1 Introduction

Some questions to ask before starting on a problem

- Extract out important keywords (what DS to use?)
- Edge cases? *e.g.* if `size==0` or `size==1`,
- Trivial cases? can just hardcode

Code styling

- CS2030 Code Styling Guide
- Google Java Styling Guide
- **Modularity**: use method to print answers inside main method

```
\\ print answer
ans = simulate(n,k,m);
printAns();
```

- **No global variables**

2 Java

How to throw exception?

```
public class MyException extends Exception {
  private int var;
  public MyException(int var) {
    this.var = var
  }
  public int getVar() {
    return this.var;
  }
}

public class Main {
  public static void main(String[] args) {
    try {
      ...
      throw new MyException(errorVar);
    } catch (MyException e) {
      System.out.println(e.getVar());
    }
  }
}
```

3 Data Structures

$$O(1) < O(\log(n)) < O(n^c) \text{ where } c < 1$$

$$O(n) < O(\log(n!)) = O(n\log(n)) < O(n^2)$$

$$O(n^k)[\text{ where } k > 2] < O(k^n)[\text{ where } k \geq 1] < O(n!)$$

How to implement Data Structures?

- Composition: use well-known DS as an attribute of the implemented DS
- Inheritance: extends well-known DS

3.1 Linked List

- Motivation: implementation of list using array needs to occupy contiguous memory space (can result in memory error)
- Variants of linked list:
  - Tailed (need to maintain `head` and `tail`)
  - Circular
  - Doubly linked (`prev` and `next` attributes for `ListNode`)
- How to find cycle?
  Answer: use fast and slow pointers

```
  slow = slow.next;
  fast = fast.next.next;
```

- **[IMPT]** Drawing pictures is very important to visualize the program!

Java API: `ArrayList` or `LinkedList`

```
\\ constructor
ArrayList<Integer> list = new ArrayList<Integer>();
```

3.2 Stack

```
// to construct an array of generics
E[] arr = (E[]) new Object[size];
/*
// does not work
E[] arr = new E[size]
*/
```

Uses:

- **[IMPT]** Converting infix to postfix expression (Lecture 4 Slide 28)
- **[IMPT]** Evaluating postfix expression

3.3 Queue

Uses:

- **[IMPT]** Breadth-first traversal of trees
- Sliding Window (especially important for contiguous blocks of stuff)

4 Recursion

[IMPT] Recipe for recursion (3 fingers)

1. General recursive case: identify simpler instances of the same problem
2. Base case: cases that we can solve without recursion
3. Be sure that we are able to reach the simplest instance so that we won't end up in infinite loop

Uses

- Insert item into sorted LinkedList
- Tower of Hanoi
- **[IMPT]** Combination (n choose k)
- Binary search
- Finding k-th smalles element (use pivot element `p`)
  move elements < `p` to the left of `p`
  move elements > `p` to the right of `p`
- Printing all permutations of a String

Overloading: same function name but with different parameters (useful in Java)

Backtracking

- Solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time
- *e.g.* Queens Lab 4B: board is fixed
  queens can be added or removed!

5 Sorting

Some definitions

- **Sort key**: use particular value of an object to do comparison and sort
- **In-place**: requires only a constant amount of extra space during the sorting process
- **Stable**: relative order of elements with the same key value is preserved by the algorithm

Some ideas used in sorting:

- Internal vs external sort
- Iterative vs recursive
- Comparison vs non-comparison based
  e.g. radix sort
- Divide and conquer

Applications

- Uniqueness testing
- Deleting duplicates
- Frequency counting
- Efficient searching

	Iterative	Recursive
Comparison	Bubble, Selection, Insertion	Quick, Merge
Non-comparison		Radix

5.1 Algorithms

5.1.1 Selection Sort

Time complexity: $O(n^2)$
Limitation: Not stable

5.1.2 Bubble Sort

Time complexity: $O(n^2)$

- Using flag: $O(n)$ `isSorted`, is the input already sorted?

5.1.3 Insertion Sort

Time complexity:

- Best case: input already sorted ($O(n)$)
- Worst case: input reversely sorted ($O(n^2)$)

5.1.4 Merge Sort

Time complexity:

- `merge(arr, left, mid, right)` is $O(right-left+1)$
- `merge` is called $\log n$ times
- Hence $O(n \log n)$

Limitations:

- Need temporary array to store values during the `merge` process (not in-place)

5.1.5 Quick Sort

Time complexity:

- `partition()`
- `quicksort(a, i, p)`
- Worst case is when it is already sorted, so the first group (elements < p) is empty: $O(n^2)$
- Best case: occurs when array is divided into 2 equal halves
  - Depth is $\log n$
  - Each level takes n comparisons (including swaps)
  - Hence $O(n \log n)$ which is also the average case

Limitation: Not stable

5.1.6 Radix Sort

Treat each data as a character string: no comparison needed
Trick: sort by unit digit → tenth digit → hundredth and so on...
Time complexity:

- Initialize 10 groups (queues) to group the elements
- Complexity is $O(dn)$ where d is the maximum number of digits of the n numeric strings in the array

Limitation: Not in-place

5.1.7 Bucket Sort

How it works:

- There are b buckets, and each element `arr` is inserted into bucket according to a function *e.g.* `(int) arr[j]*10`
- Similar to radix sort but b can be any number (base?)
  e.g. Tut 5 Q 3(b) where $N \leq$ `arr[i]` $\leq 3N$, we can have $3N$ buckets $1, 2, 3, \ldots, 3N$ so that each bucket contains only 1 element
- So only 1 pass is needed a.k.a. $O(3N)$ time
- Possible problem: takes up alot of space?

	Worst Case	Best Case	In-place?	Stable?
Selection	$O(n^2)$	$O(n^2)$		No
Insertion	$O(n^2)$	$O(n)$		
Bubble	$O(n^2)$	$O(n^2)$		
Bubble (Flag)	$O(n^2)$	$O(n)$		
Merge	$O(n \log n)$	$O(n \log n)$	No	
Radix	$O(dn)$	$O(dn)$	No	
Quick	$O(n^2)$	$O(n \log n)$		No

5.2 Java Sorting

For list/arrays:

- To convert arrays to list use `Arrays.asList`
- `Arrays.sort` or `Collections.sort`

For others: use `Collections.sort(list, compObj)`

```java
import java.util.Comparator;
class ObjComparator implements
    Comparator<Obj> {
```

```
  public int compare(Obj o1, Obj o2) {
    // if positive, o1 > o2
    // if negative, o1 < o2
    // if zero,     o1 = o2
  }
    public boolean equals(Object obj) {
      // check to see if we have the same comparator object
      return this = obj;
    }
}
```

6 Java Tricks

- OOP is important (CardGame)
  - If it involves an array, OOP is useful, methods can just modify properties/attributes of the object class (*e.g.* `reversed=true; increment=4`)
    - Especially true if only need to print statement at the end
- Invariance: property that stays constant???
  Lab 5C: Pancakes: Use number of inversions if it's even or not
- Use `StringBuilder` for return statements
  - Java StringBuilder API
  - Zigzag conversion