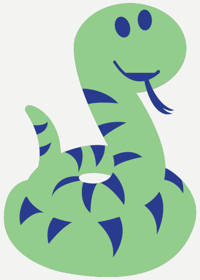
TUTORIAL 8

O()

EFFICIENCY, SEARCHING, AND SORTING

REMINDERS

- Assignment 7 due Wednesday, March 18th, at 10:00 AM
- The final exam is on April 15th, at 4:00PM.

RUNTIME REVIEW

- $O(1)$ – Constant
- $O(\log n)$ - Logarithmic
- $O(n)$ – Linear
- $O(n \log n)$ – Log linear
- $O(n^2)$ – Quadratic
- $O(2^n)$ – Exponential

USEFUL SUMMATIONS

- $\sum_{i=1}^{n} 1 = O(n)$
- $\sum_{i=1}^{n} i = O(n^2)$
- $\sum_{i=1}^{n} n = O(n^2)$
- $\sum_{i=1}^{n} \sum_{j=1}^{n} 1 = O(n^2)$

RECURRENCE RELATIONS

- $T(n) = O(1) + T(n-1) \longrightarrow O(n)$
- $T(n) = O(n) + T(n-1) \longrightarrow O(n^2)$
- $T(n) = O(1) + T(n/2) \longrightarrow O(\log n)$
- $T(n) = O(n) + 2T(n/2) \longrightarrow O(n \log n)$
  - $T(n) = O(n) + T(n/2) \longrightarrow O(n)$
- $T(n) = O(1) + T(n-1) + T(n-2) \longrightarrow O(2^n)$
  - $T(n) = O(1) + 2T(n-1) \longrightarrow O(2^n)$
  - $T(n) = O(n) + T(n-1) + T(n-2) \longrightarrow O(2^n)$
  - $T(n) = O(n) + 2T(n-1) \longrightarrow O(2^n)$

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`

- $T(n) = O(n) + T(n-1) \Rightarrow O(n^2)$

RUNTIME EXAMPLE 2

```
# 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) => O(n)

RUNTIME EXAMPLE 3

```
def fn_b(s):
    if len(s) == 0:
        return ""
    else:
        return fn_b(s[1:]) + fn_b(s[2:])
```

Count steps for:

- `s[1:]` and `s[2:]`
- Call `fn_b` recursively on a list of length `n−1` and another list of length `n−2`.
- $T(n) = O(n) + T(n-1) + T(n-2) \Rightarrow O(2^n)$

RUNTIME EXAMPLE 4

```
def fn_f(L):
    if L == []:
        return 0
    elif len(L) == 1:
        return 1
    else:
        L1 = L[0:-1:2]
        L2 = L[1:-1:2]
        return fn_f(L1) + fn_f(L2)
```

<u>Count steps for:</u>

- List slicing of `L[0:-1:2]`
- Assign `L[0:-1:2]` to `L1` and `L2`.
- Call `fn_f` recursively on list `L1` and `L2` of length `n//2`.
- T(n) = O(n) + 2T(n/2) => O(n log n)

CLICKER QUESTION 1A

- a) Determine the worst-case run-time in terms of n.

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

A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)
E. O(2^n)

CLICKER QUESTION 1B

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

```
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)
```

A. O(1)
B. O(log n)
C. O(n)
D. O(n^2)
E. O(2^n)

CLICKER QUESTION 1C

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

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

A. O(n)
B. O(log n)
C. O(n log n)
D. O(n^2)
E. O(2^n)

More runtime practice are provided on cs116 website under:

Additional Materials => Module Practice Problems => Extra Practice for Module 7

QUESTION 2: TUT5 Q1

- Write a function `record_digit(n)` that returns a list of integers of length 10, with each index from 0 to 9 represents a corresponding digit's total appearances in the integer `n`. You cannot use `L.count()`.
- For example, `record_digit(19990514)=>[1,2,0,0,1,1,0,0,0,3]`

WHAT IS THE RUNTIME?

```
def record_digit_acc(s, acc):
    if len(s) == 0:
        return acc
    else:
        acc[int(s[0])] += 1
        return record_digit_acc(s[1:], acc)

def record_digit(n):
    acc = [0]*10
    return record_digit_acc(str(n), acc)
```

Question: Can we get an O(n) runtime solution?

Suggestions to consider when trying to improve efficiency of solutions you develop:

- look at recalculated values
- can you find a way to avoid slicing a list or string
- investigate append rather than + for lists

YES!

```
def record_digit_acc(s,i,acc):
    if i == len(s):
        return acc
    else:
        acc[int(s[i])]+=1
        return record_digit_acc(s,i+1,acc)

def record_digit(n):
    acc = [0]*10
    return record_digit_acc(str(n), 0, acc)
```

Q3: WHAT IS THE RUNTIME?

```
def fn_3(n):
    if n==0:
        return 0
    elif fn_3(n-1) > n//2:
        return fn_3(n-1) - n//2
    else:
        return fn_3(n-1) + n//2
```

Possible to rewrite it with runtime O(n)?

DEFINITELY!

```
def fn_3(n):
    if n==0:
        return 0
    recur = fn_3(n-1)
    if recur > n//2:
        return recur - n//2
    else:
        return recur + n//2
```

QUESTION 3 - QUICKSORT

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

- Let **`n`** 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 + [n] + 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] + \
    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) => O(n`^{2}`)`
  - The list is already sorted
- In practice, quicksort can avoid the worst case most of the time, and, on average, runs in O(n log n) time.