METU Computer Engineering

Ceng 111 – Fall 2021
Week 11b

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.

Another example: insert an item into ordered list

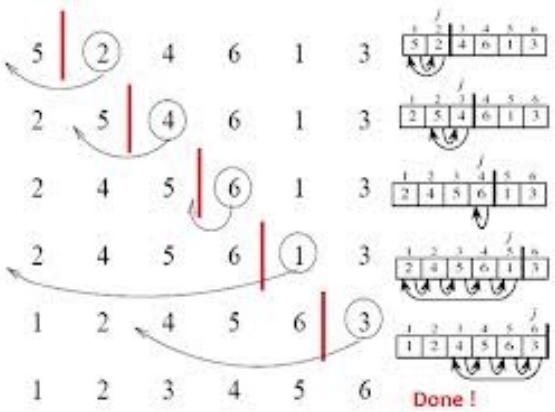
Previously on CENG111!

METU Computer Engineering

```
def insert(x, L):
    if len(L) == 0:
        return [x]
    if x < head(L):
        return [x] + L
    else:
        return [head(L)] + insert(x, tail(L))
```

Previously on CENG 111!

Another example: insertion sort




```python
def insert(x, L):
    if len(L) == 0:
        return [x]
    if x < head(L):
        return [x] + L
    else:
        return [head(L)] + insert(x, tail(L))


def insertion_sort(L):
    if len(L) <= 1:
        return L
    else:
        return insert(head(L), insertion_sort(tail(L)))
```

Previously on CENG111!

When to avoid recursion!

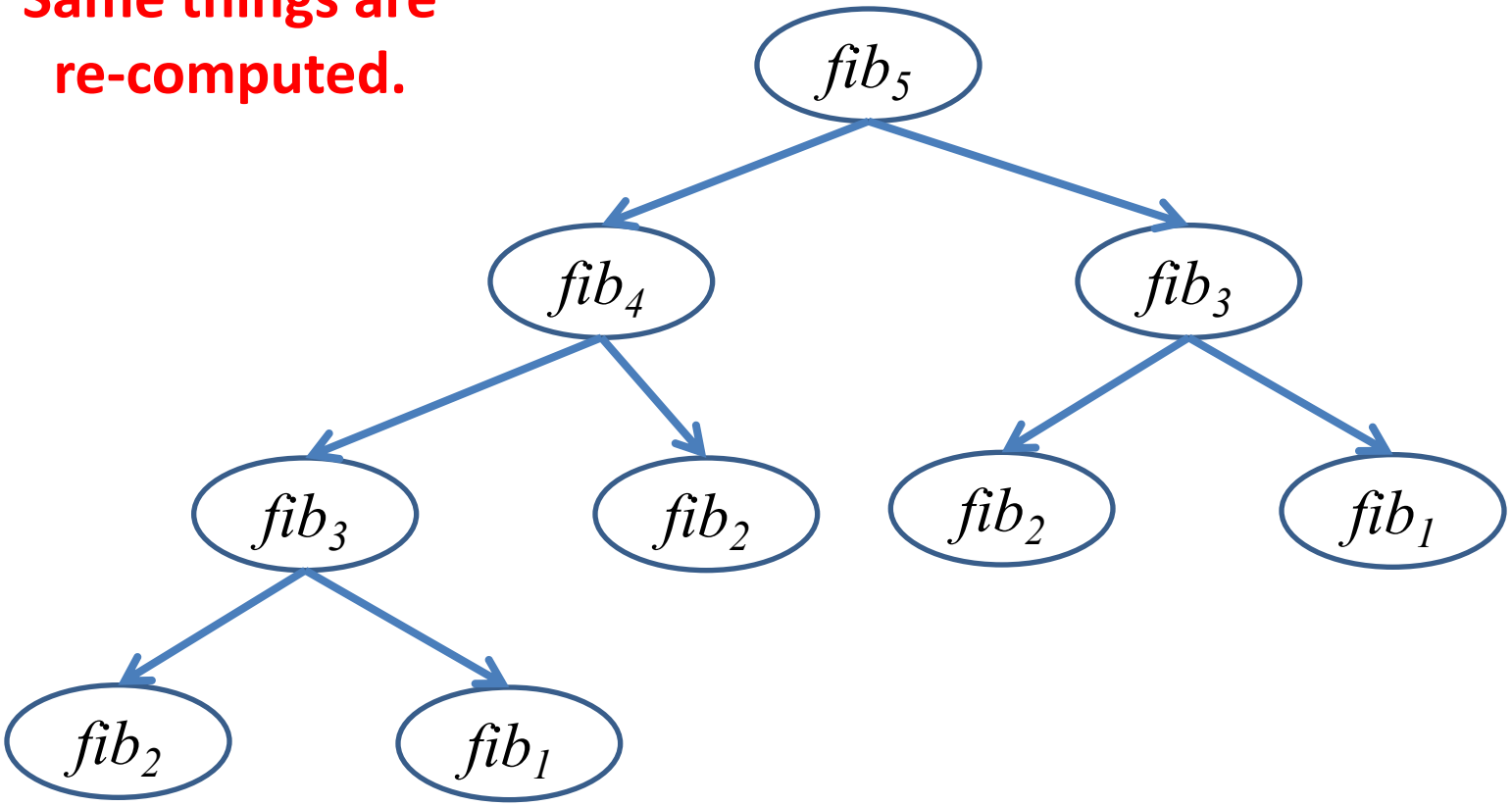
- Example: fibonacci numbers

$$
\begin{aligned}
fib_{1,2} &= 1 \\
fib_n &= fib_{n-1} + fib_{n-2} \quad \ni \quad n > 2
\end{aligned}
$$

define $fibonacci(n)$
  if $n < 3$ **then**
    return 1
  else
    return $fibonacci(n-1) + fibonacci(n-2)$

So, what is the problem with the recursive definition?

Same things are re-computed.



Alternatives to the naïve version of recursive fibonacci - 1

Previously on CENG111

METU Computer Engineering

- Store intermediate results:

```
def fib(n):
    results = [-1]*(n+1)
    results[0] = 0
    results[1] = 1
    return recursive_fib(results, n)

def recursive_fib(results, n):
    if results[n] < 0:
        results[n] = recursive_fib(results, n-1)+recursive_fib(results,n-2)
    else:
        print "using previous result"
    return results[n]
```

```
>>> fib(6)
using previous result
using previous result
using previous result
using previous result
using previous result
using previous result
```

Alternatives to the naïve version of recursive fibonacci - 2

Previously on CENG111

- Go bottom to top:
  - Accumulate values on the way

```
def fib(n):
    if n == 0 or n == 1: return n
    return fib_recursive(2, n, 1, 0)


def fib_recursive(i, n, fib_prev, fib_prev_prev):
    if i == n:
        return fib_prev + fib_prev_prev
    else:
        return fib_recursive(i+1, n, fib_prev+fib_prev_prev, fib_prev)
```

Today

- Recursion
- Iteration

Administrative Notes

- Final:
  - 5 Feb December, Saturday, 13:30

Consider these two implementations:

```
def fact1(n):
    if n == 0: return 1
    else: return n * fact1(n-1)
```

- The second implementation uses "**tail recursion**".
- tail recursion → the result of the **called function** is not used by the **calling function.**

```
def fact2(n):
    return fact_helper(n, 1)

def fact_helper(n, r):
    if n == 0: return r
    return fact_helper(n-1, r*n)
```

Tail recursion & iteration

```
def fact2(n):
    return fact_helper(n, 1)

def fact_helper(n, r):
    if n == 0: return r
    return fact_helper(n-1, r*n)
```

- Then, we can implement the tail-recursion version like on the right.

while n != 0

r = n * r

n = n - 1

Iteration

- More properly,

r = 1

while n != 0

    r = n * r

    n = n - 1

```
def fact2(n):
    return fact_helper(n, 1)

def fact_helper(n, r):
    if n == 0: return r
    return fact_helper(n-1, r*n)
```

Iteration in Python

- while statement

```
while <condition> :
        <statements>
```

- Example:

```
L = [2, 4, -10, "c"]
i = 0
while i < len(L) :
        print L[i], "@"
        i += 1
```

```
2 @
4 @
-10 @
c @
```

Iteration in Python

- for statement:

```
for <var> in <list> :
        <statements>
```

- Example:

```
for x in [2, 4, -10, "c"]:
        print x, "@"
```

```
2 @
4 @
-10 @
c @
```

Examples for Iteration

- Searching an item in a list

```
def seq_search(x, L):
    for y in L:
        if x == y: return True
```

vs.

```
def seq_search(x, L):
    i = 0
    length = len(L)
    while i < length:
        if x == L[i]: return True
        i += 1
    return False
```

Nested Loops in Python

- You can put one loop within another one
  - No limit on nesting level

```
for i in range(1,10):
        print i, ":",
        for j in range(1,i):
                print j, "-",
        print ""
```

```
1 :
2 : 1 -
3 : 1 - 2 -
4 : 1 - 2 - 3 -
5 : 1 - 2 - 3 - 4 -
6 : 1 - 2 - 3 - 4 - 5 -
7 : 1 - 2 - 3 - 4 - 5 - 6 -
8 : 1 - 2 - 3 - 4 - 5 - 6 - 7 -
9 : 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 -
```

Break statements

```
while <cond-1> :
    <statements-before-break>
    if <cond-2> :
        break
    <statements-after-break>
<statements-after-while>
```

Execution Jumps

```
for <var> in <list> :
    <statements-before-break>
    if <cond> :
        break
    <statements-after-break>
<statements-after-for>
```

Execution Jumps

“break” example

```
x = 4
List = [1, 4, -2, 3, 8]
for m in List:
        print m
        if m == x:
                print "I have found a match"
                break
```

Continue statements

```
while <cond-1> :
    <statements-before-continue>
    if <cond-2> :
        continue
    <statements-after-continue>

<statements-after-while>
```

Execution Jumps

```
for <var> in <list> :
    <statements-before-continue>
    if <cond> :
        continue
    <statements-after-continue>

<statements-after-for>
```

Execution Jumps

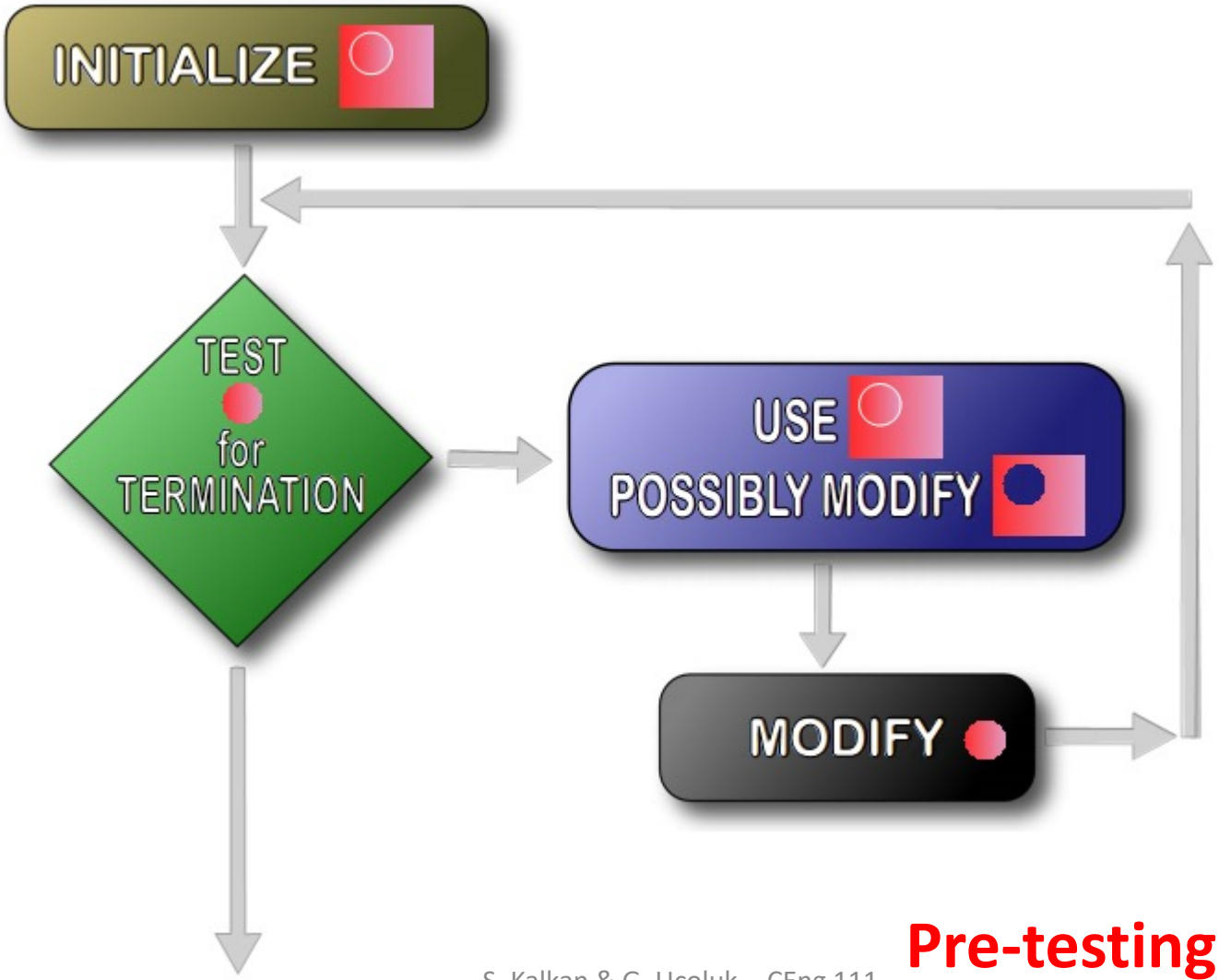
- <var> will point to the next item in the list.

Loops with “else:” parts

- The “else:” part is executed when the loop exits.
- If you use a “break” statement, the “else” part is not executed.

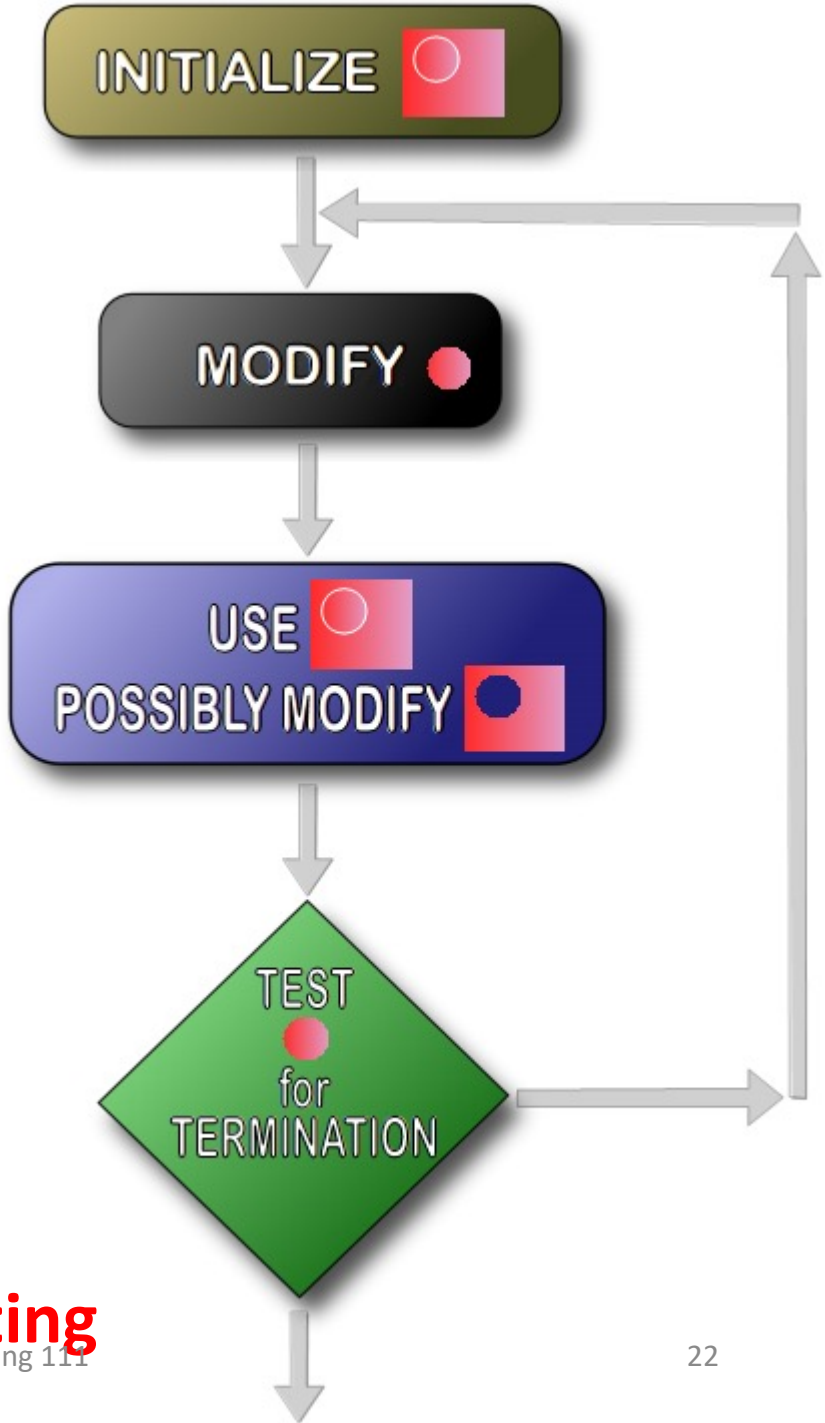
```
while <cond>:
        <statements>
else:
        <else-statements>
```

Types of Iterations



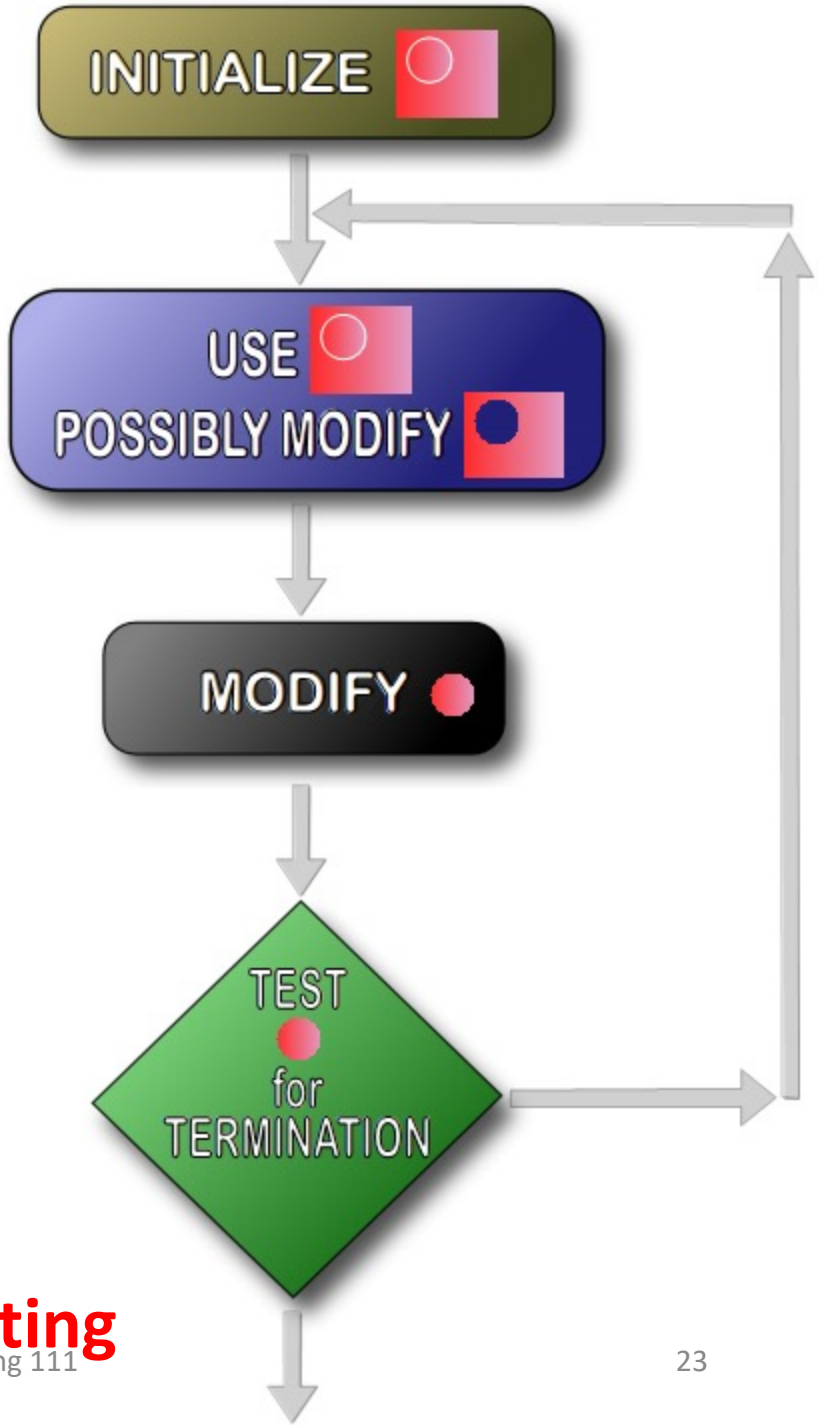
Pre-testing

Types of Iterations



Premod-posttesting

Types of Iterations



Postmod-posttesting

Examples for Iteration

- What does the following do?

```
def f(List):
    length = len(List)
    changed = True
    while changed:
        changed = False
        i = 0
        while i < length-1:
            if List[i] > List[i+1]:
                (List[i], List[i+1]) = (List[i+1], List[i])
                changed = True
            i += 1
    return List
```

Pseudo code for Bubble Sort

Step 1: While changed = True, do the following:

Step 2: - changed ← False

Step 3: - i ← 0

Step 4: - While i is less than length(L)-1, do the following:

Step 5: + if L[i] > L[i+1], then

Step 6: - swap L[i] and L[i+1]

Step 7: - changed ← True

Step 8: + Increment i by 1

Another Example for Iteration

- Naïve selection sort

```
def select_min(L):
# Find, remove and return min
    Index = 0
    Min = L[Index]
    for i, x in enumerate(L):
        if x < Min:
            Index = i
            Min = x

    L.pop(Index)
    return Min

def naive_selection_sort(L):
    Result = [0]*len(L)
    for i in range(0, len(L)):
        x = select_min(L)
        Result[i] = x

    return Result
```

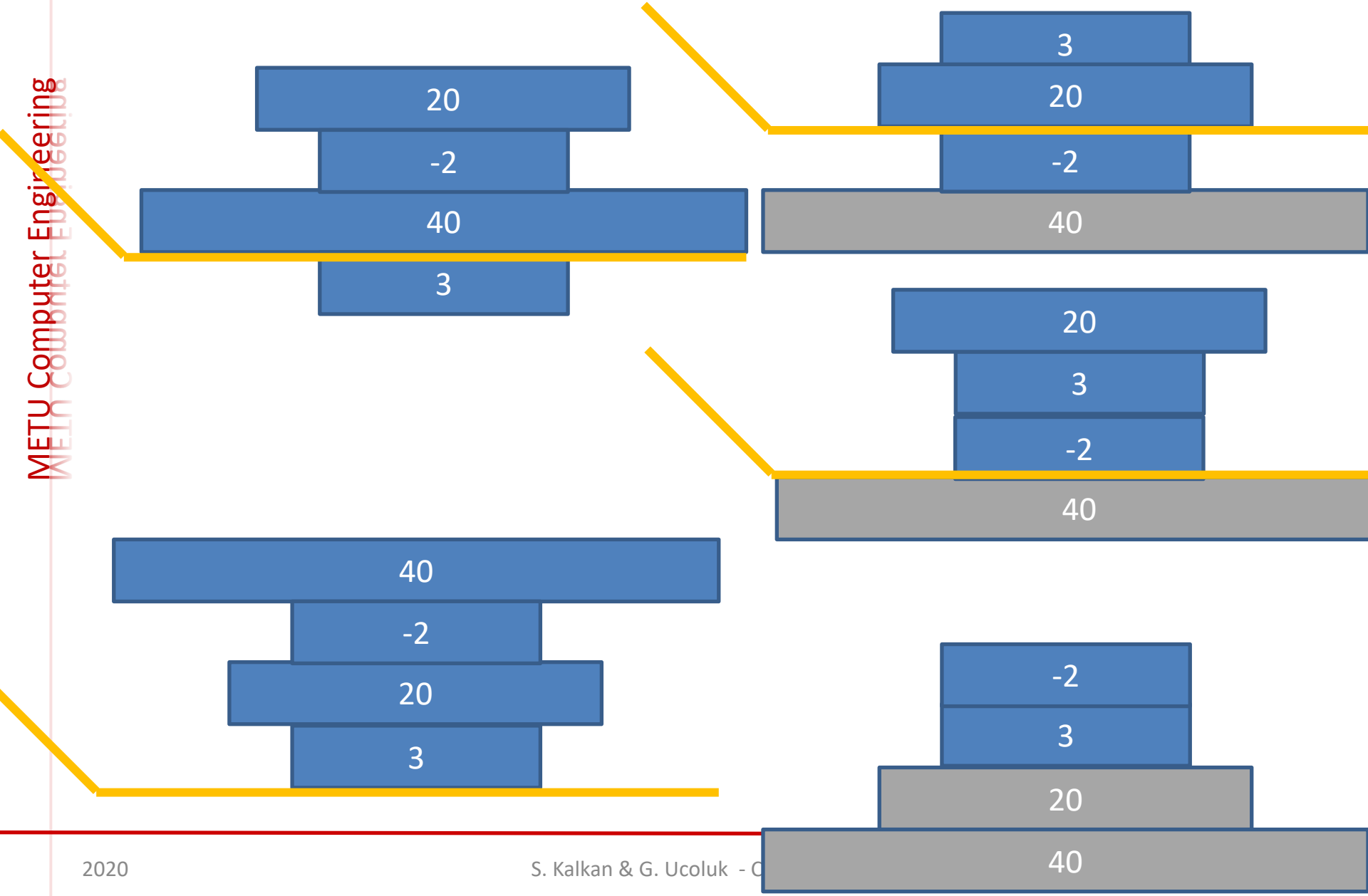
Another Example for Iteration

- A more efficient version of selection sort

```
def selection_sort(L):
    length = len(L)
    i = 0
    while i < length-1:
        j = L.index(min(L[i:]))
        (L[i], L[j]) = (L[j], L[i])
        i += 1
```

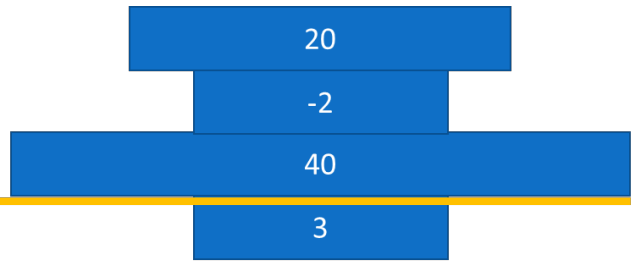
Exercise: Implement L.index(min(L[i:])) as a function

More examples for iteration: Pancake Sort



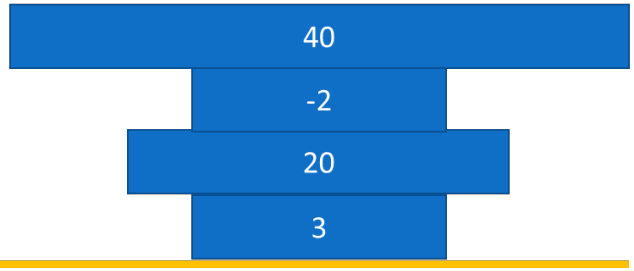
More examples for iteration: Pancake Sort

METU Computer Engineering




```
def find_max(L, s, e):
    maks = L[s]
    maks_ind = s
    for i in range(s, e):
        if L[i] > maks:
            maks_ind = i
            maks = L[i]
    return maks_ind
```

The lowest pancake is at the end of the list




```
def pancake_sort(L):
    N = len(L)
    for i in range(0, N):
        max_ind = find_max(L, 0, N-i)
        L[:max_ind+1] = L[max_ind::-1]
        L[:N-i] = L[N-i-1::-1]
    return L
```

More examples for iteration: Counting Sort

METU Computer Engineering

"To count or not to count: That's what counts"

-- a CENG111 proverb ☺

```
def csort(A):
        # Assume that the numbers are in the range 1,...,k
        k = max(A)
        C = [0] * k

        # Count the numbers in A
        for x in A:
                C[x-1] += 1

        # Accumulate the counts in C
        for i in range(1, k):
                C[i] += C[i-1]

        # Place the numbers into correct locations
        B = [0] * len(A)
        for x in A:
                B[C[x-1]-1] = x
                C[x-1] -= 1

        return B
```