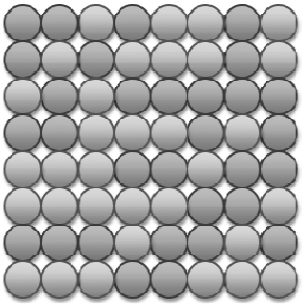
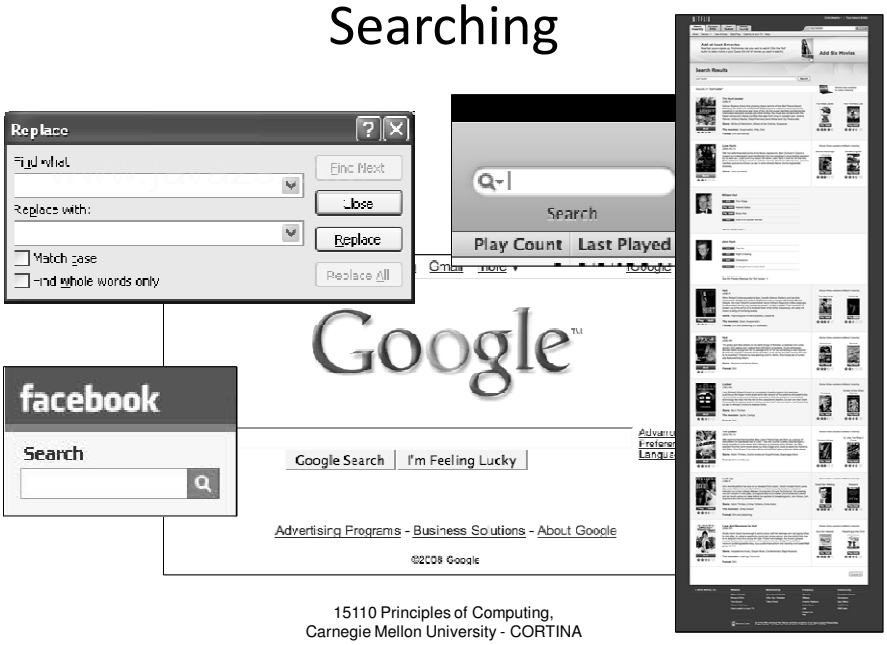


UNIT 4A
Iteration

Goals of this Unit

- Study an iterative algorithm called linear search that finds the first occurrence of a target in a collection of data.
- Study an iterative algorithm called selection 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.

Searching



Built-in Search in Ruby

```
movies = ["up", "wall-e", "toy story",
  "monsters inc", "cars", "bugs life",
  "finding nemo", "the incredibles",
  "ratatouille"]
movies.index("cars")       => 4
movies.index("shrek")      => nil
movies.index("Up")         => nil
movies.include?("wall-e") => true
movies.include?("toy")     => false
```

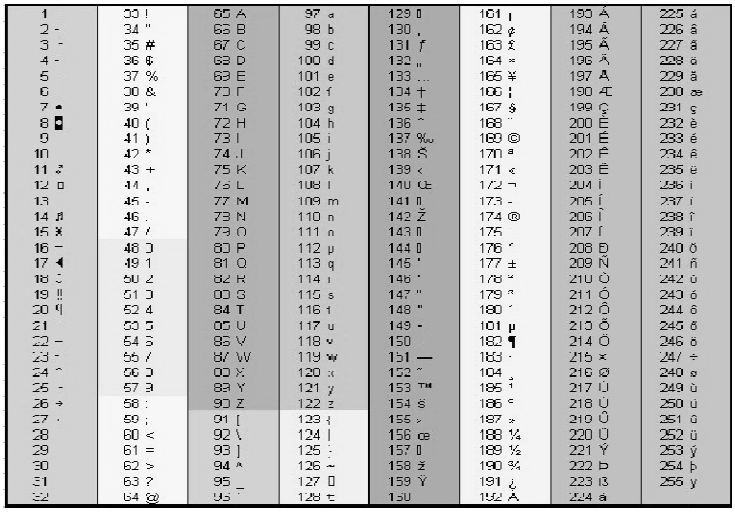
A Little More about Strings

You can use relational operators to compare strings.

Comparisons are done character by character using ASCII codes.

```
"smithers" > "burns"        => true
"homer" < "marge"           => true
"homer" < "Marge"           => false
"clancy" > "cletus"         => false
"bart" < "bartholomew"      => true
```


Extended ASCII table



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      # or index++
  end
  return nil
end
```


Sorry...

```
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?

Quite silly...

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

What's not so right about this?

Scalability

- Suppose our list contained n elements.
- In the worst case, how many elements would need to be examined during a linear search?
- In the best case, how many elements would need to be examined during a linear search?
- In the average case, how many elements would need to be examined during a linear search, if the key is present?

Scalability (cont'd)

- Suppose it takes time *t* in the worst case to search through a list using linear search.
- If we double the number of elements in the list, how long will the linear search take in the worst case?
- What if we triple the number of elements?
- In the worst case, the number of elements examined is linearly proportional to the number of elements in the list ----> linear search!