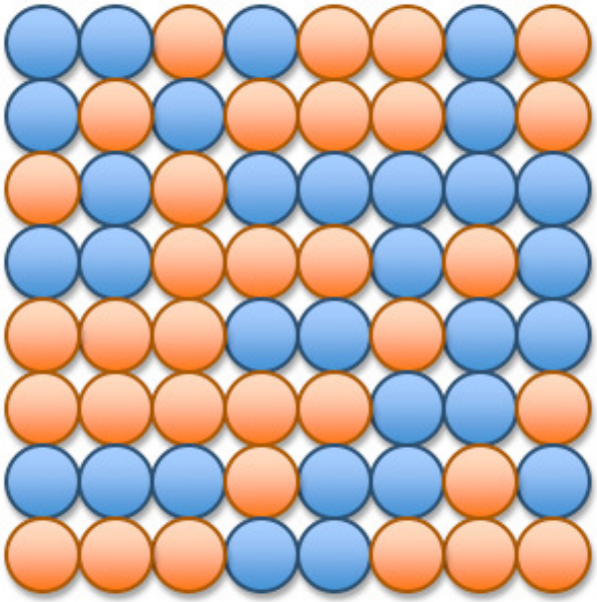


UNIT 4B
Iteration: Sorting

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
  - Ascii table

"aaa" < "baB"

97 98

5, 2, 1, 12, 23

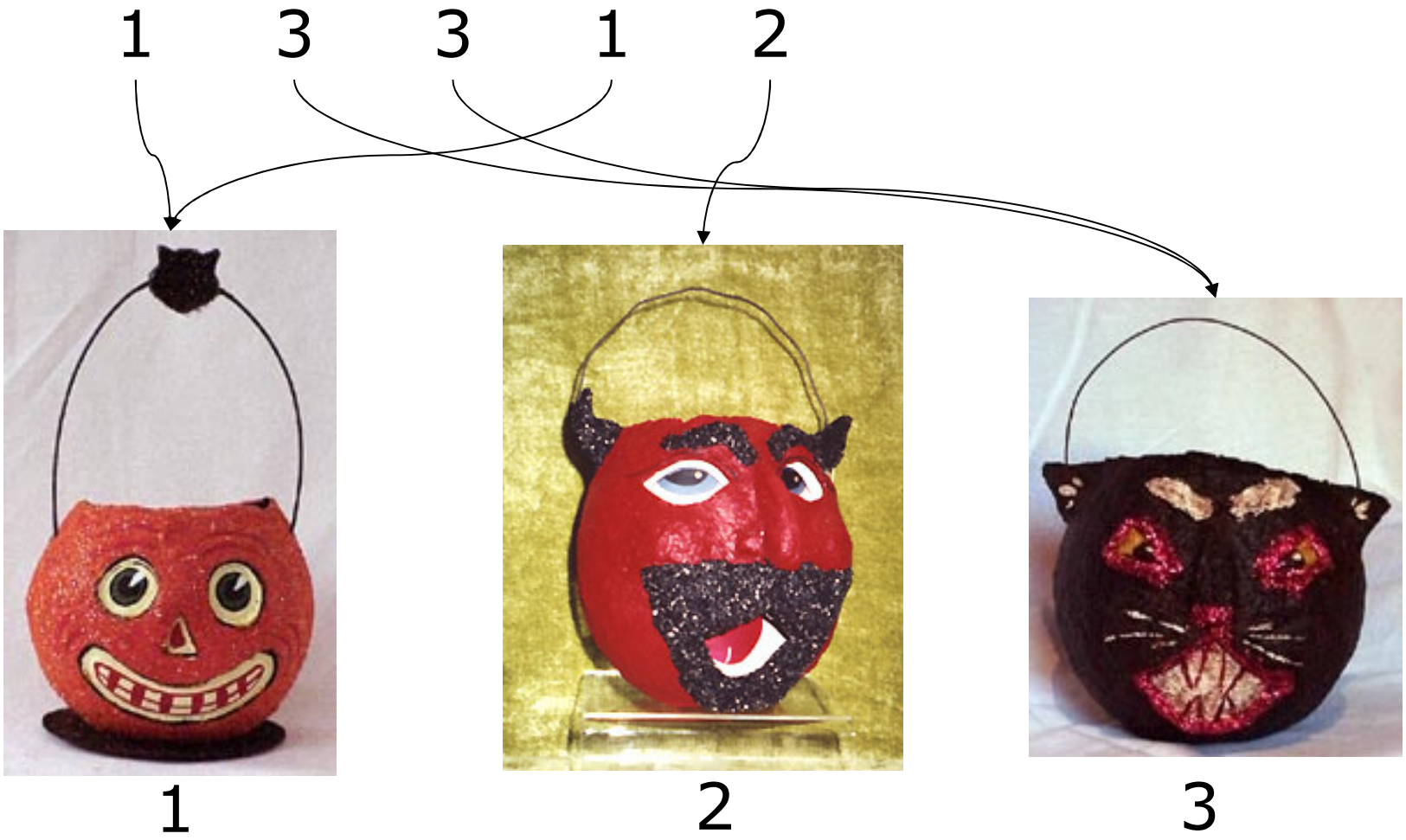
"a" < "b" < "c"

"B" < "a"

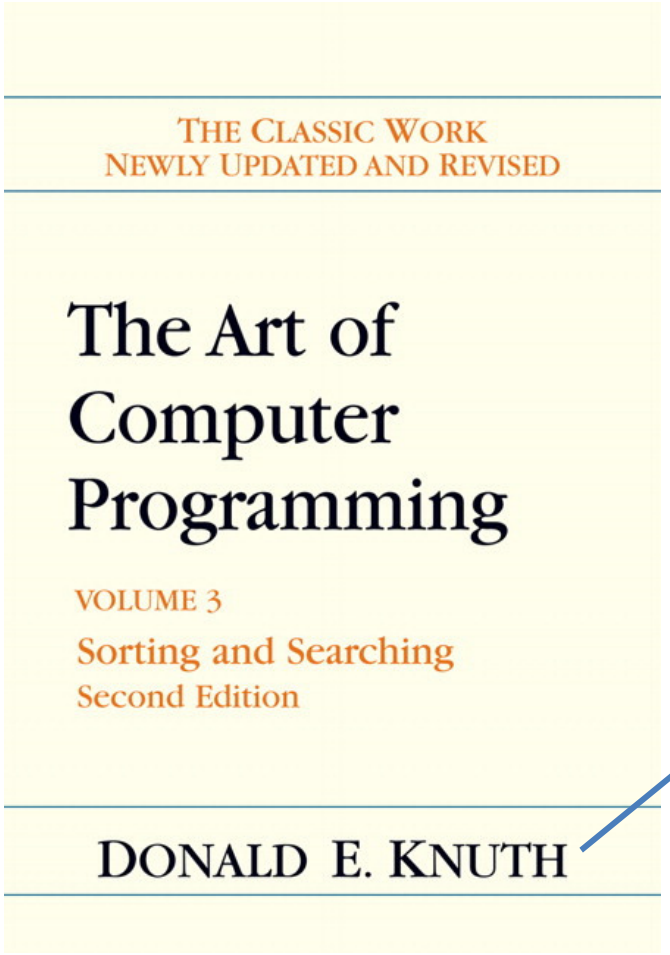
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

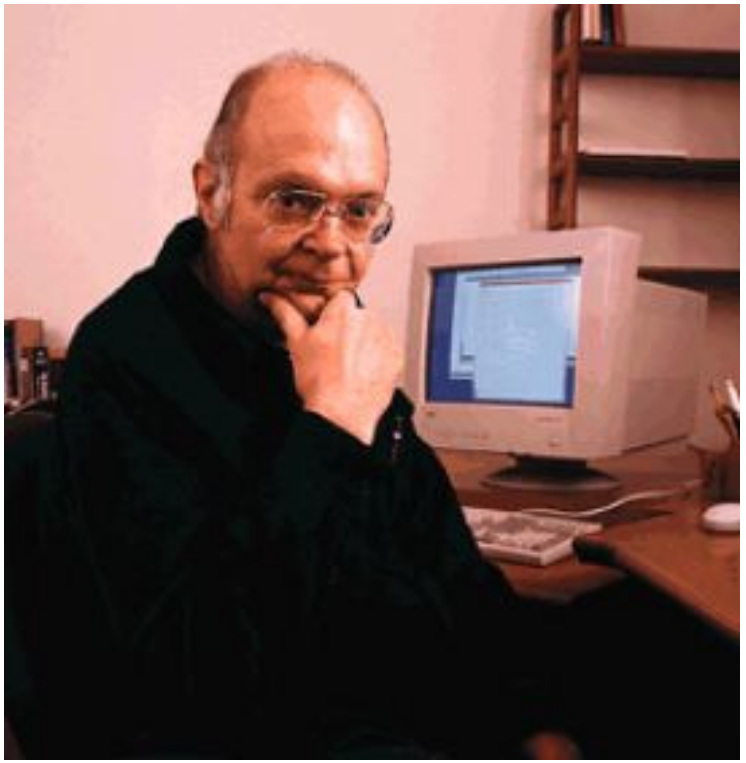
Source: www.LookupTables.com

Category Sorting



The Art of Computer Programming Volume 3: Sorting and Searching

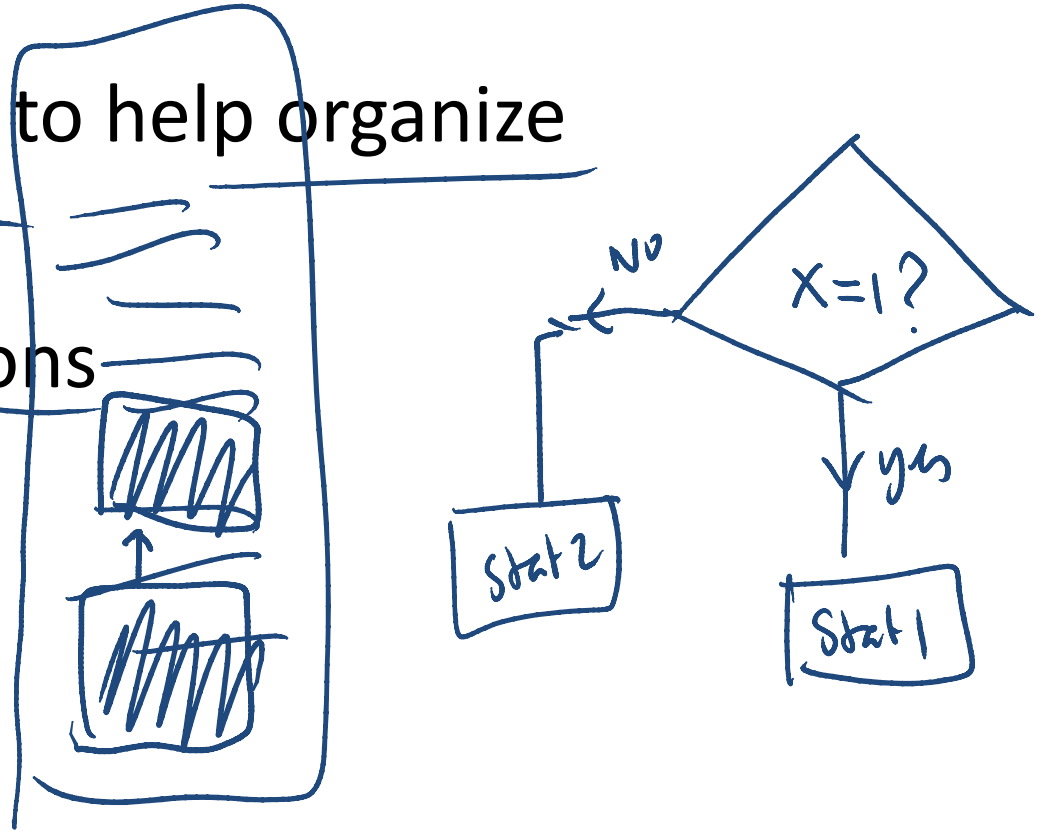




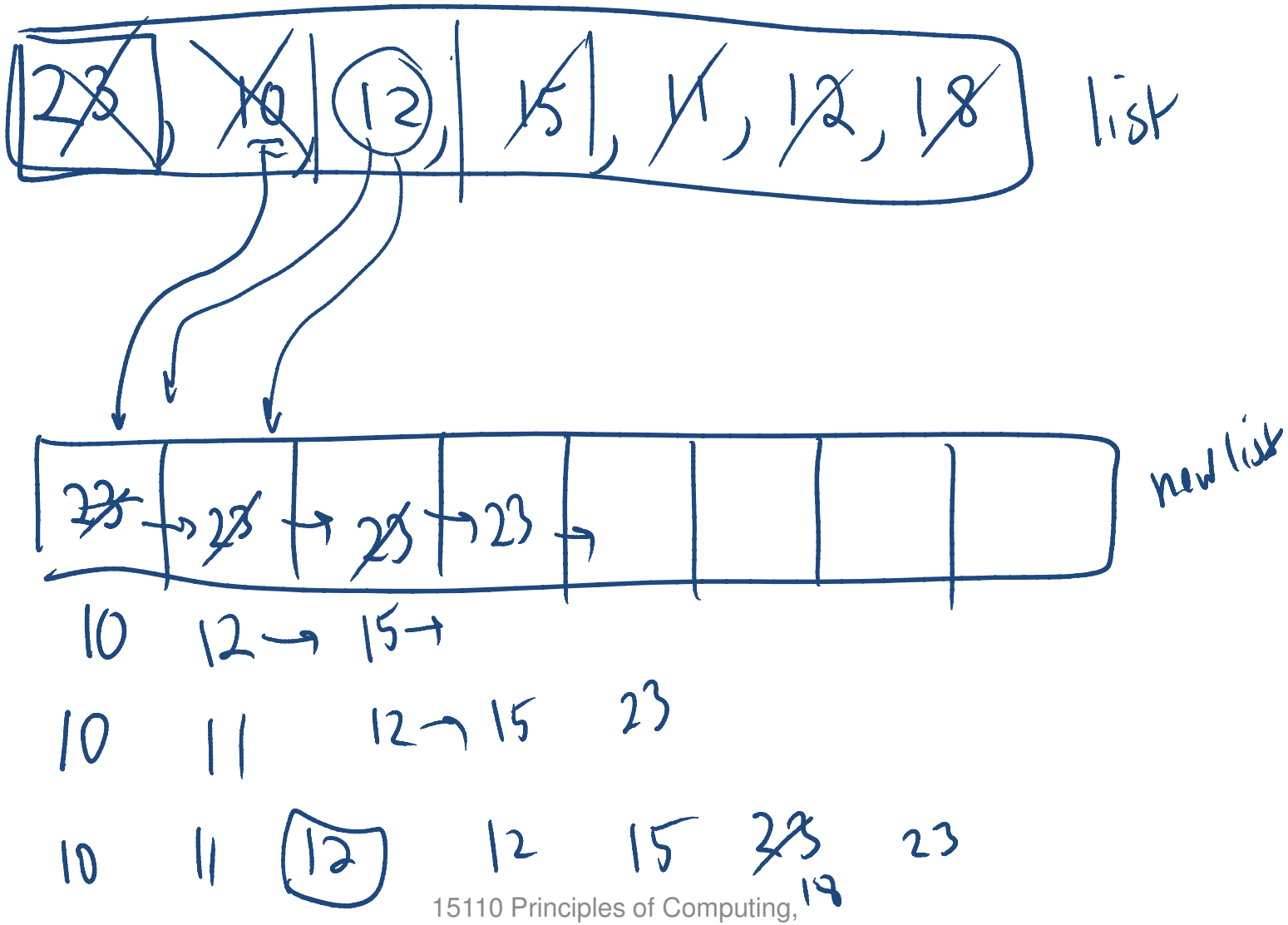
Insertion Sort

Good Practices of writing code

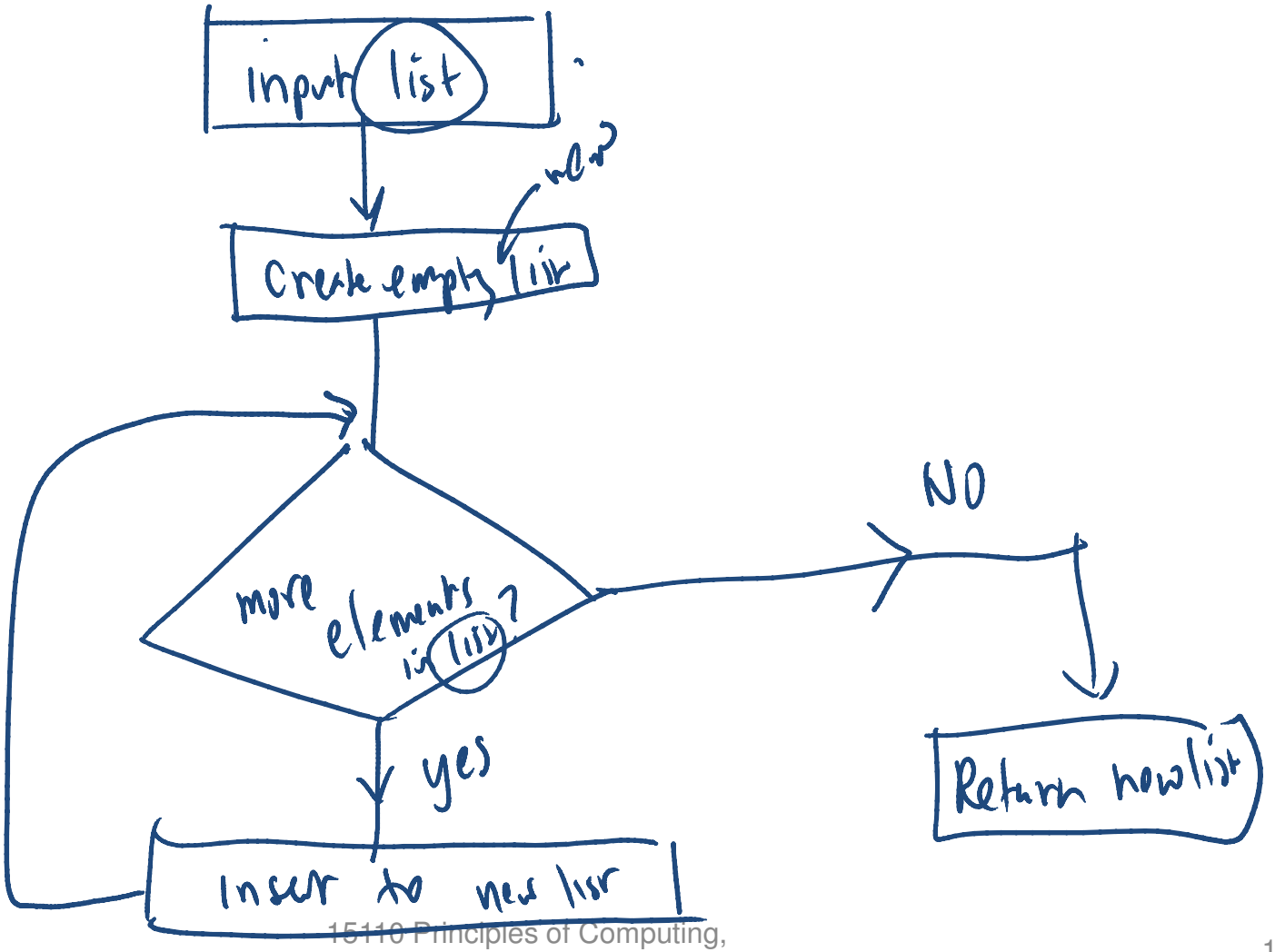
- From high level to detail
- Flow charting to help organize
- Helper functions
- testing



Insertion sort demo



To a flow chart



Flow chart to pseudo code

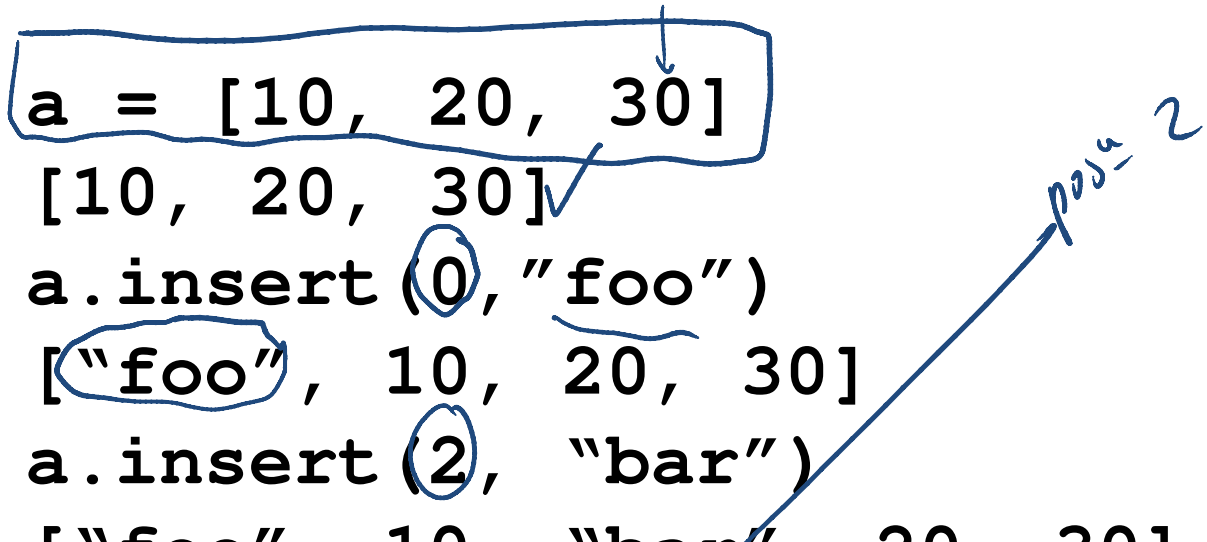
```
def isort(list)
    -- create an empty newlist
    -- for value in list
    --     insert value in-order to newlist
    -- end
end
```

From pseudo code to partial code

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"]
```

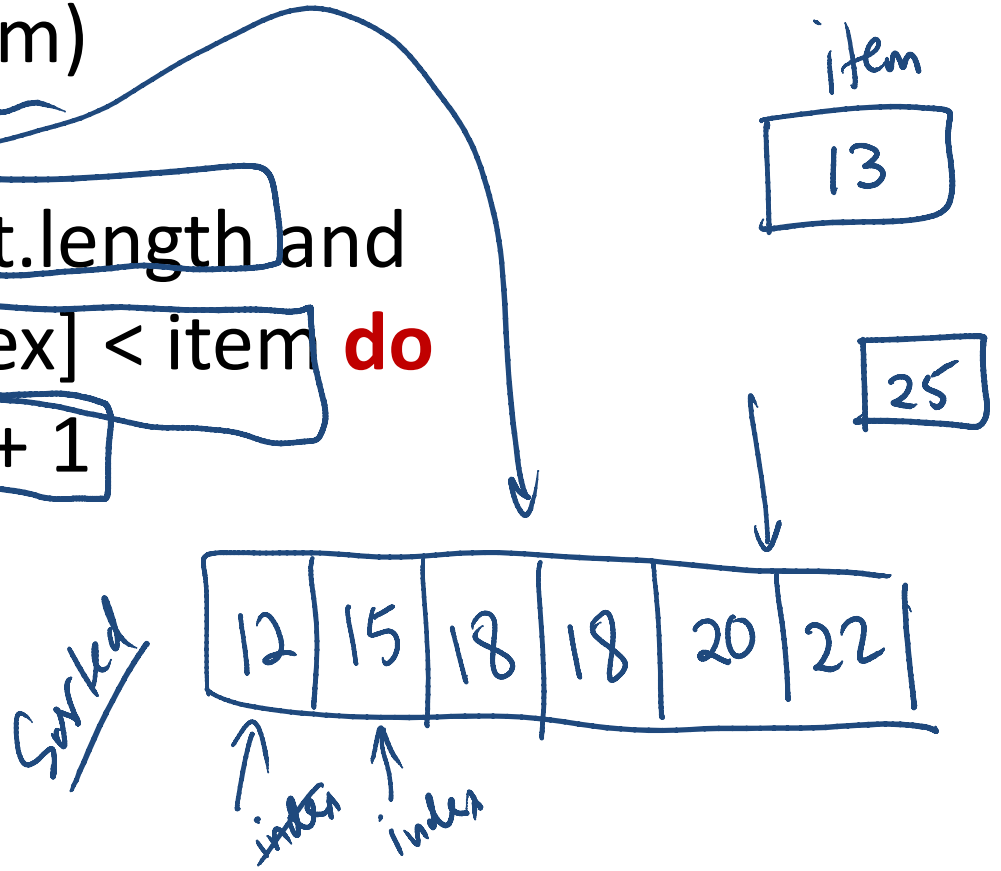


Insertion Sort, Refined

```
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
```



Testing gindex

```
>> 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
```

Tracing the code

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

list	result	val	place
[2,1,3]	[ ]	2	0
	[2]	1	0
	[1,2]	3	2
	[1,2,3]		

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

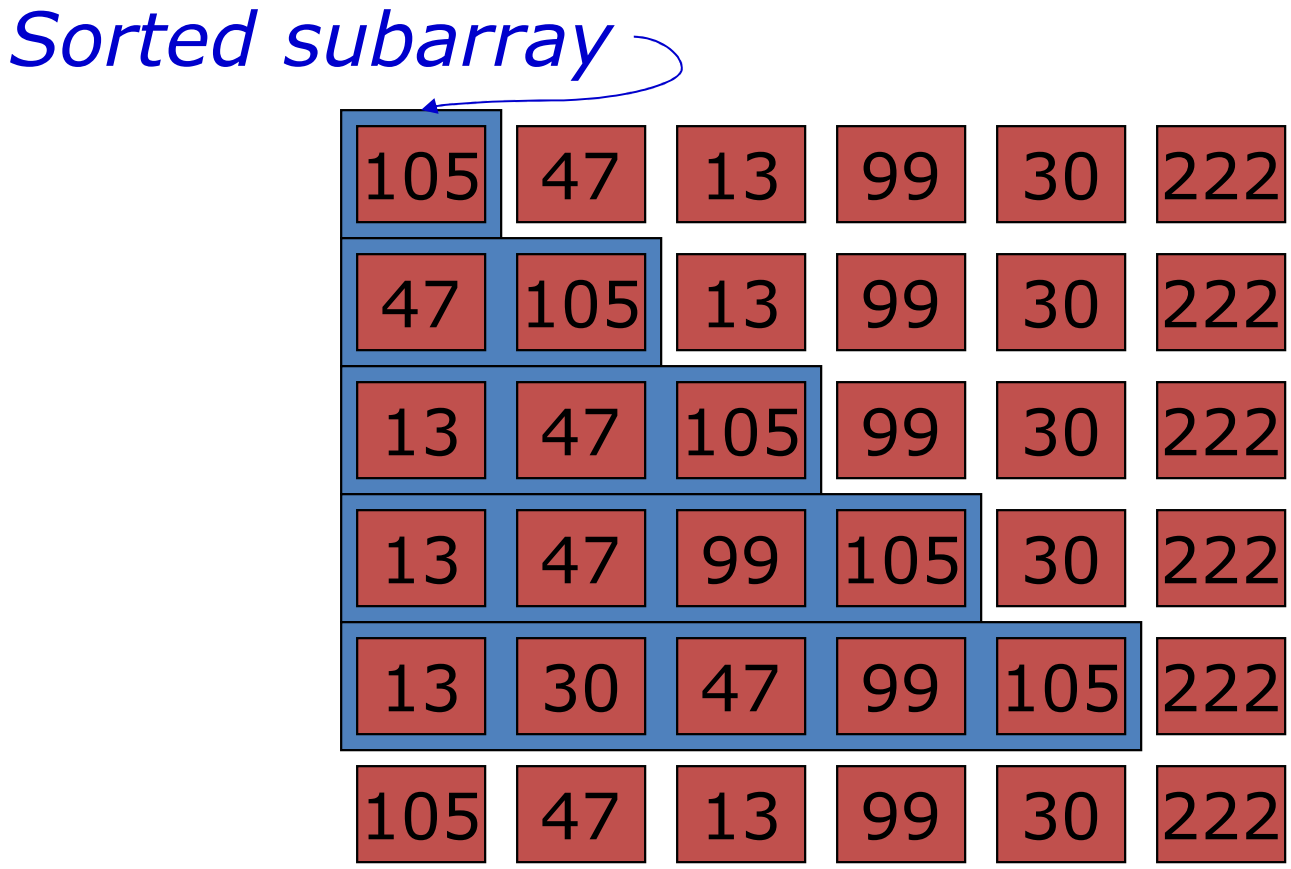
Given an array a of length n, n > 0.

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



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
      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 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
```