UNIT 4A
Iteration: Searching
Goals of this Unit

• Study an iterative algorithm called linear (sequential) search that finds the first occurrence of a target in a collection of data.

• Study an iterative algorithm called insertion sort that sorts a collection of data into non-decreasing order.

• Learn how these algorithm scale as the size of the collection grows.

• Express the amount of work each algorithm performs as a function of the amount of data being processed.

Built-in Search in Ruby

movies = ["up", "wall-e", "toy story", "monsters inc", "cars", "bugs life", "finding nemo", "the incredibles", "ratatouille"]

movies.include?("wall-e") => true
movies.include?("toy") => false
movies.index("cars") => 4
movies.index("shrek") => nil
movies.index("Up") => nil
A Little More about Strings

• You can use relational operators to compare strings: <, <=, >, >=, ==, !=

• Comparisons are done character by character using ASCII codes.

Extended ASCII table

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NULL</td>
</tr>
<tr>
<td>01</td>
<td>start</td>
</tr>
<tr>
<td>02</td>
<td>end</td>
</tr>
<tr>
<td>03</td>
<td>space</td>
</tr>
<tr>
<td>04</td>
<td>tab</td>
</tr>
<tr>
<td>05</td>
<td>linefeed</td>
</tr>
<tr>
<td>06</td>
<td>formfeed</td>
</tr>
<tr>
<td>07</td>
<td>carrige</td>
</tr>
<tr>
<td>08</td>
<td>return</td>
</tr>
<tr>
<td>09</td>
<td>vertical</td>
</tr>
<tr>
<td>10</td>
<td>horizontal</td>
</tr>
<tr>
<td>11</td>
<td>subtab</td>
</tr>
<tr>
<td>12</td>
<td>dust</td>
</tr>
<tr>
<td>13</td>
<td>悬挂</td>
</tr>
<tr>
<td>14</td>
<td>拨号</td>
</tr>
<tr>
<td>15</td>
<td>拨号</td>
</tr>
<tr>
<td>16</td>
<td>拨号</td>
</tr>
<tr>
<td>17</td>
<td>拨号</td>
</tr>
<tr>
<td>18</td>
<td>拨号</td>
</tr>
<tr>
<td>19</td>
<td>拨号</td>
</tr>
<tr>
<td>20</td>
<td>拨号</td>
</tr>
<tr>
<td>21</td>
<td>拨号</td>
</tr>
<tr>
<td>22</td>
<td>拨号</td>
</tr>
<tr>
<td>23</td>
<td>拨号</td>
</tr>
<tr>
<td>24</td>
<td>拨号</td>
</tr>
<tr>
<td>25</td>
<td>拨号</td>
</tr>
<tr>
<td>26</td>
<td>拨号</td>
</tr>
<tr>
<td>27</td>
<td>拨号</td>
</tr>
<tr>
<td>28</td>
<td>拨号</td>
</tr>
<tr>
<td>29</td>
<td>拨号</td>
</tr>
<tr>
<td>30</td>
<td>拨号</td>
</tr>
<tr>
<td>31</td>
<td>拨号</td>
</tr>
<tr>
<td>32</td>
<td>拨号</td>
</tr>
</tbody>
</table>

15110 Principles of Computing, Carnegie Mellon University - CORTINA
Exercise on String Comparison

"Steelers" > "Jets" => true
"steelers" > "Jets" => true
"Steelers" > "jets" => false
"Steelers Nation" > "Steelers" => true
"Steelers Nation" > "Steelers" => false

Containment

Design an algorithm that returns true if a list contains a desired “key”, or false otherwise.
A contains? method

```ruby
def contains?(list, key)
    index = 0
    while index < list.length do
        if list[index] == key then
            return true
        end
        index = index + 1
    end
    return false
end
```

What happens if we execute `return` before we reach the end of the method?

A contains? method – version 2

```ruby
def contains?(list, key)
    for item in list do
        if item == key then
            return true
        end
    end
    return false
end
```
A contains? method – version 3

def contains?(list, key)
    list.each { |item|
        if item == key then
            return true
        end
    }
    return false
end

A contains? method – version 4

def contains?(list, key)
    list.each { |x| return true if x == key }
    return false
end

Important note: You can use this method on keys of any type, as long as the key’s type matches the type of the elements in the array.
Search

Design an algorithm that returns the index of the first occurrence of a key in a list if the key is present, or \texttt{nil} otherwise.

\begin{verbatim}
15110 Principles of Computing,
Carnegie Mellon University - CORTINA

def search(list, key)
    index = 0
    while index < list.length do
        if list[index] == key then
            return index
        end
        index = index + 1
    end
    return nil
end
15110 Principles of Computing,
Carnegie Mellon University - CORTINA
\end{verbatim}
Alternatively?

```python
def search(list, key)
    for item in list do
        if item == key then
            return index
        end
    end
    return nil
end
```

Why can’t we do this?

Ok, but...

```python
def search(list, key)
    for item in list do
        if item == key then
            return list.index(key)
        end
    end
    return nil
end
```

What’s undesirable about this?
Comparing Algorithms and Programs

• There may be many different algorithms for solving the same problem and different implementations of them as programs

• We can compare how efficient they are both analytically and empirically

Which One is Faster?

```python
def contains1?(list, key):
    index = 0
    while index < list.length do
        return true if list[index] == key
        index = index + 1
    end
    return false
end

def contains2?(list,key):
    len = list.length
    index = 0
    while index < len do
        return true if list[index] == key
        index = index + 1
    end
    return false
end
```

list.length is executed each time loop condition is checked
list.length is executed only once and its value is stored in len
Empirical Comparison Based on Running Time

• Add the following function to our collection of contains functions from the previous page:

```ruby
def contains3?(list, key)
    list.each { |x| return true if x == key }
    return false
end
```

• Start irb

• Include RubyLabs that provides the function `time`

Measuring Runtimes

```ruby
list1 = Array(1..1000000)
list2 = []
l2string = "This is a very long and complicated string with lots of characters."
l2probe = "This is a very long and complicated string with lots of characters?"
(1..(list1.length)).each { |list2 << l2string }
puts "contains1? on list1: 
puts time { contains1?(list1, -1) }
puts "contains2? on list1: 
puts time { contains2?(list1, -1) }
puts "contains3? on list1: 
puts time { contains3?(list1, -1) }
puts "contains1? on list2: 
puts time { contains1?(list2, l2probe) }
puts "contains2? on list2: 
puts time { contains2?(list2, l2probe) }
puts "contains3? on list2: 
puts time { contains3?(list2, l2probe) }
```

Ruby iterator is faster
String comparison is expensive