REMINDERS

• Assignment 7 due Wednesday, November 14th, at 10:00 AM
RUNTIME REVIEW

• Look at the “worst case” scenario (i.e. longest runtime)
• Assume function works (i.e. will not return an error when you run it)
• Based on the assumption learned in class (and in the modules)
RUNTIME REVIEW

• O(1) – Constant
  – does not depend on the size of the input
  – For numbers:
    • Numeric operations: +, *, /, -, %, //
    • max, min
  – For list L:
    • L[0], len(L)...
    • L.append(4)...

• O(n) – Linear
  – depends on the size of the input
  – For list L:
    • L[1:], max(L), L + L, sum(L), L.remove(0)...
    • list(map(lambda x: x+1, L))
RUNTIME REVIEW

• $O(n^2)$ – Quadratic
  – time proportional to square of size of the input
  – Be careful of abstract functions:
    • `list(map(lambda k: list(range(k)), list(range(n))))`

• $O(2^n)$ – Exponential
  – As size of input increases by 1, the run time doubles
  – example: Module 5, Slide 15: fib
RUNTIME EXAMPLE 1

# Let n = len(L)
def fn(L):
    if L==[]:
        return 0
    else:
        return 1 + fn(L[1:])

Count steps for:
• Compare L with []
• Calculate L[1:]
• Call fn recursively on a list of length n-1
• Add 1 to the recursive call of fn

• T(n) = O(n) + T(n-1)
# Let n = len(L)
def fn(L):
    ans = []
    for x in L:
        if x[0] == 'A':
            ans.append(x)
    return ans

Count steps for:
• Assign [] to ans
• Loop:
  – Number of Iterations
  – Asymptotic run time of the body of loop:
    • Check if x[0] == 'A'
    • ans.append(x)
• Return ans
# Let n = len(L)
def fn(L):
    L1 = L[0::2]
    if L==[]:
        return []
    else:
        return fn(L1)

Count steps for:

- L1 = L[0::2]
- Compare L with []
- Call fn recursively on a list of length n/2
- T(n) = O(n) + T(n/2)
# Q5

def fn(n):
    if n % 2 == 0:
        return "outcome1"
    elif n % 3 == 0:
        return "outcome2"
    elif n % 5 == 0:
        return "outcome3"
    else:
        return "outcome4"

Count steps for:

- Calculate n%2
- Compare it with 0
- Calculate n%3
- Compare it with 0
- Calculate n%5
- Compare it with 0
- Return the answer
QUESTION 2 - QUICKSORT

Consider a different way of sorting a list $L$ of distinct integers:

- Let $x$ be the first element of the list
- Let $lst1$ be all the elements in the list smaller than $n$
- Let $lst2$ be all the elements in the list larger than $n$
- Recursively quicksort $lst1$ and $lst2$

- $lst1 + [x] + lst2$

Write a function `quicksort` which consumes a list of distinct integers, $lst$, and sorts it using the quicksort algorithm.
EXAMPLE

quicksort([2, 3, 1, 4, 0])

→ quicksort([1, 0]) + [2] + \\n    quicksort([3, 4])

→ (quicksort([0]) + [1]) + [2] + \\n    ([3] + quicksort([4]))

→ ([0] + [1]) + [2] + ([3] + [4])

→ [0, 1] + [2] + [3, 4]

→ [0, 1, 2, 3, 4]
RUNTIME OF QUICKSORT

- **Worst case runtime**:
  - $T(n) = O(n) + T(n-1) \Rightarrow O(n^2)$
  - The list is already sorted

- In practice, quicksort can avoid the worst case most of the time, and, on average, runs on $O(n \log n)$ time.