# Ceng 111 – Fall 2021
# Week 12a

**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.

# Tail recursion & iteration

Previously on CENG111!

METU Computer Engineering

```
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

Previously on CENG111!

# 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 @
```

METU Computer Engineering

Previously on CENG111!

# Iteration in Python

- for statement:

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

- Example:

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

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

METU Computer Engineering

Previously on CENG111!

# Break statements

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

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

METU Computer Engineering

Previously on CENG111!

# 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.

Previously on CENG111!

# 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
```

METU Computer Engineering

# Another Example for Iteration

Previously on CENG111!!

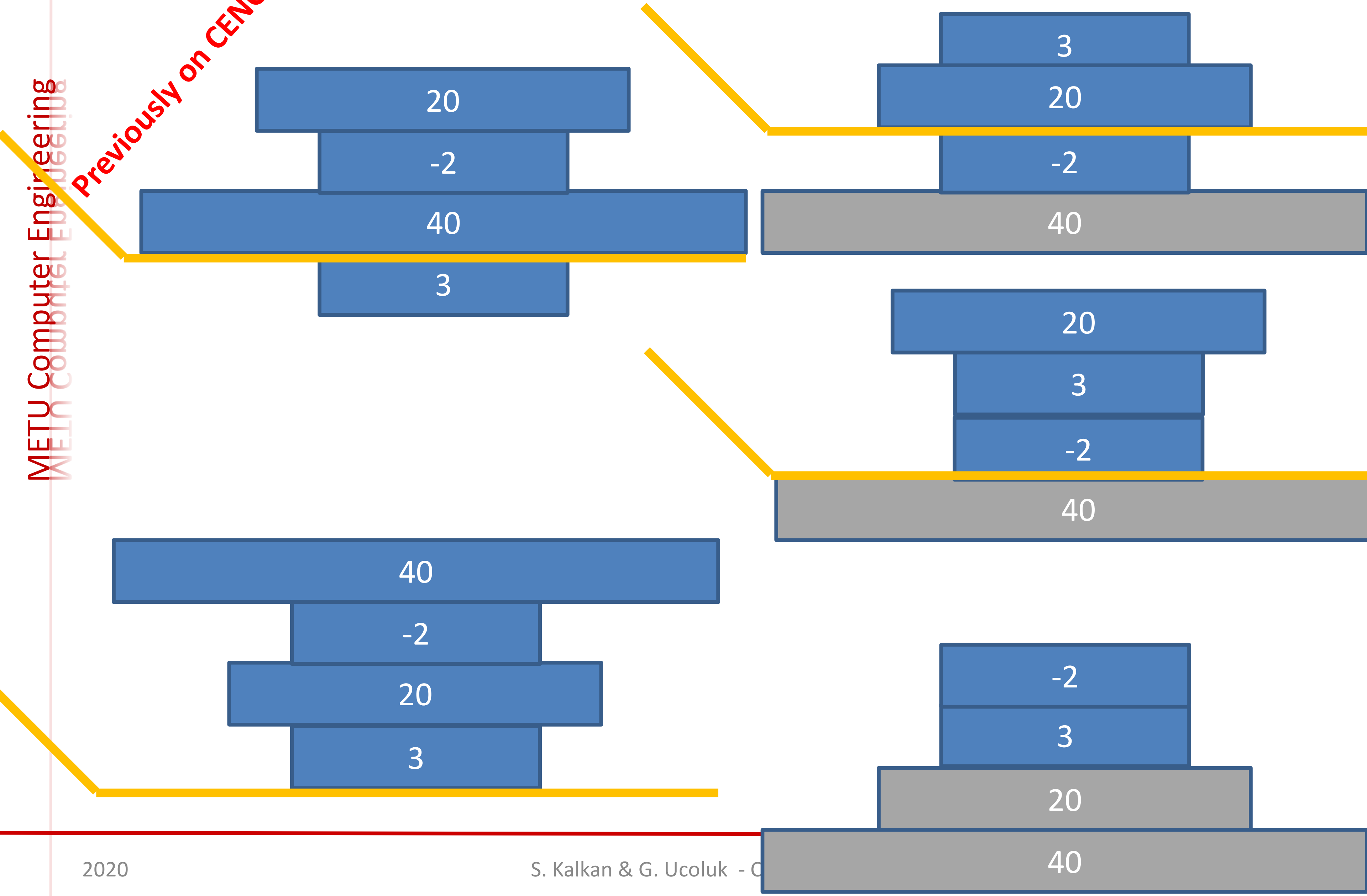
- A more efficient version of selection sort

```python
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

Previously on CENG111!



Previously on CENG111...



```
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
```

METU Computer Engineering

# Today

- Recursion vs. iteration
- Complexity

# Administrative Notes

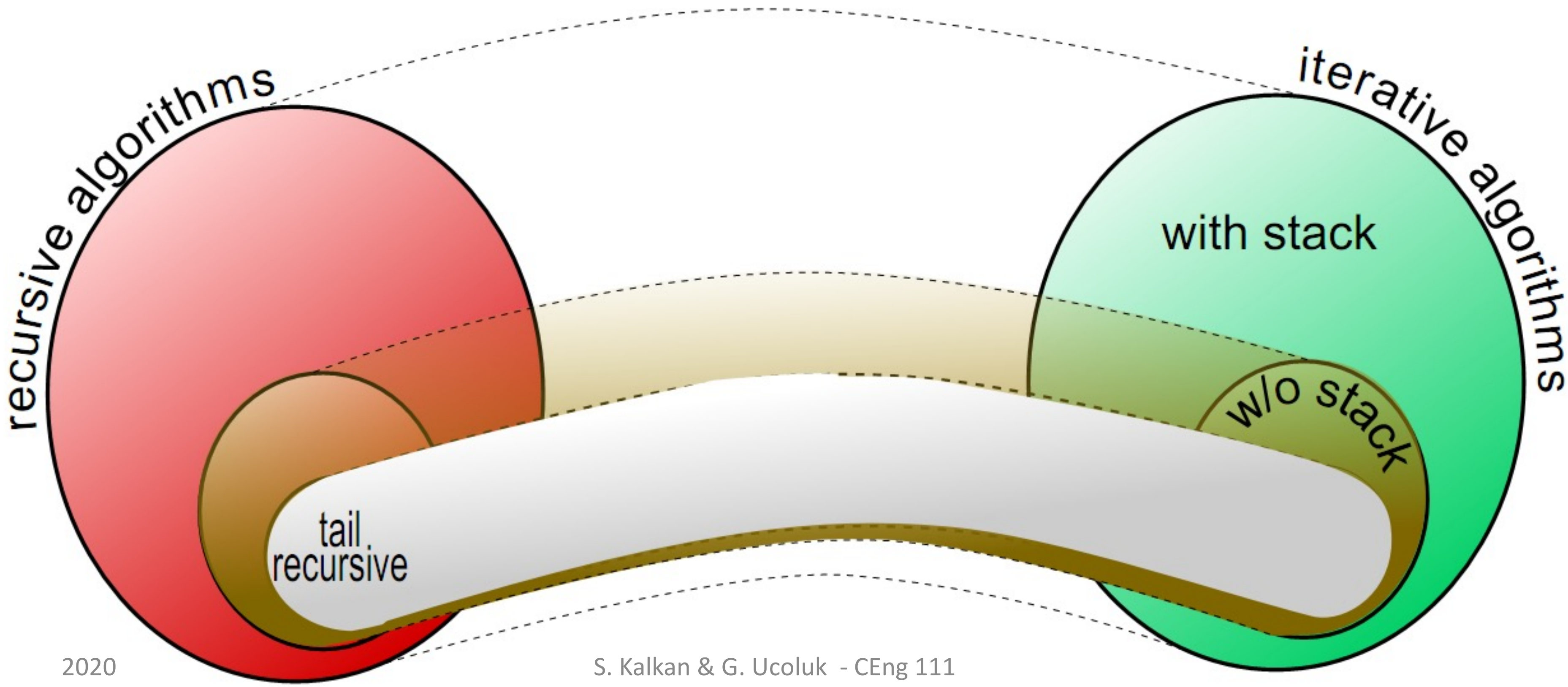


- THE3:
  - Deadline: 16 January.
- Final:
  - 5 Feb December, Saturday, 13:30

# TAIL RECURSION VS. RECURSION VS. ITERATION

# Recursion vs. Iteration

- Any recursive algorithm can be transformed into an iterative algorithm.
- The reverse is also true.

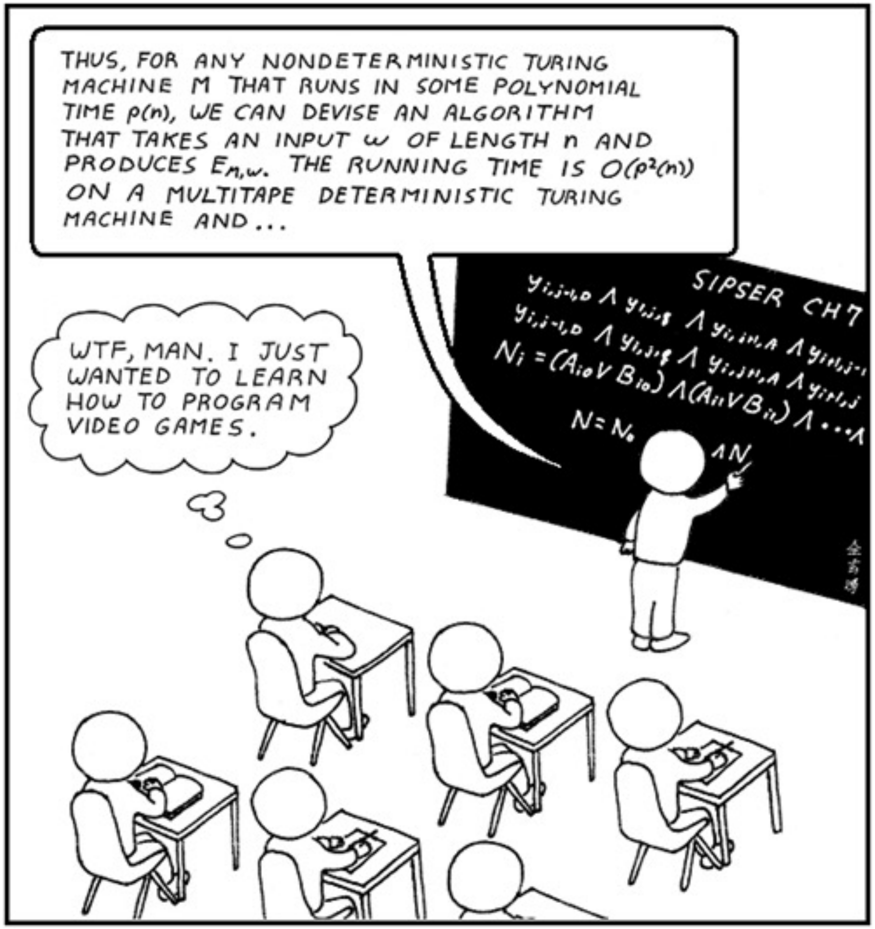


# Recursion vs. Iteration

- Which one is better?
- Better in what way?
  - Resource-wise:
    1. Iteration without stacks
    2. Iteration with stacks
    3. Recursion
  - Implementation-wise:
    1. Recursion
    2. Iteration without stack
    3. Iteration with stack

METU Computer Engineering

# TIME ANALYSIS OF ALGORITHMS



# How do you compare these two algorithms?



## Bubble sort

```
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
```

## Count sort

```
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
    i = 1
    while i < k:
        C[i] += C[i-1] 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
```

# Analyzing Performance of Algorithms

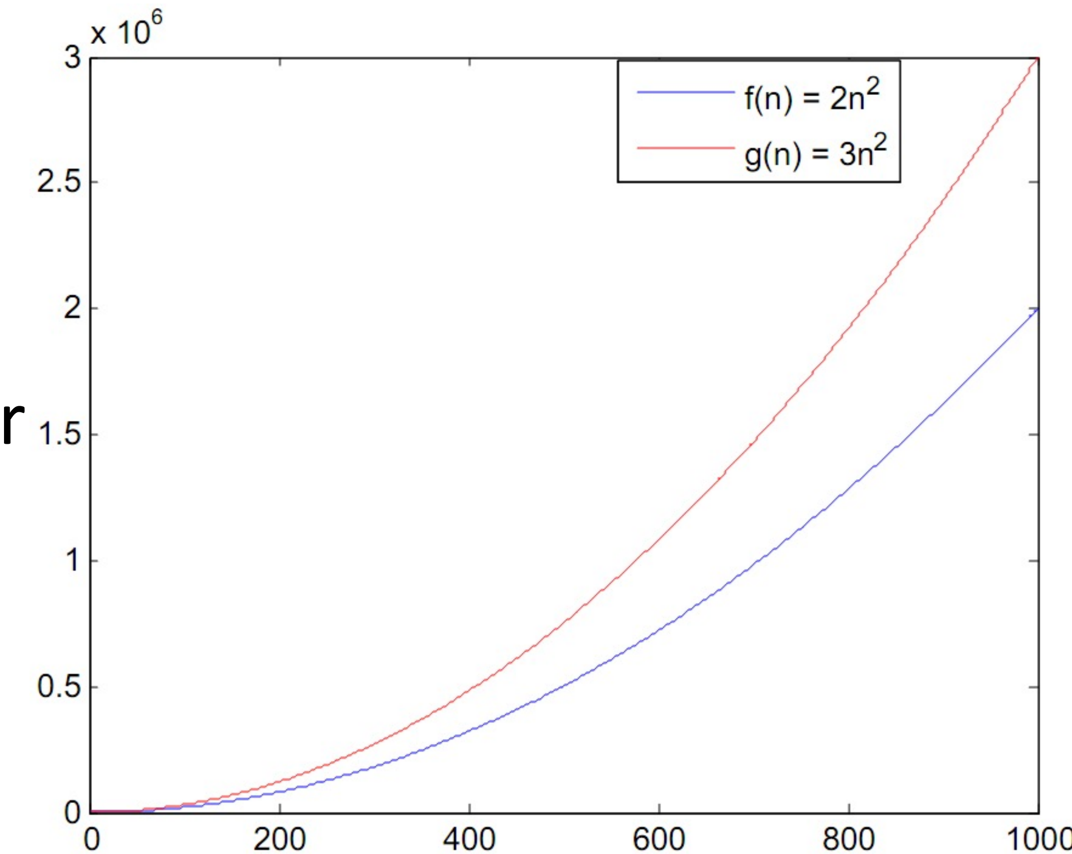


- How do you compare the performance of algorithms?
  1. Implement them and count the time they take?
  2. Count the number of main steps that affect the performance and depend on the size of the data.

# Measuring Complexity



- There are several measures for complexity.
- A measure for complexity is basically a bound for the running time of an algorithm.
- Look at $f(n) = 2n^2$
- $f(n)$ is bounded by $3n^2$



# Measuring complexity



- Consider again $f(n) = 2n^2$
- There are several functions that can bound f(n):
  1. $3n^2$, $4n^2$, $6n^2$, …
  2. $n^3$, $2n^3$, $3n^3$, …
  3. $n^4$, $2n^4$, $3n^4$, …
  4. …
  5. …
- In computational complexity, we are interested in the most "suitable" bounding function.

# Big-O Notation; O()

f(n) is O(g(n)) if and only if there exists a real constant c>0, and a positive integer $n_0$, such that $|f(n)| \leq c|g(n)|$ for all $n \geq n_0$

- Example: for $f(n) = 2n^2$, $g(n) = n^2$.
  - f(n) is O(g(n)) = $O(n^2)$
  - But, it is also $O(n^3)$ and $O(n^4)$
  - We *prefer* the *smallest*.
  - For example:

$$f(n) = 9\log n + 5(\log n)^3 + 3n^2 + 2n^3 \in O(n^3).$$