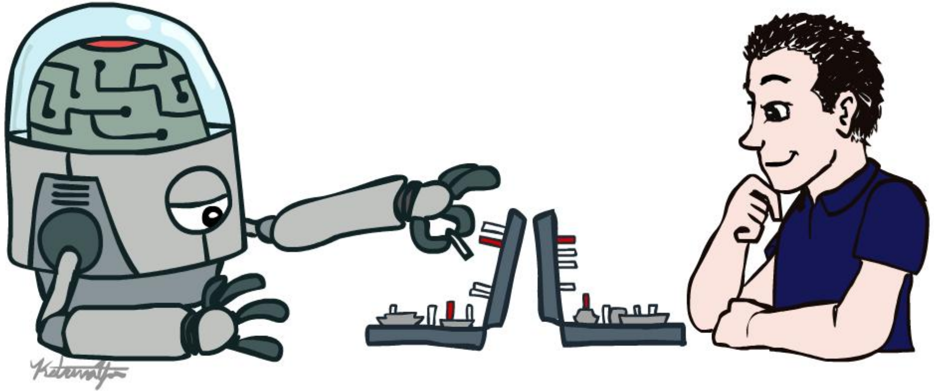
# 第一周

- **教学计划**
  - 人工智能简介（上次课）
  - Python回顾（今天）
- **任务**
  - 浏览和熟悉课程主页
  - 在自己的机器里安装Linux，Anaconda和Git，坚持使用一个学期，完成所有课程项目，自动获得**10%**的期末成绩。
  - 挑选好项目组成员（每组3~4名学生）
  - 项目0（Python热身）已经出来了，**独立完成**，尽早开始

# 人工智能导论

## Python Review and Others



# Agenda

- Runtime environment
  - Linux
  - Anaconda
  - Git
- Python review
- A couple of exercises
- Merge sort

# RUNTIME ENVIRONMENT

# Runtime environment

- Linux
  - 建议采用Linux操作系统，方便安装各种开源软件
  - 可以安装Windows/Linux双系统，启动时选择OS
  - Ubuntu：https://ubuntu.com/
- Anaconda
  - 管理软件运行环境，可以方便地在不同的环境之间切换
  - https://docs.anaconda.com/anaconda/install/
- Git
  - Free and open source distributed version control system
  - Ubuntu命令行安装：sudo apt install git
  - 可以方便地分享代码
  - https://git-scm.com/
  - Github is based Git: https://github.com/

# Linux

- 常用Linux命令
  - 路径/文件管理：pwd, cd, ls, mkdir, rmdir, mv, rm, cp, chmod, touch, find
  - 查看文件：grep, cat, head, tail, diff
  - 硬件管理：lshw, df, du, free, lsblk, cat /proc/cpuinfo, reboot, shutdown
  - 压缩文档：tar cf, tar xf, tar czf, gzip, zip, unzip
  - 进程管理：ps, jobs, kill, fg, bg, top
  - 网络：ping, traceroute, mtr, wget, curl, ip, ifconfig, ufw, nslookup
  - 用户：su, sudo, passwd, useradd, userdel, last
  - 安装：sudo apt update, sudo apt install [pkg], sudo apt remove [pkg]
  - 编辑：vim, nano, code
  - 手册：man
- Ubuntu：https://ubuntu.com/

# Anaconda

- 常用conda命令
  - 安装软件：conda install  jupyter
  - 创建环境：conda create --name py36 python=3.6
  - 激活环境：conda activate py36
  - 列出所有环境：conda env list
  - 克隆环境：conda create --clone py36 --name py36-2
  - 列出当前环境中的所有软件：conda list
  - 更新：conda update [package]
- https://docs.anaconda.com/anaconda/install/

# Git

- 常用Git命令
  - 初始化：git init
  - 克隆：git clone <repository-link>
  - 当前状态：git status
  - 查看历史：git log
  - 添加文件：git add [file/folder names]
  - 保存改动：git commit
  - 分叉：git branch
  - 换到另一个分叉：git checkout [branch-name]
  - 存入Git库：git push <remote>
  - 从Git库中取出：git pull <remote>
  - 合并：git merge
- https://git-scm.com/

# Reward

- 熟悉操作**专业**的软件开发工具和环境
- 假如你在自己的机器里安装了Linux，Anaconda和Git，并且坚持使用了一个学期，你将自动得到**10%**的期末成绩
  - Linux：5%
  - Anacoonda/Git：5%

# PYTHON

# Python

- Python
  - An interpreted, object-oriented language. (https://www.python.org/about/)
- Topics
  - Operators
  - Strings
  - Lists
  - Tuples
  - Sets
  - Dictionaries
  - Functions
  - Classes/Objects

# Operators

- Invoke python

```
(base) hyluo@einstein:~$ python
Python 3.9.13 (main, Aug 25 2022, 23:26:10)
[GCC 11.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

- Operators
  - =, +, -, *, **, /, //, %, +=, -=, /=, *=
  - &, |, ^, ~, >>, <<
  - ==, !=, >, <, >=, <=
  - and, or, not, is, is not, in, not in
- Some values
  - True, False, float(‘inf’), None, “Hello”, -1

# Strings

- Strings

```
>>> 'artificial' + "intelligence"
'artificialintelligence'
```

- Lots of build-in methods

```
>>> dir('xyz')
[...'__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith',
'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum',
'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
>>> help(str)
```

# More on strings

- Use help()

```
>>> s = 'xyz'
>>> help(s.find)
"""
Help on built-in function find:

find(...) method of builtins.str instance
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is
found,
...
>>> s.find('y')
1
```

# Lists

- Lists
  - A build-in data structure that stores a sequence of mutable items

```
>>> fruits = ['apple', 'orange', 'pear', 'banana']
>>> fruits[0]
'apple'
otherFruits = ['kiwi', 'strawberry']
>>> fruits + otherFruits
['apple', 'orange', 'pear', 'banana', 'kiwi',
'strawberry']
>>> fruits[-2]
'pear'
>>> fruits.pop()
'banana'
>>> fruits.append('grapefruit')
```

# More on lists

- Build-in methods

```
>>> dir(list)
[...
'__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'clear', 'copy', 'count',
'extend', 'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
>>> help(fruits.insert)
>>> fruits.sort(); fruits.reverse()
>>> fruits
['pear', 'orange', 'grapefruit', 'banana', 'apple']
```

# Tuples

- Tuples
  - Similar to lists, but is immutable (cannot change its content once created)
  - Can be hashed and used as keys to dictionaries

```
>>> pair = (3, 5)
>>> pair[0]
3
>>> x, y = pair
>>> x
3
>>> pair[1] = 6
TypeError: object does not support item assignment
```

# Sets

- Sets
  - Unordered list with no duplicates

```
>>> shapes = ['circle', 'square', 'triangle', 'circle']
>>> setOfShapes = set(shapes)
>>> 'circle' in setOfShapes
True
>>> setOfShapes.add('polygon')
>>> favoriteShapes = {'circle', 'triangle', 'hexagon'}
>>> setOfShapes - setOfFavoriteShapes
set(['square', 'polygon'])
>>> setOfShapes & setOfFavoriteShapes
>>> setOfShapes | setOfFavoriteShapes
```

# Dictionaries

- Dictionaries
  - key-value pairs where key must be an immutable type (string, number or tuple)

```
>>> studentIds = {'knuth': 42.0, 'turing': 56.0, 'nash': 92.0}
>>> studentIds['turing']
56.0
>>> studentIds['nash'] = 'ninety-two'
>>> studentIds.keys()
['knuth', 'turing', 'nash']
>>> studentIds.values()
[42.0, 56.0, 'ninety-two']
>>> del studentIds['knuth']
>>> studentIds
{'turing': 56.0, 'nash': 'ninety-two'}
```

# Control structures

- Python script

```
# This is what a comment looks like
fruits = ['apples', 'oranges', 'pears', 'bananas']
for fruit in fruits:
    print(fruit + ' for sale')

fruitPrices = {'apples': 2, 'oranges': 1.5, 'pears': 1.75}
for fruit, price in fruitPrices.items():
    if price < 2.00:
        print('%s cost %f a pound' % (fruit, price))
    else:
        print(fruit + ' are too expensive!')
```

# Some other features

- Functional programing: map, filter

```
>>> list(map(lambda x: x * x, [1, 2, 3]))
[1, 4, 9]
>>> list(filter(lambda x: x > 3, [1, 2, 3, 4, 5, 4, 3]))
[4, 5, 4]
```

- List comprehension construction

```
nums = [1, 2, 3, 4, 5, 6]
plusOneNums = [x + 1 for x in nums]
oddNums = [x for x in nums if x % 2 == 1]
print(oddNums)
oddNumsPlusOne = [x + 1 for x in nums if x % 2 == 1]
print(oddNumsPlusOne)
```

# Functions

- Functions

```
fruitPrices = {'apples': 2, 'oranges': 1.5, 'pears': 1.75}

def buyFruit(fruit, numPounds):
    if fruit not in fruitPrices:
        print(f"Sorry we don't have {fruit}")
    else:
        cost = fruitPrices[fruit] * numPounds
        print(f"That'll be {cost} please")

# Main Function
if __name__ == '__main__':
    buyFruit('apples', 2.4)
    buyFruit('coconuts', 2)
```

# Classes

- File person_class.py

```
class Person:
    population = 0

    def __init__(self, myAge):
        self.age = myAge
        Person.population += 1

    def get_population(self):
        return Person.population

    def get_age(self):
        return self.age
```

# Objects

- Use import

```
>>> import person_class
>>> p1 = person_class.Person(12)
>>> p1.get_population()
1
>>> p2 = person_class.Person(63)
>>> p1.get_population()
2
>>> p2.get_population()
2
>>> p1.get_age()
12
>>> p2.get_age()
63
```

# A COUPLE OF EXERCISES

# Prime numbers

- Prime numbers
- Given n, find all prime numbers <= 1000

```
def prime(n):
    ....
```

# Prime numbers - solution

- Prime numbers

```
def prime(n):
    for i in range(2, n//2+1):
        if n%i == 0:
            return False

    return True

if __name__ == ”__main__”:
    for i in range(2, 1001):
        if prime(i):
            print(i, end=” ”)
```

# Fibonacci sequence

- Fibonacci sequence
    - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
    - a[0] = 0
    - a[1] = 1
    - a[n] = a[n-1]+a[n-2]
- Given n, produce the a[n] of the Fibonacci sequence

```
def fib(n):
    ....
```

# Fibonacci Sequence - Solution 1 (iteration)

- Given n, produce the a[n] of the Fibonacci sequence

```
def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    a, b = 0, 1
    print(f"{a}\n{b}")
    for i in range(2, n+1):
        c = a+b
        a, b = b, c
        print(c)
    return c
if __name__ == "__main__":
    fib(100)
```

# Fibonacci Sequence - Solution 2 (Recursion)

- Given n, produce the a[n] of the Fibonacci sequence

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

if __name__ == "__main__":
    fib(10)
```

- This naive implementation is very inefficient
  - Try fib(100) and it takes a long time
  - why? See board for illustruation
  - How can we fix it?

# Fibonacci Sequence - Solution 3 (Recursion)

- Use a dictionary to remeber already computed value

```
fib_dict = {0: 0, 1: 1}
def fib(n):
    if n < 0: return 0
    if n in fib_dict:
        return fib_dict[n]

    fib_dict[n] = fib(n-1)+fib(n-2)
    return fib_dict[n]

if __name__ == "__main__":
    fib(100)
```

- Now, it should be as fast as solution 1

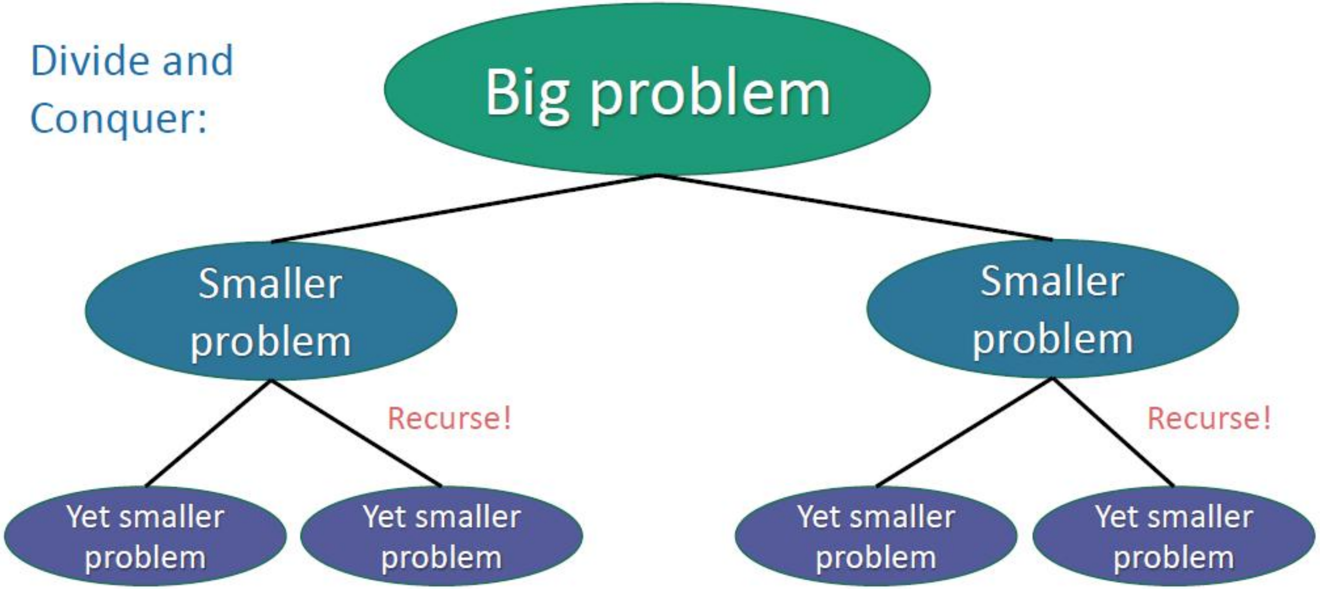
# MERGE SORT

Adopted from Stanford CS161 Algorithms

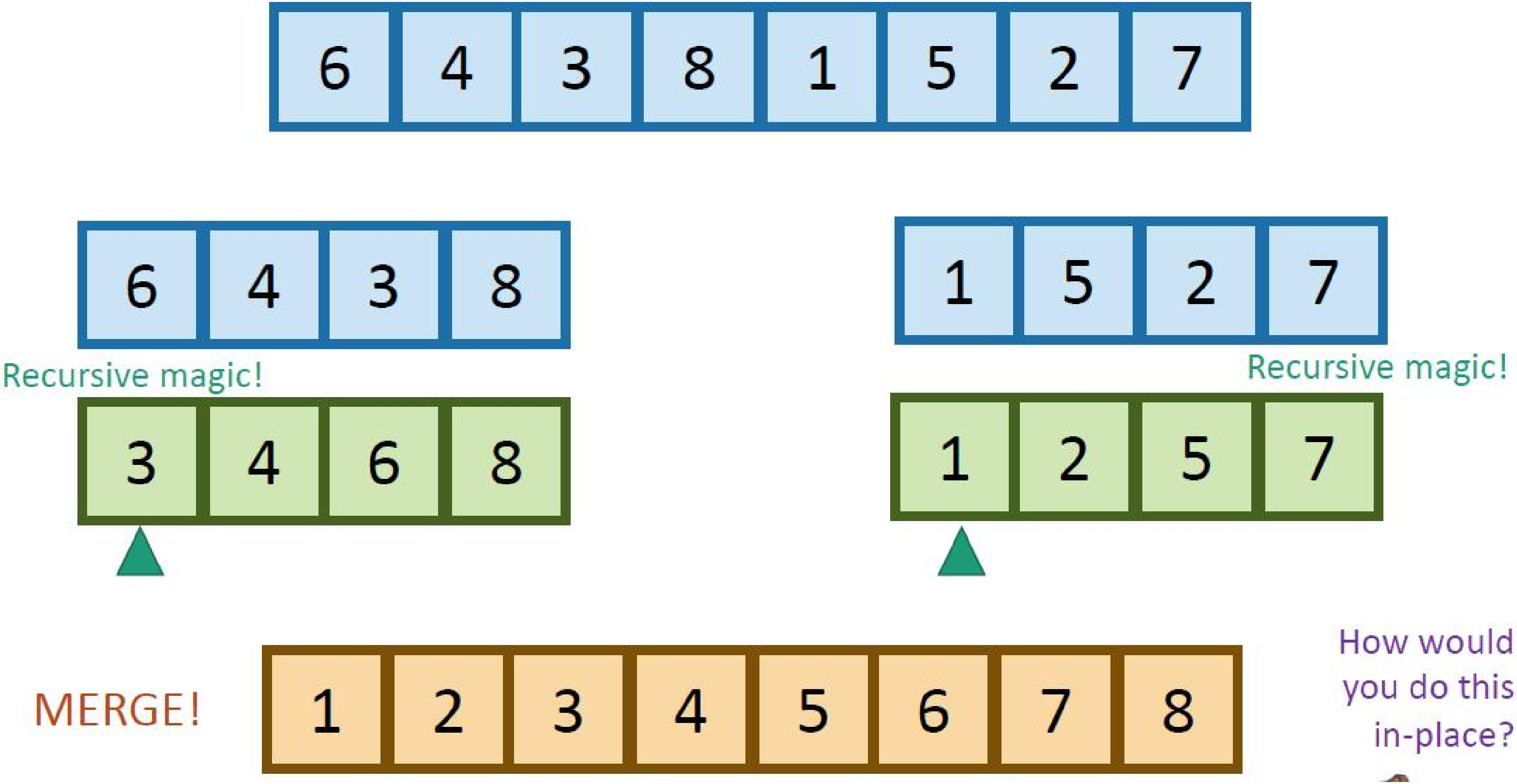
# Divide and Conquer

- MergeSort: a divide-and-conquer approach
- Recall from last time:

Divide and Conquer:



# Merge Sort



6 4 3 8 1 5 2 7

6 4 3 8

1 5 2 7

Recursive magic!

Recursive magic!

3 4 6 8

1 2 5 7

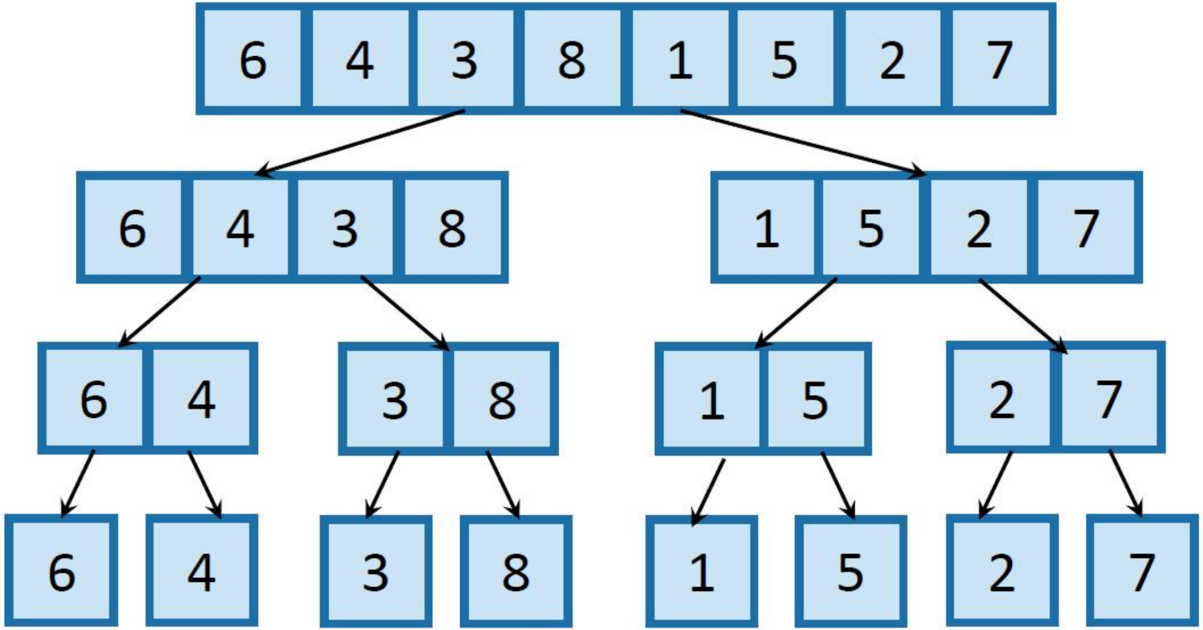
MERGE!

1 2 3 4 5 6 7 8

How would you do this in-place?

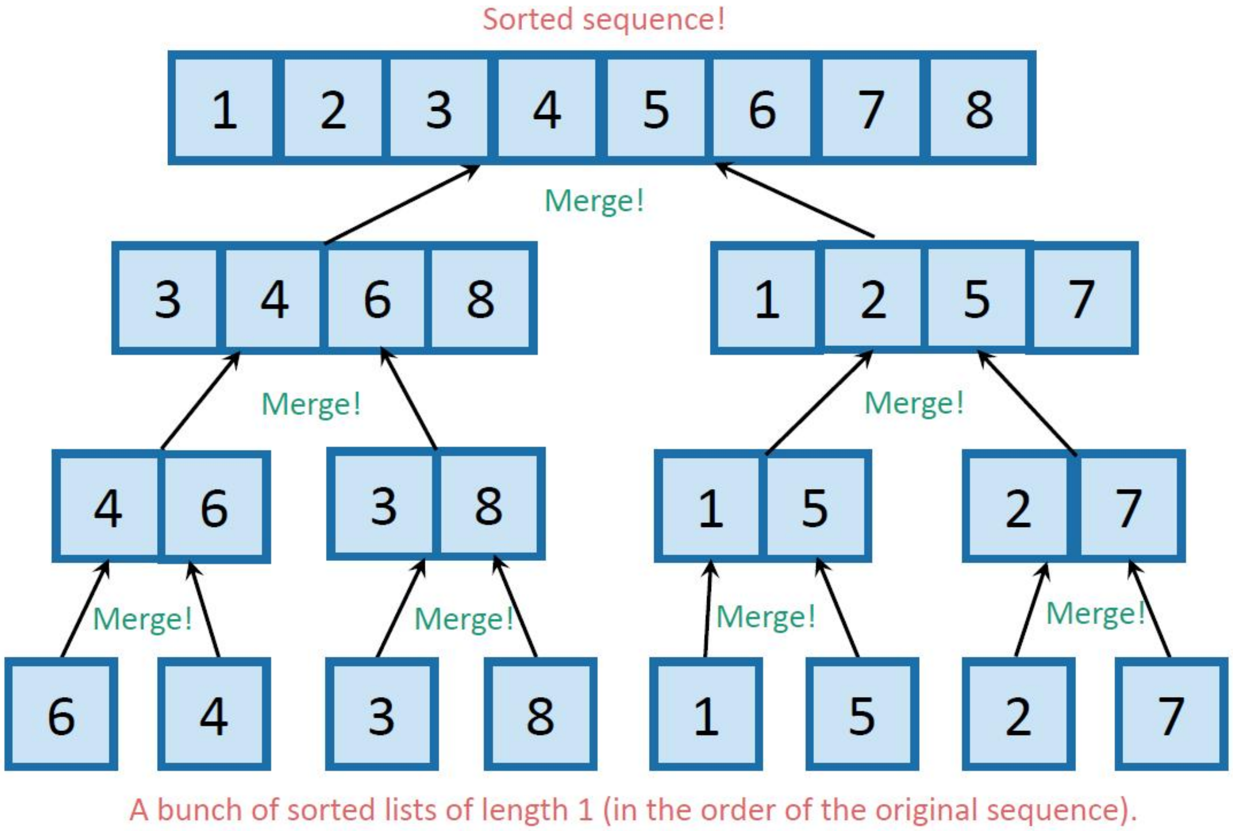
# First Divide Them Up

First, recursively break up the array all the way down to the base cases



This array of length 1 is sorted!

# Then, Merge Them All Back Up



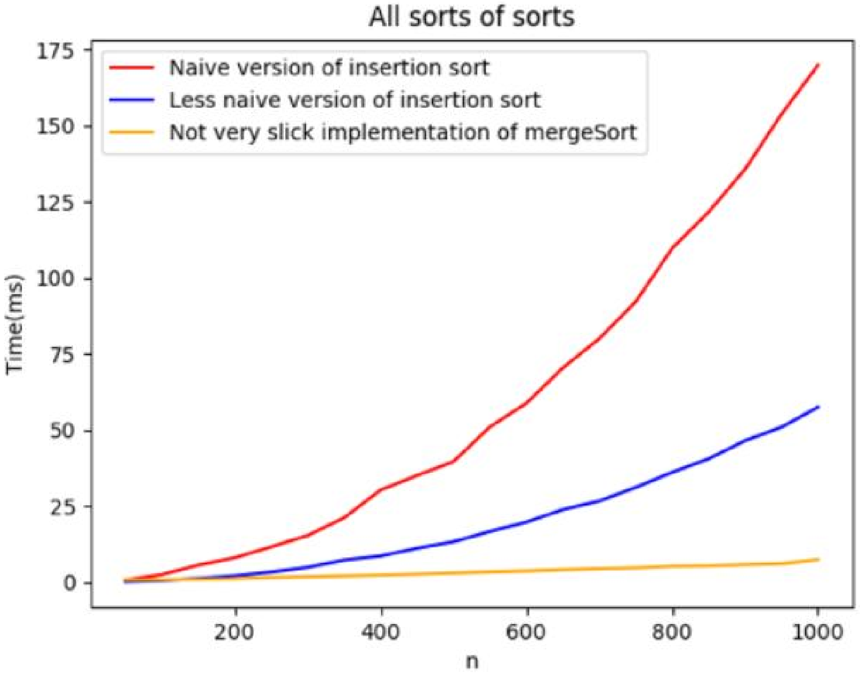
# Pseudo Code

```
MERGESORT(A):
  • n = length(A)
  • if n ≤ 1:                      If A has length 1,
                                   It is already sorted!
      • return A
  • L = MERGESORT(A[ 0 : n/2])     Sort the left half
  • R = MERGESORT(A[n/2 : n ])     Sort the right half
  • return MERGE(L,R)              Merge the two halves
```

# Two Questions

1. Does this work?
2. Is it fast?

IPython notebook says…



Empirically:

1. Seems to work.
2. Seems fast.

# Yes, It Works

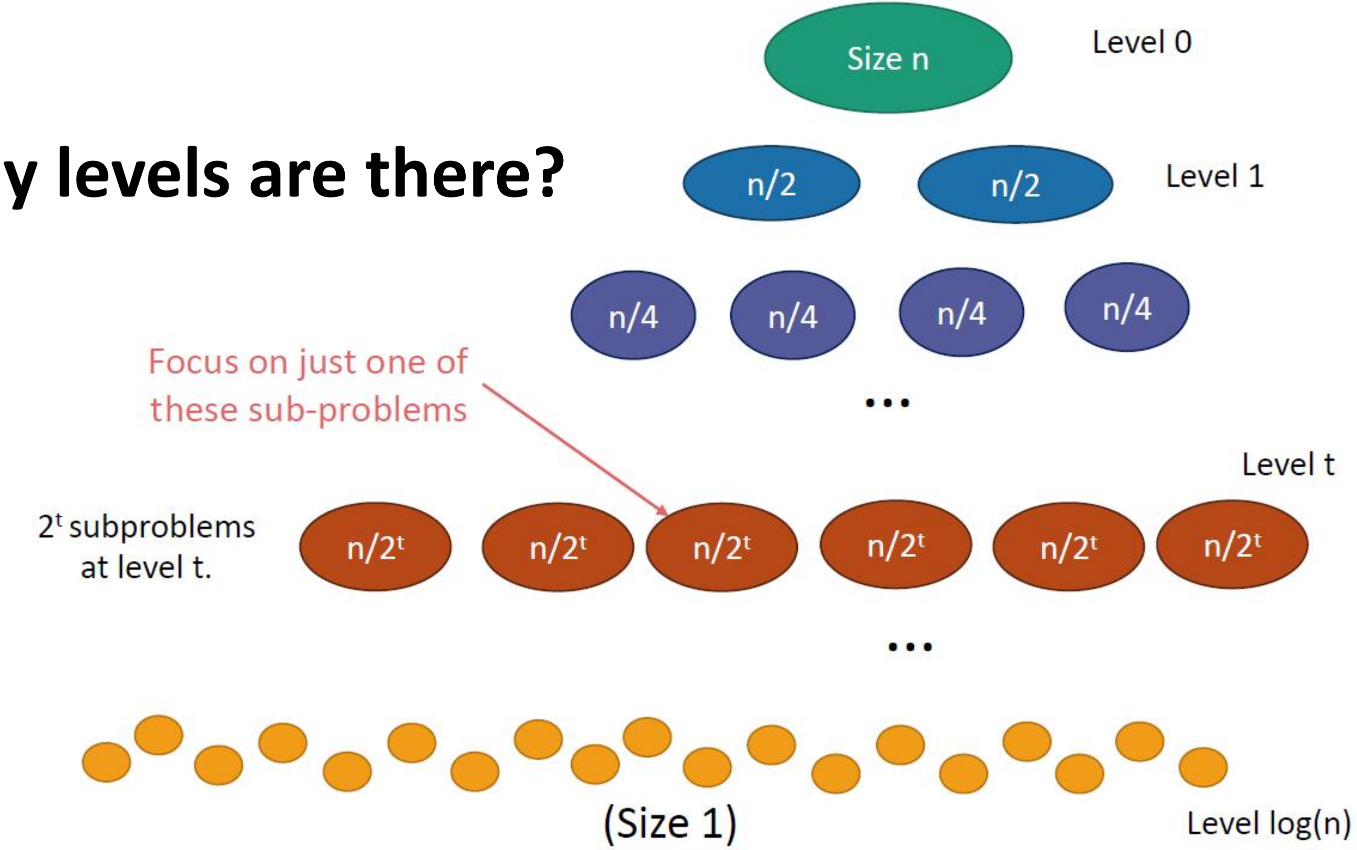
- Yet another job for…

**Proof By Induction!**

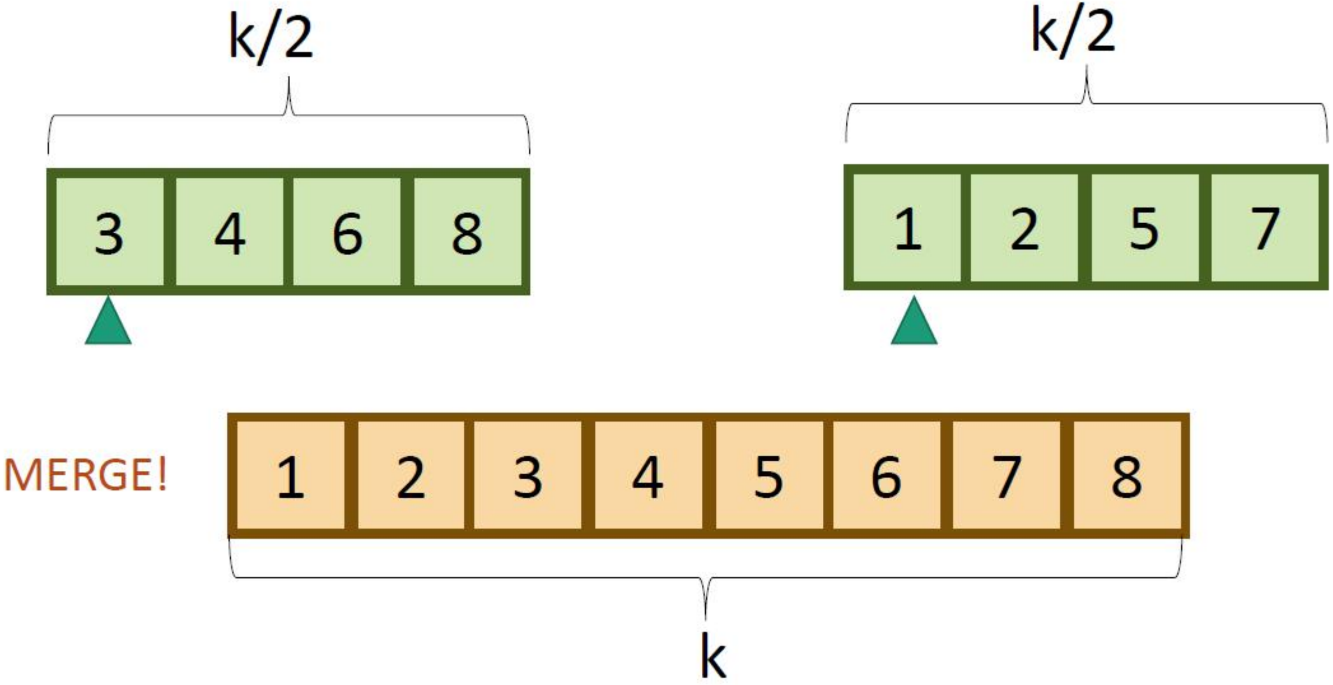
Work this out! There's a skipped slide with an outline to help you get started.

# How Fast is It?

## How many levels are there?

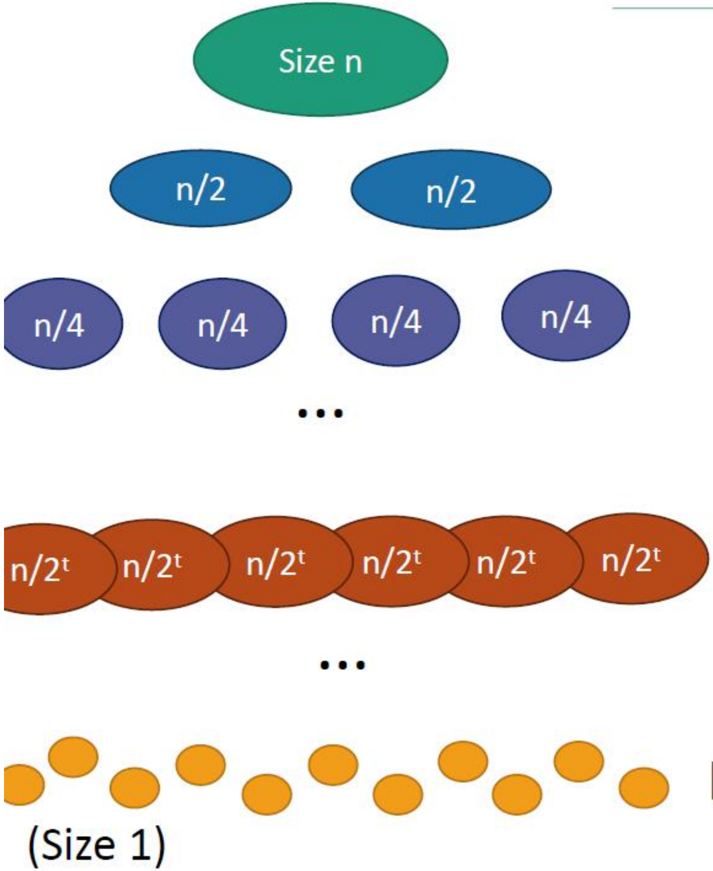


# How Much Work in Each Level?



# Recursion tree

Work this out yourself!



| Level | # problems | Size of each problem | Amount of work at this level |
|---|---|---|---|
| 0 | 1 | n | O(n) |
| 1 | 2 | n/2 | O(n) |
| 2 | 4 | n/4 | O(n) |
| ... | | | |
| t | $2^t$ | $n/2^t$ | O(n) |
| ... | | | |
| log(n) | n | 1 | O(n) |

# Total Runtime

- O(n) steps per level, at every level

- log(n) + 1 levels

- O( n log(n) ) total!

# log(n) vs. n

- Def: log(n) is the number so that $2^{\log(n)} = n.$
- Intuition: log(n) is how many times you need to divide n by 2 in order to get down to 1.

$$32, \underbrace{16, 8, 4, 2, 1}_{\text{Halve 5 times}} \quad \Rightarrow \quad \log(32) = 5$$

$$64, \underbrace{32, 16, 8, 4, 2, 1}_{\text{Halve 6 times}} \quad \Rightarrow \quad \log(64) = 6$$

log(128) = 7

log(256) = 8

log(512) = 9

….

log(**# particles in the universe**) < 280

- log(n) grows very slowly!

# Recap

- Reviews
  - Linux enviroment
  - Anaconda
  - Git
  - Python
- Programming
  - Prime numbers
  - Fibonacci sequence
  - Merge sort

# Next Time

- Ready for the real AI materials?

**Search Algorithms!**

  - Depth first search
  - Breath fist search

- Project 0 (Python warmup): individual, due next Wednesday!
- Find partners and form a group
- Runtime environment: Linux/Anaconda/Git