COMP 364: Computer Tools for Life Sciences
Algorithm design: Linear and Binary Search

Christopher J.F. Cameron and Carlos G. Oliver
Key course information

Quiz #4 will be available on Monday!

- available on MyCourses (multiple choice questions)
- Quiz #4 closes at 11:59:59 pm on Monday, October 16th
- questions cover topics from the last two weeks

Midterm

- October 24, 2017 at 7:05-9:05 PM. Location: ENGMC 204
- multiple choice, short answer, and long answer questions
- covers course material until Wednesday, October 18th
- one 8x11 double-sided cheat sheet is allowed
- TAs will hold review sessions (TBA)
What are algorithms?

"The machine learning algorithm wants to know if we’d like a dozen wireless mice to feed the Python book we just bought."

Who is putting all the Algorithm books in the horror section?"
Algorithms

Algorithm (noun): word used by programmers when they do not want to explain what they did

What are they really?

An algorithm is a predetermined series of instructions for carrying out a task in a finite number of steps

▶ or a recipe

Input → algorithm → output
Example algorithm: baking a cake

What is the input? algorithm? output?
Pseudocode is an artificial and informal language that helps programmers develop algorithms

- a text-based detail design tool

The rules of Pseudocode

1. you do not talk about Pseudocode
2. you do not talk about Pseudocode
3. ...

Whops, wrong rule set
General rules of Pseudocode

All statements showing dependency are to be indented
  ▶ includes while, do, for, if, else

Given proper input descriptions, pseudocode
  ▶ should be in sufficient detail to directly support the programming effort
  ▶ is meant to elaborate on the algorithmic detail and not just cite an abstraction
  ▶ should be in prose (not full sentences)
Example Python statements

```python
students = ['Kris', 'David', 'JC', 'Emmanuel']
grades = [75, 90, 45, 100]
for student, grade in zip(students, grades):
    if grade >= 60:
        print(student, 'has passed')
    else:
        print(student, 'has failed')

#output:
#Kris has passed
#David has passed
#JC has failed
#Emmanuel has passed
```


Algorithm 1 Student assessment

1: for each student do
2: if student’s grade ≥ 60 then
3: print ‘student has passed’
4: else
5: print ‘student has failed’
6: end if
7: end for
Search algorithms

**Search** algorithms locate an item in a data structure

- **Sorting** algorithms will be covered next lecture

**Input:** a list of (un)sorted items and value of item to be searched

**Algorithms:** linear and binary search algorithms will be covered


**Output:** if value is found in the list, return index of item
Linear search

A very simple search algorithm

- a sequential search is made over all items one by one
- every item is checked
- if a match is found, then that particular item is returned
- otherwise the search continues until the end of the sequence

Example: search for the item with value 33
Linear search #2

Starting with the first item in the sequence:

Then the next:
Linear search #3

And so on and so on...
Linear search #4

Until an item with a matching value is found:

If no item has a matching value, the search continues until the end of the sequence.
Algorithm 2 Linear search

1: **procedure** LINEAR SEARCH(*sequence*, *value*)
2:  for each *item* in *sequence* do
3:     if *item* == *value* then
4:         return *item ‘s location*
5:     end if
6:  end for
7: **end procedure**
Linear search: algorithm

**Linear search** (*sequence, value*)

- **Step 1**: set $\textit{index}$ to 0
- **Step 2**: if $\textit{index} > N_{\textit{sequence}}$ then go to Step 7
- **Step 3**: if $\textit{sequence}[\textit{index}] == \textit{value}$ then go to Step 6
- **Step 4**: increase $\textit{index}$ by 1
- **Step 5**: go to Step 2
- **Step 6**: return $\textit{value}$ is found at $\textit{index}$
- **Step 7**: return $\textit{value}$ not found
def linear_search(sequence, value):
    index = 0
    found = False
    N = len(sequence)
    while index < N and not found:
        if sequence[index] == value:
            found = True
            index += 1
    if found:
        return str(value) + " is found at " + str(index)
    else:
        return str(value) + " not found"
Binary search

A fast search algorithm (compared to linear)
- works on the principle of ‘divide and conquer’
- the sequence of items must be sorted

Looks for a particular item
- by comparing the middle most item first
- if a match occurs, then the index of item is returned
- if the middle item is greater than the item, then the item is searched in the sub-list to the left
- otherwise, the item is searched for in the sub-list to the right
- this continues until the size of the sub-list reduces to zero
Example: let’s search for the value 31 in the following sorted sequence

```
sequence = [10, 14, 19, 26, 27, 31, 33, 35, 42, 44]
low = 0
high = len(sequence) - 1
mid = low + (high-low)/2  # integer division
print (mid)  # prints: 4
```
Binary search #3

Since $\text{index} = 4$ is the midpoint of the sequence

- we compare the value stored (27)
- against the value being searched (31)

The value at index 4 is 27, which is not a match

- the value being searched is greater than 27
- since we have a sorted array
- we know that the target value can only be in the upper portion of the list
Binary search #4

\( low \) is changed to \( mid + 1 \)

Now, we find the new \( mid \) First, we need to determine the middle item:

1. \( \text{low} = \text{mid} + 1 \) \( \# 5 \)
2. \( \text{mid} = \text{low} + (\text{high} - \text{low})/2 \) \( \# \text{ integer division} \)
3. \( \text{print} \ (\text{mid}) \) \( \# \text{ prints: } 7 \)
Binary search #4

*mid* is 7 now

- compare the value stored at index 7 with our value being searched (31)

The value stored at location 7 is not a match

- 35 is greater than 31
- since it’s a sorted list, the value must be in the lower half
- set *high* to *mid* - 1
Binary search #5

Calculate the mid again
  ➤ $mid$ is now equal to 5

We compare the value stored at index 5 with our value being searched (31)
  ➤ It is a match!
Binary search #6

Remember,

- binary search halves the searchable items
- improves upon linear search, but...
- requires a sorted collection

Useful links

bisect - Python module that implements binary search
- https://docs.python.org/2/library/bisect.html

Visualization of binary search
Algorithm 3 Binary search

1: procedure BINARY_SEARCH(sequence, value)
2:     low, high = 0, N_{sequence}-1
3:     while low \leq high do
4:         mid = (low + high) / 2
5:         if sequence[mid] > value then
6:             high = mid - 1
7:         else if sequence[mid] < value then
8:             low = mid + 1
9:         else
10:             return mid
11:     end if
12: end while
13: return ‘Not found’
14: end procedure
Binary search: algorithm

**Binary search** *(sequence, value)*

Step 1 - set *low* to 0

Step 2 - set *high* to *N*\textsubscript{sequence} - 1

Step 3 - if *low* > *high*, return ‘Not found’

Step 4 - set *mid* to average of *low* and *high*

Step 5 - if *sequence*\textsubscript{*mid*} == *value*, return *mid*

Step 6 - if *sequence*\textsubscript{*mid*} < *value*,
  set *low* to *mid* + 1 and go to Step 3

Step 7 - if *sequence*\textsubscript{*mid*} > *value*,
  set *high* to *mid* - 1 and go to Step 3
Binary search: Python implementation

```python
def binary_search(sequence, value):
    low = 0
    high = len(sequence) - 1
    # ...complete as homework
```