UNIT 4B
Iteration: Sorting
Announcements

- **lab1, lab2, pa1, pa2, ps1** grades should be available now
  - Contact your TA/instructor if they are missing
- **Ps3** is due Friday Feb 8\(^{th}\) in class
- New FAQ section on the web page
  - Please visit before sending email
  - Will continue to add things
- Written exam – Wed Feb 20\(^{th}\)
  - There will be help sessions (stay tuned)
Sorting
Comparison sorting

- Keys can be compared
  - keys are comparable as a whole
- ints, strings, characters

- Ascii table

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Oct</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>1</td>
<td>1 01</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>2</td>
<td>2 02</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>3</td>
<td>3 03</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>4</td>
<td>4 04</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>5</td>
<td>5 05</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>6</td>
<td>6 06</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>7</td>
<td>7 07</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>8</td>
<td>0 10</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>9</td>
<td>9 11</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>10</td>
<td>A 12</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>11</td>
<td>B 13</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>12</td>
<td>C 14</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>13</td>
<td>D 15</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>14</td>
<td>E 16</td>
<td>00</td>
<td>space</td>
</tr>
<tr>
<td>15</td>
<td>F 17</td>
<td>00</td>
<td>space</td>
</tr>
</tbody>
</table>

Source: www.LookupTables.com

15110 Principles of Computing, Carnegie Mellon University - GUNA
Category Sorting

1 3 3 1 2
The Art of Computer Programming
Volume 3: Sorting and Searching

The Classic Work
Newly Updated and Revised

The Art of Computer Programming

Volume 3
Sorting and Searching
Second Edition

Donald E. Knuth
Insertion Sort
Good Practices of writing code

• From high level to detail

• Flow charting to help organize

• Helper functions

• testing
Insertion sort demo
To a flow chart
def isort(list):
    -- create an empty newlist
    -- for value in list
    --    insert value in-order to newlist
    -- end
    end
From pseudo code to partial code

def isort (list)
    result = [ ]
    for val in list do
        # insert val in its proper place in result
    end
    return result
end
Array Insert method

- list.insert(position, value)

```python
gt a = [10, 20, 30]
gt => [10, 20, 30]
gt => a.insert(0,"foo")
gt => ["foo", 10, 20, 30]
gt => a.insert(2, "bar")
gt => ["foo", 10, "bar", 20, 30]
gt => a.insert(5, "baz")
gt => ["foo", 10, "bar", 20, 30, "baz"]
```
def isort (list)
    result = [ ]
    for val in list do
        place = /* compute this */
        result.insert(place, val)
    end
    return result
end
Computing the place to insert

# index of first element greater than item

def gindex (list, item):
    index = 0
    while index < list.length and list[index] < item:
        index = index + 1
    return index
Testing gindex

```python
>> a = [10, 20, 30, 40, 50]
=> [10, 20, 30, 40, 50]
>> gindex(a,3)
=> 0
>> gindex(a,14)
=> 1
>> gindex(a,37)
=> 3
>> gindex(a,99)
=> 5
```
Putting it all together

def isort (list)
    result = [ ]
    for val in list do
        place = gindex(result,val)
        result.insert(place, val)
    end
    return result
end
def isort (list)
    result = [ ]
    for val in list do
        place = gindex(result, val)
        result.insert(place, val)
    end
    return result
end
Testing the code

def isort (list)
    result = [ ]
    p result  # for debugging
    for val in list do
        result.insert(gindex(result,val), val)
        p result  # for debugging
    end
    return result
end
Review/Bonus slides
Improving performance
Can We Do Better?

• isort doesn’t change its input list.
• Instead it makes a new list, called result.
• This takes twice as much memory.

• Can we write a destructive version of the algorithm that doesn’t use extra memory?
• That is the version shown in the book (see chapter 4).
Destructive Insertion Sort

Given an array $a$ of length $n$, $n > 0$.

1. Set $i = 1$.
2. While $i$ is not equal to $n$, do the following:
   (i). Insert $a[i]$ into its correct position in $a[0..i]$, shifting the other elements as necessary.
   (ii). Add 1 to $i$.
3. Return the array $a$ which will now be sorted.
def isort!(list)
    i = 1
    while i != list.length do
        move_left(list, i)
        i = i + 1
    end
    return list
end

insert a[i] into a[0..i] in its correct sorted position
def isort!(list)
    i = 1
    while i != list.length do
        -- what is true about list here?
        move_left(list, i)
        i = i + 1
    end
    return list
end

insert a[i] into a[0..i] in its correct sorted position
Moving left demo
The **move_left** algorithm

Given an array $a$ of length $n$, $n > 0$ and a value at index $i$ to be “moved left” in the array.

1. Remove $a[i]$ from the array and store in $x$.
2. Set $j = i - 1$.
3. While $j >= 0$ and $a[j] > x$, do the following:
   a. Subtract 1 from $j$.
4. Reinsert $x$ into position $a[j+1]$.
move_left in Ruby

def move_left(a, i)
    x = a.slice!(i)
    j = i-1
    while j >= 0 and a[j] > x do
        j = j - 1
    end
    a.insert(j+1, x)
end

remove the item at position i in array a and store it in x

logical operator AND: both conditions must be true for the loop to continue

insert x at position j+1 of array a, shifting all elements from j+1 and beyond over one position
Why is the algorithm correct?

• Reason with loop invariants
  – The loop invariant
    • $A[0..i-1]$ is sorted at i-th iteration
  – The loop invariant is true at the beginning of each iteration
  – Loop invariant is true after the last iteration. After the last iteration, when we go to step 3:
    $a[0..i-1]$ is sorted AND i is equal to n
iterators
Iterators

• Iterators are another way to operate on the elements of an array.

• The `each` iterator is similar to a `for` loop:

```ruby
def statement { |f| puts "Yummy " + f + "pie!" }
fruits.each { |f| puts "Yummy " + f + "pie!" }
```

This `{ }` thing is called a “block” in Ruby

• Ruby provides lots of other iterators that do cool and useful things.
Compare

Using a for loop:

```ruby
for f in fruits do
  puts “Yummy ” + f
end
```

Using an iterator:

```ruby
fruits.each { |f|  
  puts “Yummy ” + f
}
```
“Destructive” Iterators

• Some iterators modify the array. Beware!

```ruby
items = (1..10).to_a
=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

items.delete_if { |i| i.odd? }
=> [2, 4, 6, 8, 10]

items => [2, 4, 6, 8, 10]
```
Flow Charts
def nestedcode(n)
    if n<=1 then
        return nil
    end
    for i in 2..n do
        if i%2==0 then
            puts i
        end
    end
end
Tracing code
def printable(n):
    i = n
    while (i >= 1) do
        for j in 1..i do
            print j.to_int() + " "
        end
        puts
        i = i - 1
    end
end
Array Manipulation
Useful Array methods

- A.length
- A.include?(item)
- A.index(item)
- Array.at(index)
- Array.insert(index, item)
- Array.delete(index)
List equivalence

• **Problem:** Given two lists L1 and L2, return true if two lists contains the same elements (in any order)
Best Coding Practices
Grading on Code Formatting

• From now on, you will be graded on the appearance of your code.
• Proper indentation, no gratuitous blank lines. (But in long functions, blank lines can be a good way to group code into sections.)
• Why are we doing this?
  – Because we’re mean.
  – Because you cannot find the bugs in your code if you cannot read it properly.
Indenting a FOR Loop

```plaintext
for var in values do
  loop body stuff
  more loop body stuff
  even mode loop body stuff
end
```
Indenting a WHILE Loop

```plaintext
while test do
    loop body stuff
    more loop body stuff
    even mode loop body stuff
end
```
Indenting an IF

```
if test then
  some then stuff
  more then stuff
else
  some else stuff
  more else stuff
end
```
Nesting

\[ x = [3, 13, 5, 25, 4, 64] \]

```
for v in x do
    if v < 10 then
        print " ", v
    else
        print v
    end
print "\n"
end
```