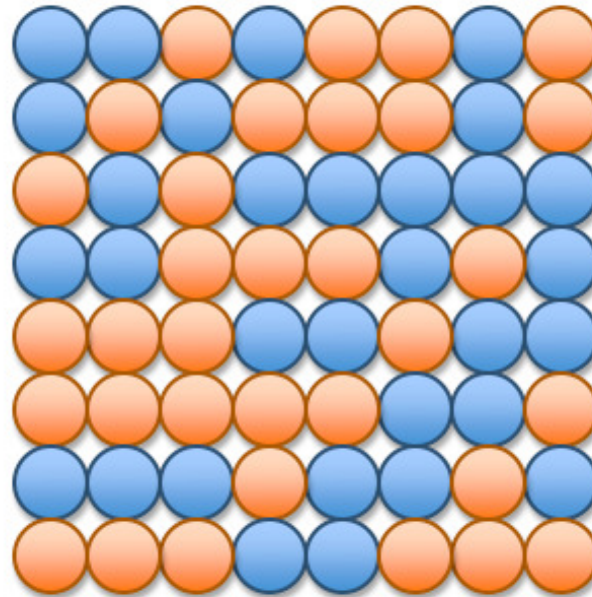




UNIT 4B

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 8th in class
- New FAQ section on the web page
 - Please visit before sending email
 - Will continue to add things
- Written exam – Wed Feb 20th
 - There will be help sessions (stay tuned)

Sorting

Comparison sorting

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

aba > aAb
 ↑ ↑ ↑ ↑

– Ascii table

12 < 23

"a" < "b"

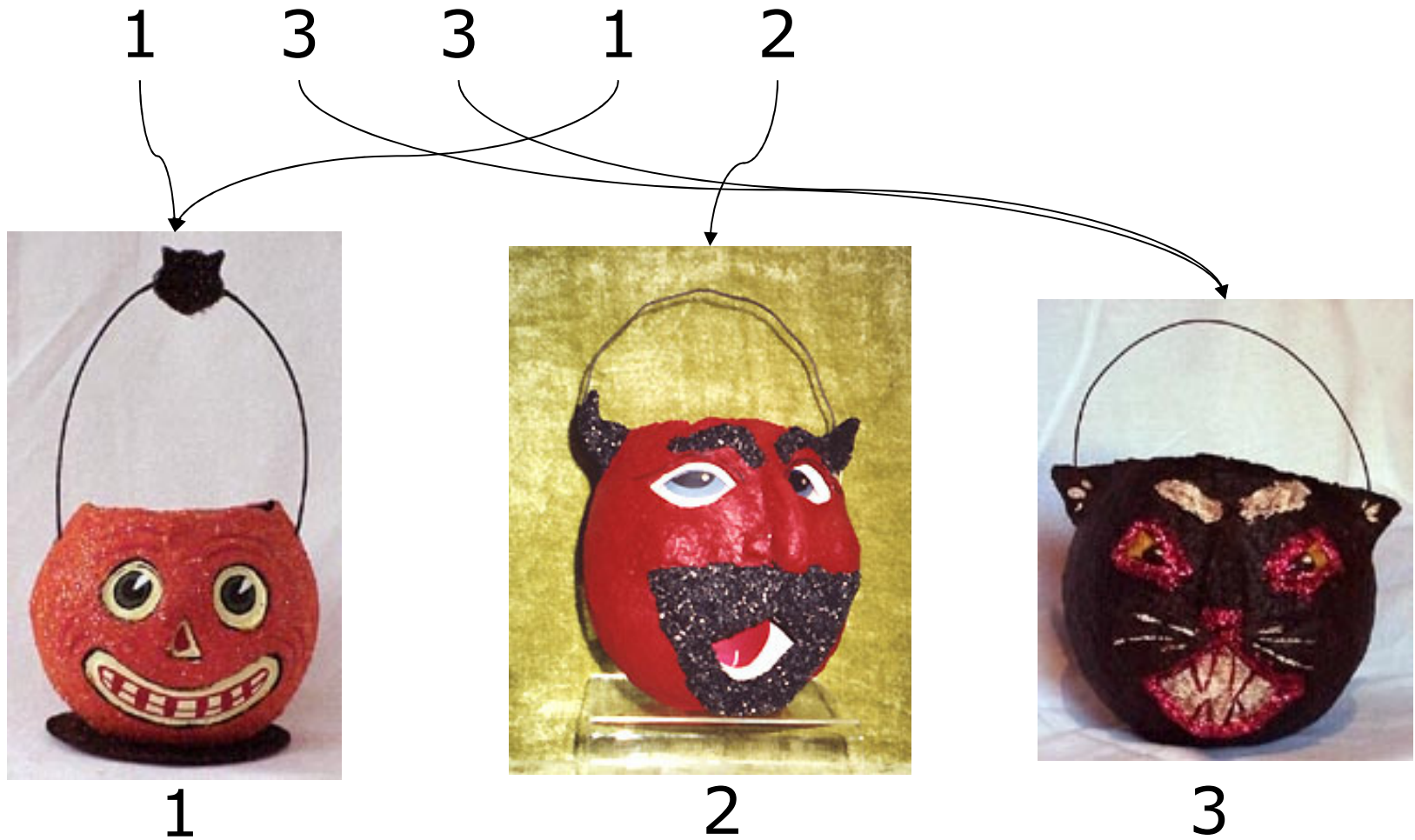
"AA" < "aa"

"a" > "A"

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	Space	32	64	100	64	140	96	60	140	96	140	96
1	1	001	SOH (start of heading)	33	21	041	!	33	65	101	65	141	97	61	141	97	141	97
2	2	002	STX (start of text)	34	22	042	"	34	66	102	66	142	98	62	142	98	142	98
3	3	003	ETX (end of text)	35	23	043	#	35	67	103	67	143	99	63	143	99	143	99
4	4	004	EOT (end of transmission)	36	24	044	\$	36	68	104	68	144	100	64	144	100	144	100
5	5	005	ENQ (enquiry)	37	25	045	%	37	69	105	69	145	101	65	145	101	145	101
6	6	006	ACK (acknowledge)	38	26	046	&	38	70	106	70	146	102	66	146	102	146	102
7	7	007	BEL (bell)	39	27	047	'	39	71	107	71	147	103	67	147	103	147	103
8	8	010	BS (backspace)	40	28	050	(	40	72	110	72	150	104	68	150	104	150	104
9	9	011	TAB (horizontal tab)	41	29	051	)	41	73	111	73	151	105	69	151	105	151	105
10	A	012	LF (NL line feed, new line)	42	2A	052	*	42	74	112	74	152	106	70	152	106	152	106
11	B	013	VT (vertical tab)	43	2B	053	+	43	75	113	75	153	107	71	153	107	153	107
12	C	014	FF (NP form feed, new page)	44	2C	054	,	44	76	114	76	154	108	72	154	108	154	108
13	D	015	CR (carriage return)	45	2D	055	-	45	77	115	77	155	109	73	155	109	155	109
14	E	016	SO (shift out)	46	2E	056	.	46	78	116	78	156	110	74	156	110	156	110
15	F	017	SI (shift in)	47	2F	057	/	47	79	117	79	157	111	75	157	111	157	111
16	10	020	DLE (data link escape)	48	30	060	0	48	80	120	80	160	112	76	160	112	160	112
17	11	021	DC1 (device control 1)	49	31	061	1	49	81	121	81	161	113	77	161	113	161	113
18	12	022	DC2 (device control 2)	50	32	062	2	50	82	122	82	162	114	78	162	114	162	114
19	13	023	DC3 (device control 3)	51	33	063	3	51	83	123	83	163	115	79	163	115	163	115
20	14	024	DC4 (device control 4)	52	34	064	4	52	84	124	84	164	116	80	164	116	164	116
21	15	025	NAK (negative acknowledge)	53	35	065	5	53	85	125	85	165	117	81	165	117	165	117
22	16	026	SYN (synchronous idle)	54	36	066	6	54	86	126	86	166	118	82	166	118	166	118
23	17	027	ETB (end of trans. block)	55	37	067	7	55	87	127	87	167	119	83	167	119	167	119
24	18	030	CAN (cancel)	56	38	070	8	56	88	130	88	170	120	84	170	120	170	120
25	19	031	EM (end of medium)	57	39	071	9	57	89	131	89	171	121	85	171	121	171	121
26	1A	032	SUB (substitute)	58	3A	072	:	58	90	132	90	172	122	86	172	122	172	122
27	1B	033	ESC (escape)	59	3B	073	;	59	91	133	91	173	123	87	173	123	173	123
28	1C	034	FS (file separator)	60	3C	074	<	60	92	134	92	174	124	88	174	124	174	124
29	1D	035	GS (group separator)	61	3D	075	=	61	93	135	93	175	125	89	175	125	175	125
30	1E	036	RS (record separator)	62	3E	076	>	62	94	136	94	176	126	90	176	126	176	126
31	1F	037	US (unit separator)	63	3F	077	?	63	95	137	95	177	127	91	177	127	177	127

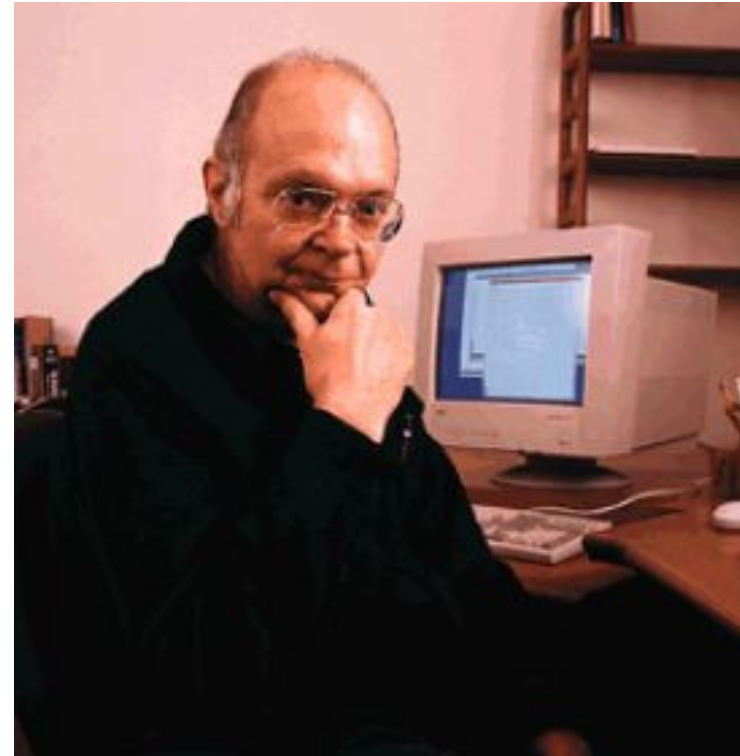
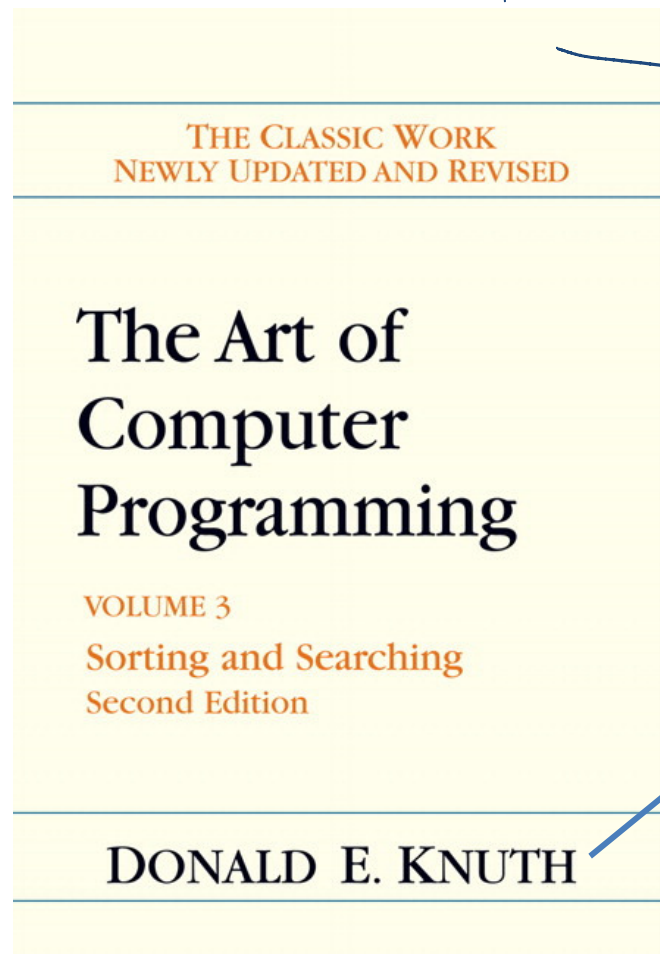
Source: www.LookupTables.com

Category Sorting



The Art of Computer Programming

Volume 3: Sorting and Searching

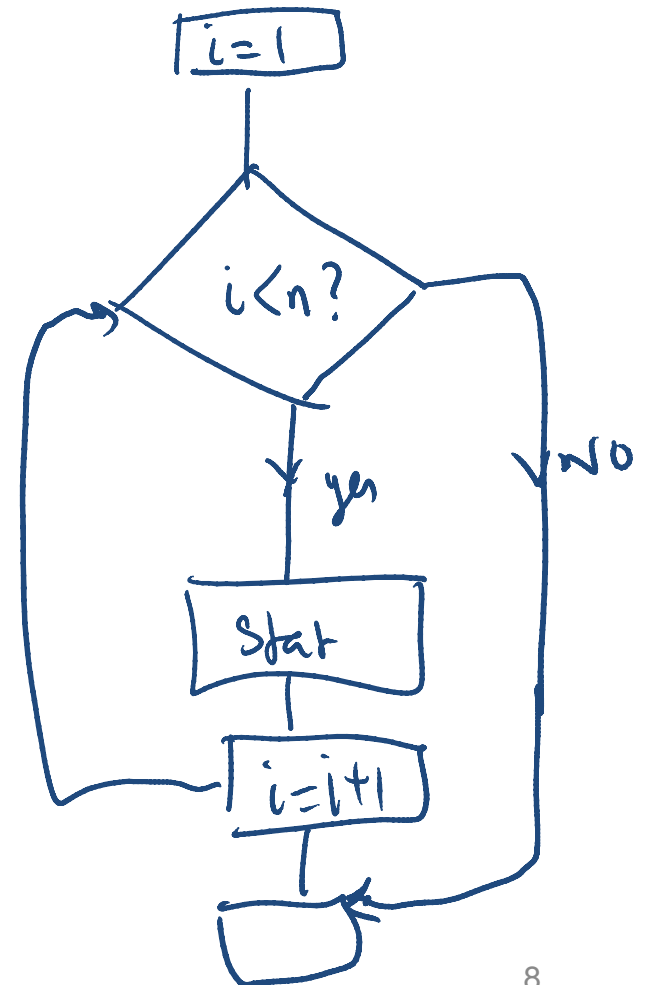


Insertion Sort

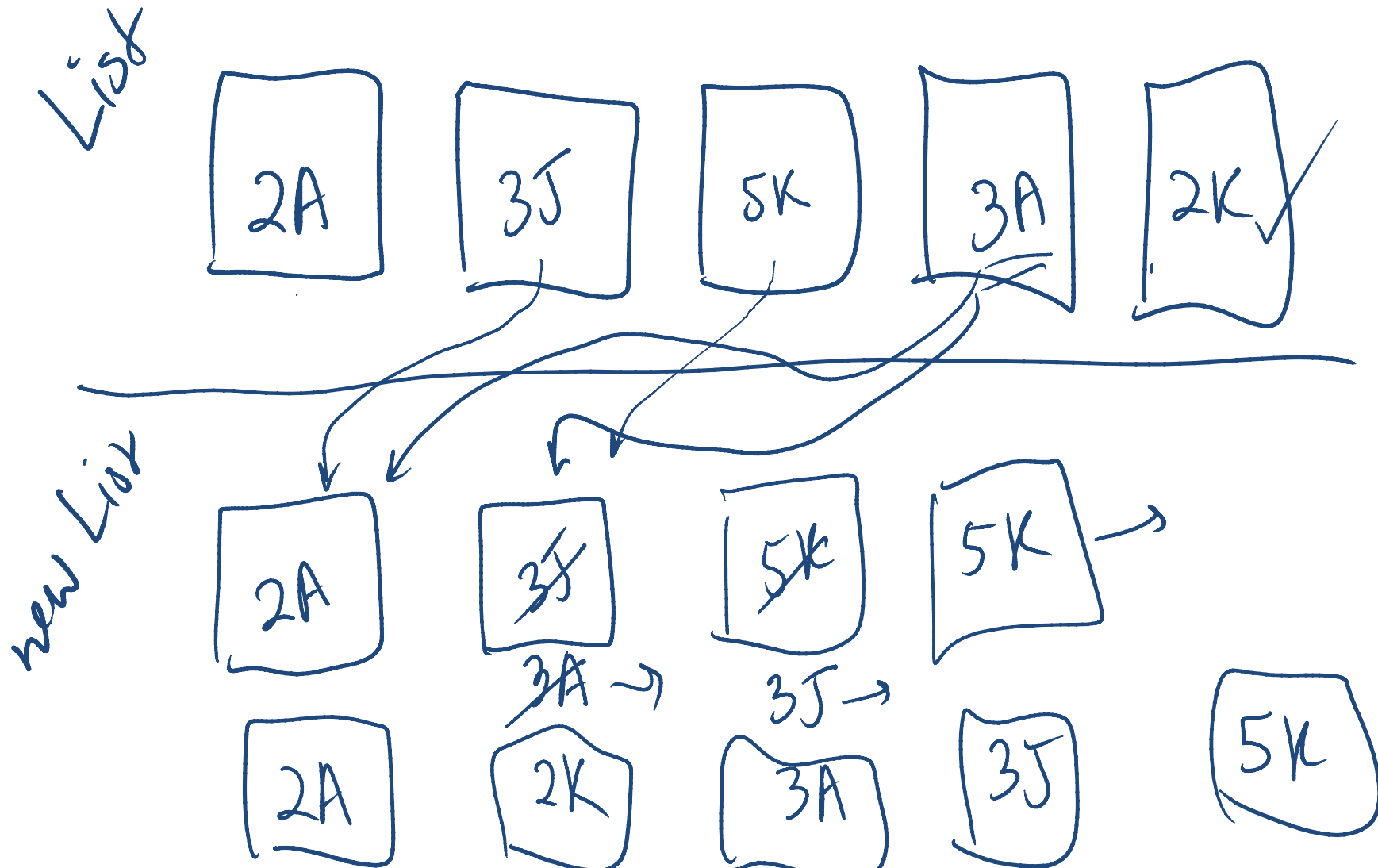


Good Practices of writing code

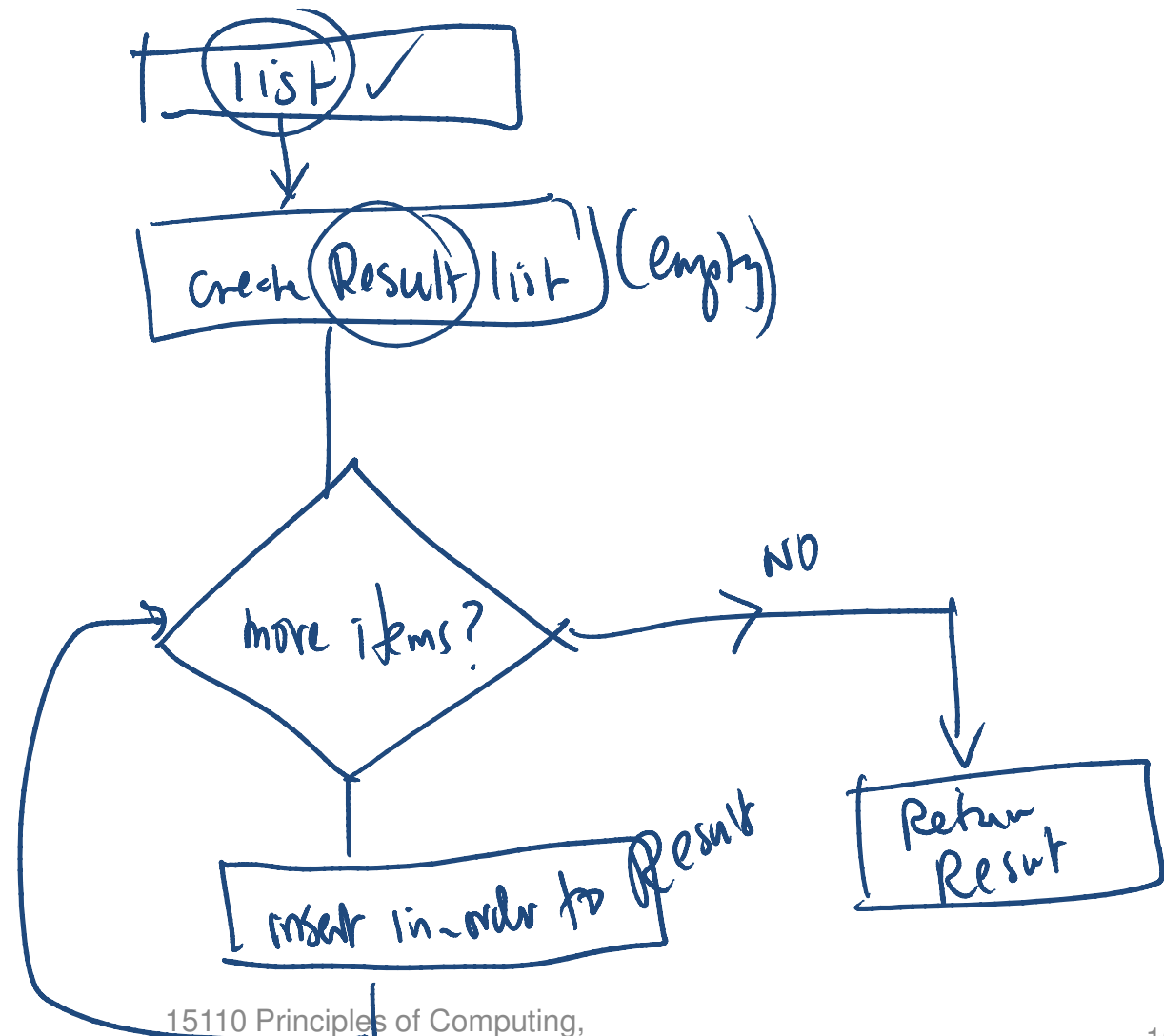
- From high level to detail
- Flow charting to help organize
- Helper functions
- testing



Insertion sort demo



To a flow chart



Flow chart to pseudo code

```
def isort(list)  
  -- create an empty newlist  
  -- for value in list  
    -- insert value in-order to newlist  
  -- end  
  return newlist  
end
```

From pseudo code to partial code

```
def isort (list) empty  
    result = [ ] ✓  
    for val in list do  
        # insert val in its proper place in result  
    end  
    return result  
end
```

Need a way to insert something into an array

Array Insert method

- `list.insert(position, value)`

```
>> a = [10, 20, 30]
```

```
=> [10, 20, 30] ✓
```

```
>> a.insert(0, "foo")
```

```
=> ["foo", 10, 20, 30]
```

```
>> a.insert(2, "bar")
```

```
=> ["foo", 10, "bar", 20, 30]
```

```
>> a.insert(5, "baz")
```

```
=> ["foo", 10, "bar", 20, 30, "baz"]
```

Insertion Sort, Refined

```
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 do
```

```
    index = index + 1
```

```
end
```

```
    return index
```

```
end
```

Tracing length = 3

list	item	index	list[index]
[2, 5, 7]	3	0	2
		1	5

exit

Testing gindex

$$n = 10^9$$

$$(10^9)^2 = 10^{18}$$

```
>> a = [10, 20, 30, 40, 50]
```

```
=> [10, 20, 30, 40, 50]
```

```
>> gindex(a, 3)
```

```
=> 0
```

```
>> gindex(a, 14)
```

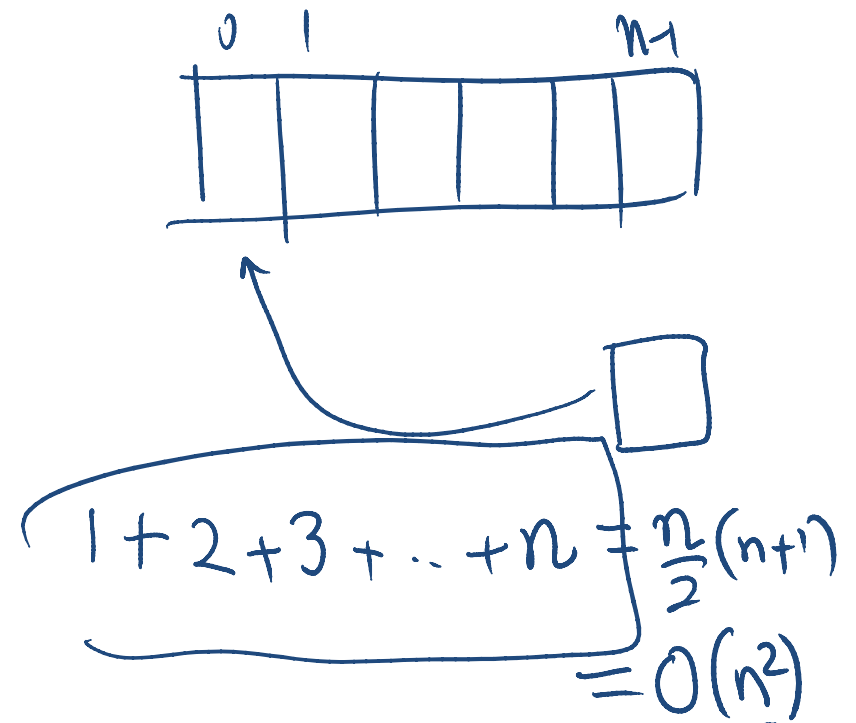
```
=> 1
```

```
>> 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
```

main

function

Tracing the code

```
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
```

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.

Insertion sort

Sorted subarray



Insertion Sort in Ruby

```
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

Loop invariants

```
def isort!(list)
  i = 1
  while i != list.length do
    -- what is true about list here?
        move_left(list, i)
        i = i + 1
    end
  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

Review/Bonus slides

iterators

Iterators

- Iterators are another way to operate on the elements of an array.
- The `each` iterator is similar to a for loop:

```
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:

```
for f in fruits do  
  puts "Yummy " + f  
end
```

Using an iterator:

```
fruits.each { |f|  
  puts "Yummy " + f  
}
```

“Destructive” Iterators

- Some iterators *modify* the array. Beware!

```
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

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

Tracing Code

```
def printtable(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

```
for var in values do  
    loop body stuff  
    more loop body stuff  
    even mode loop body stuff  
end
```

Indenting a WHILE Loop

while *test* **do**

loop body stuff

more loop body stuff

even more 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
```