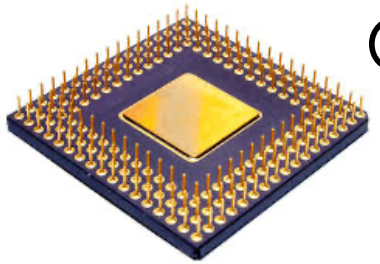
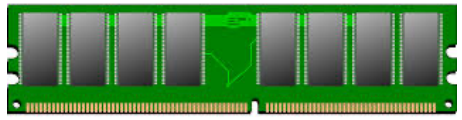
What is programming?

- Writing instructions for computers to perform tasks



CPU

Execute instructions



Memory

Data
Instructions

Instructions

- operators
- for loops
- if-else conditionals
- functions

…

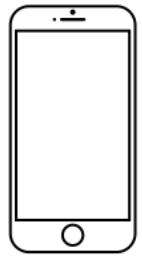
Data

- lists
- tuples
- dictionaries
- sets

…

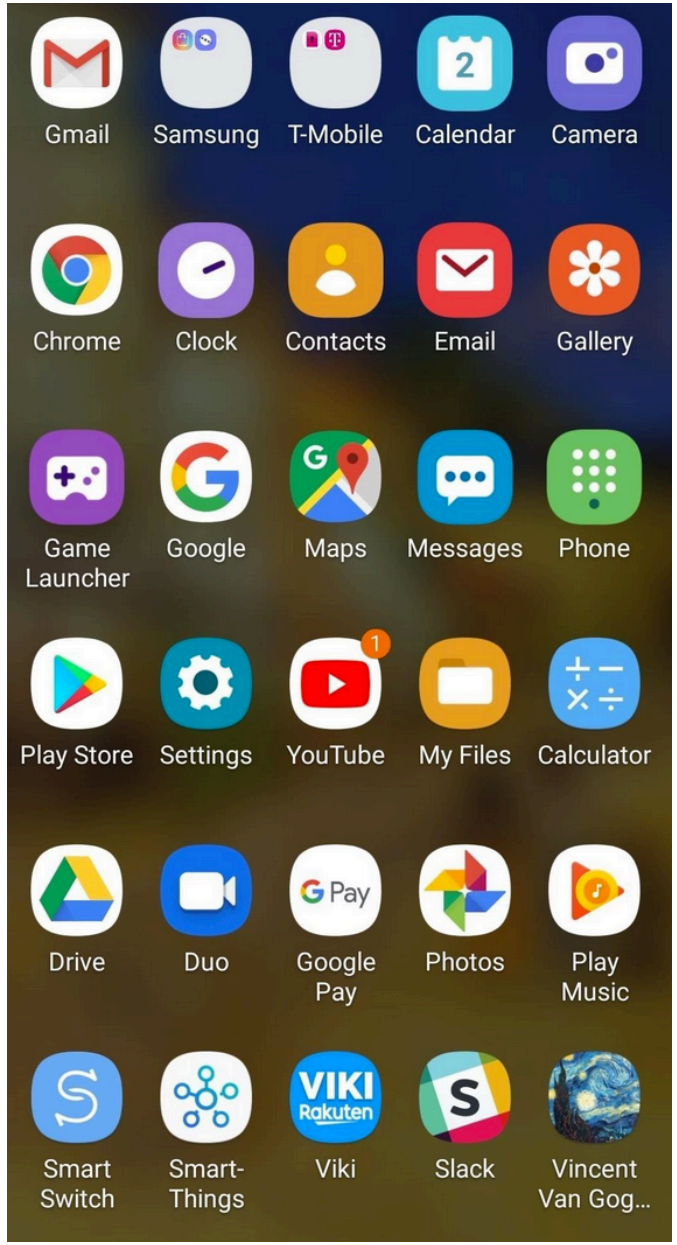
Classes

Scripts
Modules,
Packages,
Libraries




App Store
Safari
Mail
Contacts
Calendar
Reminders
Notes
Maps
Messages
FaceTime
Photo Booth
Photos
Preview
Home
iTunes
Voice Memos
GarageBand
iMovie
Pages
Numbers
Keynote
News
Stocks
Books
Dictionary
Calculator
Siri
Mission Control
System Preferences
Other
Google Chrome
Firefox
TinkerTool
Opera
Signal


heart rate
135
35:19
fitbit


Gmail
Samsung
T-Mobile
Calendar
Camera
Chrome
Clock
Contacts
Email
Gallery
Game Launcher
Google
Maps
Messages
Phone
Play Store
Settings
YouTube
My Files
Calculator
Drive
Duo
Google Pay
Photos
Play Music
Smart Switch
Smart-Things
Viki
Slack
Vincent Van Gog...

```
79
80
81      %%%%%%%%%%%%%%%%%%%%%%%%
82      % pos input
83
84 -    d = 2*pi/n_input_azimuth;
85
86 -    mamp = 0.35;
87
88 -    if use_2D_input && n_input_elevation>2
89 -        mamp = mamp*0.7;
90 -    end
91
92 -    if contains(rule_id_list{rule_id}, 'Pre-synaptic')
93 -        mamp = mamp/3;
94 -    end
95
96 -    disp(['max amp of input=' num2str(mamp)]);
97
98      %mc = xpos_radian;
99
100-    mcshift = ceil(mod(xpos_radian,2*pi)/(2*pi)*size(sim_visual_neurons,1));
101
102-    testing_if_input_is_strong_enough = 0;
103-    if testing_if_input_is_strong_enough
104-        id1 = ceil(numel(xpos_radian)/400*110);
105-        id2 = ceil(numel(xpos_radian)/400*350);
106-        mcshift(id1:id2) = mcshift(id1:id2) + ceil(size(sim_visual_neurons,1)/2);
107-    end
108
109
```

JupyterLab Reference | syn_counter.ipynb

Code | Python 3

[4]:
```
import pandas as pd              # to read in the synapse data
import numpy as np               # for numerical data
import matplotlib.pyplot as plt  # for plotting
import os                        # for OS specific operations such as directory/file manipulations

# The directory of this file. Make sure you have a "figures" sub-directory in this directory.
os.chdir('/Users/kims11/Documents/SS_research/codes/projects/synapses')
syn_path = os.getcwd()
```

[6]:
```
# V0 is based on the segmentation with the following timestamp:
#     2020-03-29T06:32:03.479060
# In Python:
# datetime.datetime(2020, 3, 29, 6, 32, 3, 479060)
# Timezone in our system is always UTC.
# One can pin a timezone in neuroglancer to view the segmentation at point in time.

#flywire_buhman_wiring_v0.csv --> 2020-03-29T06:32:03.479060 --> datetime.datetime(2020, 3, 29, 6, 32, 3, 479060)
#flywire_buhman_wiring_v1.csv --> 2020-05-02T06:26:05.780341 --> datetime.datetime(2020, 5, 2, 6, 26, 5, 780341)
#flywire_buhman_wiring_v2.csv --> 2020-05-13T23:39:05.100982 --> datetime.datetime(2020, 5, 13, 23, 39, 5, 100982)

##########################################################################
# Put the .csv data in the "syn_path" directory defined above. Then continue running code.
#syn_fn = os.path.join(syn_path, 'flywire_buhman_wiring_v0.csv'); # old version
syn_fn = os.path.join(syn_path, 'flywire_buhman_wiring_v3.csv');
print(syn_fn)

# Read the data
#syn_df = pd.read_csv(syn_fn)#, dtype="uint64") # Use "uint64" to preserve the precision of the synapse data. This reading process should take 5 minutes or l
syn_df = pd.read_csv(syn_fn), dtype=np.uint64) # Use "uint64" to preserve the precision of the synapse data. This reading process should take 5 minutes or le
```

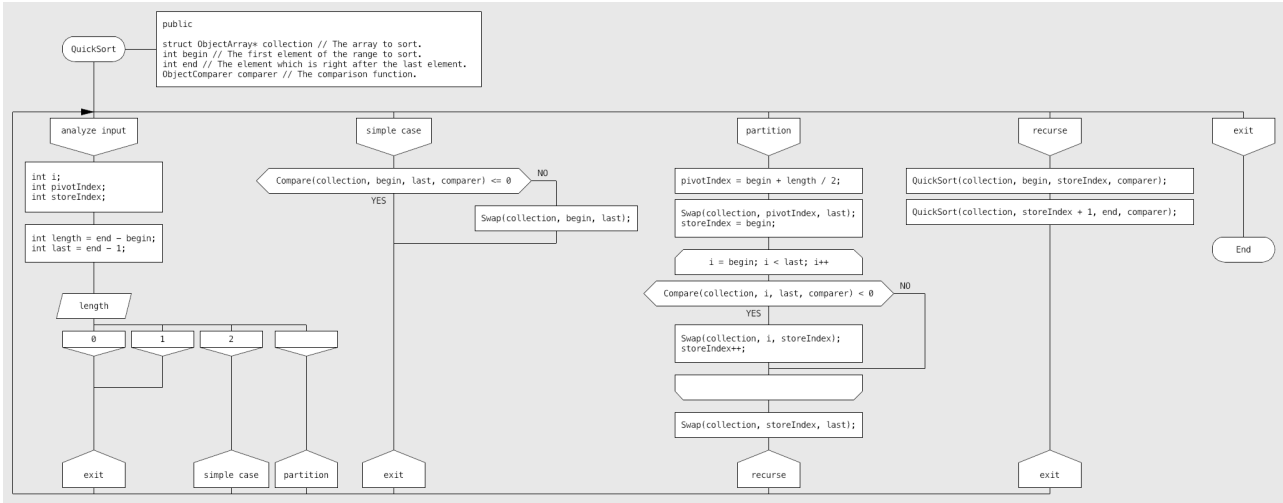
/Users/kims11/Documents/SS_research/codes/projects/synapses/flywire_buhman_wiring_v3.csv

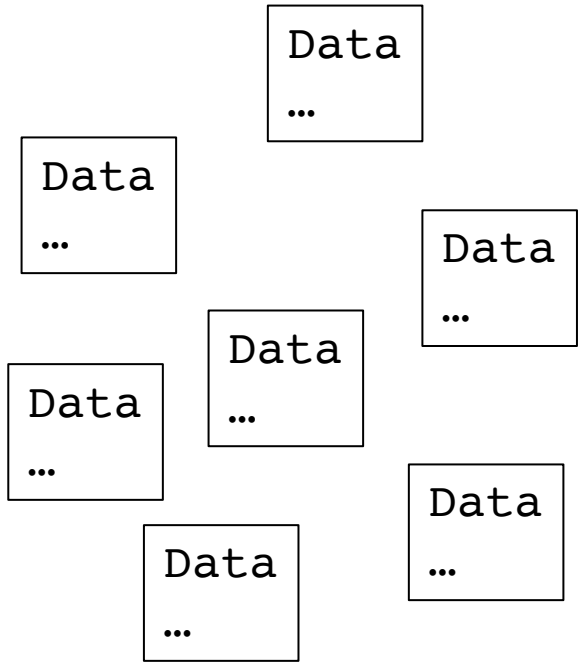
[7]:
```
# A basic Panda dataframe command
syn_df.head()
```

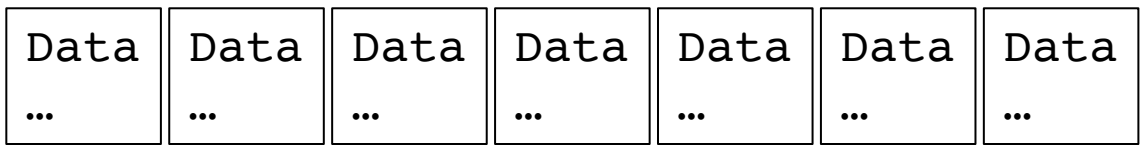
[7]:

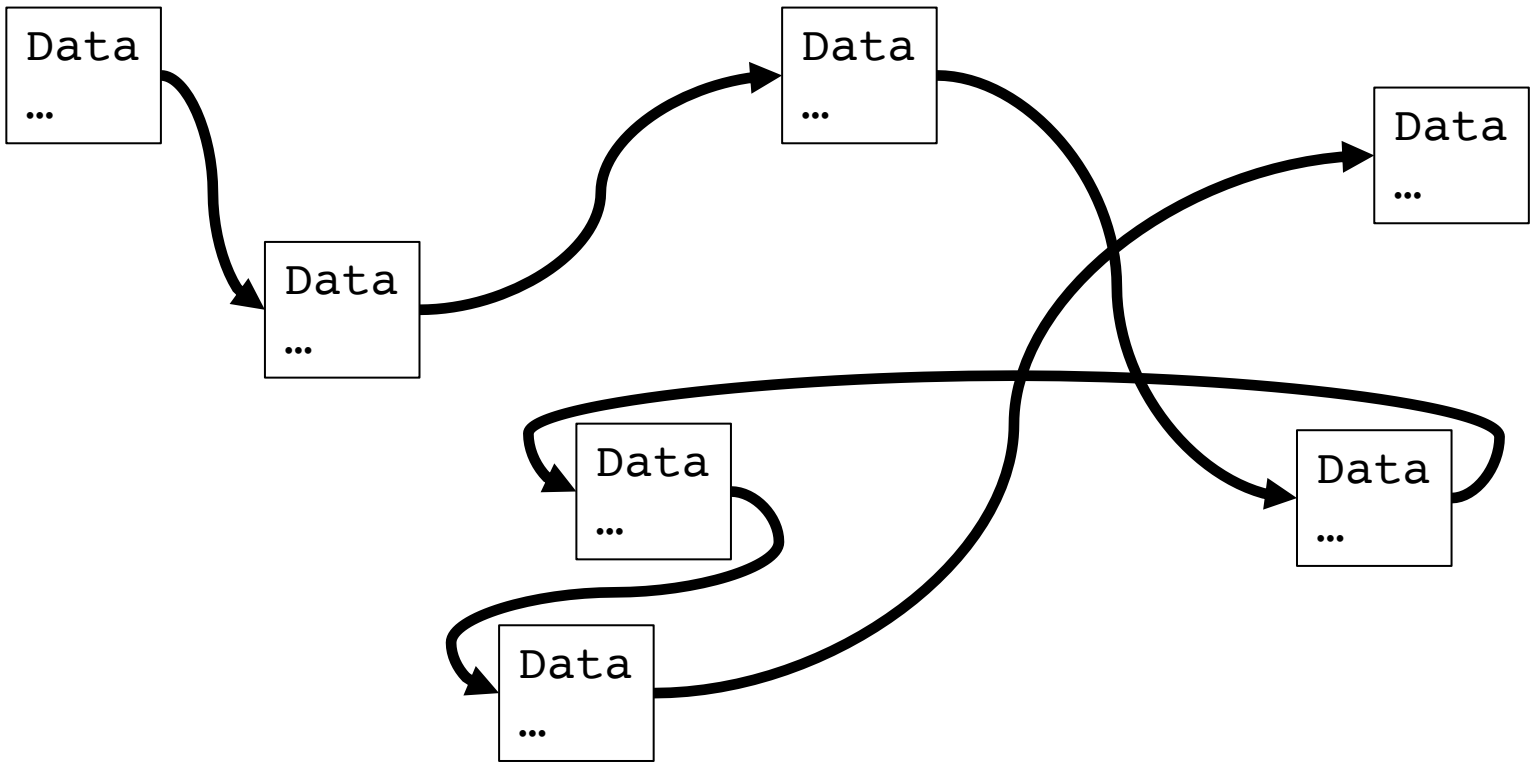
	Unnamed: 0	pre_pt_root_id	pre_pt_sv_id	pre_pt_position	ctr_pt_position	post_pt_root_id	post_pt_sv_id	post_pt_position
0	0	720575940620535651	76916748439627068	[366196 244760 73920]	[366146 244770 73900]	720575940631123034	76916748439628340	[366296 244740 73960]
1	1	720575940620535651	76916748439635341	[364864 245640 73960]	[364942 245666 73960]	720575940616174657	76916748439614229	[364708 245588 73960]
2	2	720575940616174657	76916748439614229	[364564 244904 73960]	[364506 244936 73940]	720575940545890960	76916748439623112	[364680 244840 74000]
3	3	720575940620535651	76916748439632311	[364600 244468 74200]	[364624 244530 74200]	720575940627209594	76916748439633329	[364552 244344 74200]
4	4	720575940636149614	76916748439652476	[364952 244764 74240]	[364998 244748 74260]	720575940620535651	76916748439632348	[364860 244796 74200]

[ ]:
```
# A basic Panda dataframe command
syn_df.tail()
```

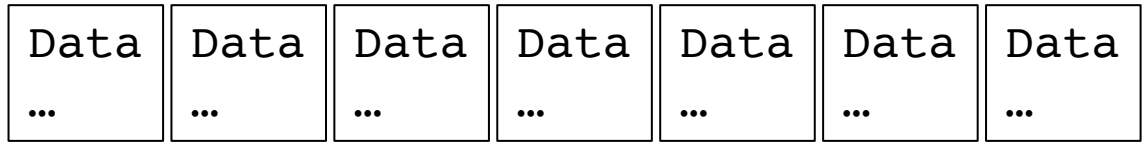


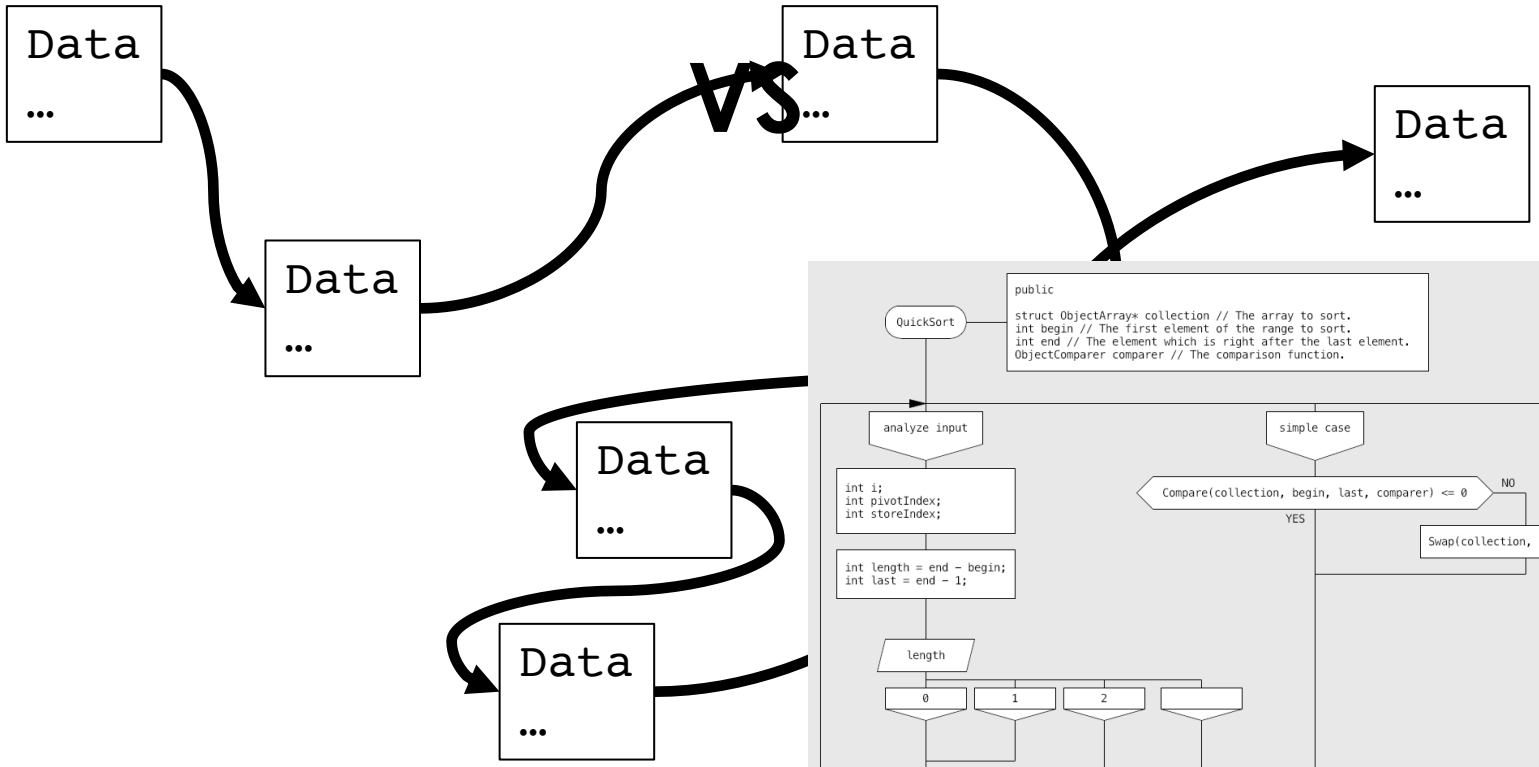

Data
…
Data
…
Data
…
Data
…
Data
…
Data
…
Data
…


Data
…
Data
…
Data
…
Data
…
Data
…
Data
…
Data
…

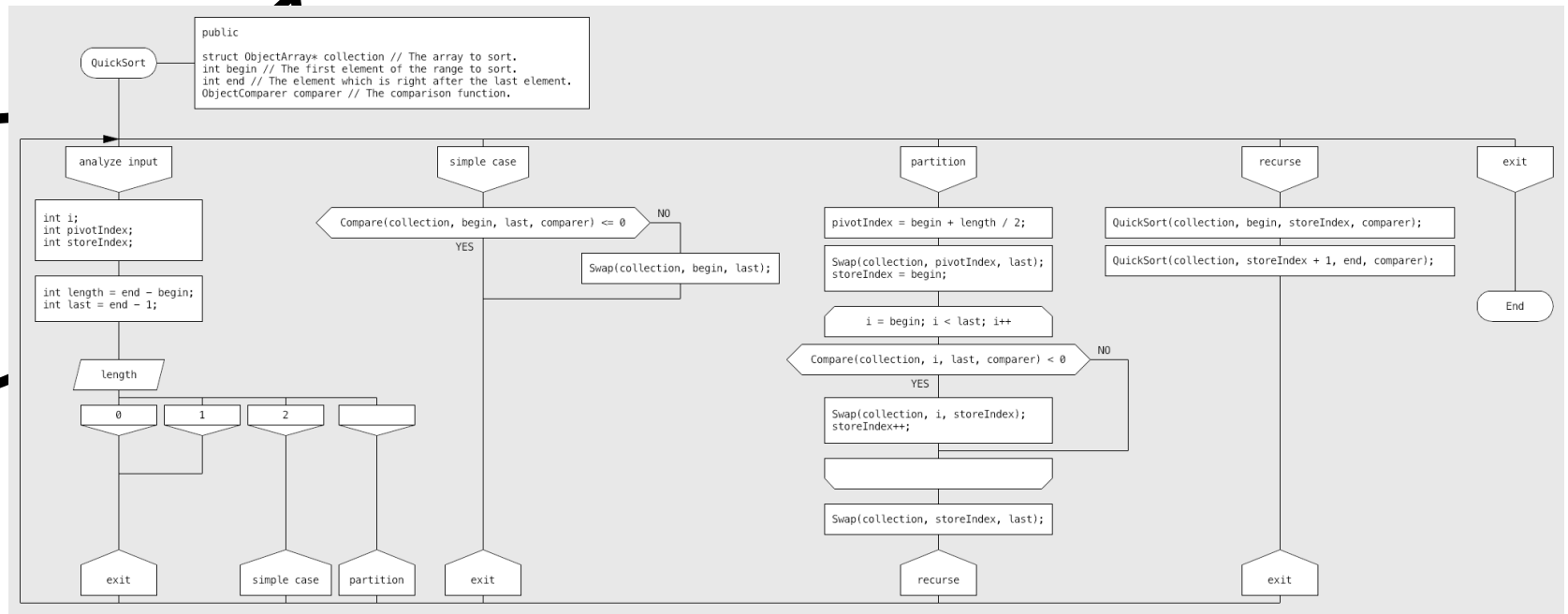

Data
…
Data
…
Data
…
Data
…
Data
…
Data
…
Data
…

Data Structures

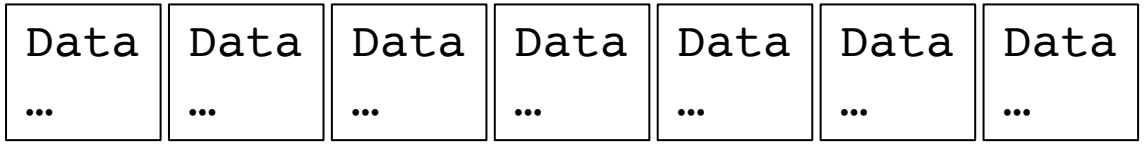




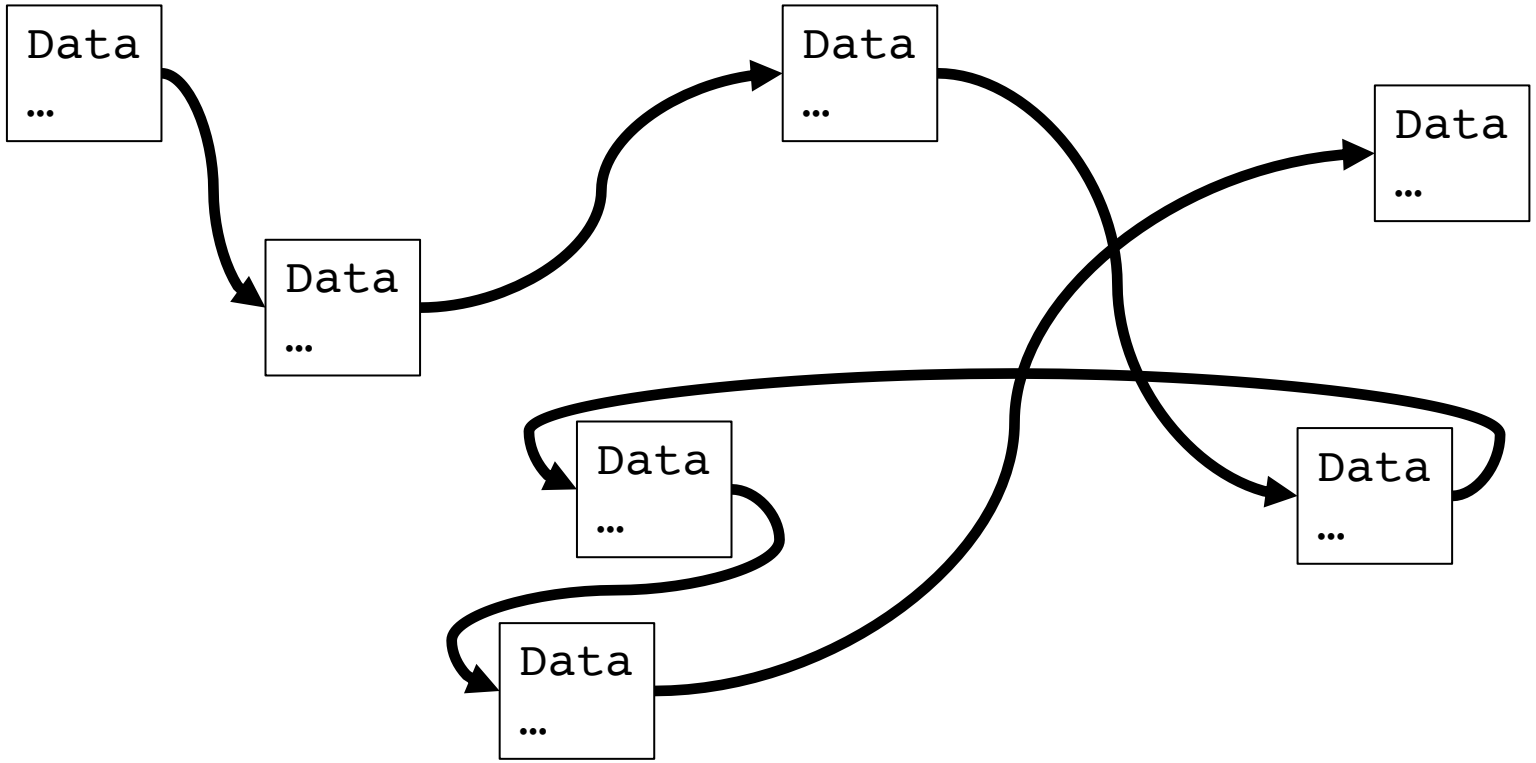
Algorithms



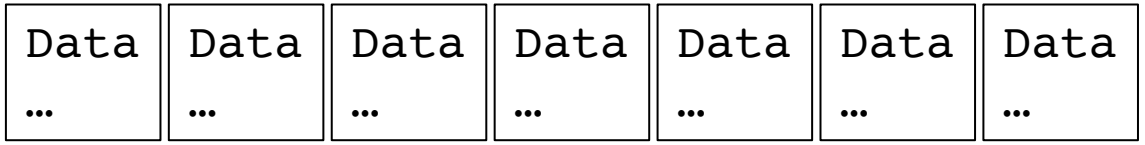
Array



Linked List



Array



Fast for search
Slow for resizing

Linked List

Slow for search
Fast for resizing

How fast is the algorithm?

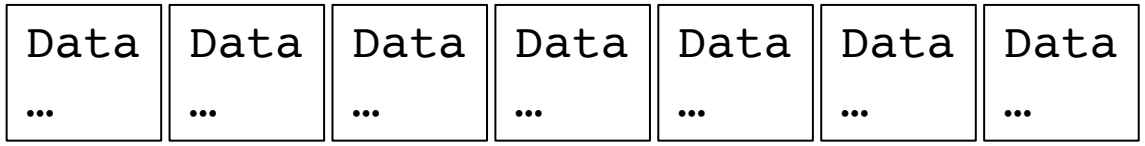
Big O

Big O

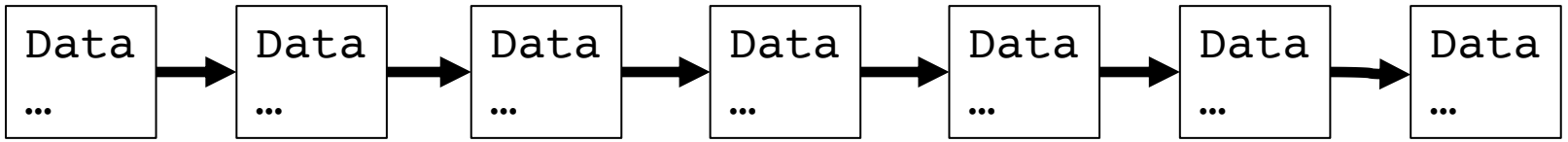
Big O notation, also called ***Landau's symbol***, describes how running time (or memory space) for your algorithm grow as the size of the input to the algorithm grows

Big O notation describes the ***asymptotic behavior*** of the algorithm

Array



Linked List




```
def add_a_number_to_all_data(data, number):
    for a in data:
        a += number
```

(This is not real code. It is conceptual logical flow of a hypothetical function.)

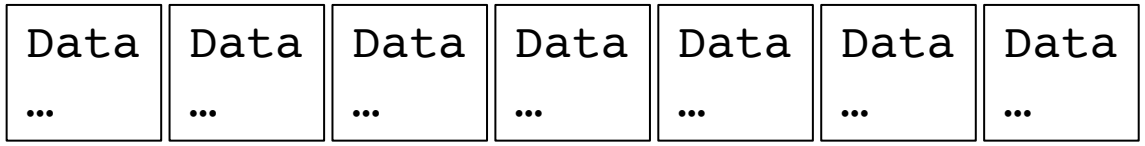
$time = a \cdot N + b$

as $N \to \infty$

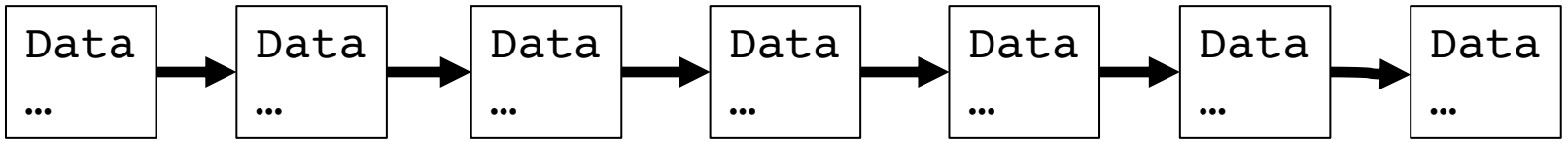
$time \sim N$

$O(N)$ (order of N)

Array



Linked List




```
def add_all_pairs_of_data(data):
    matrix = matrix_init()
    for i, a in enumerate(data):
        for j, b in enumerate(data):
            matrix(i,j) += a+b
    return matrix
```

(This is not real code. It is conceptual logical flow of a hypothetical function.)

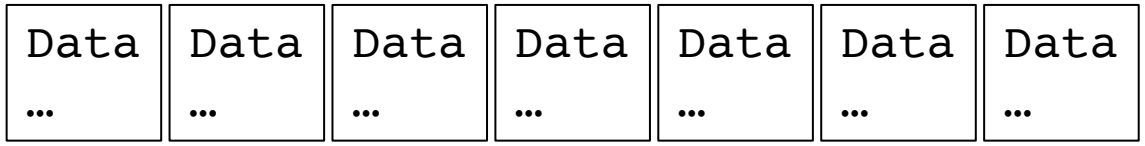
$time = a \cdot N^2 + b$

as $N \to \infty$

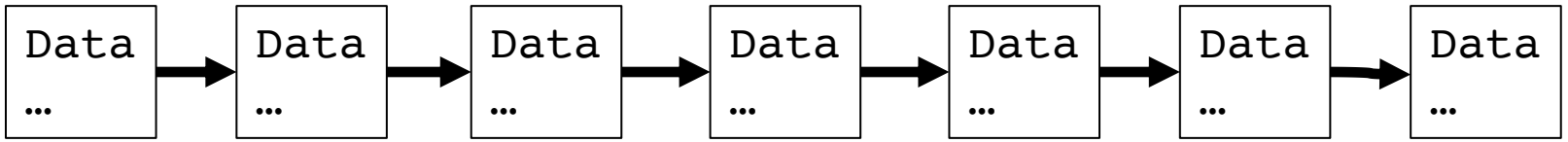
$time \sim N^2$

$O(N^2)$ (order of N squared)

Array



Linked List




```
def return_the_value_at_position_index_n_in_array(data, n):
    return data[n]
```

$O(1)$ (order of 1: constant time)

```
def return_the_value_at_position_index_n_in_linked_list(data, n):
    for i, d in enumerate(data):
        if i==n:
            return data[i]
```

On average, it needs N/2 operations.

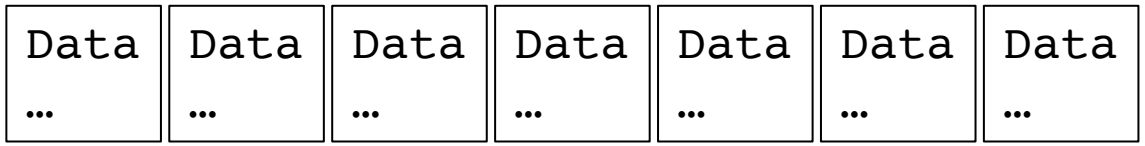
$O(N)$ (order of N)

There are common patterns in programming.

Two major categories of patterns: Data structures and Algorithms

The choice of data structure affect the implementation of algorithms and its process time.

Array



Big O

Linked List