Searching and Sorting

Searching in a List

- Search Problem. Given a list L of length n and an item e, is item *e* present in *L*?
- Function linearSearch(L, e)
- Examples

>>> linearSearch([12, 16, 23, 2, 7], 16) True

>>> linearSearch(['hello', 'world', 'silly'], 'hi') False

>>> linearSearch(['a', 'e', 'i', 'o', 'u'], 'u') True

Searching in an Unsorted List

- Search Problem. Given a list *L* of length *n* and an item *e*, is item *e* present in *L*?
- In the worst case need to look through the entire list
- O(n) algorithm

1	<pre>def linearSearch(L, e</pre>
2	n = len(L)
3	for item in L:
4	<pre>if item == e:</pre>
5	return Tru
6	return False



What if the list is sorted?

Example: Dictionary

- How do we look up a word in a dictionary?
- Words are listed in alphabetical order



The second secon

a landar of the second second based on the second s

a Trad Alla Lat. surrader, Fill a

the state of the s

A constraint of the second of

of the state of th

and a state of the bounds of all one the bound of the second of the bound of the bo

A trial of the bearing han barries, Brissian Bri

the kind or evelids many become

and a bar wat a gree control to Act of retiring a may a frequencies. Fr] 1. Act of retiring a may a frequencies. J Place of privacy i re

the color pair of press.

a balance of the second second

Analysis in a private solid, Lings, Lag.

(a) In temperature for the same of the same second distribution of the same property they are for a power and flawford with the same second se

Shrink back to my poternal cril, A line boule, with tren acrow,

a. "STRANS T. S. f. (retranslamant, Fr. Storm retereds.) The art of lopping tways. I. Vortification. (v) I had finited veryit's delays, his before summy mass the figures, the inter retranslaments of his fields, which it, app invest formewhen the gratify our imagements, on which it app invest downwith in some imagements, on which it app invest downwith in some imagements, the field The want of vowells in some imaging has been the grate original of our political authors.

tall entrumbereds, and configurity entrumes have made carrier. I would rather be an advocate for the retreachment, the discovery of this charley. Alterian RETRIBUTE, or a freterilan Lat. entrikan D. 1.

pay back t to make repayment of.

Searching in a Dictionary

- How do we look up a word in a dictionary?
- Words are listed in alphabetical order



Merriam Webster

> Let's assume we don't have these tabs to help us out

Search Algorithm

- Look at the middle page of the dictionary for our query word
- If we find our query, great!
- Otherwise:
 - If our query is later in alphabetical order to the words on the page, look for the query between the middle page and the last page
 - If our query is earlier in alphabetical order, look for the query between the middle page and the first page



How Good is This?

- **Goal:** Analyze how many pages we need to look at to look a word up in the dictionary
- Want the worst case: it's possible that I'm looking for a word that's right on the middle page
- Each time we look at the "middle" page remaining, the number of pages we need to look at is divided by 2 (Actually slightly better since we can rule out the middle page itself)
- A 1024-page dictionary requires at most 11 lookups: 1024 pages, < 512, <256, <128, <64, <32, <16, <8, <4, <2, <1 page.
- Just needed to look at 11 pages out of 1024 !
- Can we generalize this for an *n* page dictionary?



Review: Logarithm

- Logarithms are the inverse function to exponentiation
- Thus, $\log_b n$ describes the exponent to which base b must be raised to produce n
- That is, $b^{\log_b n} = n$
- Another way to look at logarithms:
 - $\log_b n$ is, essentially, the number of times n must be divided by b to reduce it to 1
- For us, important takeaway:
 - How many times can we divide n by 2 until we get down to 1
 - $\approx \log_2 n$

Log Summary

- Logarithms are defined by the relationship $n = b^{\log_b n}$
- The value $\log_h n$ is, essentially, the number of times n must be divided by b to reduce it to 1
- In CS, we often set b = 2 as we often design solutions where we divide the problem size in half
- We ignore base in Big Oh notation because they affect the value by a constant, that is,

 $\log_a n = \frac{\log_b n}{\log_b a} = \log_b n \cdot \log_a b = c \log_b n$

• Thus we write $O(\log n)$ to describe logarithmic growth with respect to the input

0









- Base cases:
 - Positive: if e == L[m], return True
 - Negative: if len(L) == 0, return False





Binary Search Code

- Let's implement this algorithm
- See Jupyter Notebook





Analysis of Binary Search

Binary Search Analysis

- Within a recursive call (function frame):
 - Constant number of steps (independent on n) and including at most one recursive call
- Total number of steps: O(# of recursive calls)

```
def binarySearch(L, e):
        if len(L) == 0:
 2
 3
             return False
        else: # recursive case
 4
             if L[mid] == e:
 5
 6
                 return True
             elif e <= L[mid]:</pre>
 7
 8
 9
             else:
10
```



Binary Search Analysis

- How many recursive calls? How many timing can we halve L until either we find the element or L has size < 1
- Size goes down by half in each recursive call:

$$n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow \cdots \rightarrow n/2^i =$$

1	<pre>def binarySearch(L, e):</pre>
2	if len(L) == 0:
3	return False
4	else: <i># recursive case</i>
5	if L[mid] == e:
6	return True
7	elif e <= L[mid]:
8	return binarySea
9	else:
10	return binarySea

- = 1

arch(L[:mid], e) arch(L[mid+1:], e)

Binary Search Analysis

- Number of recursive calls at most $\log_2 n + 1$
- Overall binary search is a $O(\log n)$ algorithm
- Grows very slowly with respect to *n* !



\log_2 (1 billion) ~ 30



The Sorting Problem

- **Problem**. Given a list of elements, sort those elements in ulletascending order.
- There are many ways to solve this problem!
- Built-in sorting functions in Python \bullet
 - sorted: returns new sorted list
 - sort(): destructive sort that sorts the list its called on
- Today: how do we design our own sorting algorithm
- **Question.** What is the best way to sort *n* items?
- We will use Big Oh to find out!

- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on





- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on





- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on





- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on





- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on





- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on





- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on





- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on



- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on





- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on



- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on



- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on





- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on



- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second \bullet position, and so on



- Find the smallest element and move it to the first position. \bullet
- Find the second-smallest element and move it to the second lacksquareposition, and so on



- Find the smallest element and move it to the first position.
- Find the second-smallest element and move it to the second lacksquareposition, and so on





• For each index i in the list L, we need to find the min item in L[i+1:], if its smaller than L[i], swap L[i] with that item





Λ

- For each index *i* in the list L, we need to find the min item in
 L[i+1:] so we can replace L[i] with that item
- In fact we need to find the position minPosition of the item that is minimum in L[i+1:]
- Reminder: how to swap values of variables **a** and **b**?
 - Using tuple assignment in Python:
 a, b = b, a
- Let's implement this algorithm

- For i = 0, inner loop runs n 1 items
- For i = 1, inner loop runs n 2 times
- . . .
- For i = n 1, inner loop runs 0 times

```
def selectionSort(L):
       """Destructive sort of list L,
 2
 3
       returns sorted list."""
       n = len(L)
 4
       for i in range(n):
 5
           minPosition = i
 6
            for j in range(i+1, n):
 7
                if L[minPosition] > L[j]:
 8
                    minPosition = j
 9
           L[i], L[minPosition] = L[minPosition], L[i]
10
11
       return L
```

- Within the inner loop we have O(1) steps (constant)
- Overall number of steps $(n-1) + (n-2) + \dots + 0$ $\leq n + (n - 1) + (n - 2) + \dots + 1$
- What is this sum?

```
def selectionSort(L):
       """Destructive sort of list L,
 2
3
       returns sorted list."""
       n = len(L)
 4
5
       for i in range(n):
           minPosition = i
 6
            for j in range(i+1, n):
 7
                if L[minPosition] > L[j]:
 8
                    minPosition = j
 9
           L[i], L[minPosition] = L[minPosition], L[i]
10
11
       return L
```

n + (n-1) + ... + 2 + 1 = n(n+1) / 2







Guassian Summation

$$S = n + (n - 1) + (n - 2) + S = 1 + 2 + \dots + (n - 2) +$$

$$2S = (n + 1) + (n + 1) + \cdots$$
$$2S = (n + 1) \cdot n$$
$$S = (n + 1) \cdot n \cdot 1/2$$

 $+ \cdots + 2 + 1$ + (n - 1) + n

(n + 1) + (n + 1) + (n + 1)

• Total number of steps

O(n(n + 1)/2)

- = O(n(n+1))
- $= O(n^2 + n)$
- $= O(n^2)$
- Is this the best we can do for sorting?
- No! Can sort in $O(n \log n)$ time !
- Next lecture: merge sort, a faster recursive sorting algorithm • that is optimal

Acknowledgments

These slides have been adapted from:

- http://cs111.wellesley.edu/spring19 and
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-<u>science/6-0001-introduction-to-computer-science-and-</u> programming-in-python-fall-2016/
- Selection sort images from: <u>https://web.stanford.edu/class/</u> archive/cs/cs106b/cs106b.1126/lectures/11/Slides11.pdf