# Week 10

**ASSIGNMENTS**

**h7** due before 10pm on Monday 4/1

**h8** available soon and due before 10pm on 4/8

**p4** available _____ and due before 10pm on Thursday 4/16

**Peer Mentors:** will help student teams with graph terminology, implementations, and algorithms, bring your own problems, or let our peer mentors suggest some

*Module: Week 10 (start on week 11 before next week)*

=> Read Team Project and install JavaFX before lecture Week 11

**THIS WEEK**

- Final Team Project (create your 4 person teams now)
- Sets
  - Notation
  - SetADT
  - Implementation
- Linear Sorts
  - Radix sort
  - Flash sort
- Graphic User Interfaces

**NEXT WEEK**

- Java FX
- more Java
-

create A-Team

- Canvas
- People -> Groups
- Search Ateam

join - tell team member

# Parsing JSON with Java

start of JSONArray.

3 JSONObject

key, value pair

```
{
    "courses": [
        {
            "number": "CS400",
            "name": "Programming III",
            "instructors": [
                {"name":"deppeler",
                "lectures":["001","002"]},
                {"name":"kuemmel",
                "lectures":["004"]}
            ]
        },
        {
            "number": "CS300",
            "name": "Programming II",
            "instructors": [
                {"name":"dahl",
                "lectures": ["001","002"] },
                {"name":"kacem",
                "lectures":["003","004"]}
                ]
        },
        {
            "number": "CS354",
            "name": "Machine Organization",
            "instructors": [
                {"name":"skrentny",
                "lectures": ["001","002"] }
                ]
        }
    ]
}
```

end of JSONArray

Json object

CSV

CS login, fname, last name.

deppeler, deb, deppler, [...]

XML

colons, value

json

double d + install

json-simple-1.1.1.jar

*Java code to "start" parse of the above JSON file*

read and interpret ← create internal java instances

```
Object obj = new JSONParser().parse(new FileReader(filepath));
JSONObject jo = (JSONObject) obj;
JSONArray courses = (JSONArray) jo.get("courses");

course = new Course[courses.size()];
for(int i = 0; i < courses.size(); i++) {
    JSONObject jsonCourse = (JSONObject) courses.get(i);
    String courseNumber = (String) jsonCourse.get("number");
    String courseName = (String) jsonCourse.get("name");
    JSONArray courseInstructors = (JSONArray) jsonCourse.get("instructors");
    ...
```

file name

internal Java d.s.

## Notation

https://www.rapidtables.com/math/symbols/Set_Symbols.html

∅
ℕ
ℝ
A∈letters
|A|
A⊆B
A⊂B
A⊈B
A⊄B
A⊇B
A⊃B
A⊉B
A⊅B
A´

## Basic Operations

A ∪ B

A ∩ B

A — B

A □ B

A**Δ**B = (A □ B) ∪ (B □ A)

# SetADT

*Operations*

```
boolean add(E e) - add if item is not present
boolean contains(Object o) - true iff o is present
boolean remove(Object o) - remove o if present
boolean isEmpty() - true if no element
int size() - returns number of elements
```

*Complexity Analysis of Implementation Options*

Assume: N is number of nodes and H is height of tree

| | insert | lookup | remove | iteration |
|---|---|---|---|---|
| Array | | | | |
| Sorted Array | | | | |
| Linked List | | | | |
| BST | | | | |
| Balanced Search Tree | | | | |
| Hash Table "good" | | | | |

De Morgan's Laws

If A and B are any two sets then:

(A union B) ' = A´ intersection B´

(A intersection B)' = A´ union B´

# Radix Sort

* not a comparison sort - sorts a sequence of character

Pre-conditions: must know the range of digits or clear in the sequence
- must know length of each sequence
- length must be same for each element

number of items (N):
range of unique digits (RANGE): {0..9} {a..z} {A..Z}
length of item's sequence of digits (LEN)

*Idea* for each position of sequence do stable sort on digit of position

LSD - MSD

Sort the following integers:

121 367 354 873 777 333 123 222 411 262 897

What is N? 11 RANGE? 10 LEN? 3

LSD - least significant digit. {0..9}

Pass 1:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | 121 | 222 | 873 | 354 | | | 367 | | |
| | 411 | 262 | 333 | | | | 777 | | |
| | | | 123 | | | | 897 | | |

O(N)

middle

Pass 2: 121 411 222 262 873 333 123 354 367 777 897 O(N)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | 411 | 121 | 333 | | 354 | 262 | 873 | | 897 |
| | | 222 | | | | 367 | 777 | | |
| | | 123 | | | | | | | |

411 121 222 123 333 354 262 367 873 777 897 ⇐ O(N)

Pass 3:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | 121 | 222 | 333 | 411 | | | 777 | 873 | |
| | 123 | 262 | 354 | | | | | 897 | |
| | | | 367 | | | | | | |

⇐ O(N)

Sorted: 121 123 222 262 333 354 367 411 777 873 897 ⇐ O(N)

6 O(N) ⇒ O(N)

# Radix Sort: algorithm

**We know:** items are stored in array A
N = A.length()
a: sequence of two digits, range is 10 [0..9]

*Algorithm*

```
List[] digitQ = new List[RANGE]

for (i = 0; i < RANGE; i++):
    digitQ[i] = new Queue()

for (pos=LEN-1; pos >= 0; pos--):
    for (j=0; j < N; j++)
        let x = digit in position pos of the item in A[j]
        digitQ[x].enqueue(A[j])

    index = 0
    for ( j=0; j < RANGE; j++ ):

        while (!digitQ[j].isEmpty()):
             A[index] = digitQ[j].dequeue()
             index++
```

O(LEN)
O(N).
sort into queue
O(R)
put back to array.

*Complexity* O(N) + O(N) + LEN * O(N) + O(N+R)
R + LEN * (N+R)
↳ O(R+N)
vs O(N log2N)

# Flashsort

- handles a large array with evenly distributed data ← if not, insertion x linear.

*Data and Variables*

```
A          - the array of elements to be sorted
n          - the number of items to sort, A.length
Amax       - the maximum value in A
Amin       - the minimum value in A
m          - the number of bins to divide items into
A[i]       - the element at index i of A
K(A[i])    - classification (bin@) for Ai
L          - an array use to track number of items in that bin
L          - later used to indicate next available index for that bin
```

pre-process data

⇐ we choose $m = N/2$

function: computes "bin"

*Setup* scan array to determine the above

$m = N/2$

L.length $= m + 1$

- $O(n)$.

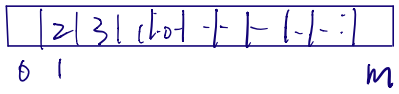
*Classification*

$K = 1 + (int)((m-1) * (A[i] - Amin) / (Amax - Amin))$

for each value: // N times

compute K

L[K] ++ ;




m = 11, 22 item.

for each bin: // m times

set value in L to be index of last value in that bin

L[m] = L[m-1] + L[m]

- $O(n)$

*Permutation* do in-place positioning of items into their "bin" range

*Insertion*

final "short range" insertion sort of array

# Flashsort Example

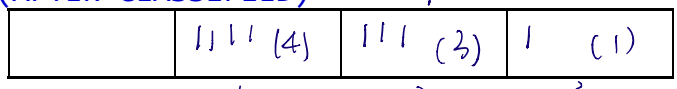
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| K(A[i]) | 2 3 | 1 | 3 2 | 1 | 1 | 2 | 1 | 2 |
| A = | 5 7 | 3 | 7 5 | 1 | 2 | 6 | 1 | 4 |
| After permutation | 3 | 2 | 1 | 1 | 6 | 5 | 4 | 7 |
| | 2 | 3 | | | | | | |
| | 1 | 2 | 3 | | | | | |

1 1 2 3 6 5 6 4 5 6 7

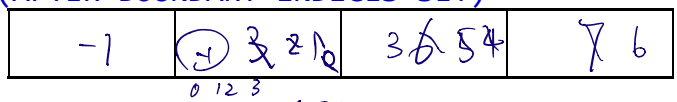
```
INITIAL SETUP
K: [2,1,3,1,1,2,1,2]
A: [5,3,7,1,2,6,1,4]
N = 8
m = 3
Amin = 1
Amax = 7
indexMax = 2

K(A[i]) = 1 + (int) ((m-1) (Ai - Amin) / (Amax - Amin) )

L (AFTER CLASSIFIED)
```




```
L (AFTER BOUNDARY INDICES SET)
```




hold 4 items

index for F: 7 6

temp: 4 7 4 5 6 2 3 7

```
// PERMUTATION ALGORITHM
swap first value in A with max value in A
set j to index 0
while j is less than last index of A:
    set F to A[j]
    get index I for F   L[K(F)]
    if j is less than i:
        Do:
            Save the value from A[i] as temp (to be the next F)
            Place F at A[i]
            Decrement the boundary index in I = L[K(F)]

            Assign temp to F to be the next value place
            Get next I for F
        Repeat while I is not j
    Increment j
```

# Graphic User Interfaces (GUIs)

Idea:

## Java FX

AWT - abstract windowing toolkit (classes that create graphic UI components)
Applets - run java code in a web browser
Swing - encapsulated common gui components - easier to use than awt packages
Java FX - latest version, has support for desktop computers and web browsers

*Why Java FX?*

# Install e(fx)clipse – Fall 2018

1. Install JDK 8
2. Install Eclipse Oxygen or Proton
3. Install Java FX project wizard
    a. launch Eclipse
    b. Go to Help -> Install New Software
    c. Search for "e(fx)clipse"
    d. Install "e(fx)clipse" and wait patiently for installation to complete (a few minutes)
    e. Accept certificate and User Agreement
    f. Restart Eclipse as directed
4. Create a Java FX Project
    a. Select File -> New -> Other
    b. Slect Java FX -> Java FX Project
    c. Enter project details as before and proceed
5. Run your Java FX application
    a. Open class Main
    b. Click Run as Application
6. View the GUI that is built by default
7. Close View

*Java FX Basics*

*Java FX Demo (in-class)*