Announcements

• Labs 17 & 18 due Friday & Monday, respectively
• Next week’s discussion sections ➔ office hours for A6 and Prelim 2
• Final Exam on May 21st 1:30-4pm. Your assigned exam session (in-person or online) will be given in CMS tomorrow. Submit a “regrade request” in CMS by May 12 if you have a legitimate reason for requesting a change.

Algorithms for Search and Sort

• Well known algorithms
  ▪ focus on reviewing programming constructs (while loop) and analysis
  ▪ will not use built-in methods such as index, insert, sort, etc.
• Today we’ll discuss
  ▪ Linear search
  ▪ Binary search
  ▪ Insertion sort
• More on sorting next lecture
• More on the topic in next course, CS 2110!

Search Algorithms

• Search for a target x in a list v
• Start at index 0, keep checking until you find it or until no more elements to check

Searching in a List (Q)

Suppose another list is twice as long as v. The expected “effort” required to do a linear search is:

A. Squared
B. Doubled
C. The same
D. Halved
E. I don’t know

How do you search for a word in a dictionary? (NOT linear search)

To find the word “tanto” in my Spanish dictionary…

while dictionary is longer than 1 page:
  Open to the middle page
  if first entry comes before “tanto”:
    Rip* and throw away the 1st half
  else:
    Rip* and throw away the 2nd half

* For dramatic effect only—don’t actually rip your dictionary! Just pretend that the part is gone.
Repeated halving of “search window”

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>3000 pages</td>
</tr>
<tr>
<td>After 1  halving</td>
<td>1500 pages</td>
</tr>
<tr>
<td>After 2  halvings</td>
<td>750 pages</td>
</tr>
<tr>
<td>After 3  halvings</td>
<td>375 pages</td>
</tr>
<tr>
<td>After 4  halvings</td>
<td>188 pages</td>
</tr>
<tr>
<td>After 5  halvings</td>
<td>94 pages</td>
</tr>
<tr>
<td>After 12 halvings</td>
<td>1 page</td>
</tr>
</tbody>
</table>

Binary Search

- Repeatedly halve the “search window”
- An item in a sorted list of length \( n \) can be located with just \( \log_2 n \) comparisons.
- “Savings” is significant!

\[
\begin{array}{|c|c|}
\hline
n & \log_2(n) \\
\hline
100 & 7 \\
1000 & 10 \\
10000 & 13 \\
\hline
\end{array}
\]

Binary Search: \( \text{target } x = 70 \)

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline
i: & 0 & v[mid] & \text{is not } x \\
mid: & 5 & v[mid] & < x \\
j: & 11 & \text{So throw away the left half...} \\
\hline
\end{array}
\]

Binary Search: \( \text{target } x = 70 \)

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline
i: & 6 & v[mid] & \text{is not } x \\
mid: & 8 & v[mid] & < x \\
j: & 11 & \text{So throw away the right half...} \\
\hline
\end{array}
\]

Binary Search: \( \text{target } x = 70 \)

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline
i: & 6 & v[mid] & \text{is not } x \\
mid: & 6 & v[mid] & < x \\
j: & 7 & \text{So throw away the left half...} \\
\hline
\end{array}
\]

Binary Search: \( \text{target } x = 70 \)

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline
i: & 7 & v[mid] & \text{is not } x \\
mid: & 7 & v[mid] & < x \\
j: & 7 & \text{So throw away the left half...} \\
\hline
\end{array}
\]
**Binary Search:** target $x = 70$

<table>
<thead>
<tr>
<th>i</th>
<th>mid</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

DONE because $i$ is greater than $j$  
→ Not a valid search window

---

**The Insertion Process**

- Given a sorted list $x$, insert a number $y$ such that the result is sorted
- Sorted: arranged in ascending (small to big) order

```
2 3 6 9
   x
```

```
2 3 6 9
```

We’ll call this process a *push down,* as in push a value down until it is in its sorted position

---

**Binary search is efficient, but we need to sort the vector in the first place so that we can use binary search**

- Many sorting algorithms out there...
- We look at **insertion sort** now
- Next lecture we’ll look at **merge sort** and do some analysis

---

**The Insertion Process**

- Given a sorted list $x$, insert a number $y$ such that the result is sorted
- Sorted: arranged in ascending (small to big) order

```
2 3 6 9
```

We’ll call this process a *push down,* as in push a value down until it is in its sorted position

---

**Sort list $b$ using Insertion Sort**

Need to start with a sorted segment. How do you find one?

```
0 1 2 3 4 5
```

$\text{Length 1 segment is sorted}$

push\_down(b, 1)  Then sorted segment has length 2  
push\_down(b, 2)  Then sorted segment has length 3  
push\_down(b, 3)  Then sorted segment has length 4  
push\_down(b, 4)  Then sorted segment has length 5  
push\_down(b, 5)  Then entire list is sorted

For a list of length $n$, call push\_down $n-1$ times.
Helper functions make clear the algorithm

```python
def swap(b, h, k):
    ...
def push_down(b, k):
    while k > 0 and b[k-1] > b[k]:
        swap(b, k-1, k)
        k= k-1
def insertion_sort(b):
    for i in range(len(b)):
        push_down(b, i)
```

Algorithm Complexity

- Count the number of comparisons needed
- In the worst case, need $i$ comparisons to push down an element in a sorted segment with $i$ elements.

**Algorithm Complexity (Q)**

Count (approximately) the number of comparisons needed to sort a list of length $n$

A. $\sim 1$ comparison
B. $\sim n$ comparisons
C. $\sim n^2$ comparisons
D. $\sim n^3$ comparisons
E. I don’t know