CMPT 120

Topic: Searching – Part 2
and Intro to Time Efficiency (Algorithm Analysis)
Last Lecture

• Intro to time efficiency and the Big O notation
• Analyzed the time efficiency of Linear Search
Learning Outcome

• At the end of this course, a student is expected to:
  • Create (design), analyze, and explain the behaviour of simple algorithms:
    • ... 
    • Describe and illustrate the operation of linear search, binary search, and $O(n^2)$ sorting algorithms
  • Analyze the running time of simple iterative algorithms
  • Compare the running time of algorithms; determine if one algorithm is more time efficient than another 

• Create (design) small to medium size programs using Python:
  • ... 
  • Create programs that search lists and strings
Today’s Menu

• We shall look at another searching algorithm
Lock-in Syndrome Activity
One possible algorithm -> binary search algorithm

- Question 1: does your word start with a letter ≤ M?
- Possible answer:
  - Yes, so we can ignore \( \frac{1}{2} \) of the alphabet
    
    \[
    \text{ABCDEFGHIJKLMNOPQRSTUVWXYZ}
    \]
  
  
  
  - No, so we can ignore the other \( \frac{1}{2} \)
    
    \[
    \text{ABCDEFGHIJKLMNOPQRSTUVWXYZ}
    \]
One possible algorithm -> binary search algorithm

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

etc...
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
How binary search works – In a nutshell

• Binary search uses the fact that the list is sorted!
  • Find element in the middle -> 9
  • Since we are looking for 7 and 7 < 9, then there is no need to search the second half of the list
    • We can ignore half of the list right away!
  • Then we repeat the above steps

• Bottom line: using binary search, we do not need to look at every element to search a list
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

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

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)

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

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.
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

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

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

• Advantages
  1. Faster because does not have to look at every element (at every iteration, ignores ½ of list)

• Disadvantages
  1. List must be sorted
  2. A bit more complicated to implement and test
Question:

- BTW, can we quickly tell if target is not in a list?
Time efficiency of binary search algorithm

• How long did it take Mr. Bauby to write his book?
• Answer: time it took to guess one letter of a word multiply by average length of a word in English (or French) multiply by number of words in JD Bauby’s book

• But, for now, let’s focus on how long it takes to find the target in list using the binary search algorithm? 😊
Time efficiency of binary search algorithm

• **Question**: What is the time efficiency of the worst case scenario of the binary search algorithm?

• **Answer**:
  1. What are the worst case scenarios?
     a) 
     b) 
  2. What are the “critical operations”?
     a) 
     b) 
  3. Now, let’s analyze binary search algorithm
     1. Figure out **count** (# of times critical operations are performed)
     2. Express **count** as a \( f(n) \)
     3. Match **count** (as a \( f(n) \)) with one of the Big O notations: \( O(1), O(\log n), O(n), O(n^2), \ldots \)
Step 1: Figure out **count** by counting # of times we do the critical operations

<table>
<thead>
<tr>
<th>Test Case:</th>
<th>iteration count:</th>
<th>== count:</th>
<th>n:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ourList = [1, 3, 4, 7, 9, 11, 12, 14, 21] target = 16</td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>ourList = [1, 3, 4, 7, 9, 11, 12, 14, 21] target = 7</td>
<td></td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>
Step 2 – Express **count** as a \( f(n) \)

- How does **count** relates to \( n \)?
- Using our example, how does **count** = 4 relates to \( n = 9 \)?
Step 2 – Express count as a $f(n)$

Binary Search Algorithm
At each iteration:
- We compare the middle element with target and if target not found
- We partition the list in half, ignoring one half and searching the other

Size of data in 1st iteration: $n$
Size of data in 2nd iteration: $n/2$
Size of data in 3rd iteration: $n/4$
Size of data in Tth iteration: 1
Step 2 – Express **count** as a $f(n)$

Note, on this slide, $N$ is used instead of $n$.

**Q?**

- How does the alg. behave?
- How many steps?
- How many divisions?

0.

1.

2.

$T$.

**Answer:** $T$ steps

**Express $T$ as a function of $N$:**

$$T = \log_2 N$$
Step 3 - Match **count** as a \( f(n) \) with ...

- ... one of the Big O notations:
  
  \( O(1), O(\log_2 n), O(n) \) or \( O(n^2) \)?
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., $n$
Comparing binary search to linear search

• How much “faster” is binary search over linear search as \( n \) goes to infinity?
  • To answer this question, let’s have a look at the diagram on next slide
Big O notation

# of times algorithm executes a “critical operation”

As $n$ goes to infinity

Length/size of data $n$

$O(n^2)$

$O(n)$

$O(\log_2 n)$

$O(1)$
Comparing binary search to linear search

• Binary search is **much** faster than linear search when searching large data

• But the data must first be sorted
  • Great if data is already sorted, but if this is not the case ...
  • How much work does this sorting requires?
Summary

• Looked at the second way of searching
  • Binary search algorithm
• Compare and contrast linear and binary searching algorithms
Next Lecture

- Sorting algorithms and their time efficiency