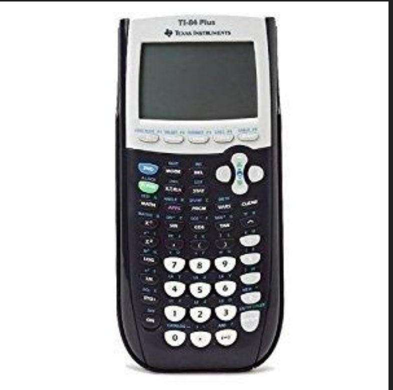
Asymptotic Complexity

How do we measure speed?



Example: Simple SELECT

```
simple_select(table, colname, value)
==
SELECT * FROM table WHERE colname = value
```

Simple SELECT

```
def simple_select(table, colname, val):
    rows = []
    for row in table:
        if row[colname] == val:
            rows.append(row)
    return rows
```

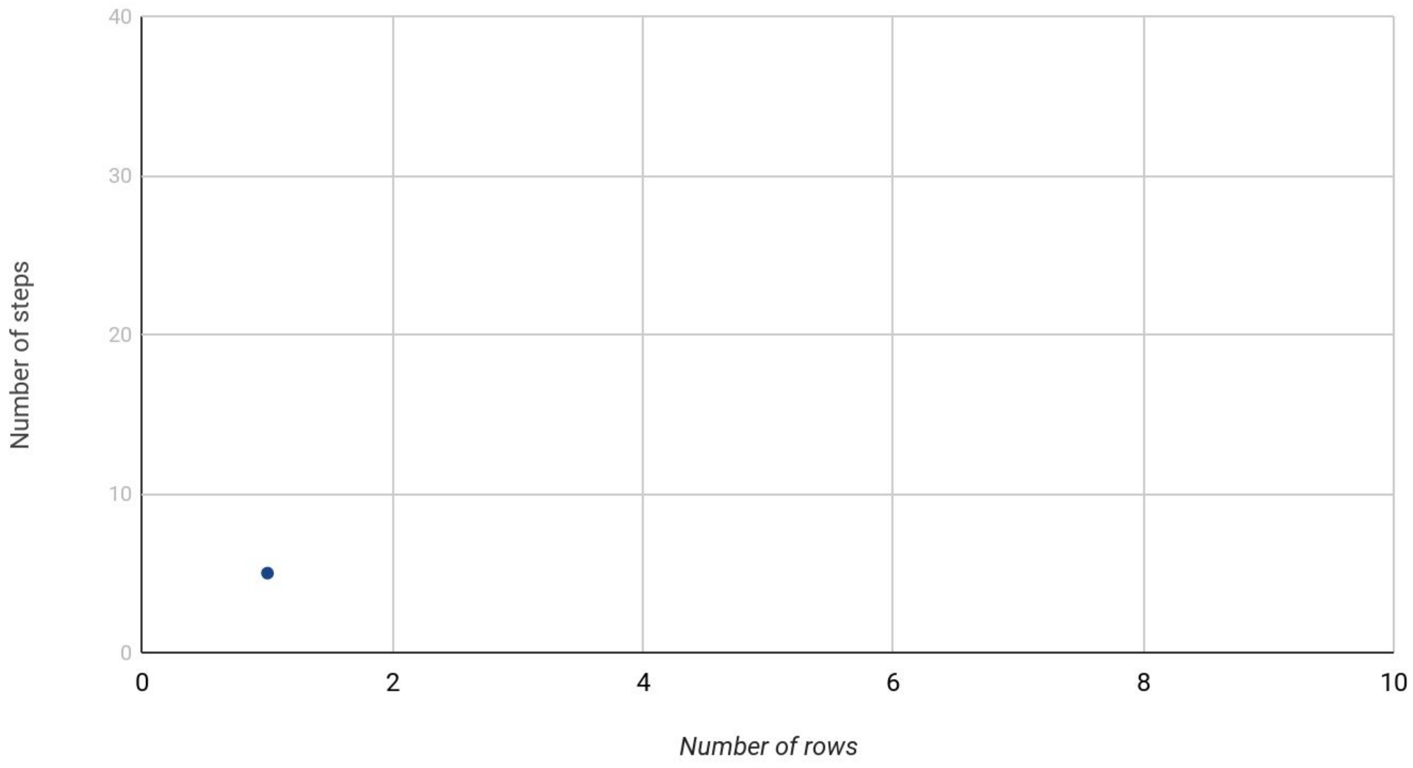
Assume `len(table) == 1`

```
def simple_select(table, colname, val):
    rows = []
    for row in table:
        if row[colname] == val:
            rows.append(row)
    return rows
```

Assume `len(table) == 1`

```
def simple_select(table, colname, val):
    rows = [] 1 +
    for row in table: Iterates once
        if row[colname] == val: +2 (dict access and ==)
            rows.append(row) +1
    return rows +1
Total steps: 5
```

simple_select performance



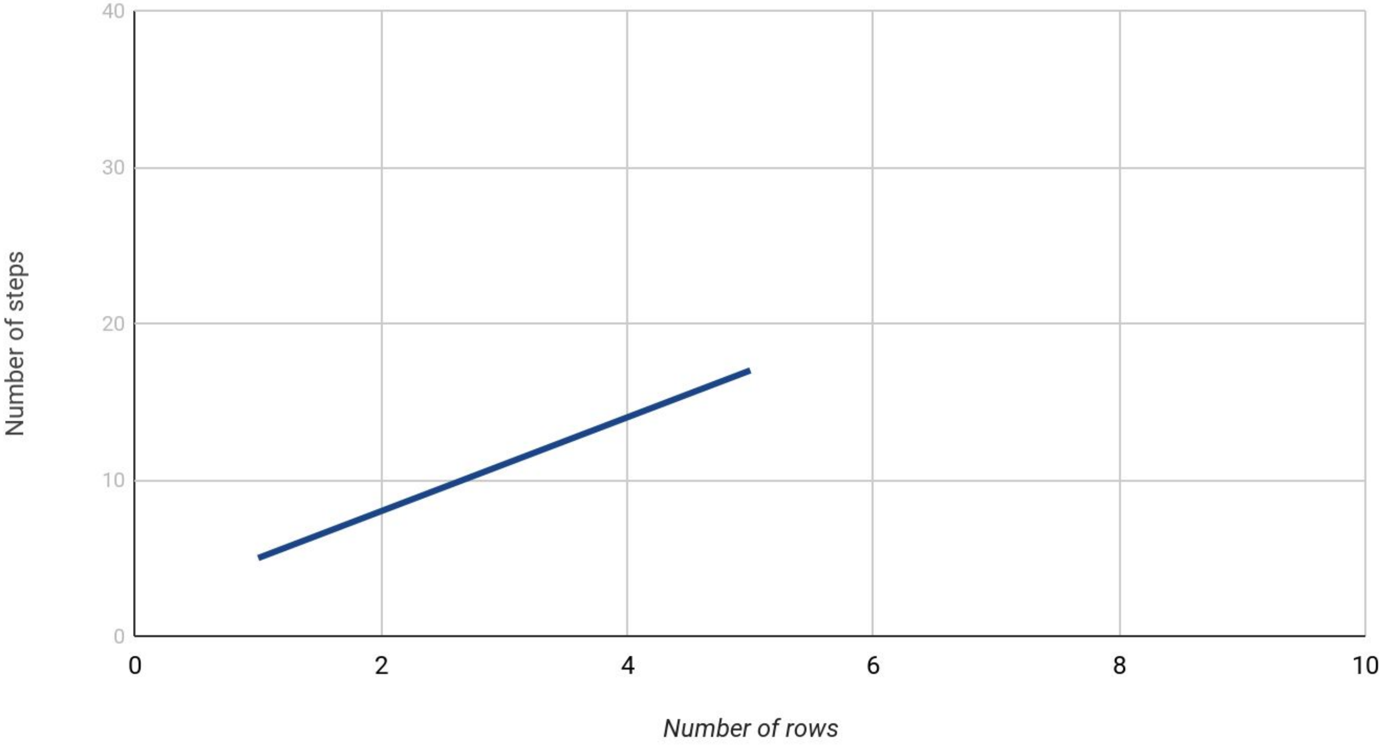
Assume `len(table) == 5`

```
def simple_select(table, colname, val):
    rows = []
    for row in table:
        if row[colname] == val:
            rows.append(row)
    return rows
```

Assume `len(table) == 5`

```
def simple_select(table, colname, val):
    rows = [] +1
    for row in table: Iterates 5 times
        if row[colname] == val: +2
            rows.append(row) +1
    return rows +1
Total steps: 1 + 5 * (2 + 1) + 1 == 17
```

simple_select performance



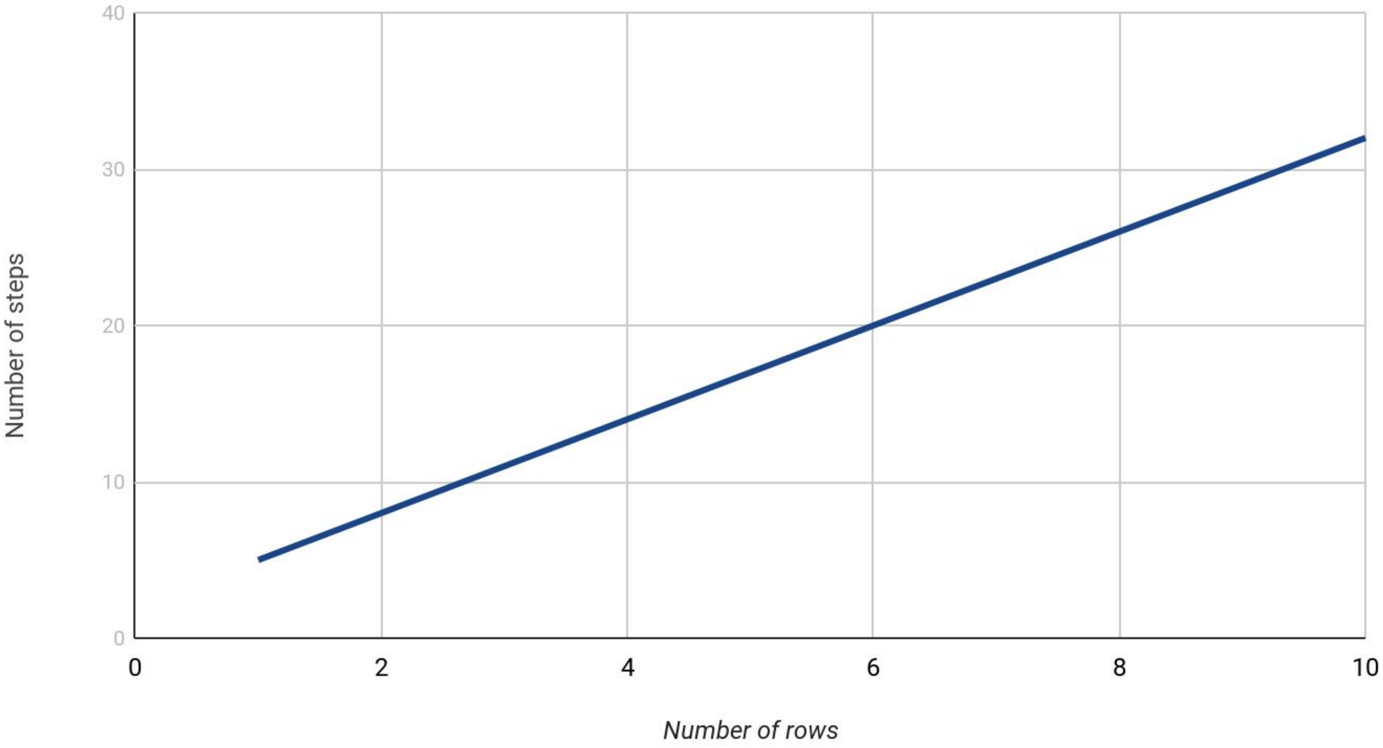
Assume `len(table) == 10`

```
def simple_select(table, colname, val):
    rows = []
    for row in table:
        if row[colname] == val:
            rows.append(row)
    return rows
```

Assume `len(table) == 10`

```
def simple_select(table, colname, val):
    rows = [] +1
    for row in table: Iterates 10 times
        if row[colname] == val: +2
            rows.append(row) +1
    return rows +1
Total steps: 1 + 10 * (2 + 1) + 1 == 32
```

simple_select performance



What’s the pattern?

Let n = len(table)

```
def simple_select(table, colname, val):
    rows = [] +1
    for row in table: Iterates n times
        if row[colname] == val: +2
            rows.append(row) +1
    return rows +1
Total steps: 1 + n * (2 + 1) + 1 == 3n + 2
```

Counting is hard.

Computer scientists are lazy.

Intuition: As the number of rows grows, the number of steps grows linearly

Simple SELECT is O(n)

Can we do better?

Consider

Name	Age
John	17
Mary	23
Alice	25
Brian	28
Bob	35
Elizabeth	39
Hailey	62

SELECT * FROM table WHERE Age = 35

Name	Age
John	17
Mary	23
Alice	25
Brian	28
Bob	35
Elizabeth	39
Hailey	62

SELECT * FROM table WHERE Age = 35

Name	Age
~~John~~	~~17~~
~~Mary~~	~~23~~
~~Alice~~	~~25~~
~~Brian~~	~~28~~
Bob	35
Elizabeth	39
Hailey	62

SELECT * FROM table WHERE Age = 35

Name	Age
~~John~~	~~17~~
~~Mary~~	~~23~~
~~Alice~~	~~25~~
~~Brian~~	~~28~~
Bob	35
Elizabeth	39
Hailey	62

SELECT * FROM table WHERE Age = 35

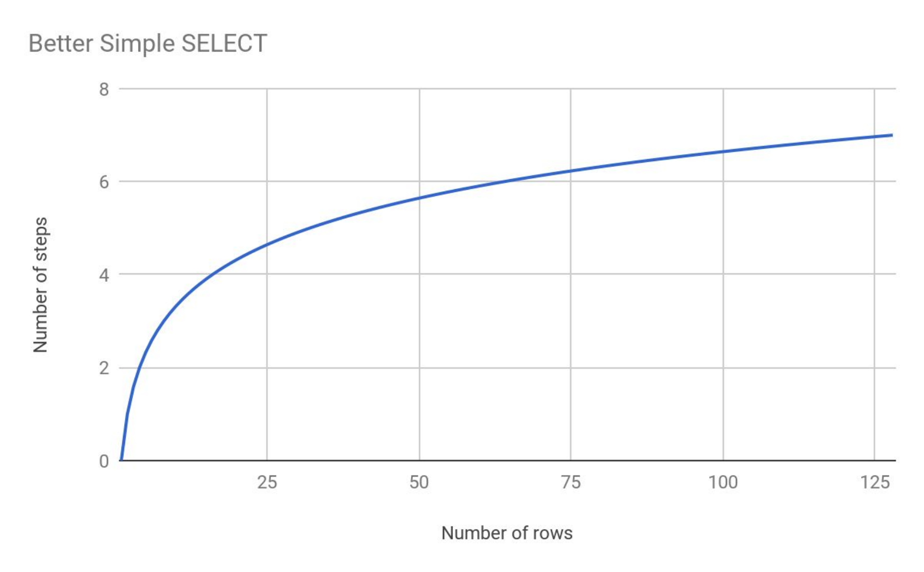
Name	Age
~~John~~	~~17~~
~~Mary~~	~~23~~
~~Alice~~	~~25~~
~~Brian~~	~~28~~
Bob	35
~~Elizabeth~~	~~39~~
~~Hailey~~	~~62~~

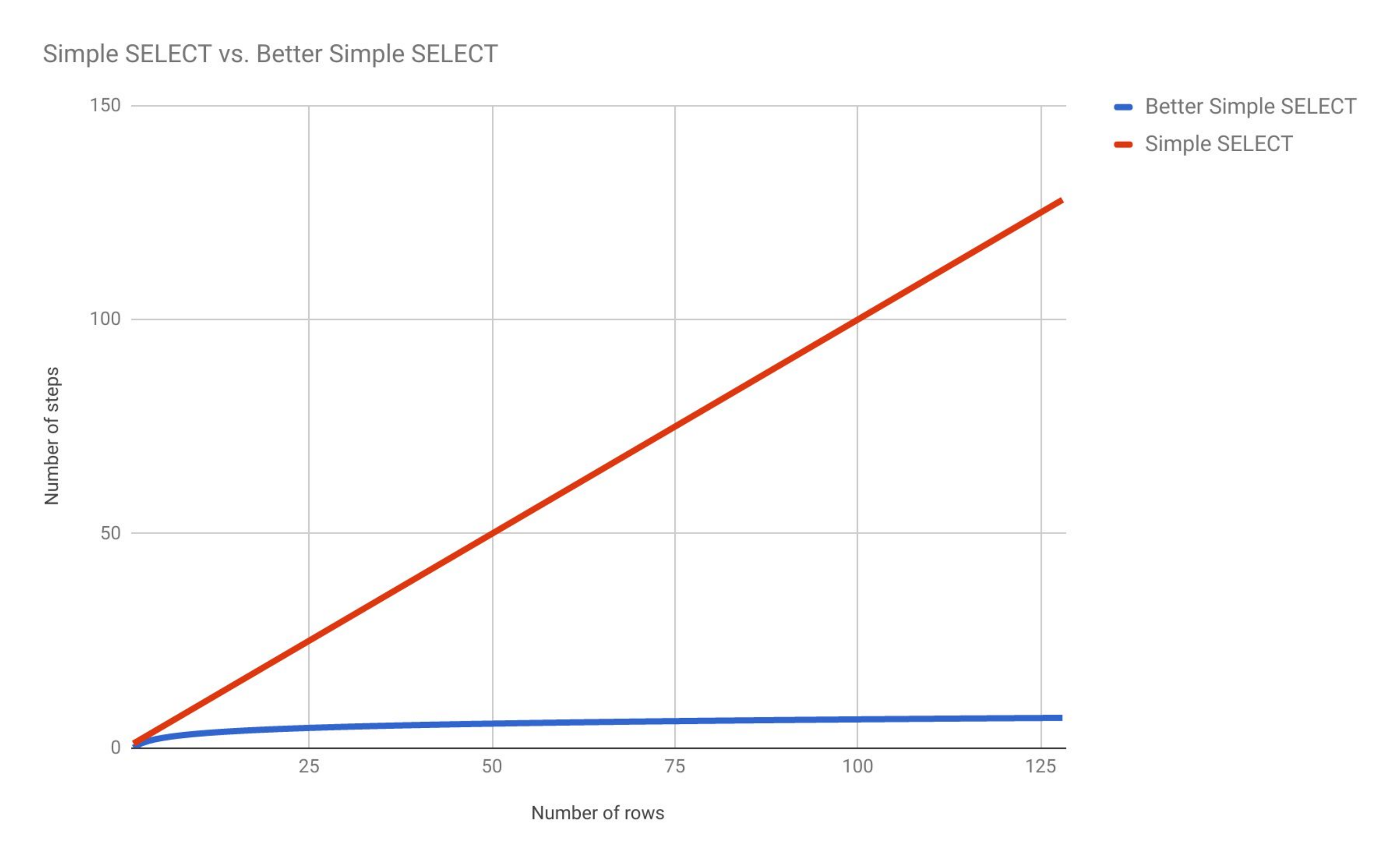
SELECT * FROM table WHERE Age = 35

Name	Age
~~John~~	~~17~~
~~Mary~~	~~23~~
~~Alice~~	~~25~~
~~Brian~~	~~28~~
Bob	35
~~Elizabeth~~	~~39~~
~~Hailey~~	~~62~~

Intuition: Doubling the number of rows only adds one extra step

Better Simple SELECT is O(log(n))




Simple SELECT vs. Better Simple SELECT
Better Simple SELECT
Simple SELECT
Number of steps
150
100
50
0
25
50
75
100
125
Number of rows

Digression: Simple JOIN

Name	Age
Brian	28
Jack	22
Hailey	62

Name	Profession
Brian	Tanner
Hailey	Robber
Rose	Paranormal Investigator

Digression: Simple JOIN

Name1	Age
Brian	28
Jack	22
Hailey	62

Name2	Profession
Brian	Tanner
Hailey	Robber
Rose	Paranormal Investigator

Name1	Age	Name2	Profession
Brian	28	Brian	Tanner
Hailey	62	Hailey	Robber

Simple JOIN

```
def join(table1, table2, colname1, colname2):
    matching_rows = []
    # TODO: For each row in table1, find all rows in table2 for which
    # row1[colname1] is equivalent to row2[colname2]
    # and add row1 and row2 to matching_rows
    return matching_rows
```

Getting started

```
git clone https://github.com/crossroads1112/join

sudo apt-get install python3-matplotlib

# Implement the join function in join.py

./plotter.py

c9 join.png
```

Solution

```
def join(table1, table2, colname1, colname2):
    matching_rows = []
    for row1 in table1:
        for row2 in table2:
            if row1[colname1] == row2[colname2]:
                matching_rows.append([row1, row2])
    return matching_rows
```

Solution

```
def join(table1, table2, colname1, colname2):
    matching_rows = []
    for row1 in table1: Iterates n times
        for row2 in table2: Iterates n times
            if row1[colname1] == row2[colname2]:
                matching_rows.append([row1, row2])
    return matching_rows
```

Questions?