Data Structures (+ Algorithms)

Goal: Many ways to solve a problem
- Learn about the most efficient ways
- Very often, this involves the data structures
  + algorithms used.

Data structure: system that stores data
Algorithm: steps (we use) to solve a problem.

1. Searching: in a sorted list of $n$ numbers, find the index of a number $x$.

   Simplest Way? Linear Search
   Time? Max # of numbers we check = $n$.

Better: Binary Search
- Like searching a phone book.
- Start at middle element, compare element to $x$.
- Stop if we find $x$;
- otherwise, search either left or right of middle.

\[
\begin{array}{c|c|c}
  i & y & \uparrow \\
  \hline
  \text{if } y > x & \text{search here} & \text{if } y < x \text{ check here} \\
  \text{if } y = x & \text{return } i & \text{search here}
\end{array}
\]
Time? Again, max # of numbers we check?

Every time we check the middle number, we either stop or discard half the list.
Since we want max, let's assume we never stop early.

<table>
<thead>
<tr>
<th>list size</th>
<th>check(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>n/2</td>
<td>1</td>
</tr>
<tr>
<td>n/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

until? [ ] (""")

Max # checks? \( \sim \log n \) (# of times we can halve n until we get 1).

2. Sorting: List of n integers between 0 and 10 (inclusive). Print the list sorted:

\[
\begin{array}{cccccc}
9 & 5 & 1 & \cdots & 4 & 7 & 1 & 3 \\
\end{array}
\]

Can we do better?
- Keep array of size 11 (countArr)
- Keep count of # of 0s, 1s, 2s, ... 10s
- Scan + print using countArr

<table>
<thead>
<tr>
<th>countArr</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 0</td>
</tr>
<tr>
<td>1 3 x2</td>
</tr>
<tr>
<td>2 0</td>
</tr>
<tr>
<td>3 1</td>
</tr>
<tr>
<td>4 1</td>
</tr>
<tr>
<td>5 1</td>
</tr>
<tr>
<td>6 0</td>
</tr>
<tr>
<td>7 1</td>
</tr>
<tr>
<td>8 0</td>
</tr>
<tr>
<td>9 1</td>
</tr>
<tr>
<td>10 0</td>
</tr>
</tbody>
</table>

Sorted: 1 1 3 4 5 7 9

Time? # of numbers we look at? n + 11.

<course info>
**COURSE INFO**

Java:

Java has data types

- (1) primitive data type - stores basic data
  - built into Java language

  int: 10, -5, 0 \([-2\text{ billion}, +2\text{ billion}]\)

  double: 5.63, -7.1, 10.0

  char: 'a', 'A', 'F', '8'

  boolean: true, false

Each data type has: range of values: set of operations

ex. int operations: + - * / % =; <, >, <=, >=

**Terminology**

- int x; \(\rightarrow\) semi-colon declaration: set variable type + name
- x=5; \(\rightarrow\) assignment/definition: set value
- int y=10 \(\rightarrow\) initialization/declare + assign
- int sum = x+y; \(\rightarrow\) initialization + operation

(Sample Code)
Object data type - stores complex data

- "Object-oriented programming"

- (a) built into Java (like primitive data)
- (b) built by programmer <later>

(b) ex. String - "stores" sequence of characters

Methods (operations): +, =, toUpperCase(), toLowerCase(), equals()

```java
String a = "Hello"
String b = "hello"

String c = a.toLowercase();

System.out.println(a.equals(b));  // false
System.out.println(c.equals(b));  // true
```

For primitive data, operations are basic.
For objects, methods are more complex.

- Might be wondering: which methods can be performed?

Class - defines (data values) + methods of an object.

⇒ Java API for String class

Notice: methods have input/output