IF
CS 180

1/8/16

T.A.: Costas
OH: W 1-2pm
BH 2432

```
int x=1;                          // a given program that
for(int i=0; i<n; i++) {          // computes 3^n
    x=3*x;
}
```

```
int x=1;        α msec
int i=0;        α msec
  i<1           β msec   (this is executed n+1 times)
{ i++           γ msec
{ x=3*x;        δ msec
```

executed n times

$2\alpha + \beta(n+1) + \gamma n + \delta n \approx O(n)$

dependent on n

$n=0 \rightarrow t = 2\alpha + \beta$

$n=1 \rightarrow t = 2\alpha + 2\beta + \gamma + \delta$

$n=10^6 \rightarrow t = 2\alpha + \beta(10^6+1) + 10^6\gamma + 10^6\delta$

$t \approx (\beta + \gamma + \delta)(10^6)$

$t \approx 3\varepsilon(10^6)$

$t \approx 3n$

$t \approx n \Rightarrow O(n)$

$\Theta(n)$ this is a tight bound

$C_1 g(n) \leq f(n) \leq C_2 g(n) \quad n > n_0$

$\Omega(n)$ this is lower bound

$c g(n) \leq f(n) \quad n > n_0$

possible for time complexity to be $O(mn)$
this is still linear -- happens when searching an array w/ dimensions m x n

ex: find the odd occuring element (appears 1 time) in a list of numbers (assume there is only one)

2, 3, 3, 4, 5, 6, 4, 5, 6 → 2 is the odd occuring element

start w/ simplest solution
compare one element w/ the rest

loop over the elements, loop over the rest
if element is not found
end
time complexity: $O(n^2)$
n elements and must compare w/ n elements

space complexity: how much extra memory you need
not necessary w/ the model of computation
if we used a counter, we can reuse it
space complexity is constant

$O(n \log n)$ is faster than $O(n^2)$

improvements:
1) sort the list of elements so identical elements are next to each other then loop through $O(n \log n)$
2) array of counters

```
for each element
    counters[element] += 1;
for each counter
    if counter is odd
        return index[counter]
```

time $O(n)$
space $O(n)$

3) use XOR
time $O(n)$
space $O(1)$
3 XOR 3 = 0
0 XOR 2 = 2

the best we can do is O(n) since we have to go through the entire array at least once

similar to #5

ex: assume an array w/ n numbers
find whether 2 numbers sum to zero (true or false)
(you can use the same number twice)

```
for (i = 0 → n)
    for (j = i → n)
        if(array[i] + array[j] == 0)
            return true
return false
```

time complexity: $O(n^2)$

improvement:

1) sort the array: check the middle to see whether the sum is greater or smaller than zero
   O(nlogn)
2) hashmap
   for element in array
   if negative element exists in hash
   terminate
   else
   HASH[element] = 1

time complexity: O(n)