

## Objectives

- Search strategies
- Broader Issues: One Laptop Per Child

Nov 30, 2007

Sprenkle - CS111

1

## Search Using **in** Review

- Iterates through a list, checking if the element is found
- Known as **linear search**
- **Implementation:**

```
def inSearch(searchlist, key):  
    for elem in searchlist:  
        if elem == key:  
            return True  
    return False
```

value	8	5	3	7
pos	0	1	2	3

What are the strengths and weaknesses of implementing search this way?

Nov 30, 2007

Sprenkle - CS111

search.py

2

## Search Using **in** Review

- Iterates through a list, checking if the element is found
- Known as **linear search**
- **Benefits:**
  - Works on *any* list
- **Drawbacks:**
  - Slow -- needs to check each element of list if the element is not in the list
  - Current implementation of **in** does not tell us where in the list it is
    - What if wanted to do something to that element?

Nov 30, 2007

Sprenkle - CS111

3

## Binary Search Review

- High-Low game
  - I'm thinking of a number between 1-100
  - You want to guess the number as quickly as possible
  - For every number you guess, I'll tell you whether you're too high or too low or if you got it right
- What is your best strategy?

Nov 30, 2007

Sprenkle - CS111

4

## Strategy: Eliminate Half the Possibilities

- Repeat until find value (or looked through all values):
  - Guess middle value of possibilities
  - If match, found!
  - Otherwise, find out too high or too low
  - Modify your possibilities
    - Eliminate the possibilities from your number and (higher or lower, as appropriate)

Nov 30, 2007

Sprenkle - CS111

5

## Searching for 8

-3	0	0	1	2	7	8	9
0	1	2	3	4	5	6	7

- Find the middle of the list
  - Positions: 0 -- 7, so mid is 3 (7/2)
- Check if the key equals the value at mid (1)
  - If so, report the location
- Check if the key is higher or lower than value at mid
  - Search the appropriate half of the list

-3	0	0	1	2	7	8	9
0	1	2	3	4	5	6	7

8 > 1, so look  
in upper half

Nov 30, 2007

Sprenkle - CS111

6

## Binary Search

- mid is 5  $((7+4)/2)$ , list[5] is 7

2	7	8	9
4	5	6	7

8 > 7, so look  
in lower half

- mid is 6  $((6+7)/2)$ , list[6] is 8

8	9
6	7

8 == 8, FOUND IT!

- What if searched for 6 instead of 8?

Nov 30, 2007

Sprengle - CS111

7

## Searching for 6

-3	0	0	1	2	7	8	9
0	1	2	3	4	5	6	7

- Will follow some of same program flow, but 6 is not in the list

- mid is 5, list[5] is 7

2	7	8	9
4	5	6	7

6 < 7, so will try in lower half  
of list

- mid is 4, list[4] is 2

2
4

6 > 2, so will try to look in upper  
half of the list, but we've already  
determined it's not there.  
How do we know to stop looking?

Nov 30, 2007

Sprengle - CS111

8

## Implementation Group Work

**def search(searchlist, key):**

"""Pre: searchlist is in sorted order.

Returns the position of key (an integer) in the  
list of integers (searchlist) or -1 if not found"""

- Trace through your program using examples
  - Start simple (small lists)
  - Do what the program says *exactly*, not what you *think* the program says

Nov 30, 2007

Sprengle - CS111

9

## One Solution

**def search(searchlist, key):**

low=0

high = len(searchlist)-1

**while** low <= high :

mid = (low+high)/2

**if** searchlist[mid] == key:

**return** mid # return True

**elif** searchlist[mid] < key:

low=mid+1

**else:**

high = mid-1

**return** -1 # return False

If you just want to  
know if it's in the list

Nov 30, 2007

Sprengle - CS111

search.py

10

## Binary Search

- Example of a **Divide and Conquer** algorithm
  - Break into smaller pieces that you can solve
- Benefits:
  - Faster to find elements (especially with larger lists)
- Drawbacks:
  - Requires that data can be compared (rather than just equal)
    - `__cmp__` method implemented by the class
  - List must be sorted before searching
    - Takes time to search

Nov 30, 2007

Sprengle - CS111

11

## Empirical Study of Search Techniques

- Goal: Determine which technique is better under various circumstances
- How long does it take to find various keys?
  - Measure by the number of comparisons
  - Vary the size of the list and the keys
  - What are good tests for the lists and the keys?

Nov 30, 2007

Sprengle - CS111

search\_compare.py

12

## Modifying Solution

```
def search(searchlist, key):
    low=0
    high = len(searchlist)-1
    mid = (low+high)/2
    while low <= high :
        if searchlist[mid] == key:
            return mid # return True
        elif searchlist[mid] < key:
            low=mid+1
        else:
            high = mid-1
            mid = (low+high)/2
    return -1 # return False
```

What if we had a list of Cards instead of a list of integers?

- What needs to be changed?
- What has to be done in the Card class?

Nov 30, 2007

Sprengle - CS111

13

## Comparing Card Objects

- What order do we want the cards in?

Nov 30, 2007

Sprengle - CS111

14

## Previously...

- Why isn't this sufficient for use with search?

```
def __cmp__(self, other):
    if self.getRank() < other.getRank():
        return -1
    if self.getRank() > other.getRank():
        return 1
    return 0
```

Nov 30, 2007

Sprengle - CS111

15

## Comparing Card Objects

Comparing by suit then rank; order is 2 Clubs, 3 Clubs, ..., Ace Clubs, 2 Diamonds, 3 Diamonds, ...

```
def __cmp__(self, other):
    if self.getSuit() < other.getSuit():
        return -1
    if self.getSuit() > other.getSuit():
        return 1
    if self.getRank() < other.getRank():
        return -1
    if self.getRank() > other.getRank():
        return 1
    return 0
```

Nov 30, 2007

Sprengle - CS111

card5.py

16

## Extensions to Solution

```
def search(searchlist, key):
    low=0
    high = len(searchlist)-1
    mid = (low+high)/2
    while low <= high :
        if searchlist[mid] == key:
            return mid # return True
        elif searchlist[mid] < key:
            low=mid+1
        else:
            high = mid-1
            mid = (low+high)/2
    return -1 # return False
```

What if we had a list of *Songs* instead of a list of integers?

- What if we wanted to check if the song's *title* matched the key and give the song back?

Nov 30, 2007

Sprengle - CS111

17

## Extensions to Solution

```
def search(searchlist, key):
    low=0
    high = len(searchlist)-1
    mid = (low+high)/2
    while low <= high :
        if searchlist[mid] == key:
            return mid # return True
        elif searchlist[mid] < key:
            low=mid+1
        else:
            high = mid-1
            mid = (low+high)/2
    return -1 # return False
```

What if we had a list of *Songs* instead of a list of integers?

- What if we wanted *all* songs that matched the title?

Nov 30, 2007

Sprengle - CS111

18

## Summary of Extensions to Solution

- Check the *title* of the Song at the midpoint
- Get the songs before and after that song in list that have the same title and put in a list
- Represent, handle when no song matches
- For “most intuitive” results:
  - Strip, lowercase the key
    - Which means what for your algorithm?
- Note: we’re not just doing “contains”
  - How could we implement that?

Nov 30, 2007

Sprenkle - CS111

19

## Search Strategies Summary

- Which search strategy should I use under the various circumstances?
  - I have a short list
  - I have a long list
  - I have a long sorted list

Nov 30, 2007

Sprenkle - CS111

20

## Search Strategies Summary

- Which search strategy should I use under the various circumstances?
  - I have a short list
    - How short? Linear (in)
  - I have a long list
    - Linear (in) -- because don’t know if in order, comparable
  - I have a long sorted list
    - Binary

Nov 30, 2007

Sprenkle - CS111

21

## A Scientific Application

Nov 30, 2007

Sprenkle - CS111

22

## Broader Issues

- One Laptop Per Child
  - An experiment on bringing cheap technology to poor children
  - Give 1, Get 1 program is on until Dec 31
- What challenges did OLPC face and how did that affect their design decisions?
- What are some other unusual features of the laptop?
- What does this technology mean for better-off countries?

Nov 30, 2007

Sprenkle - CS111

23

## Discussion

Challenge	Design Decision
Lack of Power	New, cheap battery; Consumes less power; Alternative power sources: solar power, pull cord
Software bloat	Rewrite code more compactly, efficiently
Environment	Dust proof, drop proof, light
Users - children	Simple user interfaces; tiny keyboard; light; applications keep students interested
Cost	Linux, Python, open-source tools; cheaper battery; no harddrive, CD/DVD drive

Nov 30, 2007

Sprenkle - CS111

24