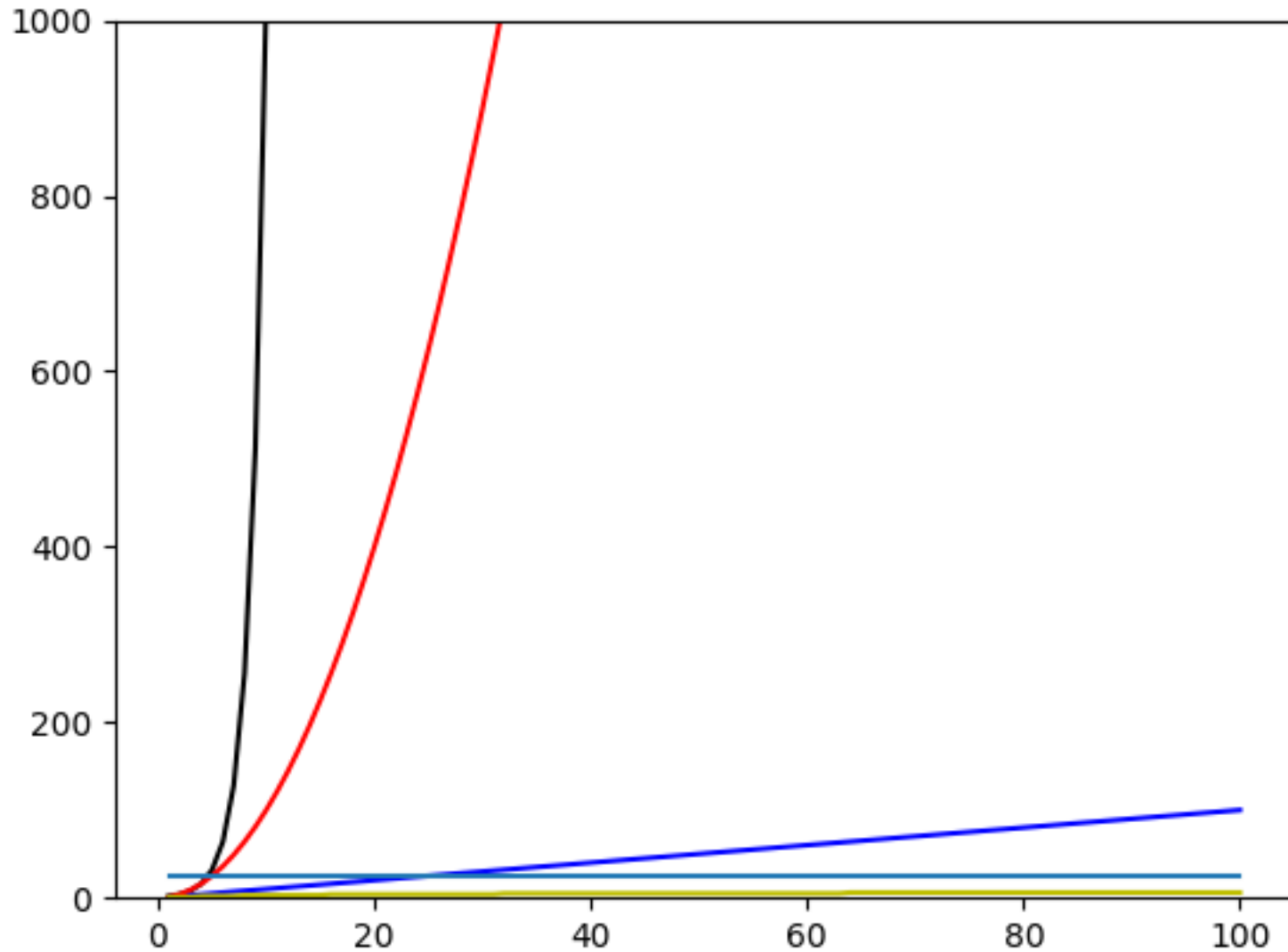





Big-O Complexity Classes



- $O(2^n)$
- $O(n^2)$
- $O(n)$
- $O(\log n)$
- $O(1)$

Big-O Complexity Classes



-  $O(2^n)$
-  $O(n^2)$
-  $O(n)$
-  $O(\log n)$
-  $O(1)$

Identifying the Big O of an algorithm

- 1) Label the key factors that drive algorithm performance
- 2) Write out the worst-case performance for each step
- 3) Identify (or reduce to) the largest terms for each factor

Big O Practice: Pattern Matching

P: word

T: There would have been a time for such a word
word word word word word word word word word **word**
word word word word word word word word word
word word word word word word word word word
word word word word word word word word word
word word word word word word word word word

Big O Practice: Simplifying efficiency

What is the big O for the following functions?

$$a(n) = n^4 + 50n + 10$$

$$b(n) = 500n \log n + 50n + \log(n)$$

$$c(n) = n^3 + 3n! + 12$$

$$d(n) = n^2 + n \log n$$

Big O Practice: Reading code

```
1 def doStuff(inList1, inList2):
2
3     c1 = 0
4     for i in inList1:
5         c1+=1
6
7
8
9     c2 = 0
10    for v1 in inList1:
11        for v2 in inList2:
12            c2+=1
13
14
15
16    return c1, c2
17
18
19
20
21
22
23
```


Big O Practice: Reading code

```
1 def doStuff2(inList):
2     ops = 0
3     size = len(inList)
4     while size > 0:
5         size = int(size / 2)
6         ops+=1
7     return ops
8
9 def doStuff3(inList1, inList2):
10    ops = 0
11    for i in inList1:
12        ops+= doStuff2(inList2)
13    return ops
14
15
16
17
18
19
20
21
22
23
```

Big O Practice: Reading code



```
1 def convert_1D_to_2D(inList, rowSize):
2     listLen = len(inList)
3     numRows = math.ceil(listLen/rowSize)
4
5     outList = []
6     count = 0
7
8     ops = 0
9     for i in range(numRows):
10        tempList = []
11
12        for j in range(rowSize):
13
14            if count >= listLen:
15                tempList.append(-1)
16            else:
17                tempList.append(inList[count])
18
19            ops+=1
20            count+=1
21
22        outList.append(tempList)
23
24    print(ops)
25    return outList
```

Next time: List Implementations and Efficiency

We've seen:

Why lists are an important fundamental data structure

The necessary functions for a list (the Abstract Data Type)

How to create and use lists using built-in methods

How to measure code performance

Understanding the actual implementation will:

Allow us to practice programming and Big O

Allow us to justify design decisions involving lists