

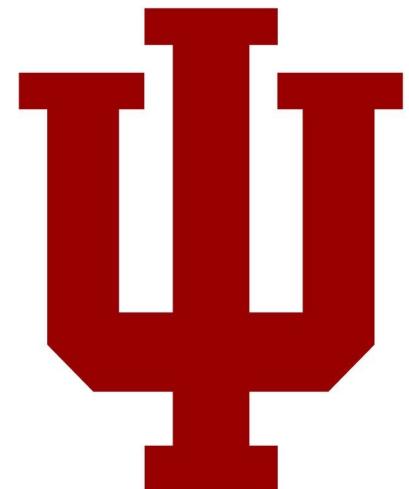
Test

# 01: Coding for Performance

Engr 315: Hardware / Software Codesign

Andrew Lukefahr

*Indiana University*



Some material taken from:

[https://github.com/trekhleb/homemade-machine-learning/tree/master/homemade/neural\\_network](https://github.com/trekhleb/homemade-machine-learning/tree/master/homemade/neural_network)

<http://cs231n.github.io/neural-networks-1/>

Course Website

engr315.github.io

Write that down!

# Slack - <https://engr-315.slack.com>

- Thanks Joe!

Send as  
Canvas  
msg.

# Office Hours

- Andrew: M/W 11-12 in 2032 Luddy or 4111 Luddy

## TA

---

Nicole Miller - ncm1@iu.edu

Caleb Cook - cookce@iu.edu

### Office Hours:

- Mon/Wed: 5pm-8pm (Nicole)
- Tue: 1:30pm-4:30pm (Caleb)
- Fri: 11:00am-2pm (Caleb) (4111)

**Location:** 3111 Luddy Hall (one floor below 4111)



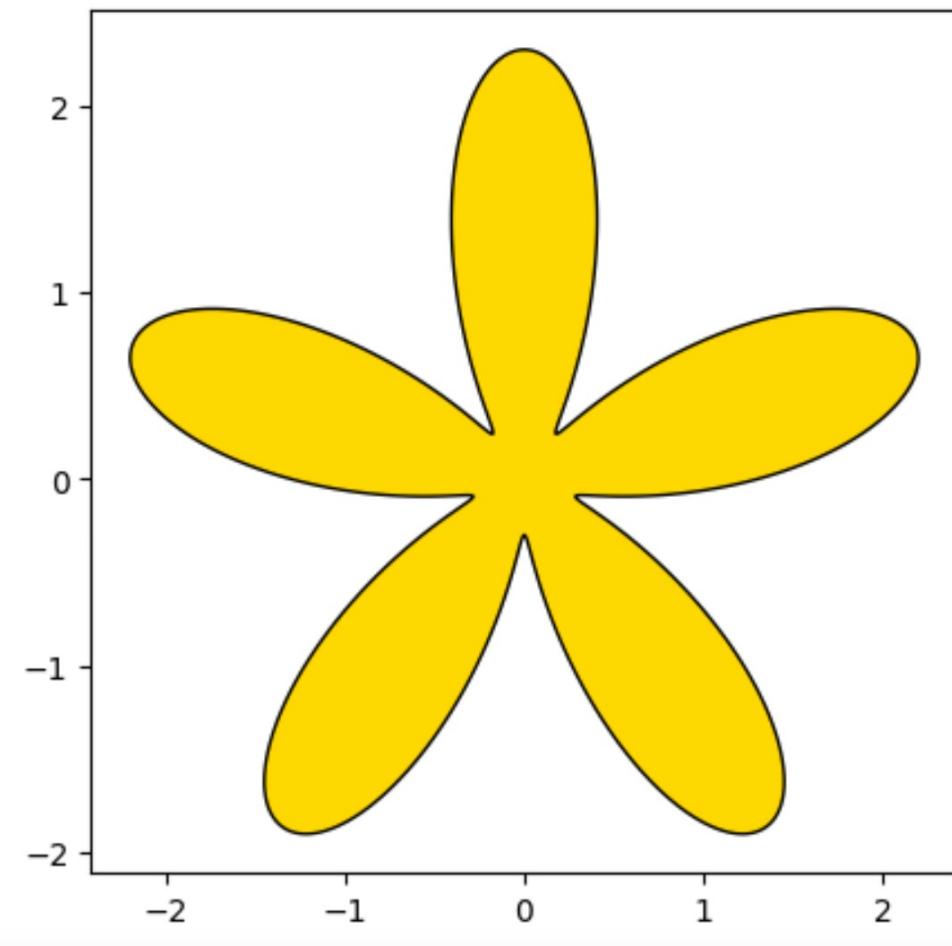
# I try to post all the code I use in class

The screenshot shows a GitHub repository page for 'Engr315/lecture\_slides'. The repository is public. The 'Code' tab is selected. The page displays three recent commits:

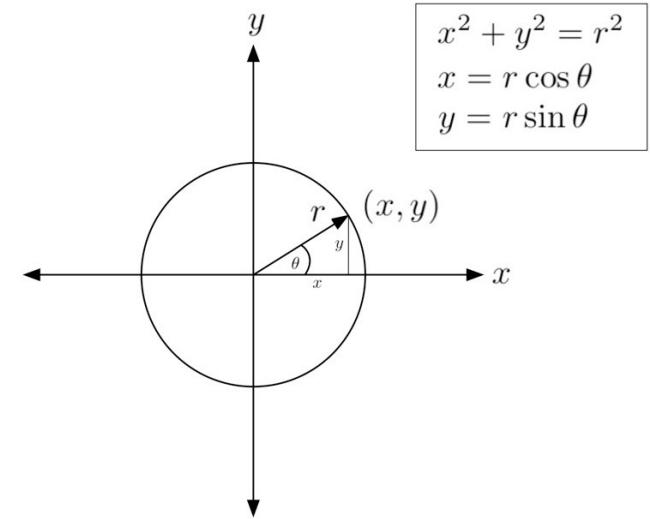
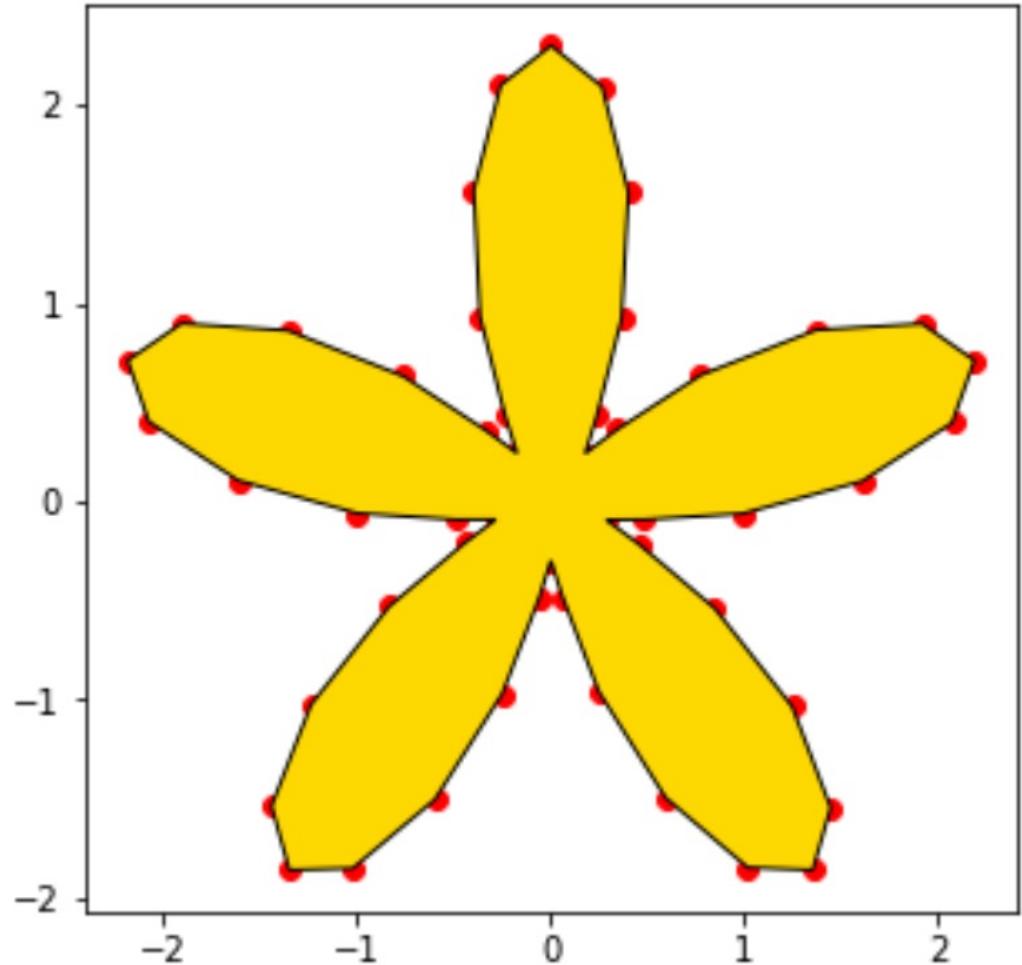
- Andrew Lukefahr Adding 00\_intro.pdf (2d15b57, 35 seconds ago, 6 commits)
- code\_00 Adding 00\_intro.pdf (35 seconds ago)
- 00\_intro.pdf Adding 00\_intro.pdf (35 seconds ago)

Remind me if (when) I forget.

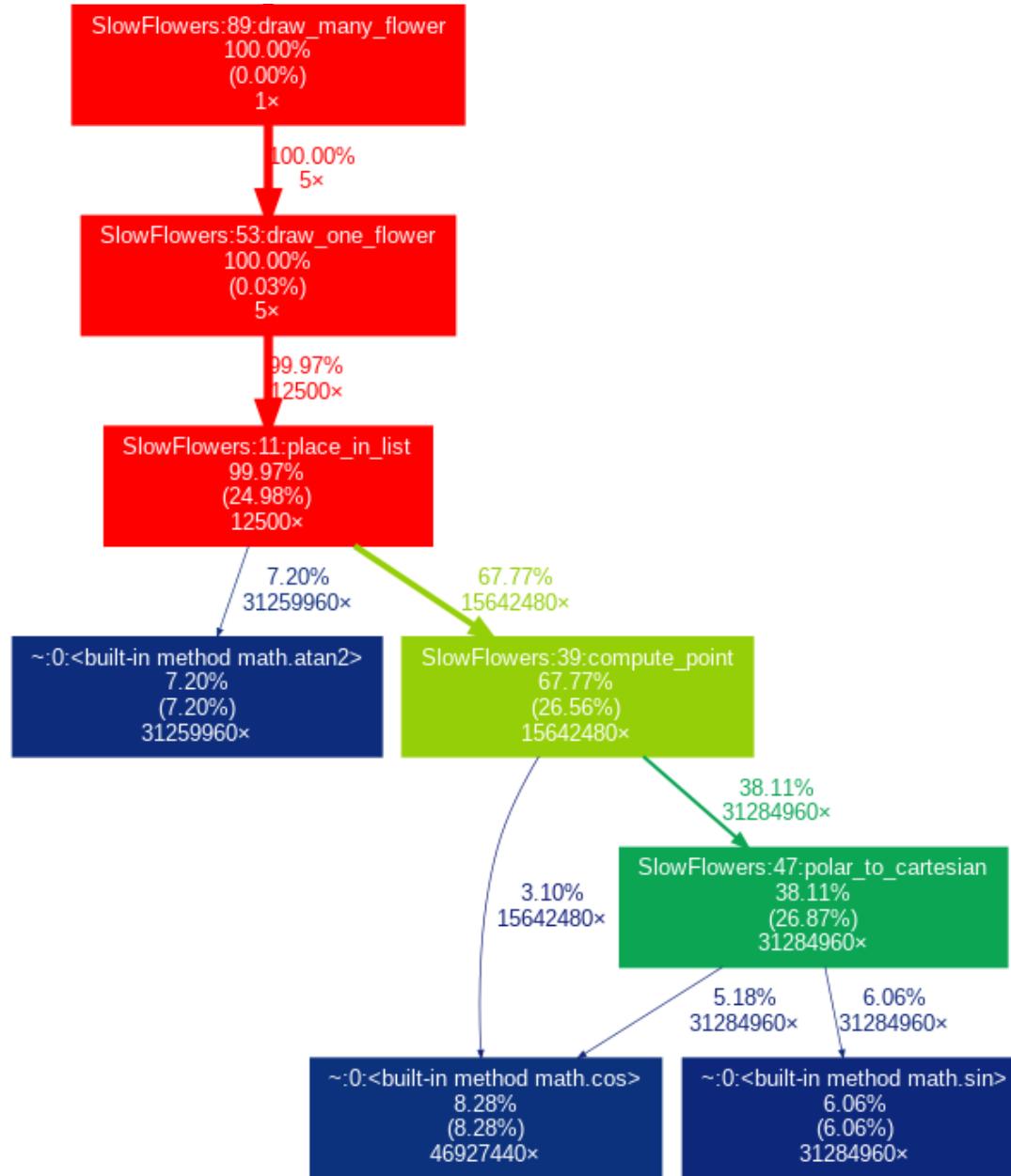
*Opt.*  
Project 1: ~~Flowers~~



# Project 1: Flowers



# Project 1: Flowers



# Project 1: Flowers

- This is optimized already:

```
oldTheta = atan2(point[0]-x_c, point[1]-y_c)  
r = (a * cos(5 * (theta - (pi/2)))) + (a * 1.3)
```

- You aren't going to accelerate math functions. Don't try.
- Figure out how to call it less.

# Project 1: “Bonus”

- Bonus: Gets you bonus points!
  - My old best time
- Better Bonus: Gets you even more bonus points!
  - (best time from student)

Good luck!

# Code Profiling

- In software engineering, **profiling** ("program profiling", "software profiling") is a form of dynamic program analysis that **measures**, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or **the frequency and duration of function calls**. Most commonly, profiling information serves to aid program optimization. [Wiki]

# Code Profiling can measure

- Program Runtimes
- Function Call Numbers/Runtimes
- Memory Usage
- Instruction Usage
- Others

# Profiling guide us on where to look to reduce runtime

```
1 def squares(n):
2     if n <= 1:
3         return [1]
4     else:
5         seq = squares(n-1)
6         seq.append(n*n)
7         return seq
```

40002 function calls (20003 primitive calls) in 0.021 seconds

Ordered by: standard name

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
20000/1	0.019	0.000	0.021	0.021	<ipython-input-8-50d13c5dd8df>:1(squares)
1	0.000	0.000	0.021	0.021	<string>:1(<module>)
1	0.000	0.000	0.021	0.021	{built-in method builtins.exec}
19999	0.002	0.000	0.002	0.000	{method 'append' of 'list' objects}
1	0.000	0.000	0.000	0.000	{method 'disable' of '_lsprof.Profiler' objects}

# Conclusion #1: Function calls are not free!

- Setup/Return overheads with function calls
  - Small
- Recursion: small overheads  $\times$  many calls
  - Can add notable overheads
- Only use recursion if don't care about performance

# Cutting recursion buys us $\sim 2x$

```
1 def squares(n):
2     if n <= 1:
3         return [1]
4     else:
5         seq = squares(n-1)
6         seq.append(n*n)
7     return seq
```

```
1 def squares2(n):
2     if n <= 1:
3         return [1]
4     else:
5         seq = []
6         for i in range(1,n):
7             seq.append(i*i)
8     return seq
```

```
1 import time
2 import sys
3 sys.setrecursionlimit(21000)
4
5 start_time = time.time()
6 squares(20000)
7 end_time = time.time()
8
9 # at the end of the program:
10 print("%f seconds" % (end_time - start_time))
```

0.009825 seconds

```
1 import time
2
3 start_time = time.time()
4 squares2(20000)
5 end_time = time.time()
6
7 # at the end of the program:
8 print("%f seconds" % (end_time - start_time))
```

0.004209 seconds

# Can we make it go even faster?

```
1 def squares2(n):
2     if n <= 1:
3         return [1]
4     else:
5         seq = []
6         for i in range(1,n):
7             seq.append(i*i)
8         return seq
```

```
1 import time
2
3 start_time = time.time()
4 squares2(20000)
5 end_time = time.time()
6
7 # at the end of the program:
8 print("%f seconds" % (end_time - start_time))
```

0.004209 seconds

```
1 import numpy as np
2 def squares3(n):
3
4     seq = np.zeros(n, dtype=np.int)
5     for i in range(1, n+1):
6         seq[i-1] = i * i
7     return seq
```

```
1 import time
2
3 start_time = time.time()
4 squares3(20000)
5 end_time = time.time()
6
7 # at the end of the program:
8 print("%f seconds" % (end_time - start_time))
```

0.003960 seconds

# ... And I'm bested!

```
1 #Thanks Drason!
2 def squares4(n):
3     return [i * i for i in range(1, n+1)]
4
5 start_time = time.time()
6 squares4(20000)
7 end_time = time.time()
8
9 # at the end of the program:
10 print("%f seconds" % (end_time - start_time))
```

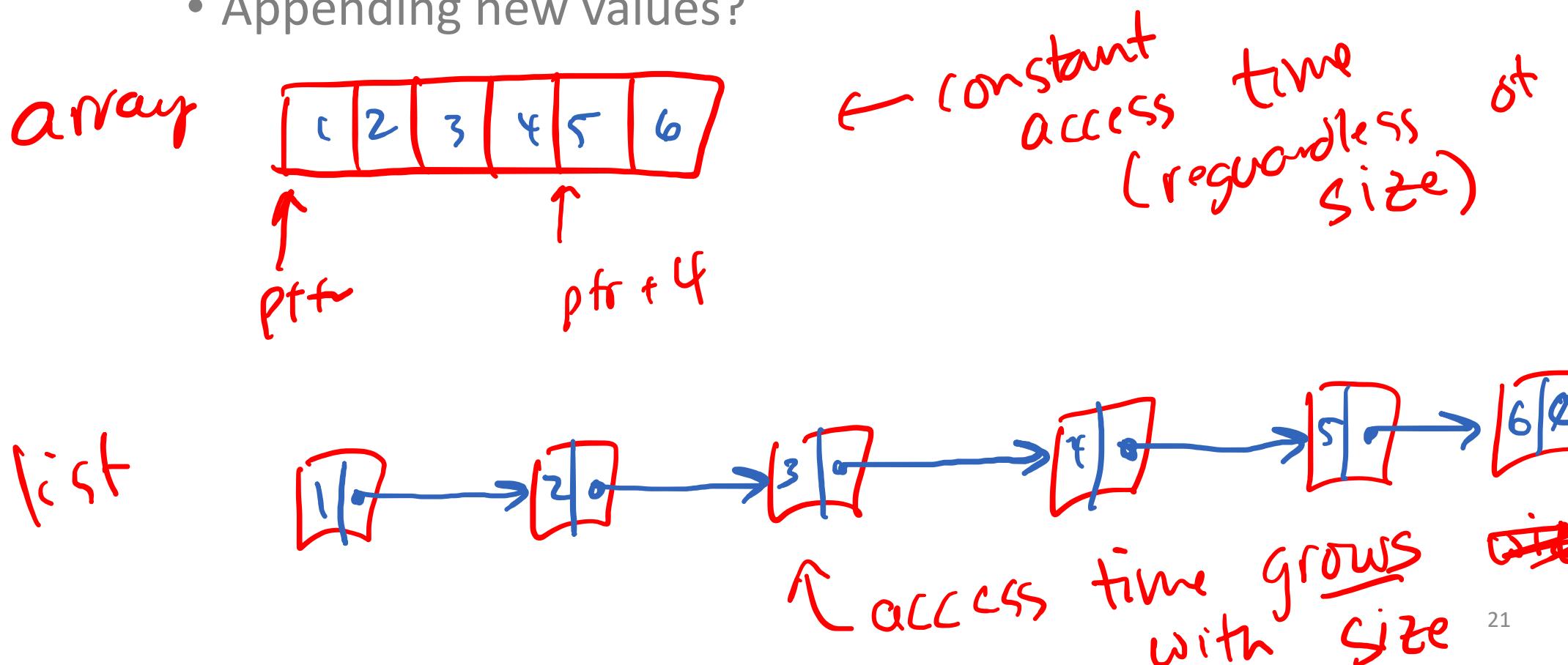
0.003010 seconds

# Conclusion #2: memory preallocation *should be faster*

- Numpy reallocates a large contiguous block 😊
    - But also zeros it out. 😞
  - List.append() allocates new memory as needed
    - Python's “List” is weird.
- comprehension*

# Array vs. Linked List: Which is faster?

- Randomly accessing a specific element?
- Appending new values?

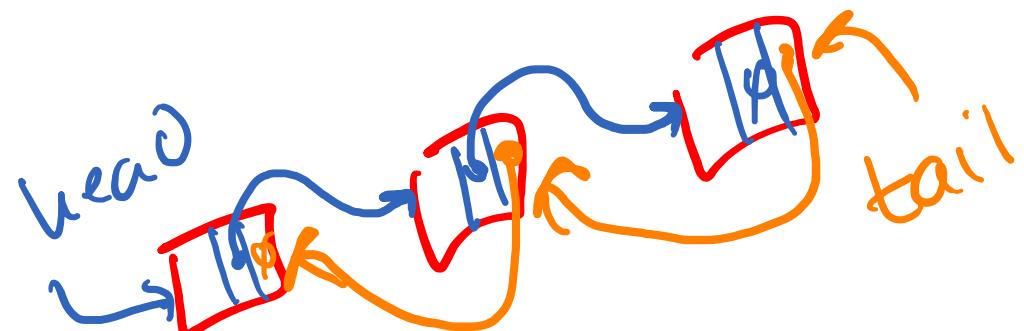


deque  $\Rightarrow$  Double Ended Queue

# Array vs. Linked List: Random Access

```
1 lst = collections.deque(nums)
2 arr = np.array(nums)
3 print (lst)
4 print (arr)
```

```
deque([5, 1, 9, 0, 3, 2, 6, 4, 8, 7])
[5 1 9 0 3 2 6 4 8 7]
```



```
1 def traverse( thing, times):
2     idx = 0
3     for i in range(times):
4         nidx = thing[idx]
5         print (i, ':', idx, '->', nidx)
6         idx = nidx
```

```
1 trips = 10
2 traverse(lst, trips)
```

```
0 : 0 -> 5
1 : 5 -> 2
2 : 2 -> 9
3 : 9 -> 7
4 : 7 -> 4
5 : 4 -> 3
6 : 3 -> 0
7 : 0 -> 5
8 : 5 -> 2
9 : 2 -> 9
```

# Array vs. Linked List: Random Access

```
1 start_time = time.time()
2 traverse(lst, trips)
3 end_time = time.time()
4
5 # at the end of the program:
6 print("True List: %f seconds" % (end_time - start_time))
7
8 start_time = time.time()
9 traverse(arr, trips)
10 end_time = time.time()
11
12 # at the end of the program:
13 print("Array: %f seconds" % (end_time - start_time))
```

```
0 : 0 -> 5
1 : 5 -> 2
2 : 2 -> 9
3 : 9 -> 7
4 : 7 -> 4
5 : 4 -> 3
6 : 3 -> 0
7 : 0 -> 5
8 : 5 -> 2
9 : 2 -> 9
True List: 0.001251 seconds
0 : 0 -> 5
1 : 5 -> 2
2 : 2 -> 9
3 : 9 -> 7
4 : 7 -> 4
5 : 4 -> 3
6 : 3 -> 0
7 : 0 -> 5
8 : 5 -> 2
9 : 2 -> 9
Array: 0.006385 seconds
```

# Array vs. Linked List: Random Access

```
1 def traverse( thing, times):
2     idx = 0
3     for i in range(times):
4         idx = thing[idx]
5
6     random.seed(1)
7     sz = 1000000
8     nums = [x for x in range(sz)]
9     random.shuffle(nums)
10    random.shuffle(nums)
11    lst = collections.deque(nums)
12    arr = np.array(nums)
13    trips = 1000
14
15    start_time = time.time()
16    traverse(lst, trips)
17    end_time = time.time()
18    print("True List: %f seconds" % (end_time - start_time))
19
20    start_time = time.time()
21    traverse(arr, trips)
22    end_time = time.time()
23    print("Array: %f seconds" % (end_time - start_time))
24
25    start_time = time.time()
26    traverse(nums, trips)
27    end_time = time.time()
28    print("Python List: %f seconds" % (end_time - start_time))
```

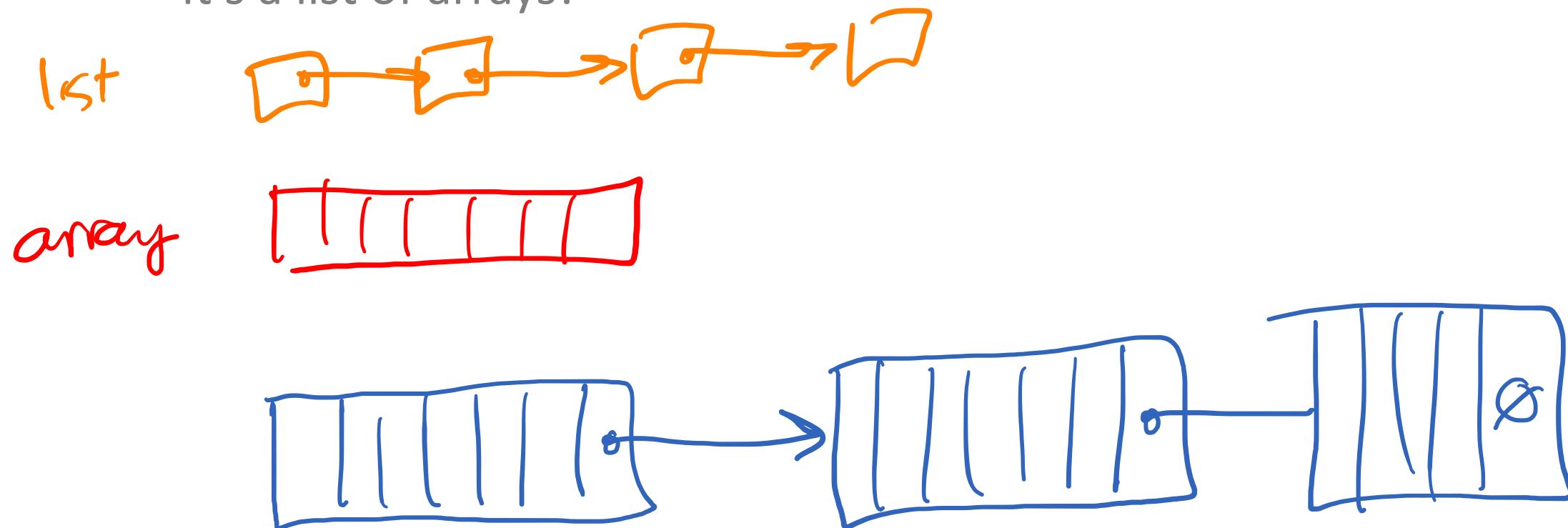
True List: 0.037878 seconds

Array: 0.000312 seconds

Python List: 0.000410 seconds

# Python’s “List” isn’t actually a “List”

- It's a list of arrays!



# Array vs. Linked List: Sequential Insert

```
1 def insert(thing, idx, values):
2     print (thing)
3     for value in values:
4         thing.insert(idx, value)
5     print (thing)
6
7 random.seed(1)
8 sz = 10
9 nums = [x for x in range(sz)]
10 random.shuffle(nums)
11 random.shuffle(nums)
12 lst = collections.deque(nums)
13 arr = np.array(nums)
14
15 idxs = int(sz/2)
16 insert(nums, idxs, [-1,-2,-3,-4])
```

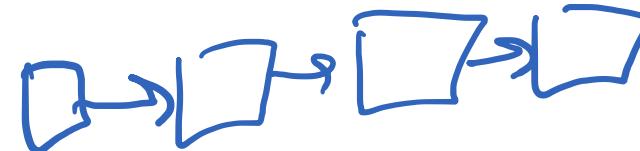
```
[5, 1, 9, 0, 3, 2, 6, 4, 8, 7]
```

```
[5, 1, 9, 0, 3, -4, -3, -2, -1, 2, 6, 4, 8, 7]
```

# Array vs. Linked List: Sequential Insert

Binary  
Search  
Trees?

Insert at: 0

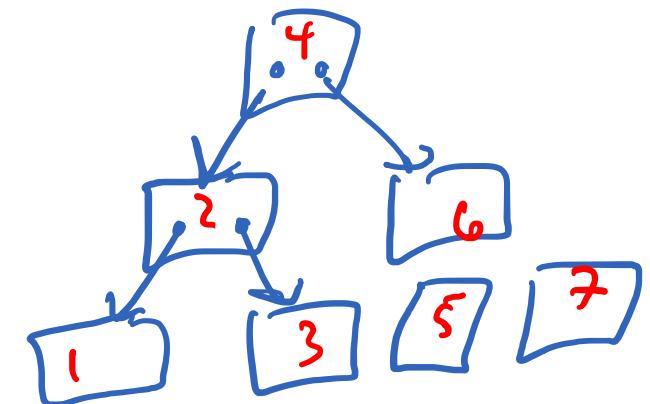


True List: 0.000085 seconds

Array: 0.335853 seconds

Python List: 0.115629 seconds

Insert at: 750000  $\sim \frac{3}{4}$



True List: 0.054327 seconds

Array: 0.336377 seconds

Python List: 0.022257 seconds

Data  
Structures  
class

# Big O Complexity

- Computational time complexity describes the change in the runtime of an algorithm, depending on the change in the input data's size.
- "How much does an algorithm's performance change when the amount of input data changes?"

# $O(1)$ – Constant Time

- “big O of 1”
- Runtime is constant, regardless of input size
- Example: `x = array[n]`

# $O(1)$ – Constant Time

Complexity class  $O(1)$  – constant time



Material taken from: <https://www.happycoders.eu/algorithms/big-o-notation-time-complexity/>



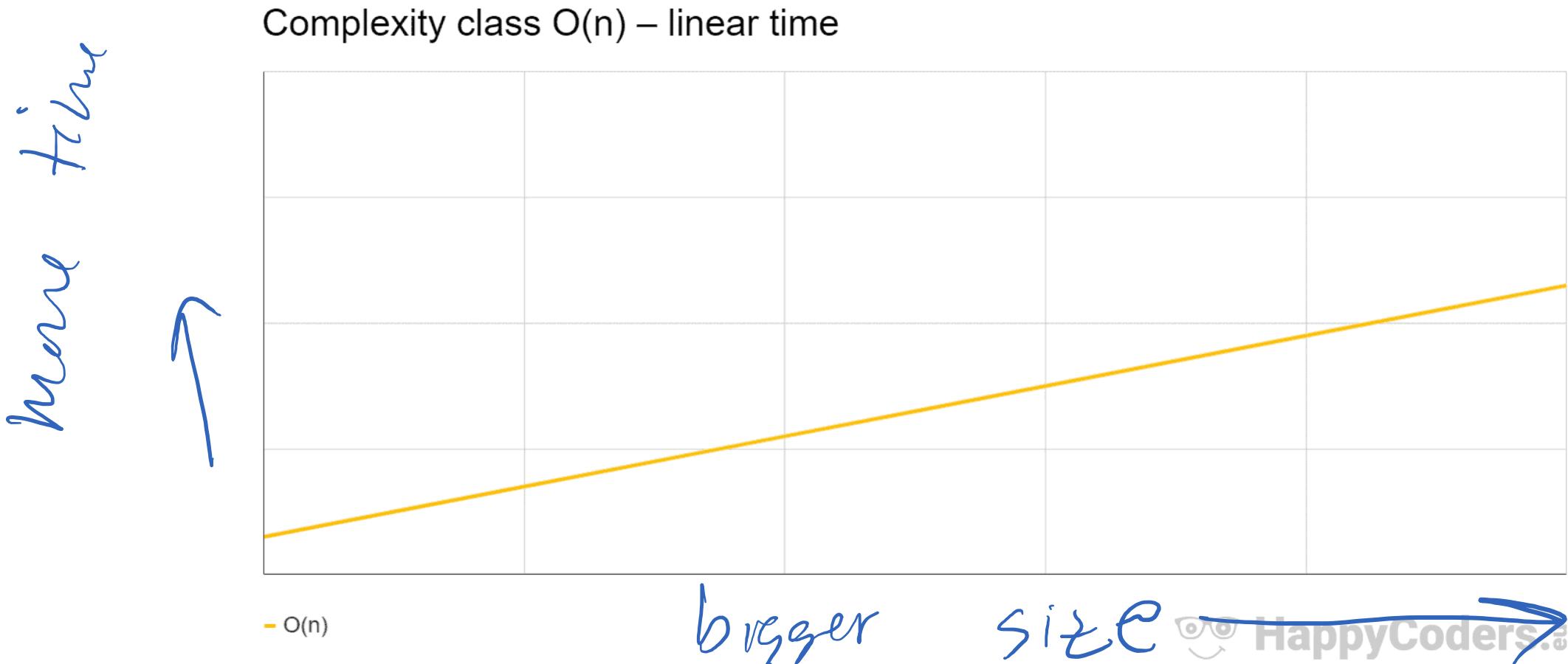
HappyCoders.eu



# $O(n)$ – Linear Time

- “big O of n”
- Runtime grows linearly with input size
- Example: Linked Lists!

# $O(n)$ – Linear Time

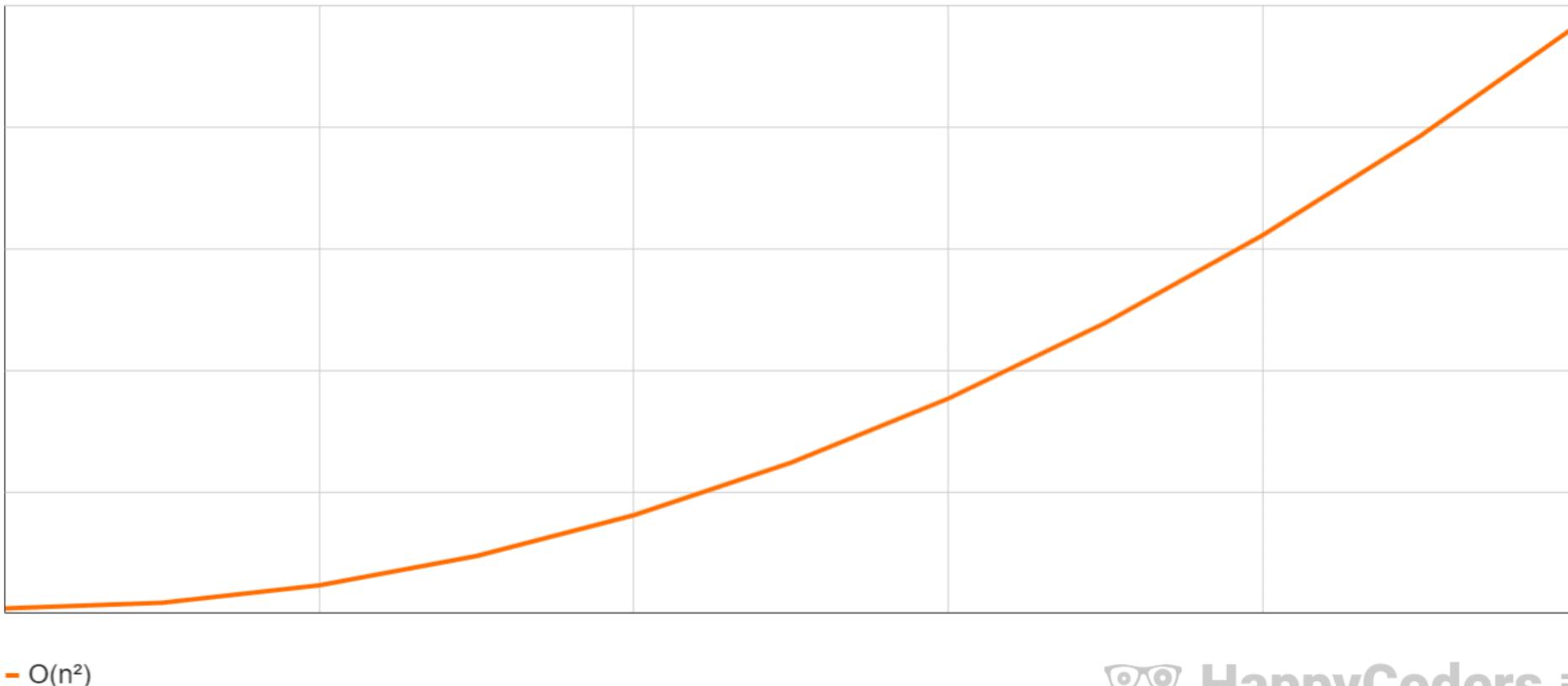


# $O(n^2)$ – Quadratic Time

- “big O of n squared”
- Runtime grows linearly with square of the input size
- Example: Bubble Sort
  - `haystack.sort(low->high)`

# $O(n^2)$ – Quadratic Time

Complexity class  $O(n^2)$  – quadratic time

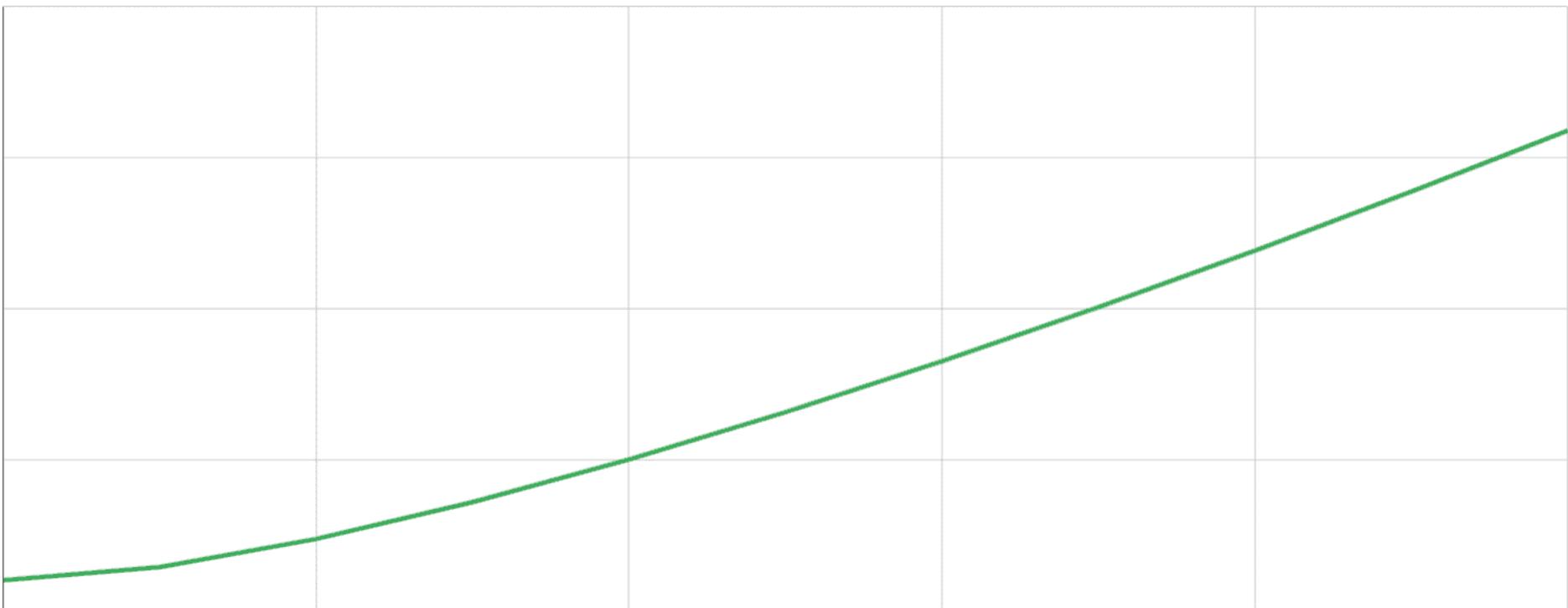


# $O(n \log n)$ – Quasilinear Time

- “big O of  $n \log n$ ”
- Runtime grows linearly and logarithmically with the input size
- Example: Good Sort
  - `haystack.sort(low->high)`

# $O(n \log n)$ – Quasilinear Time

Complexity class  $O(n \log n)$  – quasilinear time



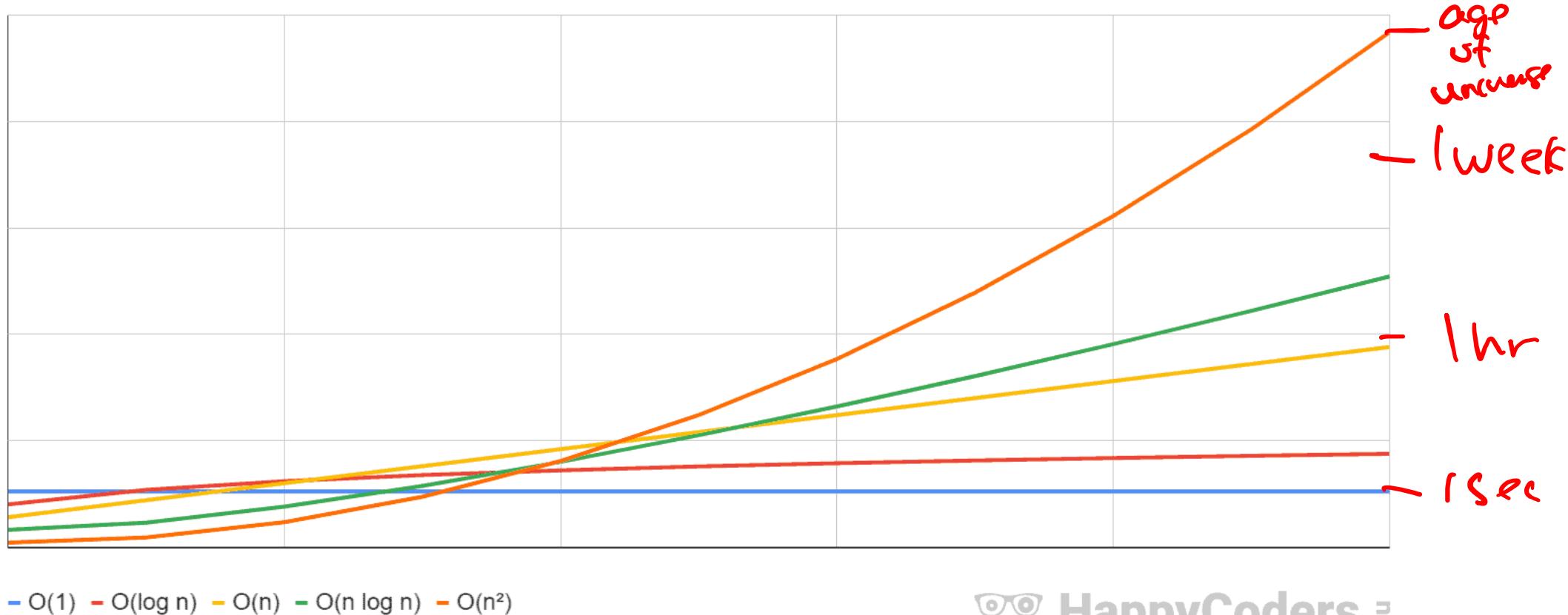
—  $O(n \log n)$



Material taken from: <https://www.happycoders.eu/algorithms/big-o-notation-time-complexity/>

# $O()$ Complexities

Comparing the complexity classes  $O(1)$ ,  $O(\log n)$ ,  $O(n)$ ,  $O(n \log n)$ ,  $O(n^2)$



HappyCoders.eu

# Conclusion #3: Think about your data structure!

- How will you be accessing your data?
  - Randomly? Sequentially?
- How will you update your data?
- Pick a data structure to minimize overheads for your access patterns

Find: The needle in the haystack.

# Find: The needle in the haystack.

NO  
libraries!

opt

```
1 def find_ignore_case( needle, haystack):
2     results = []
3     for hi in range(len(haystack)):
4         match = True
5         for ni in range(len(needle)):
6             h = haystack[hi + ni].lower()
7             n = needle[ni].lower()
8             if h != n:
9                 match=False
10                break
11        if match:
12            results.append(hi)
13    return results
```

```
28 sz=20
29 haystack = random_str(sz)
30 needle = haystack[int(sz/2):int(sz/2)+2]
31 results = find_ignore_case(needle, haystack)
32
33 print (needle)
34 print (haystack)
35 print (results)
36
```

```
sk
eiPPzDAnWiskaumnqYpl
[10]
```

# Find: The needle in the haystack.

```
1 def find_ignore_case( needle, haystack):
2     results = []
3     for hi in range(len(haystack)-len(needle)):
4         match = True
5         for ni in range(len(needle)):
6             h = haystack[hi + ni].lower()
7             n = needle[ni].lower()
8             if h != n:
9                 match=False
10            if match:
11                results.append(hi)
12        return results
13
14 random.seed(1)
15 sz=1000000
16 haystack = random_str(sz)
17 needle = haystack[int(sz/2):int(sz/2)+2]
18
19 start_time = time.time()
20 results = find_ignore_case(needle, haystack)
21 end_time = time.time()
22 print("True List: %f seconds" % (end_time - start_time))
```

True List: 1.109001 seconds

# Find: The needle in the haystack.

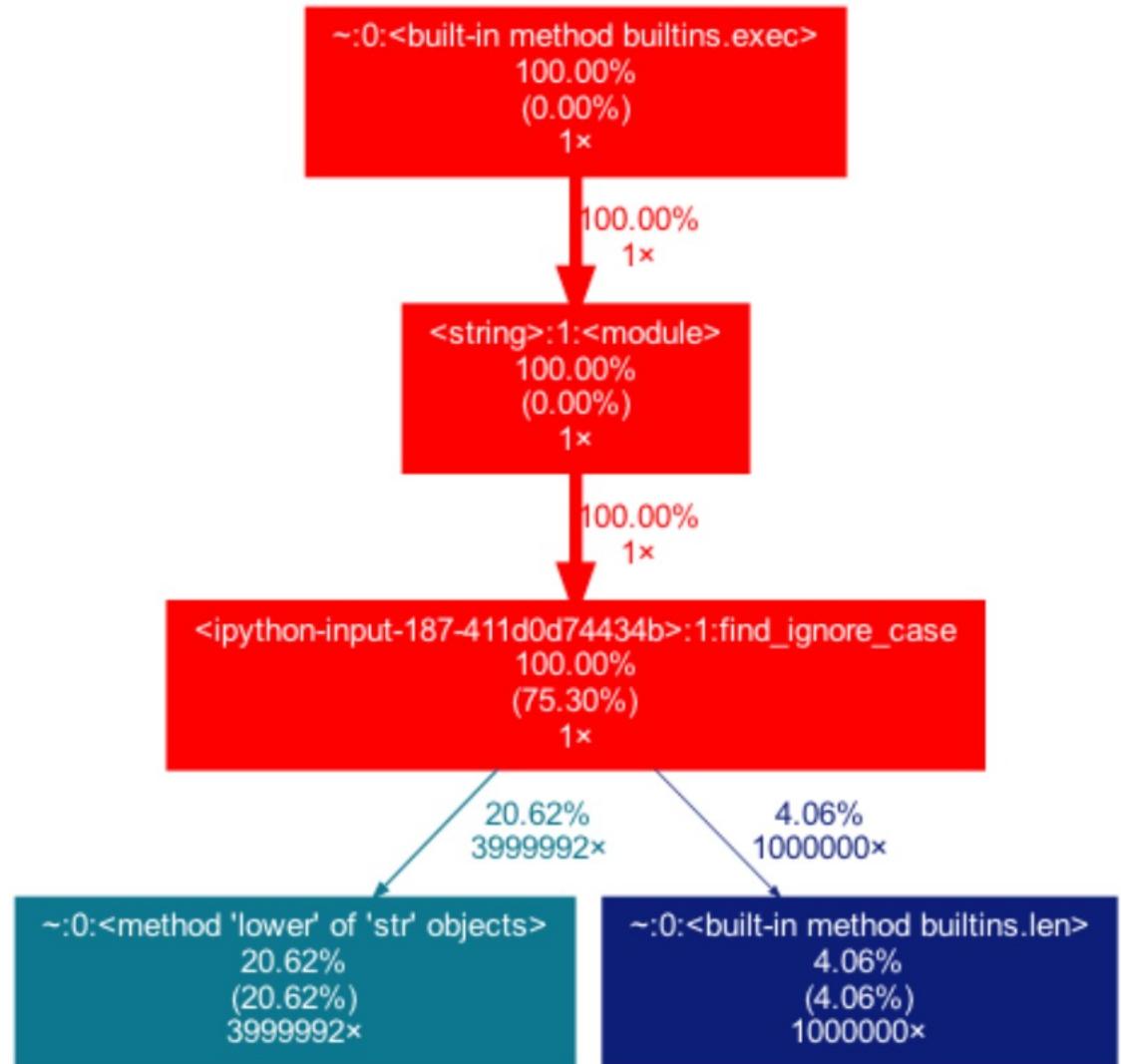
```
: 1 import cProfile
2 cProfile.run('find_ignore_case(needle, haystack)')

      5001486 function calls in 1.715 seconds

Ordered by: standard name

      ncalls  tottime  percall  cumtime  percall filename:lineno(function)
             1    1.276    1.276    1.715    1.715 <ipython-input-187-411d0d74434b>:1(find_ignore_case)
             1    0.000    0.000    1.715    1.715 <string>:1(<module>)
             1    0.000    0.000    1.715    1.715 {built-in method builtins.exec}
 1000000    0.072    0.000    0.072    0.000 {built-in method builtins.len}
     1490    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
             1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
 3999992    0.367    0.000    0.367    0.000 {method 'lower' of 'str' objects}
```

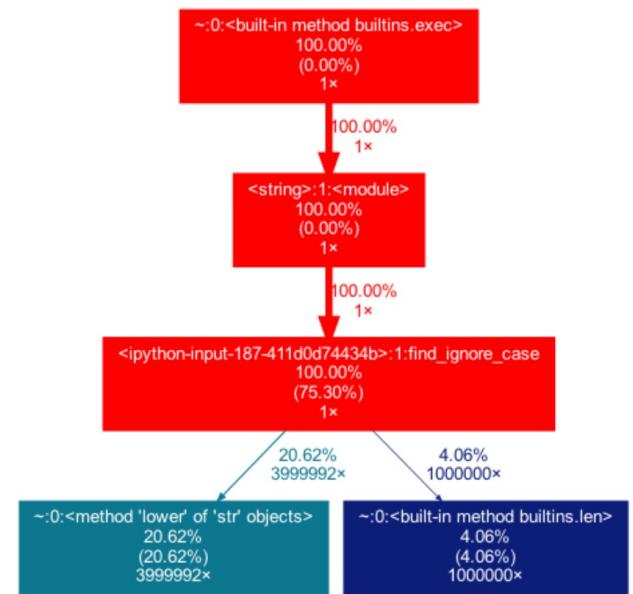
# Find: The needle in the haystack.



# Find: The needle in the haystack.

```
1 def find_ignore_case( needle, haystack):
2     results = []
3     for hi in range(len(haystack)-len(needle)):
4         match = True
5         for ni in range(len(needle)):
6             h = haystack[hi + ni].lower()
7             n = needle[ni].lower()
8             if h != n:
9                 match=False
10            if match:
11                results.append(hi)
12        return results
13
14 random.seed(1)
15 sz=1000000
16 haystack = random_str(sz)
17 needle = haystack[int(sz/2):int(sz/2)+2]
18
19 start_time = time.time()
20 results = find_ignore_case(needle, haystack)
21 end_time = time.time()
22 print("True List: %f seconds" % (end_time - start_time))
```

True List: 1.109001 seconds



- No libraries!

# Find: The needle in the haystack.

```
def find_ignore_case( needle, haystack):
    results = []
    for hi in range(len(haystack)-len(needle)):
        match = True
        for ni in range(len(needle)):
            h = haystack[hi + ni].lower()
            n = needle[ni].lower()
            if h != n:
                match=False
        if match:
            results.append(hi)
    return results
```

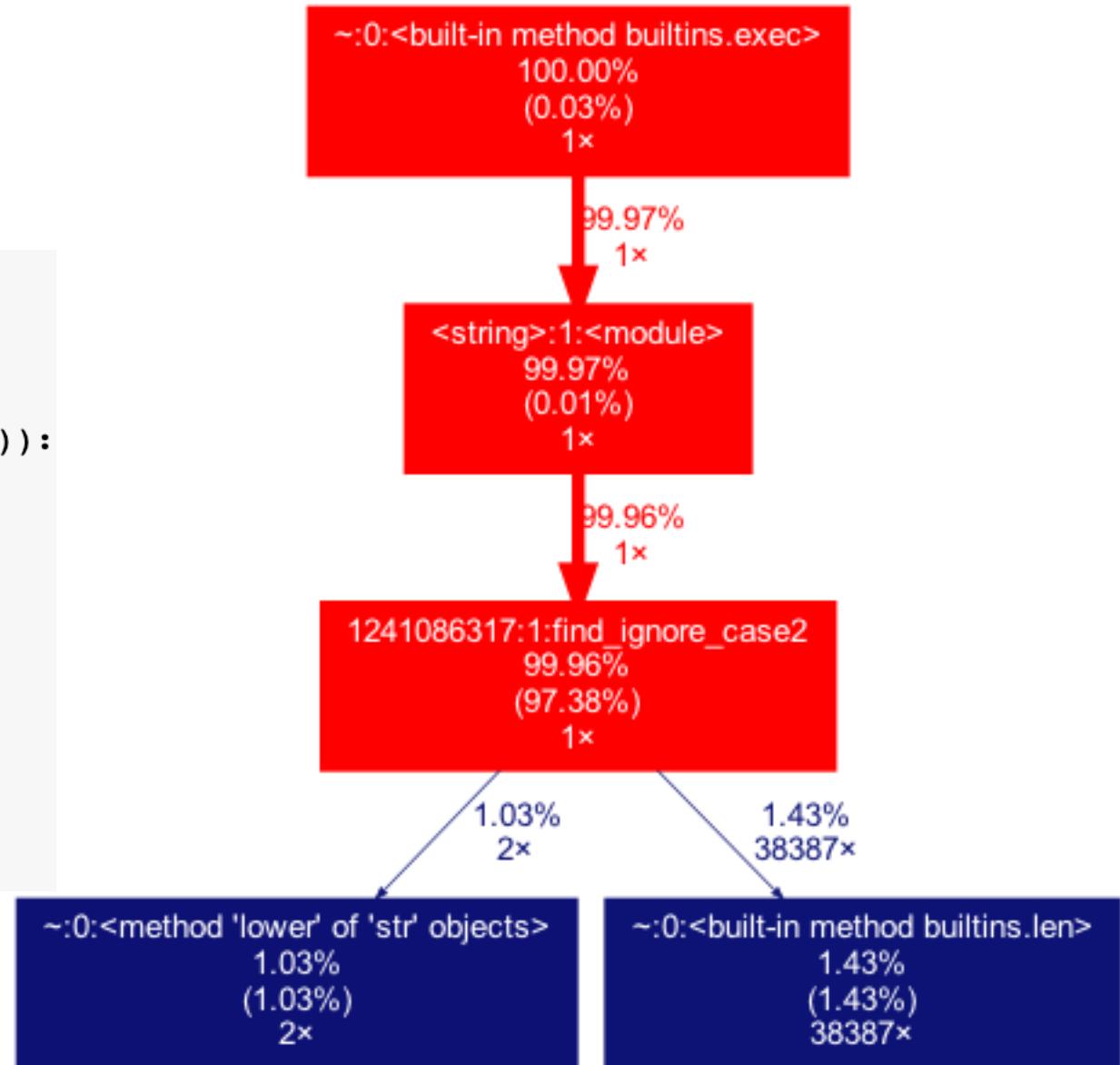
```
def find_ignore_case2( needle, haystack):
    results = []
    needle = needle.lower() # new
    haystack = haystack.lower() # new
    for hi in range(len(haystack)-len(needle)):
        match = True
        for ni in range(len(needle)):
            h = haystack[hi + ni].lower()
            n = needle[ni].lower()
            if h != n:
                match=False
        break # new
        if match:
            results.append(hi)
    return results
```

Find: 0.917540 seconds

Find2: 0.440155 seconds

# Anything else?

```
def find_ignore_case2( needle, haystack):
    results = []
    needle = needle.lower() # new
    haystack = haystack.lower() # new
    for hi in range(len(haystack)-len(needle)):
        match = True
        for ni in range(len(needle)):
            h = haystack[hi + ni].lower()
            n = needle[ni].lower()
            if h != n:
                match=False
                break # new
        if match:
            results.append(hi)
    return results
```



```
def find_ignore_case3( needle, haystack):
    results = []
    needle = needle.lower() # new
    haystack = haystack.lower() # new
    r = range(len(needle)) # new

    for hi in range(len(haystack)-len(needle)):
        match = True

        if haystack[hi] == needle[0]:
            for ni in r: # update
                h = haystack[hi + ni].lower()
                n = needle[ni].lower()
                if h != n:
                    match=False
                    break # new
            if match:
                results.append(hi)
    return results
```

Find: 0.370030 seconds  
Find2: 0.057817 seconds  
Find3: 0.053763 seconds

[New Mac Times]

```
def find_ignore_case4( needle, haystack):
    results = []
    needle = needle.lower() # new
    haystack = haystack.lower() # new
    r = range(len(needle)-1) # new

    for hi in range(len(haystack)-len(needle)):
        #match = False

        if haystack[hi] == needle[0]:
            for ni in r: # update
                h = haystack[hi + ni].lower()
                n = needle[ni].lower()
                if h == n: # new
                    #match=False
                    results.append(hi) # new
                    break # new
            #if match:
            #results.append(hi)
    return results
```

Find: 0.259516 seconds  
Find2: 0.057128 seconds  
Find3: 0.053053 seconds  
Find4: 0.048197 seconds

---

Q: How is this so much faster?

```
def find_ignore_case5( needle, haystack):
    return [haystack.find(needle)]
```

# Using built-in libraries is usually the fastest...

```
def find_ignore_case5( needle, haystack):
    return [haystack.find(needle)]
```

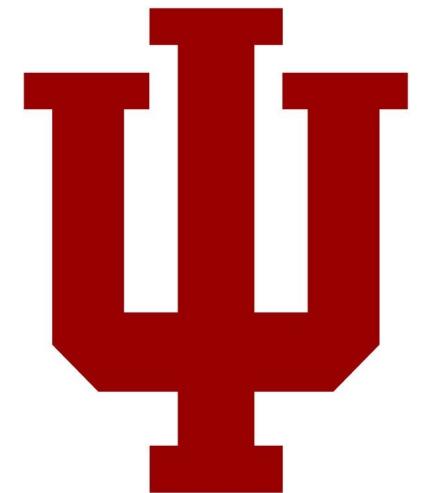
```
Find: 0.259516 seconds
Find2: 0.057128 seconds
Find3: 0.053053 seconds
Find4: 0.048197 seconds
Find5: 0.000172 seconds
```

# Coding for Performance

Engr 315: Hardware / Software Codesign

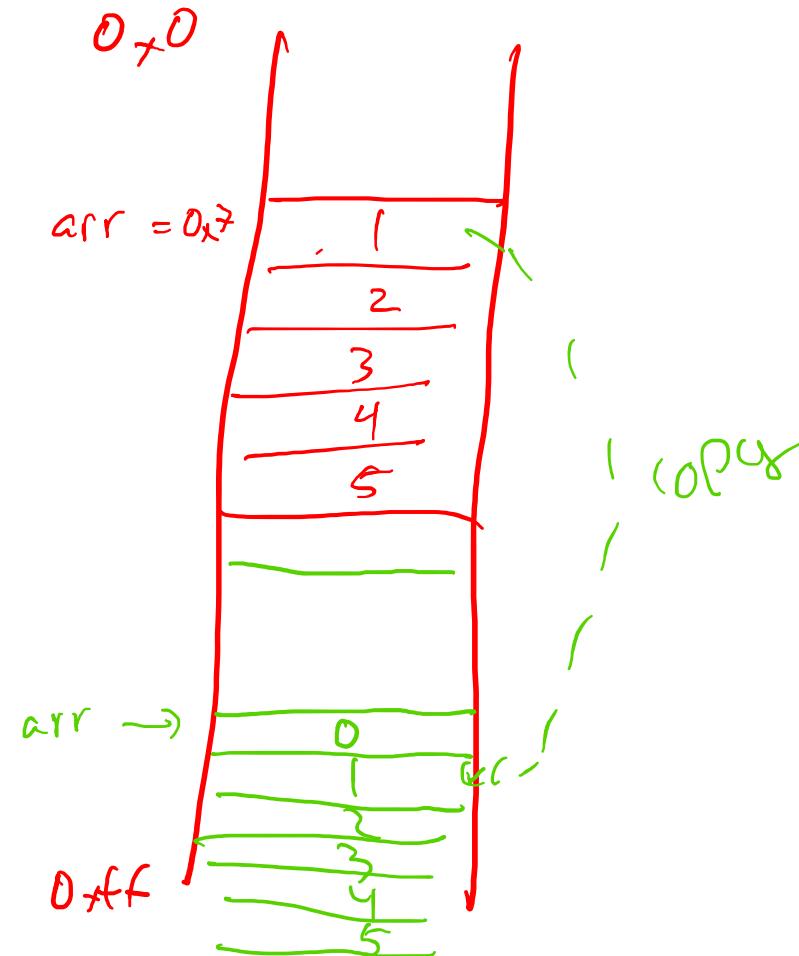
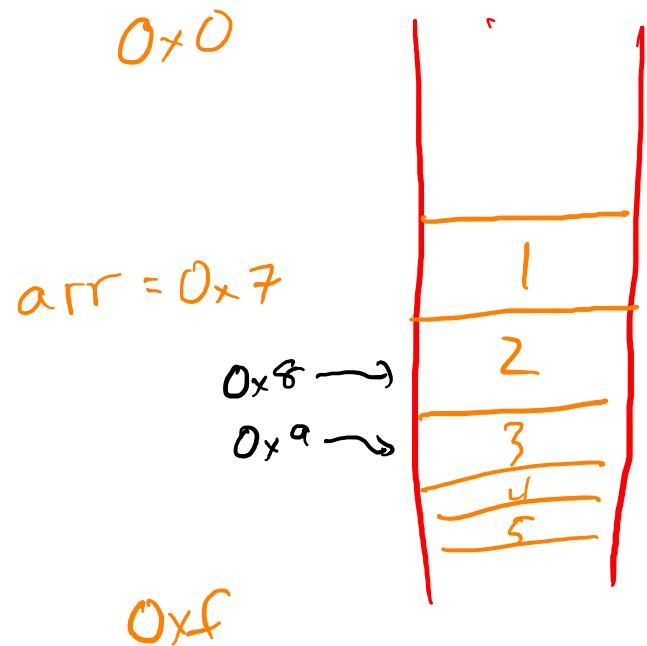
Andrew Lukefahr

*Indiana University*

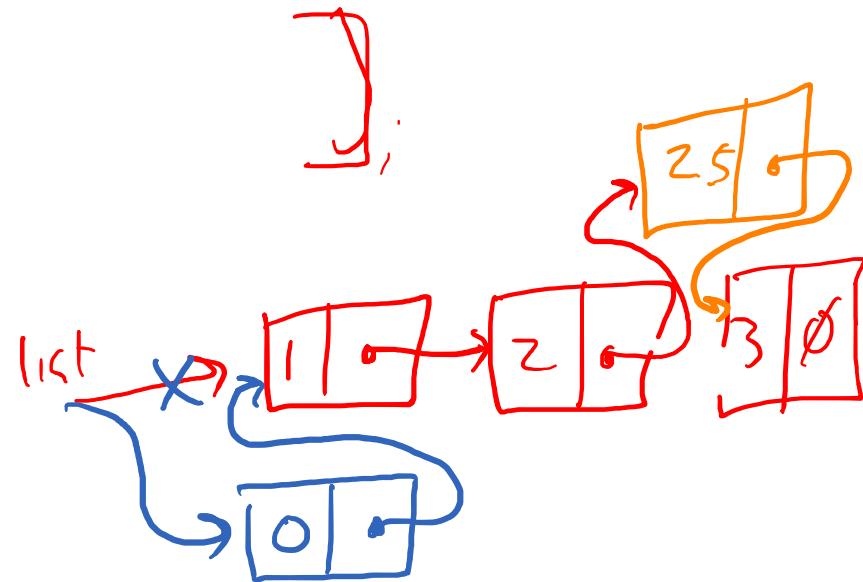
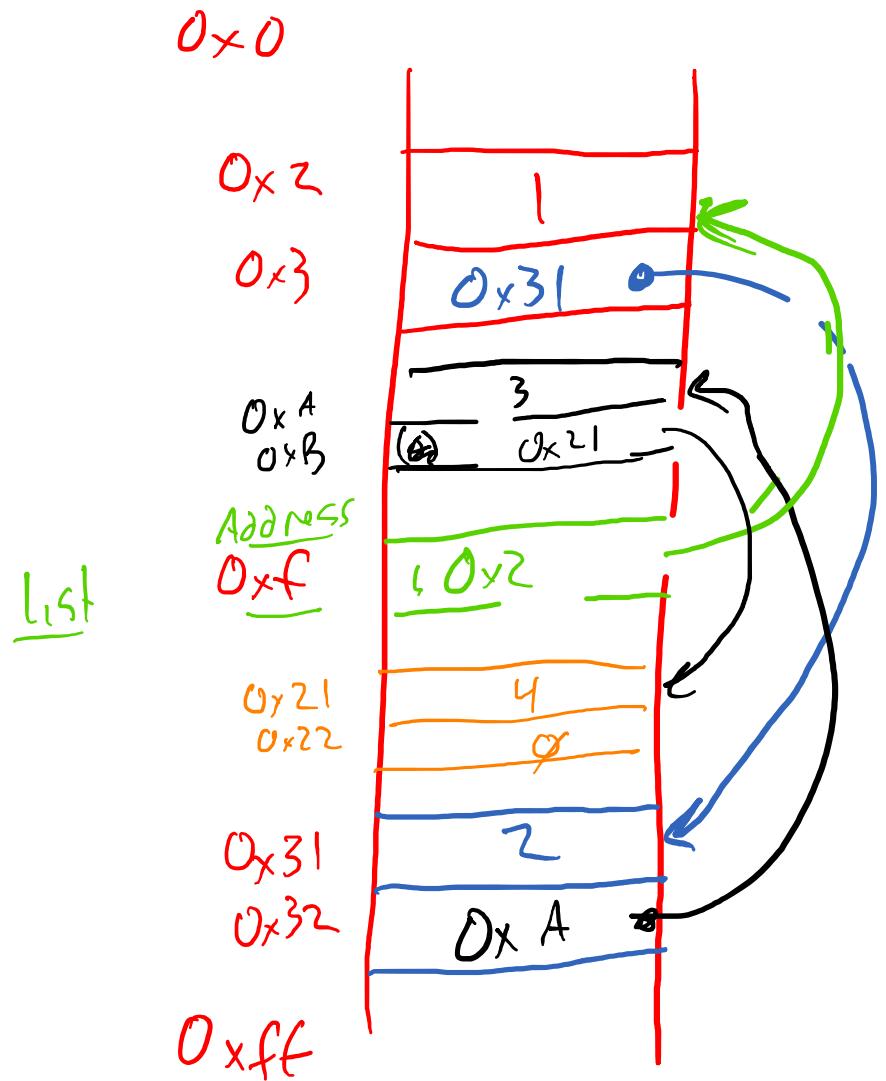


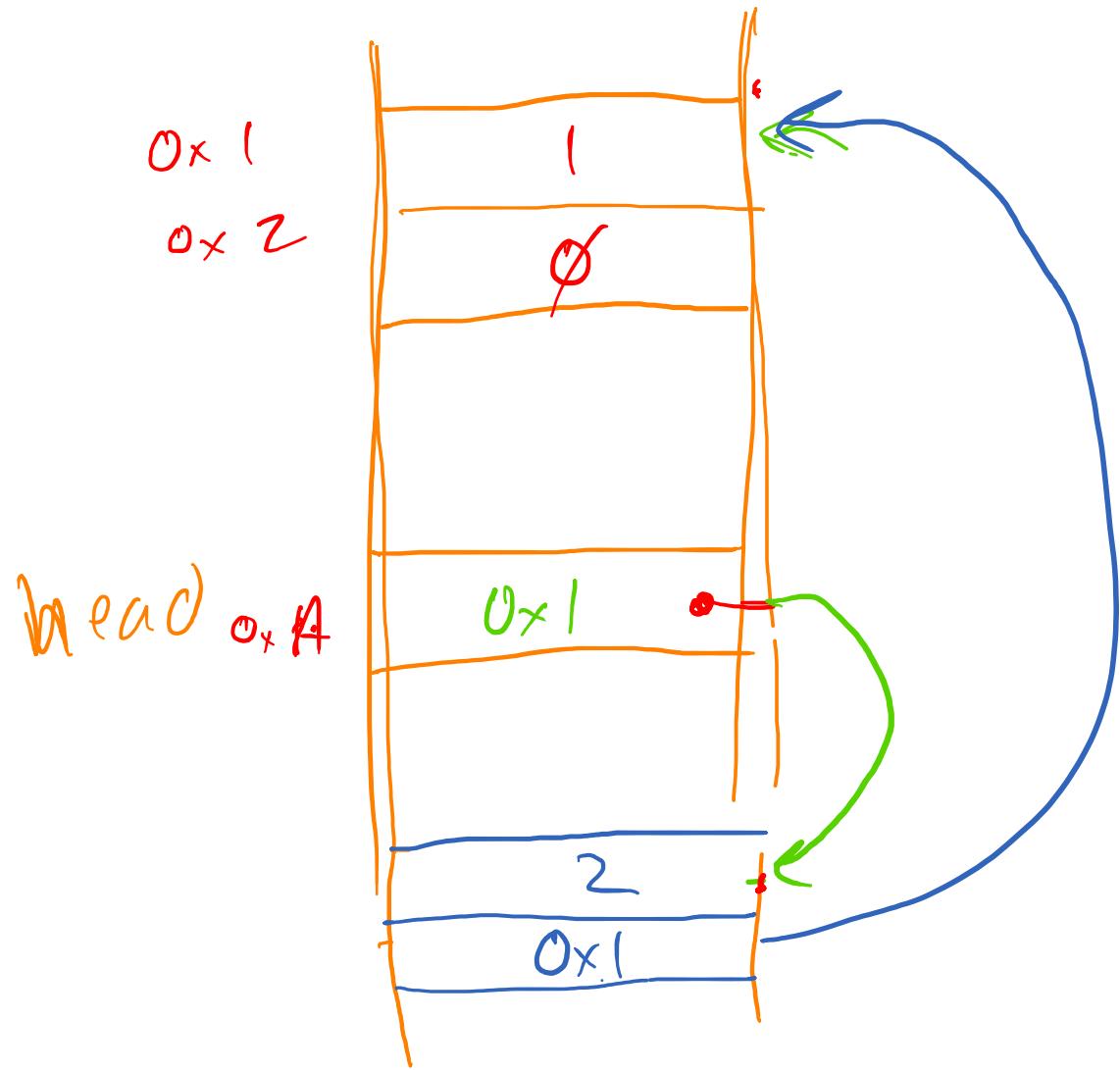
# Array

```
int arr [5] = { 1, 2, 3, 4, 5 };
```



linked list = [ 1, 2, 3,

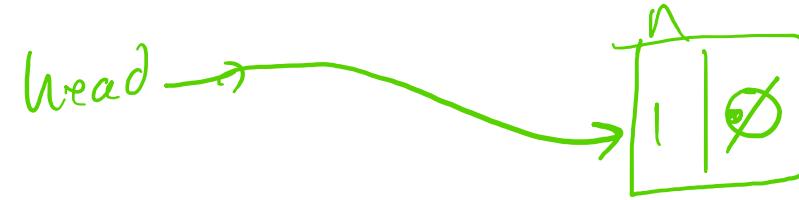




```
struct {
    int value;
    Node * next;
}; Node;
```

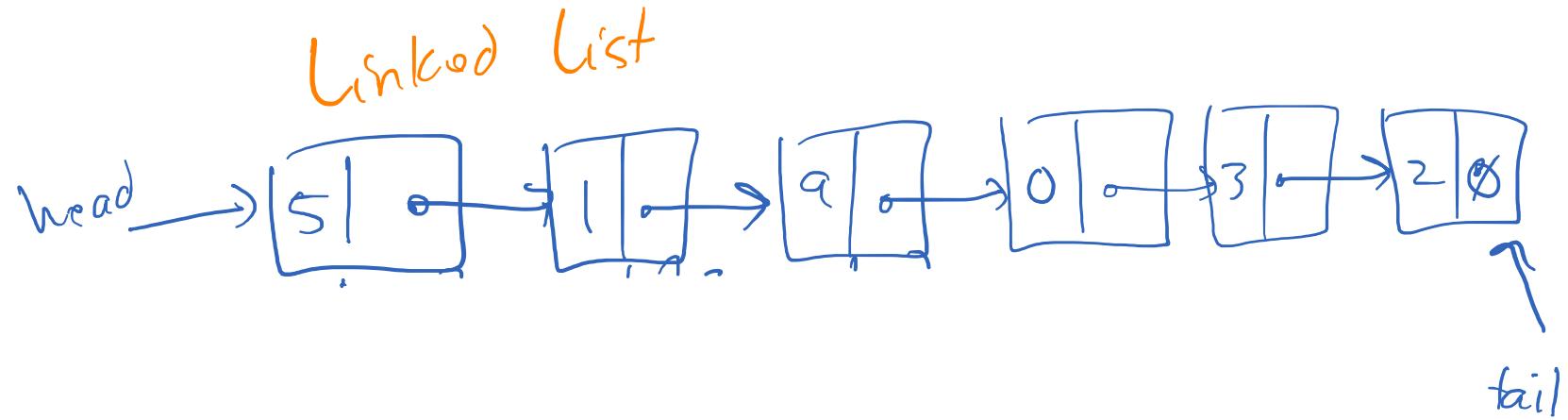
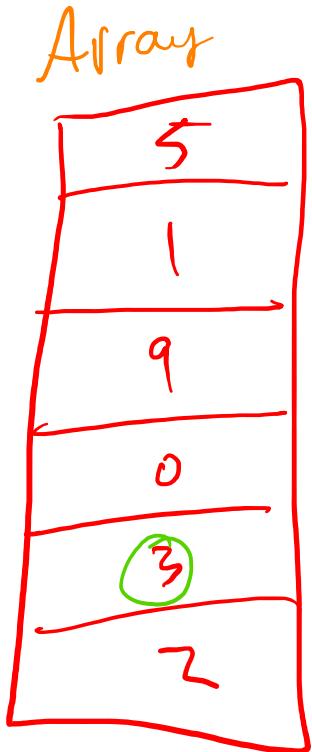
```
Node * head = NULL;
```

```
void add-node (int newValue)
{
    Node * n = malloc (sizeof(Node));
    // error check
    n->value = newValue;
    n->next = head;
    head = n;
}
```



```
int main()
{
    add-node(1);
    add-node(2);
    {
}
```

Don't forget free!



faster

go to element #4 ⇒ Array