

# CSCI 204: Data Structures & Algorithms

## Algorithm Analysis

The case of recursion

Last time we discussed the case for loops

```
for item in my_list:
    if item == search_name:
        return True
```

O(n)

```
for i in range(len(my_list)):
    for j in range(len(my_list)):
        sum += i + j + my_list[i]
```

O(n<sup>2</sup>)

```
while n > 0:
    n = n // 2 # binary search
    # fall in this pattern
```

O(log n)

```
'''Adopted for CSCI 204 from
http://interactivepython.org/runestone/static/pythonds/Sort
Xiaoning Meng
2017-09-05
'''
from random import *
def bubble_sort(alist):
    for passnum in range(len(alist)-1, 0, -1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
```

n

n-1

$$T(n) = n + (n-1) + (n-2) + \dots + (n-i) + \dots + 1 + n * C$$

$$= n * C + \sum(i)_{(for i=1 to n)} = n * C + n * (n-1) / 2$$

$$\rightarrow O(n^2)$$

Today we will discuss the case for recursion, then we'll do some exercises.

```
def bin_search(nums, target, left, right):
    """ Search the 'target' from the list 'nums'
    nums: a list of sorted numbers (could be strings as well)
    left: starting index of the list
    right: ending index of the list
    """
    if left > right: # not found
        return False
    mid = (left + right) // 2
    if nums[mid] == target:
        return True
    elif nums[mid] < target: # search for upper half
        left = mid + 1
        return bin_search(nums, target, left, right)
    else: # search for lower half
        right = mid - 1
        return bin_search(nums, target, left, right)
```

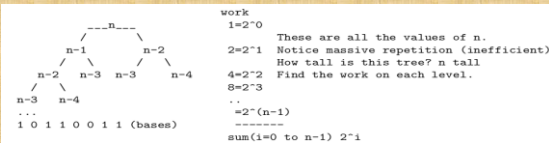
O(1)

O(n/2)

$$T(n) = C + T(n/2) = C + [C + T(n/4)] = \dots = kC + CT(1) = C \log n + B$$

So for binary search, T(n) is on the order of O(log n).

```
def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```



## Tower Of Hanoi

```
def tower_of_hanoi(n, from_rod, to_rod, aux_rod):
    if n == 1:
        print("Move disk 1 from rod ", from_rod, " to rod ", to_rod)
        return
    tower_of_hanoi(n-1, from_rod, aux_rod, to_rod)
    print("Move disk", n, "from rod", from_rod, " to rod ", to_rod)
    tower_of_hanoi(n-1, aux_rod, to_rod, from_rod)
```

$$T(n) = 1 + 2 * T(n-1) = 1 + 2 * [1 + 2 * T(n-2)] = 1 + 2 + 4 * T(n-2) = \dots$$

$$= 1 + 2 + 4 + 8 + \dots + 2^{(n-k)} * T(n-k)$$

$$O(2^n)$$