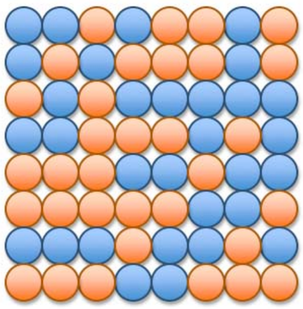
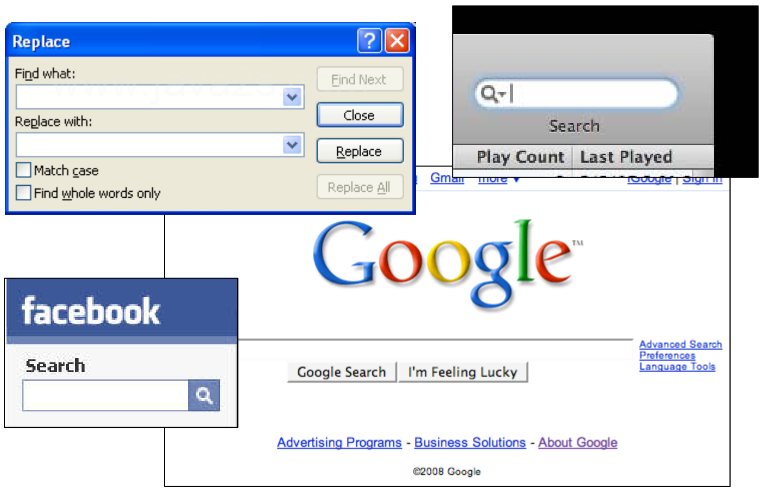


# UNIT 4A
# Iteration: Searching

## 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

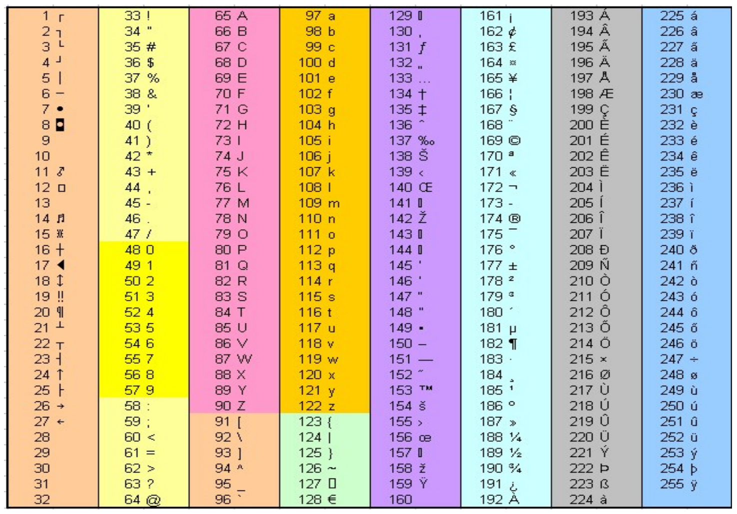
```
movies = ["up", "wall-e", "toy story", "monsters inc",
          "cars", "bugs life",  "finding nemo",
          "the incredibles", "ratatouille"]

movies.include?("wall-e") =>
movies.include?("toy") =>
movies.index("cars") =>
movies.index("shrek") =>
movies.index("Up") =>
```

## 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



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 ┌ | 33 ! | 65 A | 97 a | 129 ▯ | 161 ¡ | 193 Á | 225 á |
| 2 ┐ | 34 " | 66 B | 98 b | 130 ‚ | 162 ¢ | 194 Â | 226 â |
| 3 └ | 35 # | 67 C | 99 c | 131 ƒ | 163 £ | 195 Ã | 227 ã |
| 4 ┘ | 36 $ | 68 D | 100 d | 132 „ | 164 ¤ | 196 Ä | 228 ä |
| 5 \| | 37 % | 69 E | 101 e | 133 … | 165 ¥ | 197 Å | 229 å |
| 6 ─ | 38 & | 70 F | 102 f | 134 † | 166 ¦ | 198 Æ | 230 æ |
| 7 • | 39 ' | 71 G | 103 g | 135 ‡ | 167 § | 199 Ç | 231 ç |
| 8 ◘ | 40 ( | 72 H | 104 h | 136 ˆ | 168 ¨ | 200 È | 232 è |
| 9 | 41 ) | 73 I | 105 i | 137 ‰ | 169 © | 201 É | 233 é |
| 10 | 42 * | 74 J | 106 j | 138 Š | 170 ª | 202 Ê | 234 ê |
| 11 ♂ | 43 + | 75 K | 107 k | 139 ‹ | 171 « | 203 Ë | 235 ë |
| 12 □ | 44 , | 76 L | 108 l | 140 Œ | 172 ¬ | 204 Ì | 236 ì |
| 13 | 45 - | 77 M | 109 m | 141 ▯ | 173 - | 205 Í | 237 í |
| 14 ♫ | 46 . | 78 N | 110 n | 142 Ž | 174 ® | 206 Î | 238 î |
| 15 ☼ | 47 / | 79 O | 111 o | 143 ▯ | 175 ¯ | 207 Ï | 239 ï |
| 16 ┼ | 48 0 | 80 P | 112 p | 144 ▯ | 176 ° | 208 Ð | 240 ð |
| 17 ◄ | 49 1 | 81 Q | 113 q | 145 ' | 177 ± | 209 Ñ | 241 ñ |
| 18 ↕ | 50 2 | 82 R | 114 r | 146 ' | 178 ² | 210 Ò | 242 ò |
| 19 ‼ | 51 3 | 83 S | 115 s | 147 " | 179 ³ | 211 Ó | 243 ó |
| 20 ¶ | 52 4 | 84 T | 116 t | 148 " | 180 ´ | 212 Ô | 244 ô |
| 21 ┴ | 53 5 | 85 U | 117 u | 149 • | 181 µ | 213 Õ | 245 õ |
| 22 ┬ | 54 6 | 86 V | 118 v | 150 – | 182 ¶ | 214 Ö | 246 ö |
| 23 ┤ | 55 7 | 87 W | 119 w | 151 — | 183 · | 215 × | 247 ÷ |
| 24 ↑ | 56 8 | 88 X | 120 x | 152 ˜ | 184 ¸ | 216 Ø | 248 ø |
| 25 ├ | 57 9 | 89 Y | 121 y | 153 ™ | 185 ¹ | 217 Ù | 249 ù |
| 26 → | 58 : | 90 Z | 122 z | 154 š | 186 º | 218 Ú | 250 ú |
| 27 ← | 59 ; | 91 [ | 123 { | 155 › | 187 » | 219 Û | 251 û |
| 28 | 60 < | 92 \ | 124 \| | 156 œ | 188 ¼ | 220 Ü | 252 ü |
| 29 | 61 = | 93 ] | 125 } | 157 ▯ | 189 ½ | 221 Ý | 253 ý |
| 30 | 62 > | 94 ^ | 126 ~ | 158 ž | 190 ¾ | 222 Þ | 254 þ |
| 31 | 63 ? | 95 _ | 127 ▯ | 159 Ÿ | 191 ¿ | 223 ß | 255 ÿ |
| 32 | 64 @ | 96 ` | 128 € | 160 | 192 À | 224 à | |

## 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

```
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

```
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 **`nil`** otherwise.

## A **`search`** method

```
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
```

## Alternatively?

```
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...

```
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?

```
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:

```
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

```
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 }
print "contains1? on list1: "
puts time { contains1?(list1, -1) }
print "contains2? on list1: "
puts time { contains2?(list1, -1) }
print "contains3? on list1: "
puts time { contains3?(list1, -1) }
puts

print "contains1? on list2: "
puts time { contains1?(list2, l2probe) }
print "contains2? on list2: "
puts time { contains2?(list2, l2probe) }
print "contains3? on list2: "
puts time { contains3?(list2, l2probe) }
```

Ruby iterator is faster

String comparison is expensive