TUTORIAL 8

O( )

EFFICIENCY, SEARCHING AND SORTING ALGORITHMS
REMINDERS

• Assignment 7 due Friday, March 16\textsuperscript{th}, at 8 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
# 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
• a) Determine the worst-case run-time in terms of n.

```python
def weird(n):
    lst = []
    while n > 0:
        lst.append(n)
        n = n//2
    return lst
```

A. $O(1)$  
B. $O(n)$  
C. $O(n^2)$  
D. $O(2^n)$  
E. $O(\log n)$
**QUESTION 1B**

b) Determine the worst-case run-time in terms of $n$, where $n = \text{len}(\text{lst})$

```python
def sum_acc(lst, sum_so_far, pos):
    if pos == len(lst):
        return sum_so_far
    else:
        return sum_acc(lst, sum_so_far+L[pos], pos+1)

def sum_list2(lst):
    return sum_acc(lst, 0, 0)
```
QUESTION 1C

c) Determine the worst-case run-time in terms of n, where n = len(loi)

def evens(loi):
    return list(filter(lambda x: x%2 == 0, loi))
d) Determine the worst-case run-time in terms of n

def create_number_lists(n):
    total = []
    while n != 0:
        i = 0
        sublist = []
        while i < n:
            sublist.append(i)
            i = i + 1
        total.append(sublist)
        n = n - 1
    return total
e) Determine the worst-case run-time in terms of n, where n = len(L)

```python
def f(L):
    if L == []:
        return True
    elif L[0]%3==0 and f(L[1:]):
        return False
    else:
        return f(L[1:])
```

More runtime practice are provided on cs116 website under
- Additional Materials
  - Module Practice Problems
    - Extra Practice for Module 7
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 $\text{lst}1$ be all the elements in the list smaller than $n$
- Let $\text{lst}2$ be all the elements in the list larger than $n$
- Recursively quicksort $\text{lst}1$ and $\text{lst}2$

$\text{lst}1 + [x] + \text{lst}2$

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

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

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

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