• **Recitation**
  - Eric will host a Java primer
  - There will be some basic problems to test your I/O parsing code
  - A harder contest will be run in parallel for those who want a challenge

• **Homework 1**
  - Will be posted tomorrow (announced on the website and Piazza)
  - Due Saturday, Feb 14 at 2am (Happy Valentine’s Day!)
  - **Remember:** you may work in teams, but submit your own code
  - We will be checking!

• **Grading for HW1:**
  - Each problem is work 1 point (unless otherwise noted)
  - You either get it right (accepted) or wrong (everything else)
• We’ll be using the HUST Virtual Judge
• Remember to set up your accounts correctly!
• Make sure your username is `<netid>_CS480S15`
  • Unless you are just sitting in, in which case please avoid this convention and use whatever username you want.
• On the virtual judge, all I/O is from standard in/out
• Java programs must be entirely inside a public class called `Main` in the default package
• Both Java and C++ programs must have a `public static void main` and `int main`, respectively.
• Some problems to help you practice your I/O:
    • Replace PID with a number from 1089 to 1096
  • You may get a chance to practice during recitation (time permitting)
Nested for loops runtime.

How many \texttt{myOperation()} calls?

```java
public static void main(String args[]) {
    for (int i = 0; i < 300; i++) {
        for (int j = 0; j < 600; j++) {
            for (int k = 0; k < 200; k++) {
                \texttt{myOperation(i, j, k);
            }
        }
    }
}
```
• You receive a time limit exceeded response for an your \( O(N^3) \) solution. \( 1 \leq N \leq 100 \)
  A. Abandon the problem
  B. Improve the performance of your solution
  C. Create tricky test cases and find the bug
You receive a time limit exceeded response for an your $O(N^3)$ solution. ($1 \leq N \leq 1,000,000$)

A. Abandon the problem  
B. Improve the performance of your solution  
C. Create tricky test cases and find the bug
You receive a runtime error response. Your code runs successfully on your machine. What should you do?

A. Abandon the problem
B. Improve the performance of your solution
C. Create tricky test cases and find the bug
Exercise 1. You are given $x$, $y$, and $S$ such that $2 \leq x, y, \leq 36$ and $S \in [0-9A-Z]^+$ is a valid positive 32-bit integer written in base $x$. Output the value of $S$ in base $y$ using as few lines of code as possible.

<table>
<thead>
<tr>
<th>Sample input</th>
<th>Sample output</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 16 255</td>
<td>ff</td>
</tr>
<tr>
<td>2 8 1101111101</td>
<td>1575</td>
</tr>
<tr>
<td>16 36 c0dface</td>
<td>3cerim</td>
</tr>
</tbody>
</table>
public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String line;

    while ((line = in.readLine()) != null) {
        StringTokenizer st = new StringTokenizer(line);

        // Parse
        int x = Integer.parseInt(st.nextToken());
        int y = Integer.parseInt(st.nextToken());
        String baseXIntStr = st.nextToken();

        // Format
        int theInt = Integer.parseInt(baseXIntStr, x);
        String baseYIntStr = Integer.toString(theInt, y);
        System.out.println(baseYIntStr);
    }
}
Exercise 2. You are given an integer $x$ such that $0 \leq x < 10^9$. Print the same integer such that the number is left-padded with zeros to make a 9 digit number. Write the fewest lines of code to accomplish this.

<table>
<thead>
<tr>
<th>Sample input</th>
<th>Sample output</th>
</tr>
</thead>
<tbody>
<tr>
<td>417370</td>
<td>000417370</td>
</tr>
</tbody>
</table>
public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String line;

    while ((line = in.readLine()) != null) {
        // Parse
        int x = Integer.parseInt(line);

        // Format
        System.out.printf("%09d\n", x);

        /* Also valid: */
        // String outputString = String.format("%09d", x);
        // System.out.println(outputString);
    }
}
Exercise 3. You are given a decimal number $x$ such that $0 \leq x < 10^9$. Print the same decimal number to exactly three decimal points, rounding if necessary. Write the fewest lines of code to accomplish this.

<table>
<thead>
<tr>
<th>Sample input</th>
<th>Sample output</th>
</tr>
</thead>
<tbody>
<tr>
<td>4086.910607</td>
<td>4086.911</td>
</tr>
</tbody>
</table>
public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String line;

    while ((line = in.readLine()) != null) {
        // Parse
        double x = Double.parseDouble(line);

        // Format
        System.out.printf("%.3f\n", x);

        /* Also valid: */
        // String outputString = String.format("%.3f", x);
        // System.out.println(outputString);
    }
}
• For the following exercises, what is the runtime?
  • Given two sets A, B such that $1 \leq |A|, |B| \leq 10^6$, what is the size of A intersected with B?
  • Given a set S of n randomly scattered points $(x, y)$ such that $n < 1000$, find the greatest distance between any two points in S.
    • What if $n < 10^6$?
Static arrays

- int myArray[] = new int[10];
- Accessing and setting: O(1) operations
- Don't forget to clear between test cases
  - Arrays.fill(myArray, 0);
ArrayLists (convenient resizable arrays)

- ArrayList myList = new ArrayList();
- Constructor has one parameter, an integer
  - e.g., new ArrayList(1000) instantiates a new ArrayList with initial capacity of 1000 items
  - Default (no param): initial capacity is 10 items
- Unbounded growth (within memory limit of program)
ArrayLists (cont’d)

- Appending to list: amortized O(1) operation
  - When a resize occurs, all elements are copied to a new array, which is O(n) operations
- Inserting to list: O(n) operations
  - Elements are shifted over to accommodate
- If you reuse one between test cases, run list.clear() between runs!!
Common operations on arrays and ArrayLists

• Sorting
  • Arrays.sort(myArray) – quicksort, $O(n \log n)$
  • Collections.sort(myList) – merge sort, $O(n \log n)$

• Searching
  • Unsorted list: exhaustive search, $O(n)$
  • Sorted list: binary search, $O(\log n)$
    • Arrays.binarySearch() and Collections.binarySearch() – more later
Bitmask

- Treat a primitive int or long as a set of booleans
- C++ also has `<bitset>`, but using primitives is more efficient
- Further discussion next class
LinkedList

- O(n) time to access an indexed element
- O(n) to search for an element
- O(n) to insert (or O(1) with a ListIterator)
- Just use an ArrayList
Stack
  • LIFO operations: Push, pop

Queue
  • FIFO operations: Push, pop
  • In Java, implemented as an interface
    • Has a LinkedList data structure backend, not good to search through / insert
    • Queue<X> myQueue = new LinkedList<X>();
    • Will be used later in this class
Binary search tree

- Java's TreeSet and TreeMap, and C++'s map implement a Red-Black tree
  - Self-balancing binary tree
  - Be aware of other BSTs: AVL trees, splay trees
- Cost:
  - Insertion: myTree.put(x) – O(log n)
  - Membership: myTree.containsKey(x) – O(log n)
  - Remove: myTree.remove(x) – O(log n)
  - Fetch (TreeMap): myTree.get(x) – O(log n)
Hash table

- Java implements a standard hash table
  - Buckets (an array) of key-value objects called “Entries”
  - Keys with the same hash codes are stored in the same bucket using a linked list
    - Not LinkedList
  - Can be given an initial capacity (default 16)
  - Collision time/space trade-off regulated by the load factor (default 0.75)
    - How full the table can become before capacity is automatically increased
Hash table (cont’d)

- Cost:
  - Insertion, fetch, removal, membership: expected $O(1)$
  - Depends on a good hash function
    - If you make a custom class, ensure you override the `hashCode()` so collisions are minimized
    - Eclipse is your friend: Source → Generate `hashCode()` and `equals()`
- `HashMap` and `HashSet`
Linked hash table

- Like a regular hash table, but a doubly linked list is maintained through all entries
- Convenience class for efficiently traversing hash table keys
  - for (Entry<K, V> e : myHashTable.getEntries())
- Java: LinkedHashMap, LinkedHashSet
- Iteration order:
  - Order in which elements were added
- Cost of iterating:
  - Linear in size
Heap

- Tree structure
- Each element:
  - Is larger than its parent
  - Is smaller than its children
- Java: PriorityQueue, a binary heap
- Operations:
  - Add: Put the element in the tree – $O(\log n)$
  - Poll: Remove and return top element from the heap tree, i.e., the smallest element – $O(\log n)$
Heap

- Stored in contiguous memory for fast lookup

```
<table>
<thead>
<tr>
<th>idx</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Parent:** \((\text{idx} - 1) \gg 1\)
Heap

- Add 2, percolate up

```
+---+---+---+
| 3 |   |   |
+---+---+---+
| 4 |   | 6  |
+---+---+---+
| 5 | 8  | 7  |
```

```
<table>
<thead>
<tr>
<th>idx</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
```

**Parent**: \((\text{idx} - 1) >>> 1\)
Heap

- Compare against parents: swap 2 and 6

![Heap Diagram]

<table>
<thead>
<tr>
<th>idx</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

**Parent:** (idx - 1) >>> 1
Heap

- Compare against parents: swap 2 and 3

```
<table>
<thead>
<tr>
<th>idx</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
```

**Parent:** $(idx - 1) \ggg 1$
A set of nodes connected by edges
- Directed, undirected
- Cyclic, acyclic

Represented by:
- Adjacency Matrix
- Adjacency List
- Edge List

Much more later
Motivation:

- You want a data structure to quickly union two or more disjoint sets (union)
- You want to quickly find what set an element belongs to (find)

How to do this efficiently

- Make a forest of trees for each element
- The root of the tree is the set identifier
Union:

function Union(x, y)
  xRoot := Find(x)
  yRoot := Find(y)
  xRoot.parent := yRoot

Find:

function Find(x)
  if x.parent != x
    x.parent := Find(x.parent)
  return x.parent
Union-Find:
  • Start with 8 disjoint sets (trees)
Union:

• If you want to union sets 1 and 2, add tree 2 as a child of tree 1
Union:

- If you want to union sets 3 and 5, add tree 5 as a child of tree 3
Find:

- Are 2 and 6 in the same set? Check their roots.
Union:
• Union 2 and 6.
Union:
- Union 5 and 6
Find with path compression:

- During the find operation, point each traversed node to its root directly
Given an integer $v$ and a list $S$ such that $1 \leq |S| \leq 10^6$, find two integer $a$, $b$ in $S$ such that $a+b = v$
Competitive Programming 2.1 to 2.4.2