CSE 115
Introduction to Computer Science I
Road map

▶ Review (sorting) ◀

Persisting data

Databases
Sorting

Given a sequence of values that can be ordered, sorting involves rearranging these values so they go from smallest to largest (or largest to smallest).

Example:

\[17, 93, 12, 44, 82, 81, 22, 73\]

all mixed up

Sorting rearranges items:

\[12, 17, 22, 44, 73, 81, 82, 93\]

increasing

smallest → largest
**selection sort**

Given a list of values, repeatedly select the smallest value and push it to the end of a list of sorted values.

N items $\rightarrow$ $N^2$ time

double input $\rightarrow$ quadruple time

**merge sort and Tim sort**

Given a list of values, split into in left and right partitions of roughly equal size. Sort each partition, then merge the two sorted partitions.

Tim sort uses merge sort for large inputs, insertion sort (similar to selection sort) for small inputs.

N items $\rightarrow$ $N \log_2 N$ time

double input $\rightarrow$ slightly more than double time
Custom sorting

Suppose we have data on students stored in a dictionary:

```
[  { "fname": "Sally", "lname":"Smith", "pn":"342083", "age":"23" },
   { "fname": "Barb",  "lname":"Woods", "pn":"934850", "age":"21" },
   { "fname": "Bo",    "lname":"Meele", "pn":"393847", "age":"22" },
   { "fname": "Amy",   "lname":"Fable", "pn":"705834", "age":"21" }
]
```

We could sort these dictionaries in many ways: by any of the fields, or by combinations of fields.

How do we specify what the sorting key(s) should be?
Custom sorting

students = [
  { "fname": "Sally", "lname":"Smith", "pn":"342083", "age":"23" },
  { "fname": "Barb",  "lname":"Woods", "pn":"934850", "age":"21" },
  { "fname": "Bo",    "lname":"Meele", "pn":"393847", "age":"22" },
  { "fname": "Amy",   "lname":"Fable", "pn":"705834", "age":"21" }
]

Function to return length of first name:

```python
def fNLen(V): return len(V["fname"])
```

How to sort by length of first name:

```python
students.sort(key = fNLen)
```
Custom sorting

The sort function can take a comparator argument, which specifies how to determine the relative order of two data elements.

The relative order is determined by a function which takes two data elements x and y as input and returns:

- a negative value (typically -1) if x comes before y,
- a positive value (typically 1) if x comes after y,
- zero if x and y belong at the same place in the order.
Custom sorting

```javascript
var students = [
    { "fname": "Sally",  "lname":"Smith", "pn":"342083", "age":"23" },,
    { "fname": "Barb",   "lname":"Woods", "pn":"934850", "age":"21" },,
    { "fname": "Bo",     "lname":"Meele", "pn":"393847", "age":"22" },,
    { "fname": "Amy",    "lname":"Fable", "pn":"705834", "age":"21" }
];

function fNLen(X,Y) {
    if (X["fname"].length < Y["fname"].length) { return -1; }
    if (X["fname"].length > Y["fname"].length) { return 1; }
    return 0;
}

students.sort(fNLen);
```

Function to determine relative order of lengths of first names:

```javascript
function fNLen(X,Y) {
    if (X["fname"].length < Y["fname"].length) { return -1; }
    if (X["fname"].length > Y["fname"].length) { return 1; }
    return 0;
}
```

How to sort by length of first name:

```
students.sort(fNLen);
```
Road map

Review (sorting)

▶ Persisting data ▶

Databases
Persisting data

Central Processing Unit

Random Access Memory

persistent storage
(e.g. file or database)
Persisting data

text file - stream of characters

CSV file - fields separated by comma

database - can support highly efficient operations on data
Road map

Review (sorting)

Persisting data

▶ Databases ◀
SQLite

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

https://www.sqlite.org/about.html
**SQLite**

SQLite is an in-process *library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine*. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

https://www.sqlite.org/about.html
import sqlite3

Start by importing the library.
import sqlite3

csqlite3.connect('atest.db')

Open or create a database file.
This file is not human-readable.
import sqlite3

conn = sqlite3.connect('atest.db')

Store returned value in a variable.
import sqlite3

conn = sqlite3.connect('atest.db')

cur = conn.cursor()
import sqlite3

conn = sqlite3.connect('atest.db')

cur = conn.cursor()

do things to database

A cursor allows us execute commands on the database.
import sqlite3

conn = sqlite3.connect('atest.db')
cur = conn.cursor()

do things to database

conn.commit()
import sqlite3

conn = sqlite3.connect('atest.db')

cur = conn.cursor()

do things to database

conn.commit()

conn.close()
Commands

create a new table

CREATE TABLE IF NOT EXISTS name columnNames

name – the name of the table

columnNames – the names of columns in the database table

Example:
'CREATE TABLE IF NOT EXISTS movies (title, director, year)'

Execute on cursor:
cur.execute(
    'CREATE TABLE IF NOT EXISTS movies (title, director, year)'
)
Commands

insert rows into table

```
INSERT INTO table VALUES (x, y, ... z)
```

table – the name of the table

x, y, ... z – the values for each column

Example:

```
'INSERT INTO movies VALUES ("Jaws", "Spielberg", 1975)'
```

Execute on cursor:
```
cur.execute(  
    'INSERT INTO movies VALUES ("Jaws", "Spielberg", 1975)'
)
```
Commands with safe substitution

As a function

```python
def insert(title, director, year):
    cur.execute('INSERT INTO movies VALUES (?,?,?)', (title, director, year))

'?' is a placeholder that is used for safe replacement – we'll talk more about security later this week.
```

Calling function

```python
insert("Jaws", "Spielberg", 1975)
```

Read more about tuples:
https://docs.python.org/3.3/tutorial/datastructures.html#tuples-and-sequences
Commands
get rows from table

SELECT * FROM table

table - the name of the table

Example:
'SELECT * FROM movies'

Execute on cursor:
cur.execute(
    'SELECT * FROM movies'
)
Commands
get matching rows from table

SELECT * FROM table WHERE constraint

table – the name of the table
constraint – constraint to match rows

Example:
'SELECT * FROM movies WHERE year=2002'

Execute on cursor:
cur.execute(
    'SELECT * FROM movies WHERE year=2002'
)
Commands
function to select rows

As a function

def get_all_by_year(year):
    return cur.execute('SELECT * FROM movies WHERE year=?', (year,))

Calling function

get_all_by_year(2002)

This is a singleton tuple. The comma is required.
Using result

print('Movies by year 2002')
for entry in get_all_by_year(2002):
    print(entry)