Review Week: Session #1

Overview

- Runtime analysis overview
- In class walkthrough
- In class exercise

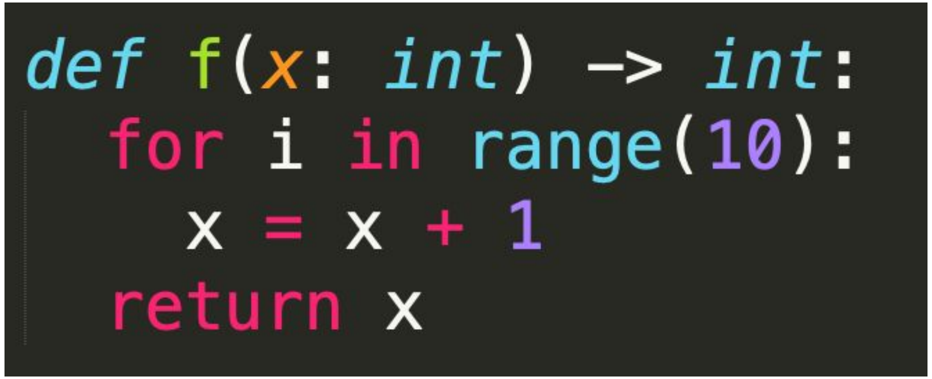
Big O Analysis

Big O Analysis

- Very important part of the interview
  - Helps you quickly evaluate different approaches
  - Will reflect poorly if you can't do this
- Both space and time complexity are important!
  - Space complexity is a parallel concept
- Should be able to talk about time/ space trade offs for different approaches
- Be clear what the variables signify in your analysis
  - What is N?

Constant: O(1)

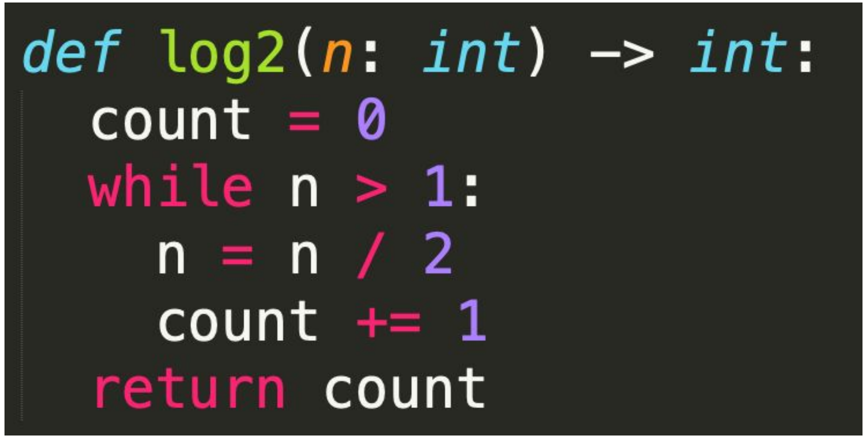
Runtime is does not scale with the input.



```python
def f(x: int) -> int:
  for i in range(10):
    x = x + 1
  return x
```

Logarithmic: O(log n)

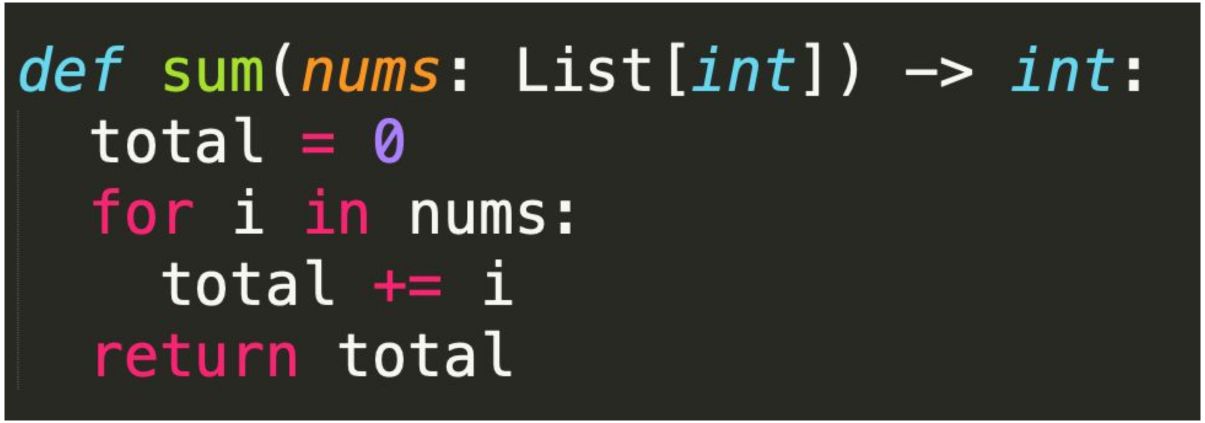
Applies to solutions in which the number of elements in the problem space gets divided by some factor each time.



```python
def log2(n: int) -> int:
  count = 0
  while n > 1:
    n = n / 2
    count += 1
  return count
```

Linear: O(n)

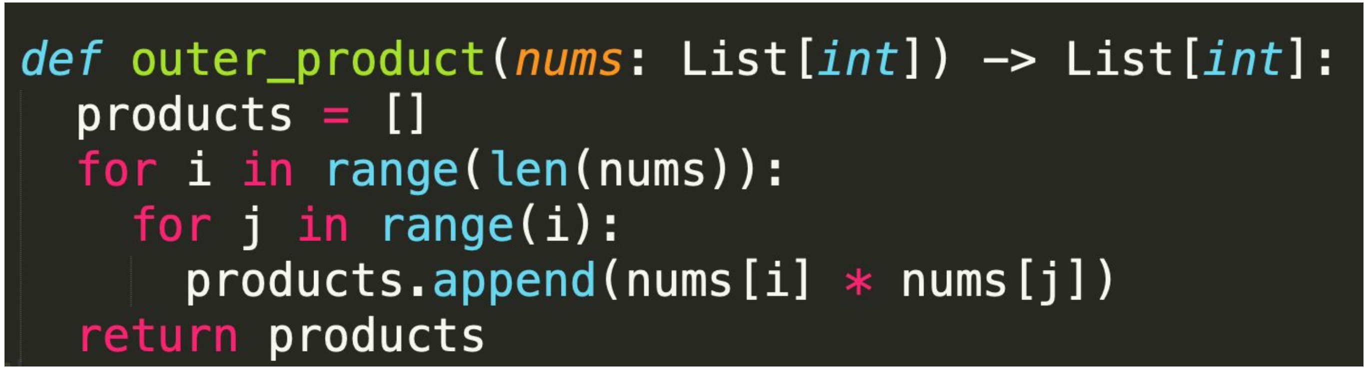
Grows proportionally with the size of the input.



```
def sum(nums: List[int]) -> int:
  total = 0
  for i in nums:
    total += i
  return total
```

Quadratic: $O(n^2)$

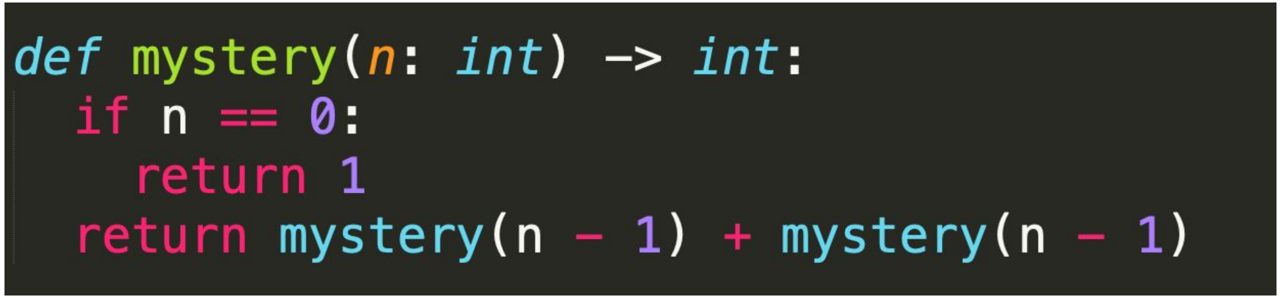
- Common for double for loops
- Note this series: $1 + 2 + 3 + \ldots + n$ is also $O(n^2)$



```python
def outer_product(nums: List[int]) -> List[int]:
  products = []
  for i in range(len(nums)):
    for j in range(i):
      products.append(nums[i] * nums[j])
  return products
```

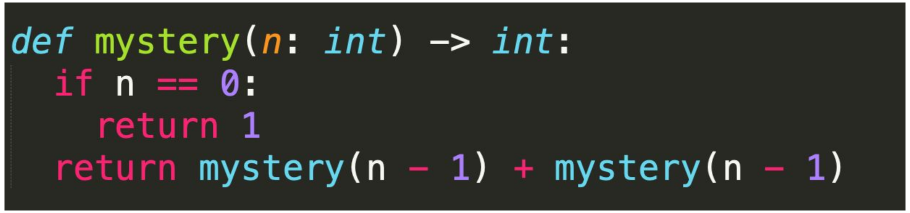
Exponential: $O(2^n)$

Can happen with recursive solutions

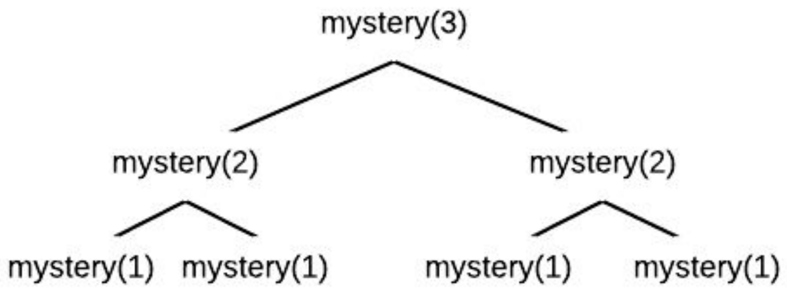


```python
def mystery(n: int) -> int:
  if n == 0:
    return 1
  return mystery(n - 1) + mystery(n - 1)
```

Exponential: $O(2^n)$



```python
def mystery(n: int) -> int:
  if n == 0:
    return 1
  return mystery(n - 1) + mystery(n - 1)
```



Big O Analysis Rules

1. Different steps get added together
    a. You can generally drop non dominant terms. $O(N^2 + N) \rightarrow O(N^2)$
    b. Note! Preprocessing that is less than the dominant term doesn't impact the overall runtime.
2. Drop the constants
    a. $O(2N) \rightarrow O(N)$
3. Specify different variables for different inputs or input dimensions

Big O Analysis Recap

1. Be **very** comfortable doing time and space complexity analysis
2. Know all the runtimes associated with common algorithms, data structures
3. Use the order of runtimes to help you reason about improvements
    a. $O(1)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(2^n)$
4. Practice doing this with every problem you work on!
5. Multiple solutions with time/ space tradeoffs may exist
    a. Discuss this with your interviewer!
6. Guide
    a. https://guides.codepath.com/compsci/Big-O-Complexity-Analysis

In Class Walkthrough

Understand the Problem

Problem:

You have k lists of sorted integers in ascending order. Find the smallest range that includes at least one number from each of the k lists. We define the range [a,b] is smaller than range [c,d] if $b-a < d-c$ or $a < c$ if $b-a == d-c$.

Understand the Problem

Problem:
You have k lists of sorted integers in ascending order. Find the smallest range that includes at least one number from each of the k lists. We define the range [a,b] is smaller than range [c,d] if b-a < d-c or a < c if b-a == d-c.

Example 1:
Input:
```
[[4,10,15,24,26],
 [0,9,12,20],
 [5,18,22,30]]
```

Output: `[20,24]`

Understand the Problem

Problem:
You have k lists of sorted integers in ascending order. Find the smallest range that includes at least one number from each of the k lists. We define the range [a,b] is smaller than range [c,d] if b-a < d-c or a < c if b-a == d-c.

Example 1:
Input:

```
[[4,10,15,24,26],    24 from here
 [0,9,12,20],        20 from here
 [5,18,22,30]]       22 from here
```

Output: `[20,24]`

Understand the Problem

Problem:
You have k lists of sorted integers in ascending order. Find the smallest range that includes at least one number from each of the k lists. We define the range [a,b] is smaller than range [c,d] if b-a < d-c or a < c if b-a == d-c.

What would be the solution to this?

```
[[1, 22, 24, 27, 28, 35, 36],
 [7, 8, 12, 32, 37, 39],
 [2, 5, 14, 18, 23, 29]]
```

Understand the Problem

Problem:
You have k lists of sorted integers in ascending order. Find the smallest range that includes at least one number from each of the k lists. We define the range [a,b] is smaller than range [c,d] if b-a < d-c or a < c if b-a == d-c.

What would be the solution to this?

```
[[1, 22, 24, 27, 28, 35, 36],
 [7, 8, 12, 32, 37, 39],
 [2, 5, 14, 18, 23, 29]]
```

Answer

```
[28, 32]
```

Understand the Problem

Problem:
You have k lists of sorted integers in ascending order. Find the smallest range that includes at least one number from each of the k lists. We define the range [a,b] is smaller than range [c,d] if b-a < d-c or a < c if b-a == d-c.

What would be the solution to this?

```
[[1, 22, 24, 27, 28, 35, 36],   28 from here
 [7, 8, 12, 32, 37, 39],        32 from here
 [2, 5, 14, 18, 23, 29]]        29 from here
```

Answer

```
[28, 32]
```

Understand the Problem

Problem:
You have k lists of sorted integers in ascending order. Find the smallest range that includes at least one number from each of the k lists. We define the range [a,b] is smaller than range [c,d] if b-a < d-c or a < c if b-a == d-c.

Key things to note

- Lists are sorted
- Lists are various sizes
- There may be duplicate elements
- Example is quite overwhelming

Brute Force Approach

Steps

1. Generate all possible ranges from all items
2. For every range, check to see if there is a number in each list that falls in that range.
3. Keep track of the smallest one, and update accordingly

Runtime? N = number of items in the matrix

Number of Ranges * Checking Every Item = $O(N^2) * O(N) = O(N^3)$

Inefficiencies

- Redundant range checks
  - Once we find a valid range, we keep still check larger ranges
- We don't take advantage of the lists being sorted

Working through an example

Working through an example

```
[[1, 2, 6],
 [5, 7],
 [2, 8, 20]]
```

Working through an example

Let’s look at the first element in each list

```
[[1, …],
 [5, …],
 [2, …]]
```

Working through an example

Let’s look at the first element in each list

```
[[1, …],
 [5, …],
 [2, …]]
```

We are ignoring the other elements for now!

Working through an example

Let’s look at the first element in each list

```
    [[1, …], min
     [5, …], max
     [2, …]]
```

```
Best range: [1, 5]
```

Working through an example

```
    [[1, 2, …],
     [5, 7],
     [2, 8, …]]

Best range: [1, 5]
```

Working through an example

Option #1: In the first row, swap the 1 to 2

```
    [[1, 2, ...],
     [5, 7],
     [2, 8, ...]]

Range for these numbers: [2, 5]
Previous range: [1, 5]
```

Working through an example

Option #2: In the second row, swap the 5 to 7

```
    [[1, 2, …],
     [5, 7],
     [2, 8, …]]
```

```
Range for these numbers: [1, 7]
Previous range: [1, 5]
```

Working through an example

Option #3: In the last row, swap the 2 to 8

```
    [[1, 2, …],
     [5, 7],
     [2, 8, …]]
```

```
Range for these numbers: [1, 8]
Previous range: [1, 5]
```

Working through an example

Option 1: First Row

```
[[1, 2, …],
 [5, 7],
 [2, 8, …]]
```

Range for these numbers: `[2, 5]`

Option 2: Second Row

```
[[1, 2, …],
 [5, 7],
 [2, 8, …]]
```

Range for these numbers: `[1, 7]`

Option 3: Third Row

```
[[1, 2, …],
 [5, 7],
 [2, 8, …]]
```

Range for these numbers: `[1, 8]`

Working through an example

Option 1: First Row

```
[[1, 2, …],
  [5, 7],
  [2, 8, …]]
```

Range for these numbers: [2, 5]

Option 2: Second Row

```
[[1, 2, …],
  [5, 7],
  [2, 8, …]]
```

Range for these numbers: [1, 7]

Option 3: Third Row

```
[[1, 2, …],
  [5, 7],
  [2, 8, …]]
```

Range for these numbers: [1, 8]

Working through an example

```
[[1, 2, 6],
 [5, 7],
 [2, 8, 20]]
```

```
Best range: [2, 5]
```

Working through an example

```
    [[1, 2, 6], Swap 2 to 6
     [5, 7],
     [2, 8, 20]]

Best range: [2, 5]
```

Working through an example

```
    [[1, 2, 6],
     [5, 7],
     [2, 8, 20]] Swap 2 to 8

Best range: [2, 5]
```

Working through an example

```
    [[1, 2, 6],
     [5, 7],  Swap 5 to 7
     [2, 8, 20]]


Best range: [2, 5]
```

Working through an example

```
    [[1, 2, 6],
     [5, 7],  Swap 5 to 7
     [2, 8, 20]]
```

Best range: ~~[2, 5]~~ [6, 8]

Working through an example

```
    [[1, 2, 6],
     [5, 7],  Swap 5 to 7
     [2, 8, 20]]
```

Best range: ~~[2, 5]~~ [6, 8]

🎉 Done 🎉

High Level Plan

1. Look at the first elements in each list as the initial working set
2. Calculate the range for this working set by finding the min & max
3. Finding the smallest number from the working set.
4. Swap the smallest number with its neighbor (next in the list)
5. Calculate the new range from this new working set from the min and max.
    a. If it's better than the old range, store this new one.
6. Repeat until the last element of a list is the minimum of a range and we're done!

High Level Plan

1. Look at the first elements in each list as the initial working set
2. Calculate the range for this working set by finding the min & max
3. Finding the smallest number from the working set.
4. Swap the smallest number with its neighbor (next in the list)
5. Calculate the new range from this new working set from the min and max.
    a. If it's better than the old range, store this new one.
6. Repeat until the last element of a list is the minimum of a range and we're done!

What is the runtime of this plan?

Use N to represent of elements

Use k to represent the number of lists

High Level Plan

1. Look at the first elements in each list as the initial working set
2. Calculate the range for this working set by finding the min & max
3. Finding the smallest number from the working set.
4. Swap the smallest number with its neighbor (next in the list)
5. Calculate the new range from this new working set from the min and max.
    a. If it's better than the old range, store this new one.
6. Repeat until the last element of a list is the minimum of a range and we're done!

What is the runtime of this plan? O(N * k)

Use N to represent of elements

Use k to represent the number of lists

In Class Exercises

In Class Exercises

- 60 minutes
- Create a shared repl.it
- Rotate between members with UMPIRE
- Two questions
  - Basic Calc (Hard)
  - Asteroid Collision
- Questions? Message me on slack!

Recap

- Basic Calc
    - Tricky stack based problem
    - Really good practice for these types of problems!
- Asteroid Collision
    - Good stack practice problem

Back to the walkthrough…

Understand the Problem

Problem:
You have k lists of sorted integers in ascending order. Find the smallest range that includes at least one number from each of the k lists. We define the range [a,b] is smaller than range [c,d] if $b-a < d-c$ or $a < c$ if $b-a == d-c$.

High Level Plan

1. Look at the first elements in each list as the initial working set
2. Calculate the range for this working set by finding the min & max
3. Finding the smallest number from the working set.
4. Swap the smallest number with its neighbor
5. Calculate the new range from this new working set from the min and max.
    a. If it's better than the old range, store this new one.
6. Repeat until we hit the end of a list, we're done!

What is the runtime of this plan? O(N * k)

Use N to represent of elements

Use k to represent the number of lists

High Level Plan

1. Look at the first elements in each list as the initial working set
2. Calculate the range for this working set by finding the min & max
3. Finding the smallest number from the working set.
4. Swap the smallest number with its neighbor
5. Calculate the new range from this new working set from the min and max.
    a. If it's better than the old range, store this new one.
6. Repeat until the last element of a list is the minimum of a range and we're done!

Can we do EVEN better?

High Level Plan

1. Look at the first elements in each list as the initial working set
2. Calculate the range for this working set by finding the min & max
3. Finding the smallest number from the working set.
4. Add the smallest number's neighbor to the new working set.
5. Calculate the new range from this new working set from the min and max.
    a. If it's better than the old range, store this new one.
6. Repeat until we hit the end of a list, we're done!

Improvements:

- Keep a running max
- Use a heap!

Pseudocode

```
put first elements of the list in a min heap
get range & max of those nums
while heap is not empty:
     remove a num from the heap, use it a new min
     calculate the new range, running max - new min
     update the best range if this new range is smaller
     get the list corresponding to this min
     if this min num has no neighbor in its list (end of list):
          STOP and return the best range
     else:
          move one index down in that list and fetch the neighbor
          compare new element: set new max to be this new element if needed
          add to min heap
```

Pseudocode

```
put first elements of the list in a min heap
get range & max of those nums
while heap is not empty:
     remove a num from the heap, use it a new min
     calculate the new range, running max - new min
     update the best range if this new range is smaller
     get the list corresponding to this min
     if this min num has no neighbor in its list (end of list):
          STOP and return the best range
     else:
          move one index down in that list and fetch the neighbor
          compare new element: set new max to be this new element if needed
          add to min heap
```

Pseudocode

```
put first elements of the list in a min heap
get range & max of those nums
while heap is not empty:
     remove a num from the heap, use it a new min
     calculate the new range, running max - new min
     update the best range if this new range is smaller
     get the List corresponding to this min
     if this min num has no neighbor in its list (end of list):
          STOP and return the best range
     else:
          move one index down in that list and fetch the neighbor
          (need to store which row, index in row in the heap)
          compare new element: set new max to be this new element if needed
          add to min heap
```

Space & Time Complexity

There are k lists and N total number of elements:

- Space complexity: `O(k)`
- Time complexity: `O(N * log(k))`

Implementation

1. Try to implement this on your own before looking at the solution!
2. Solution will be posted in the portal

Recap: Weeks 1 & 2

Linked Lists

- Dummy Head
- Multiple Pass
- Linked List Two Pointer

Data structures

- Hash Tables
- Stacks and Queues
- Heaps

Reminders

- Panel on Saturday!
  - Submit your Q&A questions in the sli.do (check email)
- HackerRank will test **both** linked lists and core data structures
- Lots of problems to review
  - Warm up problem
  - Walkthrough problems
  - Group Exercises
  - Post assessment Questions