

Algorithms and Data Structures for Data Science

Lists Implementation

CS 277

February 5, 2024

Brad Solomon



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

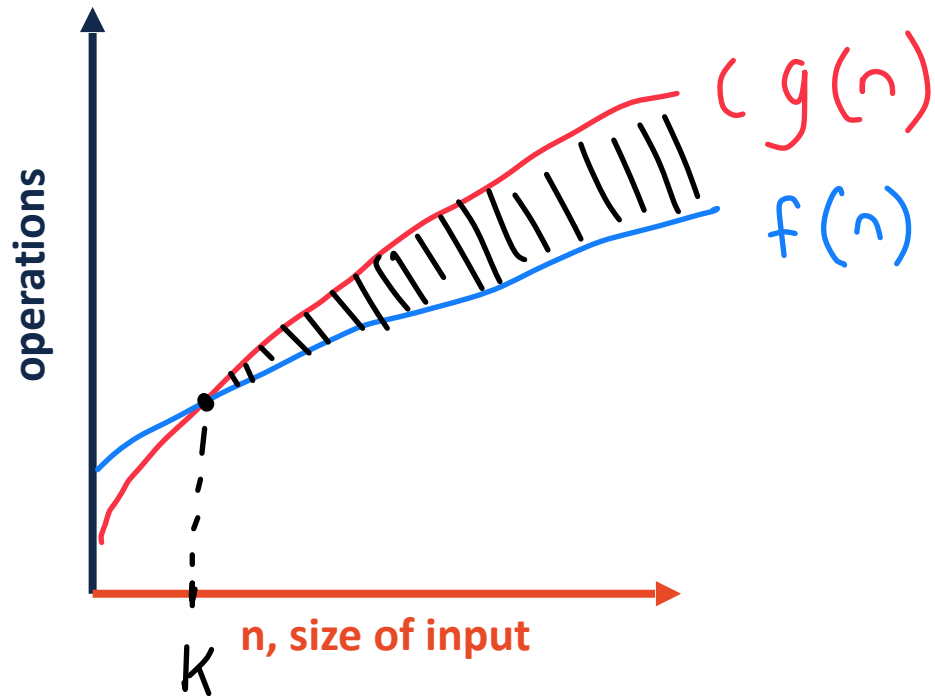
Learning Objectives

Re-introduce and review Big O

Discuss list implementation strategies in the context of Big O

Big-O notation

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

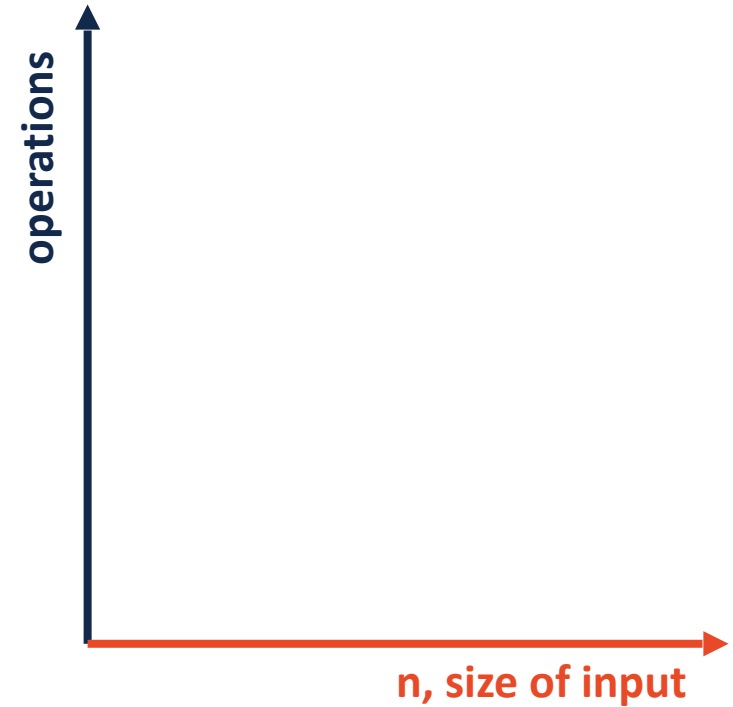


1) $cg(n)$ is an upper bound on $f(n)$

2) This is true for all input values larger than some arbitrary k

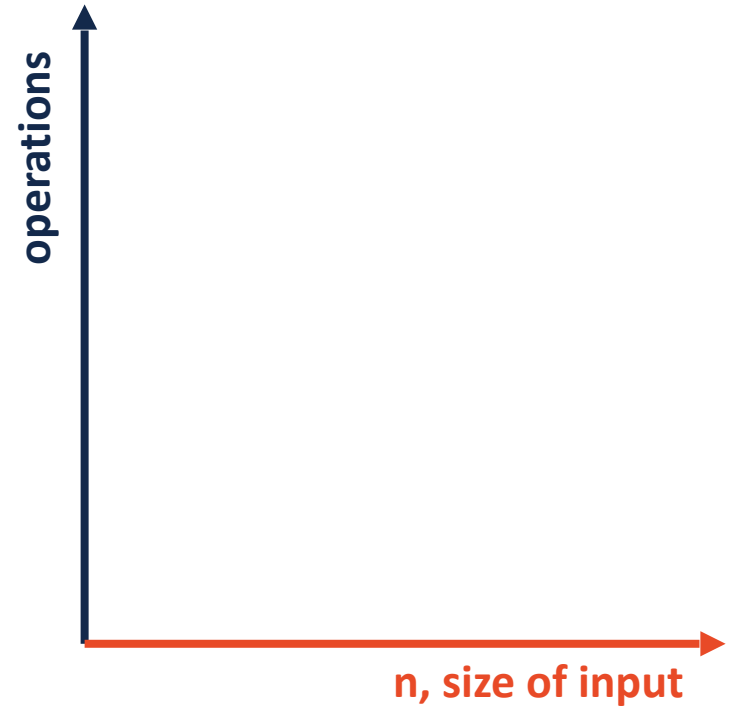
Constant Time, $O(1)$

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



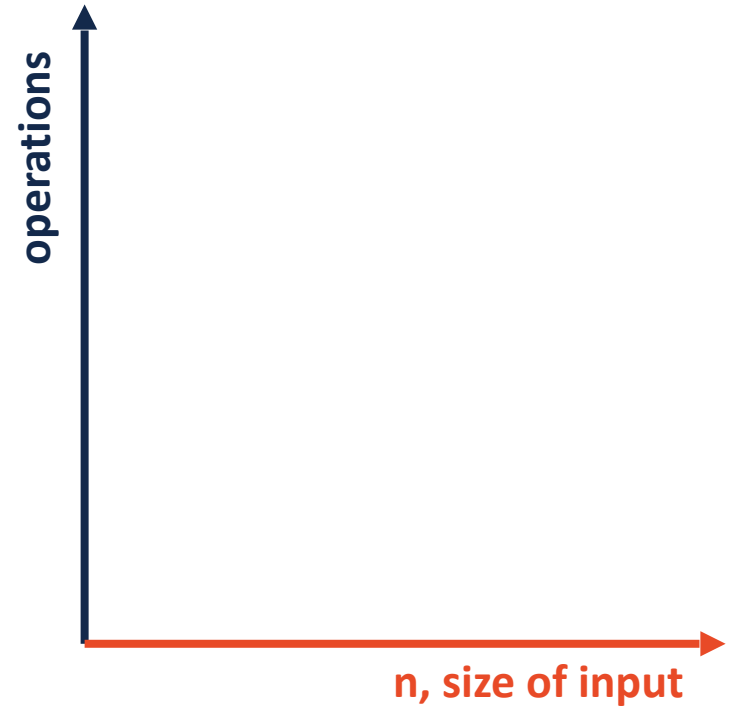
Logarithmic Time, $O(\log n)$

```
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))
```



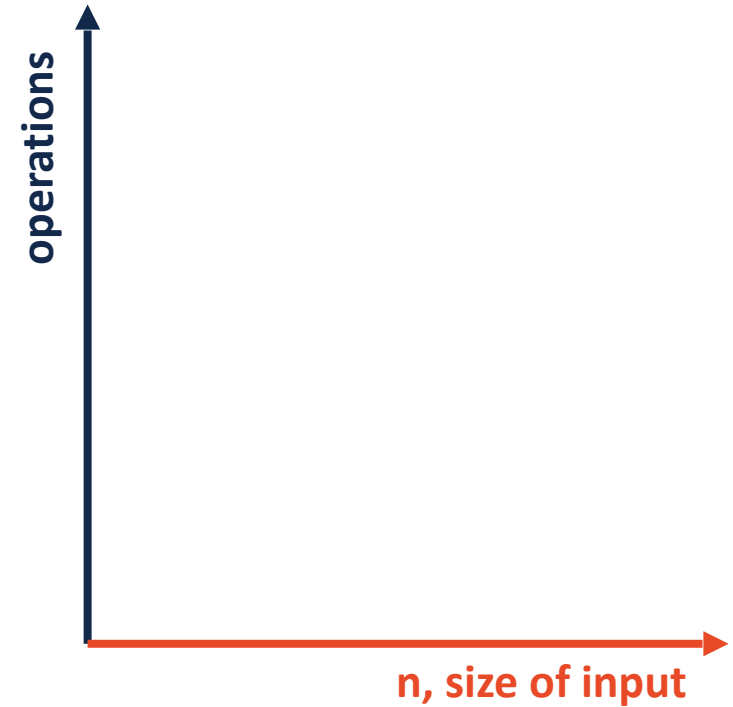
Linear Time, $O(n)$

```
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))
```

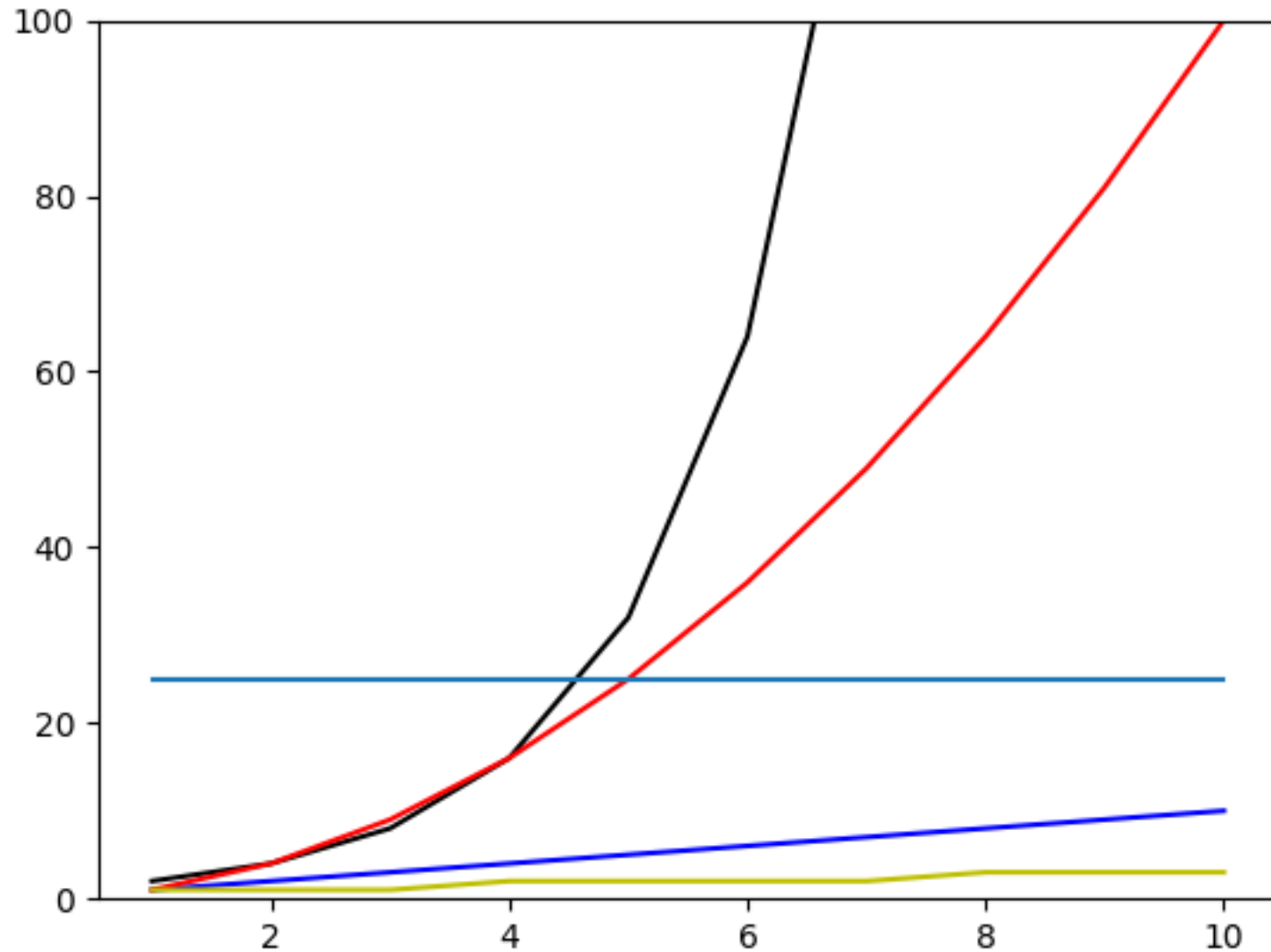


Quadratic Time, $O(n^2)$

```
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))
```

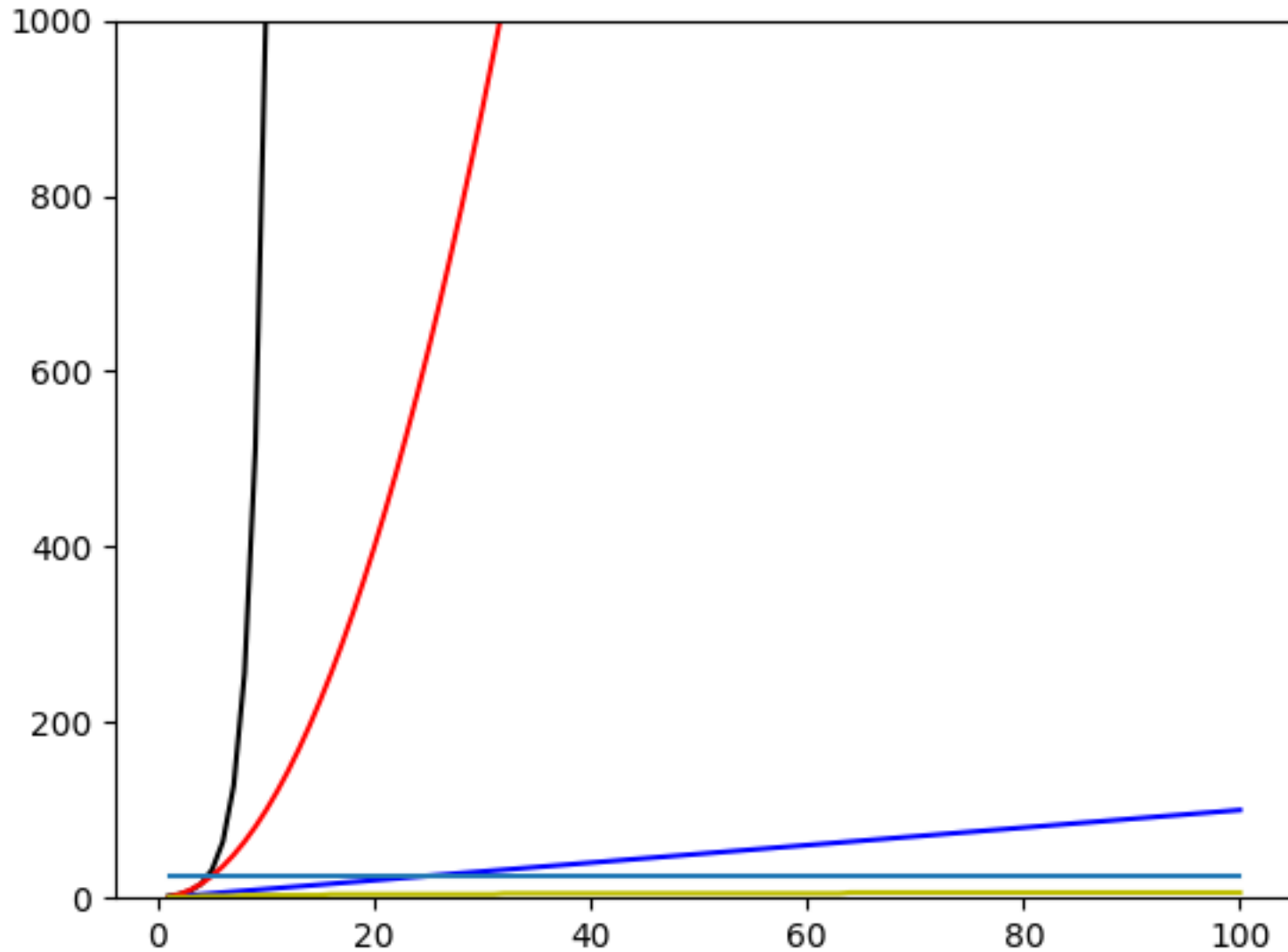





Big-O Complexity Classes



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

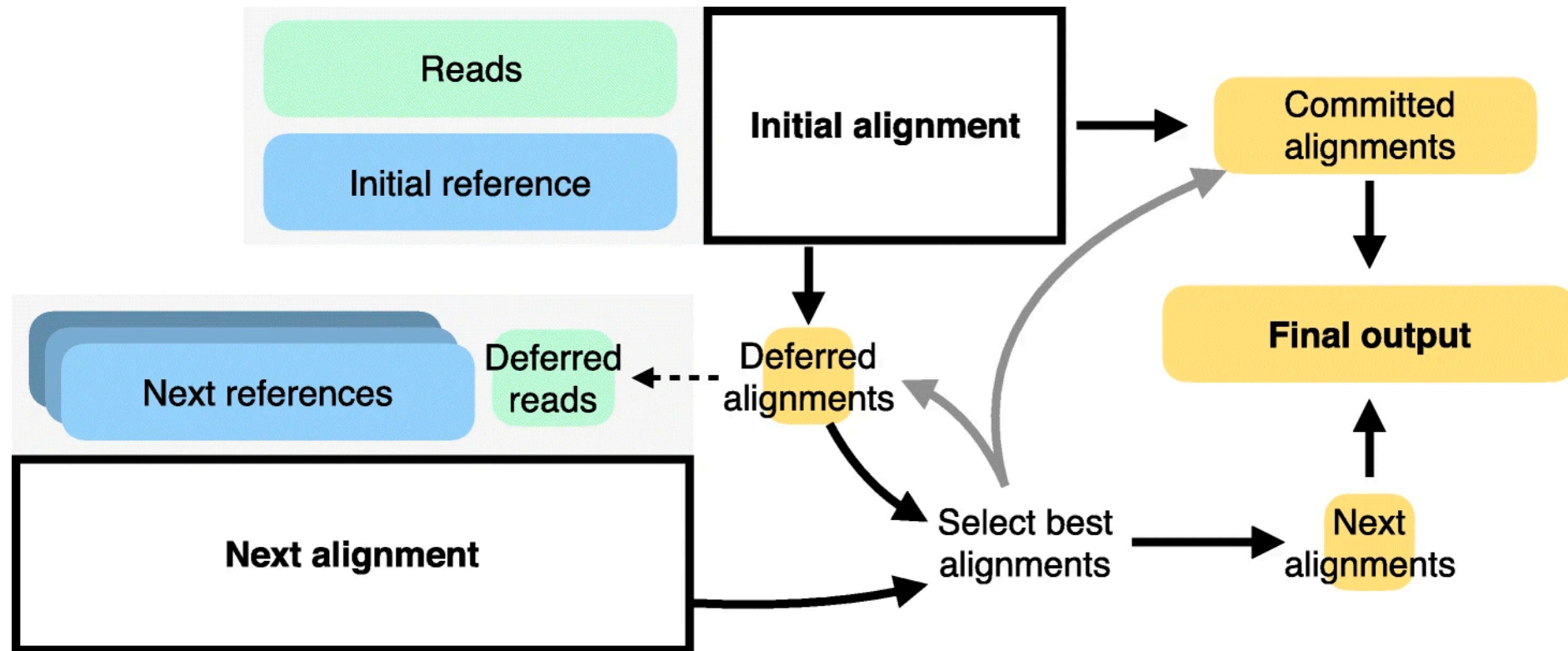
Big-O Complexity Classes



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

Big O: Communicating Efficiency

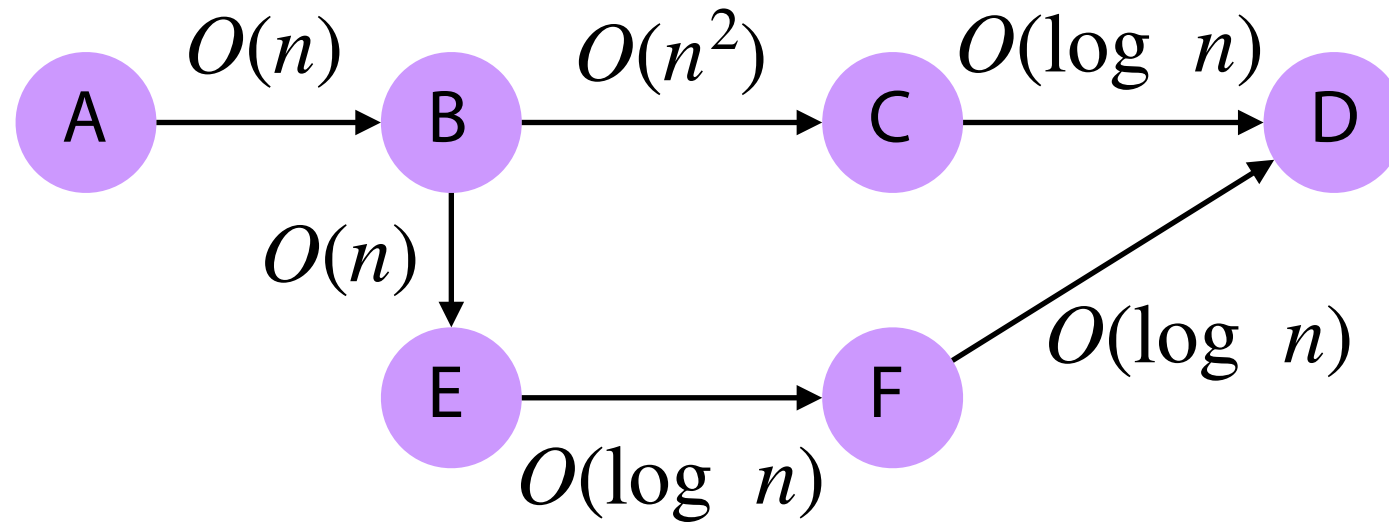
I want to find the most efficient data processing pipeline



Reference flow: reducing reference bias using multiple population genomes. Chen et al 2021

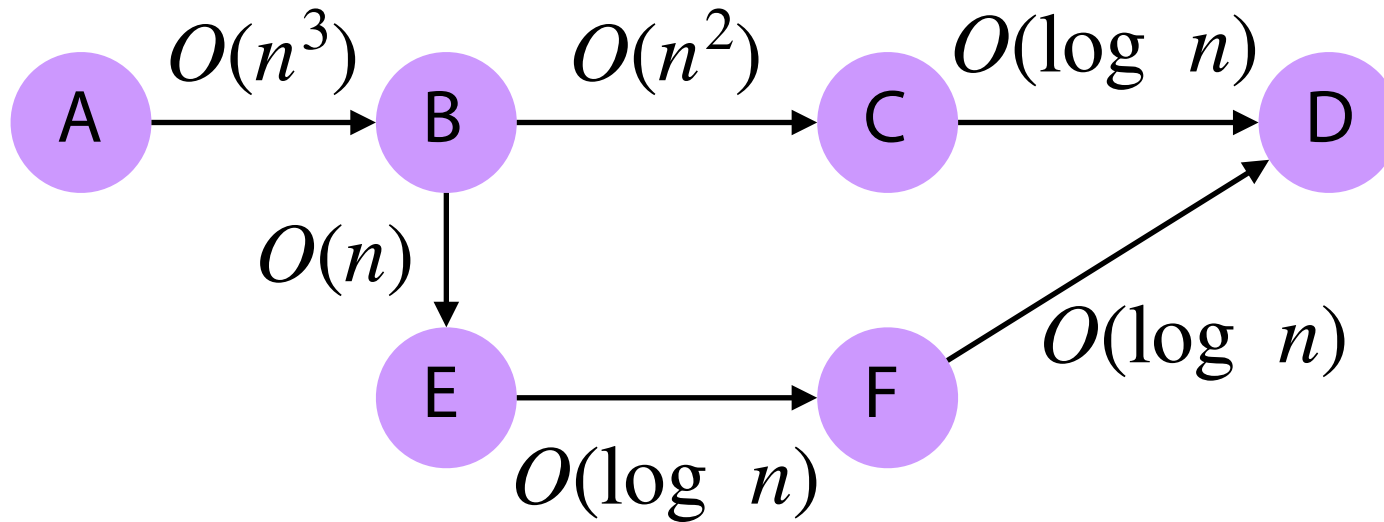
Big O: Communicating Efficiency

I want to find the most efficient data processing pipeline



Big O: Communicating Efficiency

I want to find the most efficient data processing pipeline



Big O: Communicating Efficiency

I want to design the best approach to finding the minimum value.

Given n items, what is my Big O to find the min?

Given n items which are sorted, what is my Big O to find the min?

What is the runtime to sort an unsorted dataset?

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: Identifying key factors (variables)

Imagine I have a PNG and want to count the number of blue pixels

What are the variables I need to keep track of?

If I look at each pixel individually, what is my Big O?

Big O Practice: Simplifying to highest order

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

Big O of the List ADT

A minimally functional list must have the following functions:

Constructor: `__init__()`

Insert: `append(x)` `insert(i, x)`

Delete: `remove(x)` `pop()`

Index `__getitem__()` `[]` `index(x)`

Size() `len(list)`

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])
```

(Theoretical) List Implementations

1. Array List

2. Linked List

Creating an array list

```
1 l1 = [None]*5
2
3 l2 = [0, "1", 2.0, "W", (1, 2,)]
4
5 np1 = np.zeros(5)
6
7 np2 = np.empty(5)
```

Conceptual Idea 1:



Conceptual Idea 2:



Python lists are stored as PyObjects

```
1 l1 = [None]*5
2
3 l2 = [0, "1", 2.0, "W", (1, 2,)]
4
5 m1 = np.zeros(5)
6
7 m2 = np.empty(5)
8
9
10
11
12
13
14
```



Python List Memory Allocation

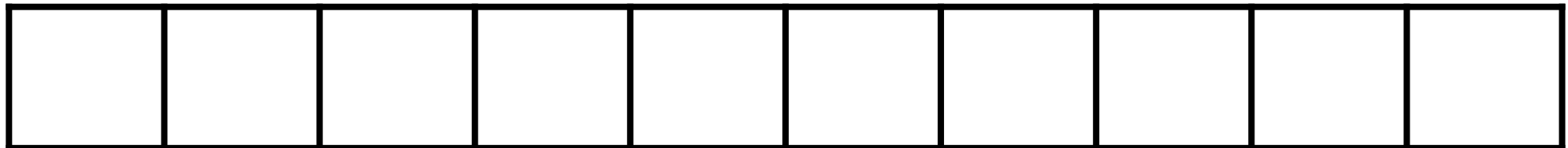


```
1 l1 = [1, 2, 3]
2 print(sys.getsizeof(l1))
3
4 l2 = [(1, 2, 3,), (4, 5, 6), (7, 8, 9)]
5 print(sys.getsizeof(l2))
6
7 bigString=""
8 for i in range(10**4):
9     bigString+="A"
10
11 print(sys.getsizeof(bigString))
12
13 l3 = [bigString]
14
15 print(sys.getsizeof(l3))
16
17
```



Array __getitem__()

1	12[3]
2	
3	mp1[3]



Array 'Add'

Python command to add to back of list:



Python command to add to front of list:



In-Class Brainstorm

We don't want to have to remake a new array every time we 'add'.

What sorts of things can we do?

Resize Strategy: x2 elements every resize

We have to find a good tradeoff between overallocation and efficiency!





Resize Strategy: x2 elements every resize

Python List Size Implementation

```
1 memory_size = {}
2
3 for length in range(50):
4     lst = []
5     for length_loop in range(length):
6         lst.append(length_loop)
7         memory_size[length] = sys.getsizeof(lst)
8
9 print(memory_size)
10
```

```
{0: 56, 1: 88, 2: 88, 3: 88, 4: 88, 5: 120, 6: 120, 7: 120, 8: 120, 9: 184, 10: 184, 11:
184, 12: 184, 13: 184, 14: 184, 15: 184, 16: 184, 17: 248, 18: 248, 19: 248, 20: 248, 21:
248, 22: 248, 23: 248, 24: 248, 25: 312, 26: 312, 27: 312, 28: 312, 29: 312, 30: 312, 31:
312, 32: 312, 33: 376, 34: 376, 35: 376, 36: 376, 37: 376, 38: 376, 39: 376, 40: 376, 41:
472, 42: 472, 43: 472, 44: 472, 45: 472, 46: 472, 47: 472, 48: 472, 49: 472}
```

Python List Size Implementation

```
1 memory_size = {}
2
3 for length in range(50):
4     lst = []
5     for length_loop in range(length):
6         lst.append(length_loop)
7         memory_size[length] = sys.getsizeof(lst)
8
9 print(memory_size)
10
```

```
{0: 56, 1: 88, 2: 88, 3: 88, 4: 88, 5: 120, 6: 120, 7: 120, 8: 120, 9: 184, 10: 184, 11:
184, 12: 184, 13: 184, 14: 184, 15: 184, 16: 184, 17: 248, 18: 248, 19: 248, 20: 248, 21:
248, 22: 248, 23: 248, 24: 248, 25: 312, 26: 312, 27: 312, 28: 312, 29: 312, 30: 312, 31:
312, 32: 312, 33: 376, 34: 376, 35: 376, 36: 376, 37: 376, 38: 376, 39: 376, 40: 376, 41:
472, 42: 472, 43: 472, 44: 472, 45: 472, 46: 472, 47: 472, 48: 472, 49: 472}
```

Numpy List Implementation

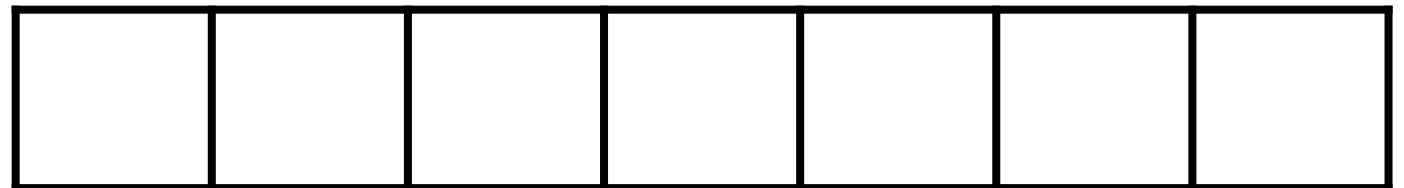
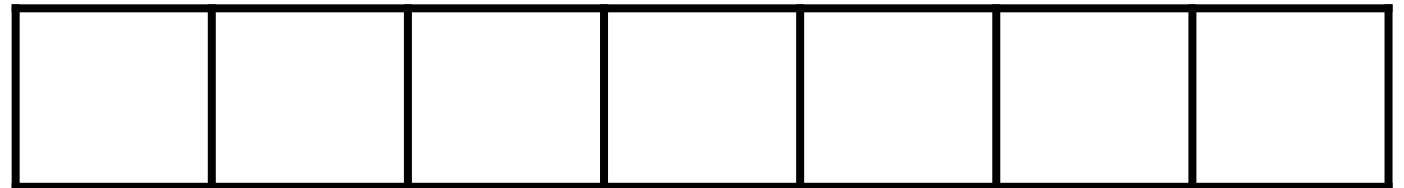


```
1 nms = {}
2 for length in range(50):
3     npa = np.array([])
4     for length_loop in range(length):
5         npa = np.append(npa, length)
6     nms[length] = sys.getsizeof(npa)
7
8 print(nms)
9
10
```

```
{0: 112, 1: 120, 2: 128, 3: 136, 4: 144, 5: 152, 6: 160, 7: 168, 8: 176, 9: 184, 10: 192,
11: 200, 12: 208, 13: 216, 14: 224, 15: 232, 16: 240, 17: 248, 18: 256, 19: 264, 20: 272,
21: 280, 22: 288, 23: 296, 24: 304, 25: 312, 26: 320, 27: 328, 28: 336, 29: 344, 30: 352,
31: 360, 32: 368, 33: 376, 34: 384, 35: 392, 36: 400, 37: 408, 38: 416, 39: 424, 40: 432,
41: 440, 42: 448, 43: 456, 44: 464, 45: 472, 46: 480, 47: 488, 48: 496, 49: 504}
```


Array remove()

```
1 l = [1, 2, 3, 4, 5, 6, 7]
2 l.pop()
3
4 l.remove(1)
5
6 l.pop(3)
7
8
```



Experiment on your own: Python remove

Practice your Python programming and confirm your hypothesis

How does Python's list size change as you remove objects? Numpy's?

Array Implementation



	Array
Look up given an input position	
Search given an input value	
Insert/Remove at front	
Insert/Remove at arbitrary location	