Lecture 27: Sorting & Searching

CS 1110

Introduction to Computing Using Python

http://www.cs.cornell.edu/courses/cs1110/2018sp
Announcements

• Academic Integrity:
  ▪ Remember to cite your sources

• Assignment 5
  ▪ Due 11:59pm on ***Wednesday*** May 9th
  ▪ Please see webpage for updates/clarifications

• Last Lab due this week in your Lab Section.

• Final Exam
  ▪ May 17th, 9am-11:30am
  ▪ **Location:** Barton Hall Central and East
  ▪ Conflict assignment on CMS
Final Exam Review Session

- Hollister B14 from 12-3:30

Saturday May 12:
- Lists/Sequences (12-1)
- Loop invariants/sequence algorithms (1-2)
- Problem solving session (2-3:30)

Sunday May 13:
- Call frames (12-1)
- Classes (1-2)
- Problem solving session (2-3:30)
Last Week’s Plan of Attack

- Insertion Sort
- Partition
- Quick Sort

Where we left off
Sorting with Partitions

- **Idea:** Pick a *pivot* element $x$
- Partition sequence into $\leq x$ and $\geq x$

\[ \begin{array}{c}
0 \\
b \quad x \\
? \\
n
\end{array} \]

Now partition this and this, too

*Keep recursing*...

\[ \begin{array}{c}
0 \\
b \quad \leq x \quad x \quad \geq x \\
n
\end{array} \]

\[ \begin{array}{c}
0 \\
b \quad \text{Sorted!} \\
n
\end{array} \]
def quick_sort(b, h, k):
    '''Sort the array fragment b[h..k]'''
    if k<=h:
        return
    i = partition(b, h, k)
    # INV: b[h..i-1] <= b[i] <= b[i+1..k]
    # Sort b[h..i-1] and b[i+1..k]
    quick_sort(b, h, i-1)
    quick_sort(b, i+1, k)

https://www.youtube.com/watch?v=m1PS8IR6Td0
QuickSort in the real world

```
DEFINE JOB INTERVIEW QUICKSORT (LIST):
  OK SO YOU CHOOSE A PIVOT
  THEN DIVIDE THE LIST IN HALF
  FOR EACH HALF:
    CHECK TO SEE IF IT'S SORTED
      NO, WAIT, IT DOESN'T MATTER
    COMPARE EACH ELEMENT TO THE PIVOT
      THE BIGGER ONES GO IN A NEW LIST
      THE EQUAL ONES GO INTO, EH
      THE SECOND LIST FROM BEFORE
    HANG ON, LET ME NAME THE LISTS
      THIS IS LIST A
      THE NEW ONE IS LIST B
    PUT THE BIG ONES INTO LIST B
    NOW TAKE THE SECOND LIST
      CALL IT LIST, UH, A2
      WHICH ONE WAS THE PIVOT IN?
    SCRATCH ALL THAT
      IT JUST RECURSIVELY CALLS ITSELF
    UNTIL BOTH LISTS ARE EMPTY
      RIGHT?
    NOT EMPTY, BUT YOU KNOW WHAT I MEAN
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```
Today’s Plan of Attack

• Quick Sort
• Linear Search
• Binary Search
Linear Search

- **Vague:** Find first occurrence of $v$ in $b[h..k-1]$.
- **Better:** Store an integer in $i$ to make this post-condition true:
  
  post:  
  1. $v$ is not in $b[h..i-1]$  
  2. $i = k$ OR $v = b[i]$
**Linear Search**

- **Vague**: Find first occurrence of \( v \) in \( b[h..k-1] \).
- **Better**: Store an integer in \( i \) to make this post-condition true:
  
  **post**:  
  1. \( v \) is not in \( b[h..i-1] \)  
  2. \( i = k \)  OR  \( v = b[i] \)

\[ \begin{array}{c}
\text{pre: } b \\
\hline
h & ? & k
\end{array} \]

\[ \begin{array}{c}
\text{inv: } b \\
\hline
h & i & \text{v not here} & ? & k
\end{array} \]

\[ \begin{array}{c}
\text{POST: } b \\
\hline
h & i & v & ? & \text{OR} & b & \hline
& & \text{v not here} & k = i
\end{array} \]
def linear_search(b, v, h, k):
    
    """Returns: first occurrence of v in b[h..k-1]"""
    
    # Store in i index of the first v in b[h..k-1]
    i = h

    # invariant: v is not in b[0..i-1]
    while i < k and b[i] != v:
        i = i + 1

    # post: v is not in b[h..i-1]
    # i >= k or b[i] == v
    return i if i < k else -1

Analyzing the Loop

1. Does the initialization make inv true?

2. Is post true when inv is true and condition is false?

3. Does the repetend make progress?

4. Does the repetend keep the invariant inv true?
Binary Search: What’s the Invariant?

- Look for $v$ in sorted sequence segment $b[h..k]$.
  - **Precondition:** $b[h..k-1]$ is sorted (in ascending order).
  - **Postcondition:** $b[h..i-1] < v$ and $v \leq b[i..k]$

Called **binary search** because each iteration of the loop cuts the array segment still to be processed in half.
Binary Search

<table>
<thead>
<tr>
<th>h</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Example: b [3 3 3 3 3 4 4 6 7 7]

- if v is 3, set i to 0
- if v is 4, set i to 5
- if v is 5, set i to 7
- if v is 8, set i to 10
Binary Search

\[ i = h; \ j = k + 1; \]

Looking at \( b[i] \) gives linear search from left.
Looking at \( b[j-1] \) gives linear search from right.
Looking at middle: \( b[(i+j)/2] \) gives binary search.

while \( i != j \):

Looking at \( b[i] \) gives linear search from left.
Looking at \( b[j-1] \) gives linear search from right.
Looking at middle: \( b[(i+j)/2] \) gives binary search.

\[ \text{pre: } b \quad ? \]

\[ \text{inv: } b < v \quad ? \quad >= v \]

\[ \text{post: } b < v \quad >= v \]
**Binary Search**

```python
def bsearch(b, v):
    i = 0
    j = len(b)

    while i < j:
        mid = (i+j)//2

        if b[mid] < v:
            i = mid + 1
        else:
            j = mid

    if i < len(b) and b[i] == v:
        return i
    else:
        return -1
```

**pre:**
- `b` is a list
- `i` and `j` are indices

**inv:**
- `b` is a list
- `i < j` and `b < v` and `b >= v`

**post:**
- `b` is a list
- `b < v` and `b >= v`
def bsearch(b, v):
    i = 0
    j = len(b)
    # invariant; b[0..i-1] < v, b[i..j-1] unknown, b[j..] >= v
    while i < j:
        mid = (i+j)//2
        if b[mid] < v:
            i = mid+1
        else:
            #b[mid] >= v
            j = mid

    if i < len(b) and b[i] == v:
        return i
    else:
        return -1

Analyzing the Loop

1. Does the initialization make \texttt{inv} true?

2. Is \texttt{post} true when \texttt{inv} is true and \texttt{condition} is false?

3. Does the repetend make progress?

4. Does the repetend keep the invariant \texttt{inv} true?
def rbsearch(b, v):
    """ len(b) > 0 """
    return rbsearch_helper(b, v, 0, len(b))

def rbsearch_helper(b, v, i, j):
    if i >= j:
        if i < len(b) and b[i] == v:
            return i
        else:
            return -1
    mid = (i + j) / 2
    if b[mid] < v:
        return rbsearch_helper(b, v, mid + 1, j)
    else:
        return rbsearch_helper(b, v, i, mid)