CMPT 120: Introduction to Computing Science and Programming 1

Searching

Copyright © 2018, Liaqat Ali. Based on CMPT 120 Study Guide and Think Python - How to Think Like a Computer Scientist, mainly. Some content may have been adapted from earlier course offerings by Diana Cukierman, Anne Lavergn, and Angelica Lim. Copyrights © to respective instructors. Icons copyright © to their respective owners.
Today’s Topics

1. Searching
2. Linear Search
3. Binary Search
Introduction to Searching

• Have you ever used Ctrl-F keys?
  ▫ We use it to search a value.
  ▫ How to search a value – how to search it fast?

• **Searching**: Locating an item in a list of data.

• Two of search algorithms are:
  1. **Linear** or Sequential Search.
  2. **Binary** Search.
     • Half-interval search.
     • Logarithmic search.
Linear Search

• Starting at the first element, this algorithm steps through an array \textit{sequentially}, examining each element until it locates the desired value.

  Suppose, an array \texttt{list} contains following values:

  \[
  \begin{array}{ccccccc}
  17 & 23 & 5 & 11 & 2 & 29 & 3 \\
  \end{array}
  \]

  ▫ To search a \texttt{value 11}, Linear Search compares 17, 23, 5, and 11.
  ▫ Say, we define two variable:
    ▫ \texttt{VALUE = 11}
    ▫ \texttt{found = False}
  ▫ How you will perform this Linear Search?

Liaqat Ali, Summer 2018
Linear Search

- **Algorithm:**

```python
list = [45, 12, 34, 2, 5, 40]
Set search value = 2

for i in range(len(list)):
    if list[i] is equal to search value
        return i

return -1
```
Linear search algorithm

linearSearch(list, target)

    set result to value TARGET_NOT_FOUND
    set targetNotFound to value true

    if list not empty
        set currentElement to first element of list
        while targetNotFound AND have not looked at every element of list
            if currentElement == target
                set result to current element
                set targetNotFound to false
            otherwise
                set currentElement to next element of list

    return result
Linear Search

- **Algorithm:**

  - Worst-case performance: $O(n)$
  - Best-case performance: $O(1)$
  - Average performance: $O(n)$
Linear Search - Tradeoffs

• Benefits:
  ▫ Easy algorithm to understand
  ▫ List can be in any order

• Disadvantages:
  ▫ Inefficient (slow): for a list of $N$ elements it examines:
    • $N/2$ elements on average for value in array,
    • $N$ elements for value not in array.
Binary Search

- **Binary Search** is another search algorithm.
- It requires array elements to be ordered (sorted).

1. Divides the array into **three** sections:
   - i. middle element
   - ii. elements on one side of the middle element
   - iii. elements on the other side of the middle element

2. If the middle element is the correct value, done. Otherwise, go to step 1. Using only the half of the array that may contain the correct value.

3. Continue steps 1. and 2. until either the value is found or there are no more elements to examine.
Binary search algorithm

**Question 1:** does your word start with a letter $\leq M$?

**Possible answer:**

- **Yes**, so we can ignore $\frac{1}{2}$ of the alphabet

  \[A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P\ Q\ R\ S\ T\ U\ V\ W\ X\ Y\ Z\]

- **No**, so we can ignore the other $\frac{1}{2}$

  \[A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P\ Q\ R\ S\ T\ U\ V\ W\ X\ Y\ Z\]
Next question:
• **Question 2:** does your word start with a letter ≤ G?

• Possible answer:
  ▫ Yes, so we can ignore ½ of the alphabet
    
    A B C D E F G H I J K L M
  ▫ No, so we can ignore the other ½
    
    A B C D E F G H I J K L M

OR

• **Question 2:** does your word start with a letter ≤ T?

• Possible answer:
  ▫ Yes, so we can ignore ½ of the alphabet
    
    N O P Q R S T U V W X Y Z
  ▫ No, so we can ignore the other ½
    
    N O P Q R S T U V W X Y Z
One possible algorithm -> binary search algorithm

Next question:
• Question 3: does your word start with a letter <= D?
• Possible answer:
  ▫ Yes, so we can ignore ½ of the alphabet
    A B C D E F G
  ▫ No, so we can ignore ½ of the alphabet
    A B C D E F G

OR
Question 3: does your word start with a letter <= J?
    H I J K L M
Question 3: does your word start with a letter <= Q?
    N O P Q R S T
Question 3: does your word start with a letter <= W?
    U V W X Y Z
eetc...

Anne_Lavergn2, Summer 2017.
Another example

• Suppose we have a sorted list:

1  3  4  7  9  11  12  14  21

• Using Binary Search algorithm, we can search for target = 7 without having to look at every element.
1. We start with a list and a target = 7

   1  3  4  7  9  11  12  14  21

2. We find the middle element

   1  3  4  7  9  11  12  14  21

3. Is this element == target?
   • Yes, then we are done!
   • No, then we throw away half of the list in which we know target cannot be located (grey part)

   1  3  4  7  9  11  12  14  21

   and we consider only the part in which target could be located.

   We repeat steps 2 and 3 until we found target or run out of list.
2. We find the *middle* element

3. Is this element == target?
   - Yes, then we are done!
   - No, then we throw away half of the list in which we know target cannot be located (grey part)

and we consider only the part in which target could be located

We repeat steps 2 and 3 until we found target or run out of list.
2. We find the *middle* element

\[
\begin{array}{c}
1 & 3 & 4 & 7 & 9 & 11 & 12 & 14 & 21
\end{array}
\]

3. Is this element == target?
   - Yes, then we are done!
   - No, then we throw away half of the list in which we know target cannot be located (grey part)

\[
\begin{array}{c}
1 & 3 & 4 & 7 & 9 & 11 & 12 & 14 & 21
\end{array}
\]

and we consider only the part in which target could be located

We repeat steps 2 and 3 until we found target or run out of list.

Anne_Lavergn2, Summer 1017.
2. We find the *middle* element

1 3 4 7 9 11 12 14 21

3. Is this element == target?
   - Yes, then we are done! 😊
Binary Search - Tradeoffs

• Benefits:
  ▫ Much more efficient than linear search. For array of N elements, performs at most $\log_2 N$ comparisons.
  ▫ Faster because does not have to look at every element (at every iteration, ignores $\frac{1}{2}$ of list).

• Disadvantages:
  ▫ Requires that array elements be sorted
Time efficiency of binary search algorithm

Result of binary search algorithm efficiency analysis:
- The worst case scenario of the **binary search algorithm** is of order \( \log_2 n \) i.e., has a time efficiency of \( O(\log_2 n) \).

since the time required (i.e., number of times the critical operation is executed) by the binary search algorithm (under the worst case scenario) to find “target” in a list of length \( n \) is proportional to the log of the number of elements in the list, i.e., \( n \).
Binary search **algorithm**: Time Complexity

- **Worst-case space complexity**: \( O(1) \)
- **Worst-case performance**: \( O(\log n) \)
- **Best-case performance**: \( O(1) \)
- **Average performance**: \( O(\log n) \)
Binary Search

Set `first` to 0
Set `last` to the last subscript in the array
Set `found` to false
Set `position` to -1

While `found` is not true and `first` is less than or equal to `last`
   Set `middle` to the subscript half-way between `array[first]` and `array[last]`.
   If `array[middle]` equals the desired value
      Set `found` to true
      Set `position` to `middle`
   Else If `array[middle]` is greater than the desired value
      Set `last` to `middle` - 1
   Else
      Set `first` to `middle` + 1
   End If.
End While.

Return `position`.
Binary Search algorithm - iterative

PreCondition: data must be sorted

binarySearch(list, target)
    set position to value TARGET_NOT_FOUND
    set targetNotFound to value true
    if list not empty
        while targetNotFound AND have not looked or discarded every element of list
            find middle element of list
            if middle element == target
                set position to position of target in original list
                set targetNotFound to false
            else
                if target < middle element
                    list = first half of list
                else
                    list = last half of list
        return position

We ignore 2nd half of the list and middle element
We ignore 1st half of the list and middle element

Anne_Lavergn2, Summer 1017.
Questions?