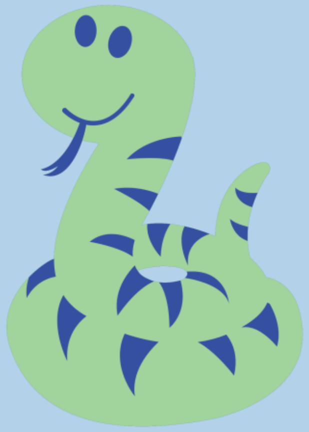
TUTORIAL 7 EFFICIENCY



REMINDER

- Assignment 6 is due next Wednesday, 10 AM.
- Final is on April 15th, at 4:00 pm.

CLICKER QUESTION 1

How was the midterm?

A. Pretty smooth and there was enough time
B. Overall easy but long
C. A bit difficult in some questions but overall reasonable length and difficulty
D. Overall difficult.
E. ☹

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 assumptions learned in class (and in the modules)

RUNTIME REVIEW

- O(1) – Constant
  - does not depend on the size of the input
  - `Comparison operations: >, >=, <, <=, ==`
  - Value assignment: (i.e. x = 4)
  - For numbers:
    - Numeric operations: `+, *, /, -, %, //`
    - `max, min`
  - For list L:
    - `L[0],L[1], L[2],…, len(L)…`
    - `L.append(4)…`

- O(n) – Linear
  - depends on the size of the input
  - For list L(assume the length of L is n):
    - `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`

USEFUL SUMMATIONS

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

RUNTIME EXAMPLE 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`
- $\sum_{i=1}^{n} 1$ = O(n)

RUNTIME EXAMPLE 2

```
# Let n be a natural number
def fn(n):
     ans = 1
     collection = list(range(2*n))
     for x in collection:
          if x%10==1:
               ans = ans + 1
          else:
               ans = ans + 2
     return ans
```

Count steps for:

- Assign `1` to ans
- `list(range(2*n))`
- Assign value to collection
- Loop:
  - Number of Iterations
  - Asymptotic run time of the body of loop:
    - Calculate `x%10`
    - Check `if x%10 ==1`
    - `ans = ans + 1` (or `ans = ans +` 2)
- Return `ans`
- O(n) + $\sum_{i=1}^{n} 1$ => O(n) +O(n)=>O(n)

RUNTIME EXAMPLE 3

```
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
- O(1)

RUNTIME EXAMPLE 4

```
def fn(L):
    return len(list(filter(lambda x:
                             x == max(L),L)))
```

Count the steps for:

- `max()`
- Check `x == max(L)`
- Filter
- `len()`
- O(n) * O(n) => O(n^2)

CLICKER QUESTION 1A

a) 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))
```

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

CLICKER QUESTION 1B

b) 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
```

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

ITEM DEFINITION

A `Card` is a list of length 2 where

- the first item is an integer between 1 and 13, inclusive, representing the `value` of the card, and
- the second item is a string ("hearts", "spades", "clubs", or "diamonds") representing the `suit` of the card.

Example: `[1, "hearts"]` represents the ace of hearts

LOOP: TUTORIAL 4 Q1

Use loops, write a function `create_cards` that consumes two lists with same length, which are a list of card values (integers between 1 and 13), and a list of suit values (one of the four suit strings), and returns a list of `Card`, created pair-wise from the consumed lists (`values` and `suits`).

For example,

```
create_cards([4,1,10],["hearts", "diamonds", "clubs"])
      =>[[4,"hearts"], [1, "diamonds"], [10, "clubs”]]
```

WHAT IS THE RUNTIME OF CREATE_CARDS?

```
def create_cards(values, suits):
    acc = []
    for i in range(len(values)):
        acc.append([values[i],suits[i]])
    return acc
```

A. O(n)

B. O(n^2)

C. O(2^n)

WHAT IS THE RUNTIME IF WE USE ABSTRACT LIST FUNCTION?

```
def create_cards(values,suits):
    return list(map(lambda x, y:[x,y],values,suits))
```

LOOP: TUTORIAL 5 Q2

Using loops, write a function `count_max` that consumes a nonempty list of integers `alon` and returns the number of times the largest integer in `alon` appears.

Note: - `max` and `L.count()` cannot be used in this question.

- Your function can only pass through the list once

For example,

```
count_max([1, 3, 5, 4, 2, 3, 3, 3, 5]) => 2
```

since the largest element of the list, `5`, appears twice. Your function should pass through the list only once.

WHILE LOOP SOLUTION

```
def count_max(alon):
    current_max = alon[0]
    max_occur = 0
    while L != []:
        if L[0] > current_max:
            current_max = alon[0]
            max_occur = 1
        elif L[0] == current_max:
            max_occur += 1
        L = L[1:]
    return max_occur
```

What is the runtime?

Let us solve the question with runtime **O(n)**!

FOR LOOP SOLUTION

```
def count_max(lon):
    current_max = lon[0]
    max_occur = 0
    for each in lon:
        if each > current_max:
            current_max = each
            max_occur = 1
        elif each == current_max:
            max_occur += 1
    return max_occur
```