Quiz 5
Project 2 will be due next Wednesday at 2:30. If you don’t come to lab because you’re still working on your project, you get 0%. (If you have to miss due to illness, then you must bring an official excuse, e.g. from a doctor, nurse, or parent with telephone numbers.)

Midterm March 10: more details soon
But everything we cover up through and including March 1.

A. Searching and Sorting

We will explore both in non-recursive and recursive versions.

Read Ch. 16 but look at my Python code which translates directly from the C/C++/Java code that is in the Zybooks.

B. Searching

Non-recursive and recursive: non-recursive is faster and uses less memory.

If the list to be searched is not sorted, then you should use linear search, i.e. search in a line.

There is no real quick and simple way to do it.

You could use a probabilistic algorithm, i.e. randomly choose indexes.

Look at binary search, both recursive and non-recursive version.

See the code.

Do binary search (non-recursive) by tracing.
my list = [1, 2, 4, 6, 10, 13, 15, 19, 28]

len(my list) // 2

[13, 15, 19, 28]

len( ) // 2

[28]

len( ) // 2

[ ]

len( ) // 2

n

target = 4

[1, 2, 4, 6]

4 // 2

2

N // 4

target < list[mid]

target = list[mid]

target = list[mid]

N // 2

Worst case running time

time complexity

for Binary search is

\[ \log_2(N) \]
**C. Big-Oh notation**

Suppose you have N numbers to search.

<table>
<thead>
<tr>
<th>N numbers</th>
<th>Seq. Search</th>
<th>Bin. Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>1024</td>
<td>10</td>
</tr>
<tr>
<td>2048</td>
<td>2048</td>
<td>11</td>
</tr>
<tr>
<td>4096</td>
<td>4096</td>
<td>12</td>
</tr>
<tr>
<td>8192</td>
<td>8192</td>
<td>13</td>
</tr>
<tr>
<td>16384</td>
<td>16384</td>
<td>14</td>
</tr>
<tr>
<td>1048576</td>
<td>1048576</td>
<td>20</td>
</tr>
</tbody>
</table>

Worst-case time required as a function of input size:

\[
T_{\text{seq}}(N) = N
\]

\[
T_{\text{bin}}(N) = \log_2(N)
\]

How much smaller is \(\log_2(N)\) than \(N\) for very large \(N\)? It is infinitesimally smaller!

\[
\lim_{N \to \infty} \frac{\log_2 N}{N} = 0
\]

So a function that grows at logarithm speeds is unbelievably better than one that grows at \(N\).

We will investigate this more clearly soon.

**D. Other complexity measures**

There are several different complexity measures:

For linear searching:

<table>
<thead>
<tr>
<th>Case</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>best case</td>
<td>1</td>
</tr>
<tr>
<td>average case</td>
<td>(N/2)</td>
</tr>
<tr>
<td>worst case</td>
<td>(N)</td>
</tr>
</tbody>
</table>
$T(N) = N$

$T(N) = \log_2(N)$
There are other things that we measure:

<table>
<thead>
<tr>
<th>Time</th>
<th>How many seconds it will take before you know the answer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>How many bytes of RAM it will require when it works.</td>
</tr>
<tr>
<td>Complexity</td>
<td>How difficult the algorithm is for a human to understand.</td>
</tr>
<tr>
<td>Hardware</td>
<td>How many cores the algorithm requires (parallel computing engines.)</td>
</tr>
</tbody>
</table>

### E. Trade-offs

You never get anything for free! You only decide how much you’re willing to pay, and what you are willing to sacrifice.

The cost of binary search is that the list must be in sorted order already.

If the list changes frequently, then binary search may not be a good idea because you have to continually resort, or at least put into sorted order.

You could put into sorted order by starting at the 0 index and traveling forward until you find a value larger than the insertion value. Then stick the new value there. The T(N) function for this is N in the worst case.

Binary search is easy to understand but it costs in terms of memory and even some time.

Generally you trade time against space.

- Algorithm A: uses little space but is quite slow.
- Algorithm B: uses a lot of space but is very very fast.

However there is no general statement we can make because sometimes

- Algorithm C: uses little space and is very fast.

One of the important things about Big-Oh notation and complexity analysis is to use good mathematical logic to make accurate statements. Can’t just run a few examples, use a little timing numbers.
Insert a new element into a sorted list and keep it in sorted order:

1, 2, 4, 6, 7, 9, 11, 13, 14, 20

8 → 3

\[ T(N) = N \] worst-case time complexity