

Algorithms and Data Structures for Data Science

Lists 3

CS 277

September 20, 2021

Brad Solomon



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

reflections on mp_racing

Submission for mp_racing is end of semester

Key concepts: File IO / Conditionals and Loops / Logic

Avg: 51 / 60 (Bonus raised average significantly)

reflections on lab_debug

No student feedback given

Seems to be successful — both in time and score

Avg: 14.67 / 15

reflections on lab_proficiency



Limiting factor is time

Converting 1D to 2D most missed question

Count peaks most answered question

We will continue to move forward on course content

Learning Objectives

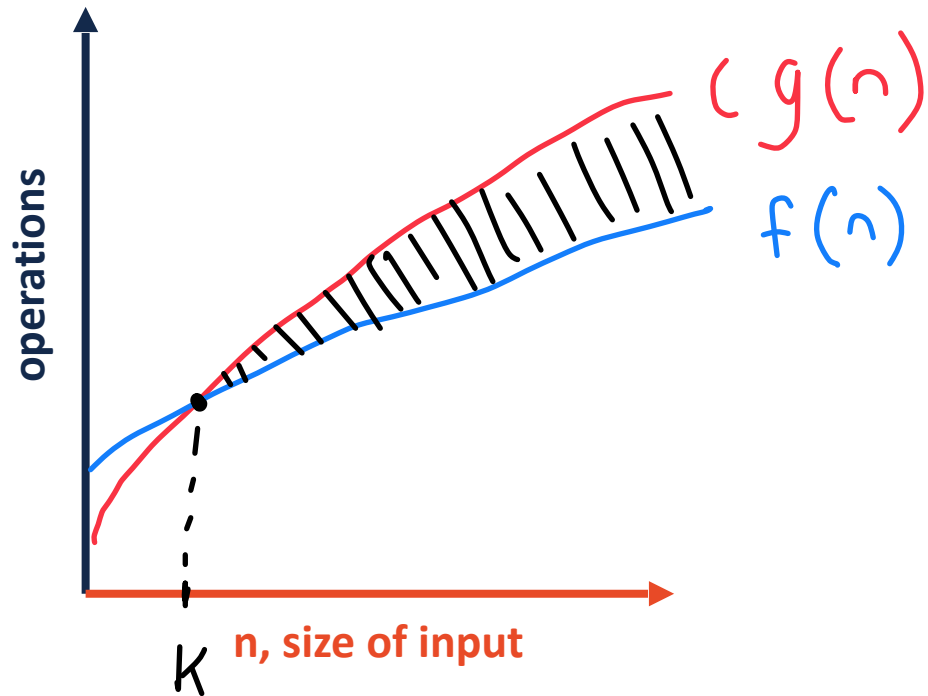
Review asymptotic notation

Discuss implementation tradeoffs for lists

Introduce alternative ordered 1D data structures

Big-O notation

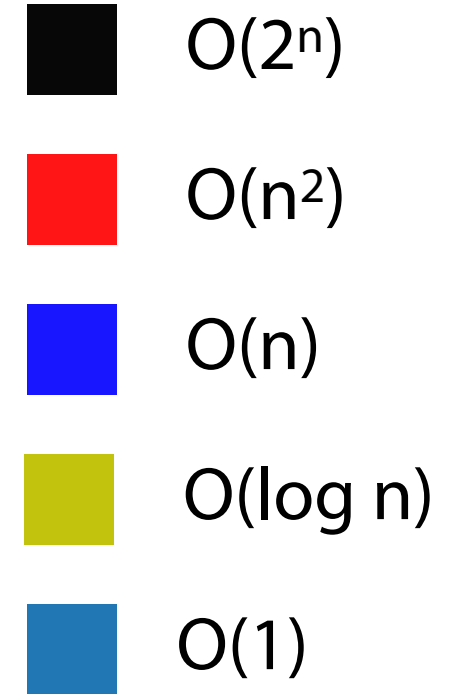
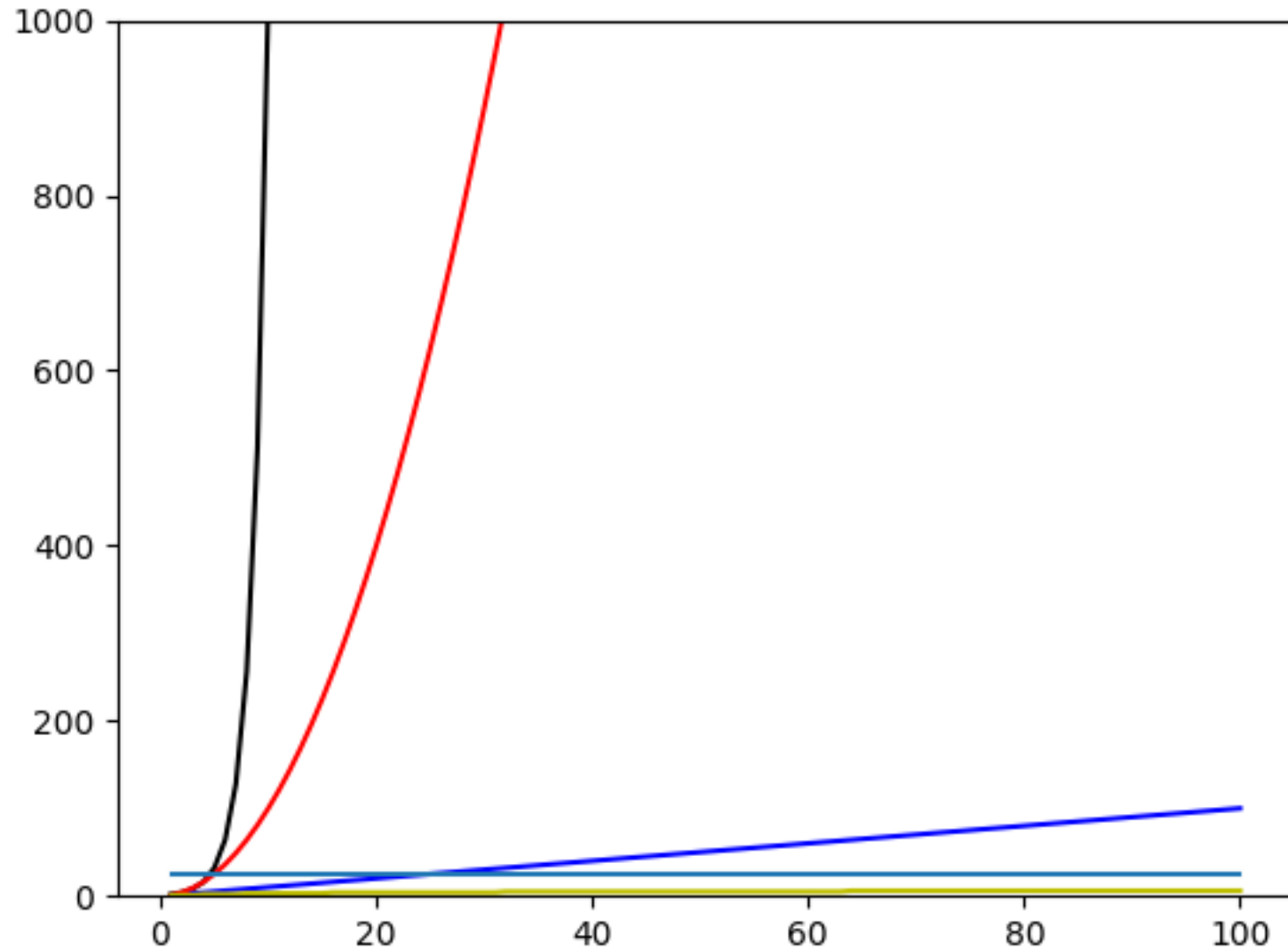
$f(n)$ is $O(g(n))$ iff $\exists c, k$ such that $f(n) \leq cg(n) \forall n > k$



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

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

Big-O Complexity Classes



count_words_in_line

```
1 def count_words_in_line(inFile):
2     countList = []
3
4     with open(inFile) as myFile:
5
6         for line in myFile:
7             count = 0
8             line = line.strip()
9             spline = line.split(" ")
10
11            for word in spline:
12                word = word.strip()
13                if word:
14                    count+=1
15
16            countList.append(count)
17
18    return countList
19
20
21
22
23
```


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

count_peaks

```
1 def countPeaks(inList):
2     count = 0
3
4     for i in range(len(inList)):
5         pv = nv = 0
6
7         if i == 0:
8             pv = -1
9         if i == len(inList)-1:
10            nv = -1
11
12        if pv != -1:
13            pv = inList[i-1]
14        if nv != -1:
15            nv = inList[i+1]
16
17        cv = inList[i]
18        if cv > pv and cv > nv:
19            count+=1
20
21    return count
22
23
```

convert_1D_to_2D



```
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     for i in range(numRows):
9         tempList = []
10
11         for j in range(rowSize):
12
13             if count >= listLen:
14                 tempList.append(-1)
15             else:
16                 tempList.append(inList[count])
17
18             count+=1
19
20         outList.append(tempList)
21
22     return outList
23
```

List Functions

Index Look up: Given index, return value at index

Add: Given value, insert value at front of list

Insert: Given value and index, insert value at index position in list

Remove: Given value, remove first instance in list

Pop: Given index, remove and return value at index position

List Implementations

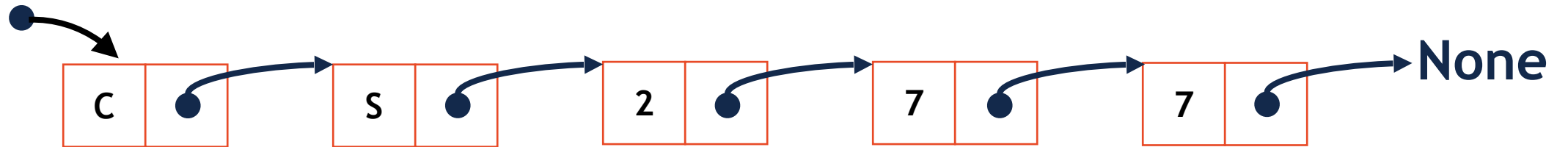
1. Linked List

2. Arrays

Linked List

look_up()

Head

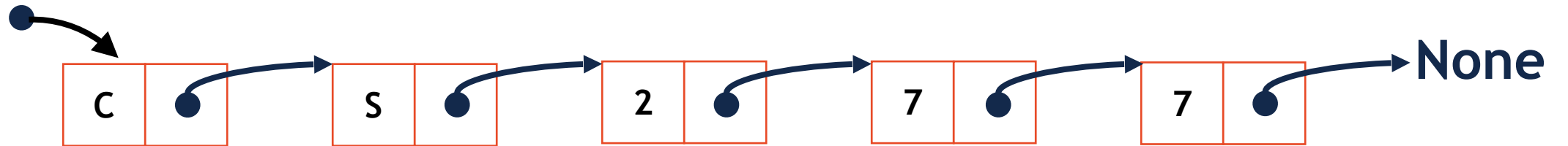


```
1 class linkedList:
2
3     def look_up(self, pos):
4         curr = self.head
5
6         i = 0
7         while(curr and i < pos):
8             curr = curr.next
9             i+=1
10
11         if i == pos:
12             return curr
13
```

Linked List

insert()

Head

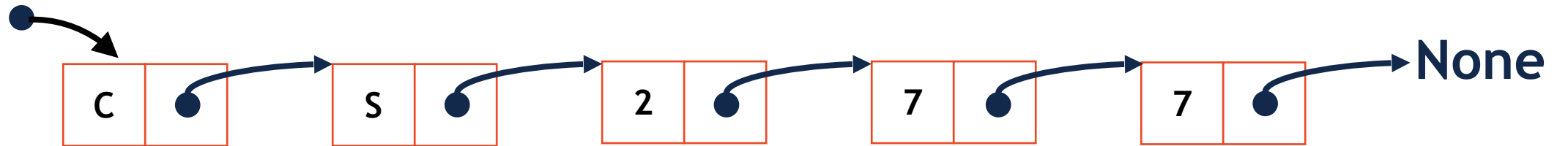


```
1 class linkedList:
2
3     def insert(self, data, pos=0):
4         if (pos == 0):
5             self.add(data)
6         else:
7             prev = self.index(pos-1)
8
9             temp = prev.next
10
11             prev.next = Node(data, temp)
12
13
```

Linked List

remove()

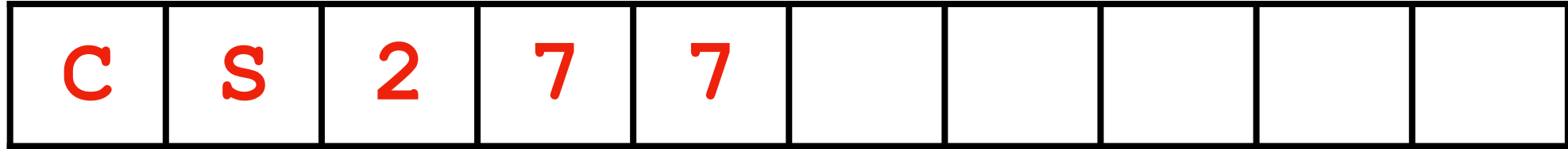
Head



```
1 def remove(self, data):
2     prev = None
3     curr = self.head
4
5     while(curr):
6         if curr.data == data:
7             if prev == None:
8                 self.head = self.head.next
9                 break
10            else:
11                prev.next = curr.next
12                break
13        else:
14            prev = curr
15            curr = curr.next
16
```


Array

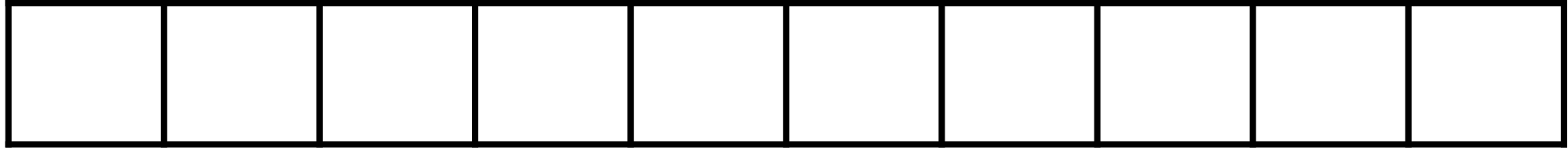
look_up()



```
1 def look_up(self, pos):  
2     return self.array[pos]  
3
```

Array

`insert()`



```
1 def insert(self, data, pos=0):  
2     self.array.insert(pos, data)  
3
```

Resize Strategy: +2 elements every resize





Resize Strategy: +2 elements every resize

Resize Strategy: x2 elements every resize





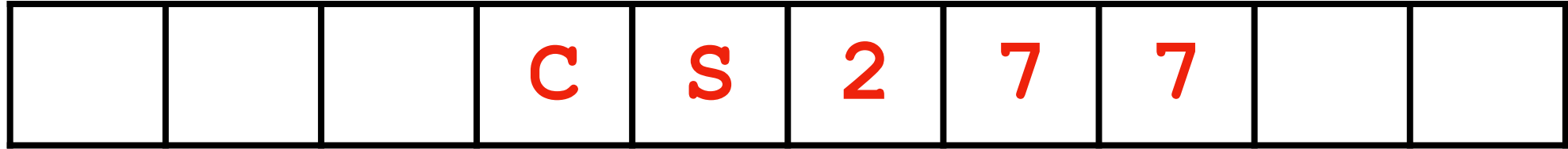
Resize Strategy: x2 elements every resize

Observing built-in resize

```
1 import sys
2
3 l = []
4 n = 4
5
6 size = sys.getsizeof(l)
7 print("empty list:", size)
8
9 for i in range(n):
10     l.append(i)
11
12 nsize = sys.getsizeof(l)
13 print("n={} list size: {}".format(n, nsize))
14
15 c = (nsize-size)//8
16 print("capacity:", c)
17 print("length:", len(l))
18
19
20
21
22
23
```

Array

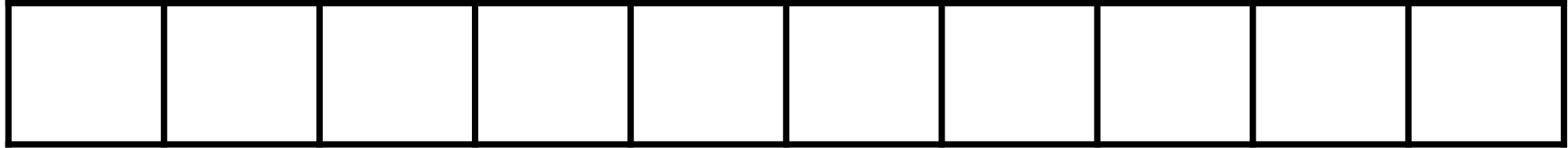
`insert()`



```
1 def insert(self, data, pos=0):  
2     self.array.insert(pos, data)  
3
```


Array

`remove()`



```
1 def remove(self, data):  
2     self.array.remove(data)  
3
```

Array Implementation



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



Organizing a to-do list



Queue ADT

Order:

Functions:

Runtime:



Stack ADT

Order:

Functions:

Runtime: