CS-UY 1134 Lab 4
Search Algorithms

Agenda

- Binary search algorithm
    - Visual representation
    - Code
- More on asymptotic analysis

The Binary Search Algorithm

- Motivation: What is the index of a specific value in an **ordered** list?
- Simple solution: Linearly iterate through list. Worst case runtime?

The Binary Search Algorithm

- Motivation: What is the index of a specific value in an **ordered** list?
- Simple solution: Linearly iterate through list. Worst case runtime?

  O(n). But we can do way better.

The Binary Search Algorithm

- Motivation: What is the index of a specific value in an **ordered** list?
- Simple solution: Linearly iterate through list. Worst case runtime? O(n). But we can do way better.
- A better solution: binary search
  - O(logn)
  - It’s what humans do when we try to open a page in a book. We don’t go over every single page from page 1. We make a guess and open somewhere in the middle.

A Visual Representation

Problem: Find the index referencing a value of 15

Step 1: Set ’Low’ and ‘High’ pointers

2	4	7	11	15	32	58	60
0	1	2	3	4	5	6	7

Low High

Problem: Find the index referencing a value of 15

Step 2: Calculate middle ('Mid') index

2	4	7	11	15	32	58	60
0	1	2	3	4	5	6	7

Low Mid High

$$mid = \left| \frac{low + high}{2} \right|$$

= (0 + 7) // 2 = 3

Problem: Find the index referencing a value of 15

Step 3: Compare value at mid index against target value

2	4	7	11	15	32	58	60
0	1	2	3	4	5	6	7

Low Mid High

11 != 15

Target value does not match value at mid index, so we ask: is the value at Mid higher or lower than our target value? What should we do with that information?

Problem: Find the index referencing a value of 15

Step 4: Adjust Low and High Indexes

- 11 is lower than the target value, so we know not to look at any indexes at or lower than Mid.
- We change or focus to the greater half of the list, recalculating our low index as: Low = Mid + 1

2	4	7	11	15	32	58	60
0	1	2	3	4	5	6	7

~~Low~~ Mid Low High

Problem: Find the index referencing a value of 15

Step 5: Rinse and repeat, steps 1 through 4

- Recalculate the value of mid: What should the new value be?

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

2	4	7	11	15	32	58	60
0	1	2	3	4	5	6	7
			Mid	Low			High

Problem: Find the index referencing a value of 15

Step 5: Rinse and repeat, steps 1 through 4

2	4	7	11	15	32	58	60
0	1	2	3	4	5	6	7
			~~Mid~~	Low	Mid		High

- Our value at Mid still doesn't match our target value - so we recalculate our low and high indexes.
- 32 is greater than 15, meaning we must discard the index Mid and all indexes greater than Mid.
- What should our new High index be?

Problem: Find the index referencing a value of 15

Step 5: Rinse and repeat, steps 1 through 4

2	4	7	11	15	32	58	60
0	1	2	3	4	5	6	7
				Low	Mid		~~High~~
				High			

Problem: Find the index referencing a value of 15

Step 5: Rinse and repeat, steps 1 through 4

2	4	7	11	15	32	58	60
0	1	2	3	4	5	6	7

Low ~~Mid~~

High

Mid

- Our value at Mid matches our target value!
- We are done, and can return the value of Mid, which is currently 4

Problem: What if we were instead searching for 16?

2	4	7	11	15	32	58	60
0	1	2	3	4	5	6	7

Low

High

Mid

Next step would be the same as before: 15 is lower than 16, so we adjust our Low pointer

Problem: What if we were instead searching for 16?

2	4	7	11	15	32	58	60
0	1	2	3	4	5	6	7

~~Low~~ Low

High

Mid

- Our Low index is now greater than our High index, an ‘invalid’ condition
- We can use this condition to know that the target value was not in the list, and we should stop searching

Binary Search: The Code

```
def binary_search(lst, x):
    # Set pointer locations
    low = 0
    high = len(lst) - 1

    while low <= high:
        # Calculate midpoint
        mid = (low + high) // 2

        # 3 cases
        # 1. We find the index of x
        if lst[mid] == x:
            return mid
        # 2. x is in lower half
        elif lst[mid] > x:
            high = mid - 1
        # 3. x is in upper half
        else:
            low = mid + 1

    # *4. x is not in lst
    return -1
```

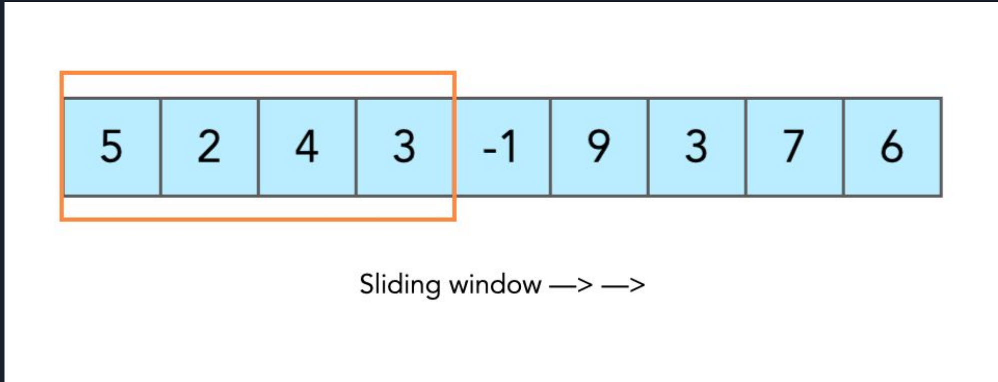
1. Initialize low and high pointers
2. Calculate midpoint
3. Check against target value; move pointers accordingly
4. Repeat 2 and 3
5. If low surpasses high, target value is not in the list . Some possible return values are then -1, None, or the size of the list, but use care when returning -1. (Why?)

Two pointers - sliding window

- One of the lab questions: given an array of length n and an integer k, we want to find <u>maximum sum</u> in a **contiguous** subarray of length n/k
- Ex) lst = [1, 12, -5, -6, 50, 3], k = 2 -> 47 (-6 + 50 + 3)
- Brute force approach: calculate needed info from scratch in every possible subarray
- Ex) max(sum(lst[0:3]), sum(lst[1:4]), sum(lst[2:5]), sum(lst[3:6]))
- Slow

Two pointers - sliding window

- Better approach: use two pointers with fixed distance to create a "window" and move it, while keeping track of needed info in a variable
- Each iteration, update variable solely based on what value just left the window, and what value just entered the window
- Again, this is one of your lab questions!



Asymptotic Analysis

- Last week, we analyzed the asymptotic runtime of some cases where the runtime per iteration is consistent
- Many times, overall runtime = (# of iterations) * (runtime per iteration)
- For example,

```
def func(n):
    for i in range(n):          # O(n) iterations
        print("Hello World.")   # O(1) per iteration
```

This function has an overall runtime of O(n * 1) = O(n)

Asymptotic Analysis

- What if the runtime per iteration changes?
- For example,

```
def print_triangle(n):
    for i in range(1, n + 1):   # O(n) iterations
        print('*' * i)          # O(i) per iteration
```

  In this function, the runtime of the line inside the loop depends on the value of i, which changes every iteration.
- How do we calculate the runtime of this function?
- Start by writing out the individual runtimes per iteration

```python
def print_triangle(n):
    for i in range(1, n + 1):
        print('*' * i)
```

Value of i	Runtime of iteration
1	1
2	2
3	3
…	…
n - 1	n - 1
n	n

Total runtime = Σ(individual runtimes) = 1 + 2 + 3 + … + n-1 + n = n * (n+1) / 2 = O(n^2) (arithmetic sequence)

Time to work on your labs! Have Fun!

Reminder: HW2 is out! You might want to start early.