Data Structures

Preliminary	Basic	Intermediate	Advanced	
Array	Linked List	Priority Queue	Binary Indexed Tree	
String	Queue	Disjoint Set	Segment Tree	
Set	Stack	Min/Max Heap - Sorted Map & Set	Merkle	
Hash Table	Stack	Trie	Indexed Priority Queue	
	Binary Search Tree	Monotonic Queue/Stack		
	Graph - Bipartite			

Algorithms & Approaches

Sorting	Strings / Arrays	Basic
Selection	Two-Pointers - Sliding Window - Growing/Shrinking SW - Catch-Up Condition	Recursion
Insertion	Slow & Fast Runners	BFS
Shell	Sorting & Searching - Binary Search Variation - Sort, then Solve	DFS (3 types)
Merge	Hash Something Techniques - Hash the Running Computation - Forward-Backward Running Computation: Bi-Directional - Hash the Elements - Increment / Decrement Counts	Tree Traversal
Quick		Linear Search
Counting		Binary Search
Bucket		
Radix		
3-Way Quicksort		
Bubble		

Intermediate	Advanced
Dynamic Programming	Knuth-Morris-Pratt (KMP) pattern matching
Greedy	Minimum Spanning Tree - Prim: Lazy & Eager - Kruskal
Backtracking	Maximum Network Flow: Ford-Fulkerson
Shortest Paths: - Djikstra - Bellman Ford	
Topological Sorting - DFS - Kahn's	
QuickSelect - DSelect	

Big-O Complexity

Constant	O(1)	Quartic	O(n^5)
Logarithmic	O(log n)	Quintic	O(n^5)
Polylogarithmic	$O(\log(n)^c)$	Sextic	O(n^6)
Linear	O(n)	Polynomial	$O(n^c)$
Linearithmic	O(n log n)	Exponential	$O(c^n)$
Quadratic	$O(n^2)$	Factorial	O(n!)
Cubic	O(n^3)		

Java

Strings

```
String s = "a*b*c";
s.charAt(i);
s.length();
s.substring(0, 1);// [0, 1)
s.substring(1);   // [1, s.length)
s.equals("b");
// return -1 bc s comes 1st in lexicographical order
s.compareTo("b*c*d");
```

```
new String(new char[ n times ]).replace("\0", "String To Be Repeated");
Integer.toBinaryString(Integer.parseInt(a, 2));

public String replaceFirst(String regex, String replacement)
Str.replaceFirst("(.*)Tutorials(.*)", "AMROOD"));
```

```
s.trim(); // Remove tailing and padding spaces
s.lastIndexOf('a');
new
StringBuilder(second).reverse().toString();
```

```

toUpperCase();
toLowerCase();

s.replaceAll("[^a-zA-Z0-9]", "")
```

```
.indexOf()

char[] chars = s.toCharArray();
Arrays.sort(chars);
String temp = new String(chars)

char[] arr = s.toCharArray();
String[] arr = s.split("\\*") // when delimiter is '*'
String[] arr = s.split("\\.") // when delimiter is '.'
String res = String.join(String delimiter, List<String> data); // use the delimiter to
concatenate the string in data.
Objects.equals(Object a, Object b); // (1)if both parameters are null return true
                                    // (2)if exactly one parameter is null return false
                                    // (3)return the result of invoking the equals() method
of the first parameter passing it the second parameter
                                    // This behaviour means it is "null safe".
```

String Builder

```
 * StringBuilders not override equals.
 * However, it does implement Comparable
   (since Java 11).
 * Therefore, you can check s1.compareTo(s2) == 0
   to find out if the string representations are
   equal.
```

```
StringBuilder sb = new StringBuilder(s);
StringBuilder sb = new StringBuilder();
sb.append("a");
// sb.insert(int offset, char c)
// or sb.insert(offset, str)
sb.insert(0, "a");
sb.setCharAt(index, ‘c’);
```

String, Char, Int Conversion

```
Integer.parseInt(s);
Integer.valueOf(s);
String.valueOf(int)
String str = new String(chArray);
String[] arr = list.toArray(new
String[list.size()]);
 List<String> list = Arrays.asList(arr);
Arrays.asList("first", "second");
```

```
sb.deleteCharAt(int index);
sb.reverse();
sb.toString();
sb.length();
str.length();
sb.indexOf();
```

Arrays

```
/**
 * Maximum size of an array: 2,147,483,647
 */
int[] arr = new int[10];
Arrays.sort(arr);
Arrays.fill(arr, -1);
public void helper(int[] nums);
helper(new int[]{1, 2});
Arrays.asList(array)
int[] newArray = Arrays.copyOf(oldArray, oldArray.length)
Collections.shuffle(Arrays.asList(array))

int m = (right + left) / 2
int m = left + ((right - left) / 2)
```

int Arrays

```
new int[]{1, 2, 3}
Stream<int[]> stream = Stream.of(height)
array[index]
Arrays.asList(array).indexOf(4)
Arrays.sort(nums);
Arrays.stream(tab).min().getAsInt()
Arrays.stream(tab).max().getAsInt()
Arrays.asList(array).contains(key)

Arrays.stream(nums).boxed().collect(Collectors.toList())
Arrays.stream(nums).boxed().collect(Collectors.toSet())
list.stream().mapToInt(i -> i).toArray()

int sum = Arrays.stream(array).sum()
double average = Arrays.stream(array).average().orElse(Double.NaN)

int[] reversedArray = IntStream.rangeClosed(1, array.length)
                               .map(i -> array[array.length - i])
                               .toArray()
int[] uniqueArray = Arrays.stream(array).distinct().toArray()
```

Two Dimensional int Arrays

```
int[][] arr = new int[rows][cols];

int[][] arr = list.stream()
    .map(row -> row.stream()
        .mapToInt(Integer::intValue)
        .toArray())
    .toArray(int[][]::new);

// Sorting a 2D int array in ascending order based on the first element of each subarray:
Arrays.sort(arr, Comparator.comparingInt(a -> a[0]));
```

```
// Sorting a 2D int array in descending order based on the first element of each subarray
Arrays.sort(arr, (a, b) -> Integer.compare(b[0], a[0]));

// Sorting a 2D int array based on a custom comparator:
class comp implements Comparator<int[]> {
    public int compare(int[] a, int[] b) {
        return a[0] - b[0];
    }
}
Arrays.sort(arr, new MyComparator());
```

Lists

```
/**
 * LinkedList, ArrayList, Singly Linked List, Doubly Linked List
 */
List<Integer> list = new ArrayList<>();
list.add(14);
// list.add(int index, int value);
list.add(0, 10);
list.get(int index);
list.remove(list.size() - 1);
// replaces element at index and returns original
list.set(int index, int val);
// return first index of occurrence of specified element in the list; -1 if not found
list.indexOf(Object o);
// return a sublist within range [fromIndex, toIndex)
list.subList(int fromIndex, int toIndex);
Collections.sort(list);
// ascending order by default
Collections.sort(list, Collections.reverseOrder());
// descending order
Collections.sort(list, new Comparator<Integer>() {
   @Override
   public int compare(Integer o1, Integer o2) {
   // the Integer can be any Object instead
   return o1 - o2;// 0->1
   // return o2-o1; 1->0
   }
});

// with nodes
Collections.sort(array, (x, y) -> Integer.compare(x.val, y.val));

list.forEach(num -> system.out.println(num));
// traverse the list and print out by using lamda function

List<List<Integer>> list = Arrays.stream(arr)
```

```
    .map(row -> Arrays.stream(row)
        .boxed()
        .collect(Collectors.toList()))
    .collect(Collectors.toList());

// Copying part
Arrays.copyOfRange(nums, i, i+k);
// Copying
new ArrayList<>(list);

// Declaring with vals:
new ArrayList<>(Arrays.asList("xyz", "abc"));
new ArrayList<>(Arrays.asList(array));
```

HashMap	HashSet

HashMap<Character, Integer> map = new HashMap<Character, Integer>();
map.put('c', 1);
map.get('c');
map.getOrDefault(key, defaultValue);
map.remove('c'); // remove key and its value
map.computeIfAbsent(key, mappingFunction);
map.computeIfAbsent(key, k -> new HashSet<>()).add(val);
map.computeIfAbsent(key, k -> new ArrayList<>()).add(val);
if (map.containsKey('c'))
if (map.containsValue(1))
for (Character d : map.keySet())
for (Integer i : map.values())
	HashSet<Integer> set = new HashSet<Integer>();
	set.add(10);
	set.remove(10);
	if(set.contains(10))
	set.size();
	set.isEmpty();
	// setA keeps the intersection of original setA & setB;
	setA.retainAll(setB);
	setB.removeAll(setC);
	setC.addAll(setD);
	setC.containsAll(setD);
	Object[] arr = setA.toArray();

for(Map.Entry<Character, Integer> entry : map.entrySet()){
    entry.getKey();
    entry.getValue();
}
// traverse key-value pair using lamda expression to
// print out info
map.forEach((k,v) -> System.out.println("key: "+k+" value:"+v));
 	
map.isEmpty();
map.size();
 	
// using a multidimensional array is faster than:
Using a map<Integer, Set<Integer>> map
 	
List<String> candidates = new ArrayList<>(cnt.keySet());
 	
// used to combine multiple mapped values for a key using the given mapping function:
	// this works for up to 10 elements:
	Map<String, String> test1 = Map.of(
	    "a", "b",
	    "c", "d"
	);
	
	// this works for any number of elements:
	import static java.util.Map.entry;
	Map<String, String> test2 = Map.ofEntries(
	    entry("a", "b"),
	    entry("c", "d")
	);

```
map.merge(key, value, BiFunction remappingFunction)
```

```
LinkedHashMap<String, Integer> sorted = map.entrySet()
                .stream()
                .sorted(Map.Entry.<String, Integer>comparingByValue()
                        .reversed()
                        .thenComparing(Map.Entry.comparingByKey()))
                .collect(Collectors.toMap(
                                e -> e.getKey(),
                                e -> e.getValue(),
                                (e1, e2) -> null,
                                () -> new LinkedHashMap<String, Integer>()
                        )
                );

List<String> topK = new ArrayList<>(map.keySet());
Collections.sort(topK, (w1, w2) -> map.get(w1).equals(map.get(w2)) ? w1.compareTo(w2) :
map.get(w2) - map.get(w1));
```

TreeMap

```
 * https://www.geeksforgeeks.org/treemap-in-java/
 * Java.util.TreeMap uses a red-black tree
 * null key or null value is not permitted
 * Always stores key-value pairs which are in sorted order on the basis of the key
 * Descending order: Switch the x & y vals: (x, y) -> Integer.compare(y, x)
```

```
// key's ascending order (default) - lexicographical order
TreeMap<Integer, String> map = new TreeMap<>();
// descending order
TreeMap<Integer, Integer> m = new TreeMap<>(Collections.reverseOrder());
```

```
map.put(2, "b");
map.put(1, "a");
map.put(3, "c");
```

```
// traverse in "a" "b" "c" order
for(String str : map.values())

// traverse in  1, 2, 3 order
for(Integer num : map.keySet())
```

```
// return the max key that < k
treeMap.lowerKey(k);

// return the min key that >= k
treeMap.floorKey(k);

// return the min key that > k
```

```
treeMap.higherKey(k);

// return the max key that <= k
treeMap.ceilingKey(k);

// returns the 1st (lowest) key currently in this map.
treeMap.firstKey();
```

```
// Returns a view of the portion of this map whose keys are strictly less than toKey.
SortedMap<K,V> portionOfTreeMap = treeMap.headMap(K toKey);

// Returns a view of the portion of this map whose keys are less than or equal to toKey.
NavigableMap<K,V> map = treeMap.headMap(toKey, true);
```

TreeSet	

Instantiating	`// sort in ascending order by default`
	`Set<Integer> treeSet = new TreeSet<>();`

Get	`// return greatest element that is < e, or null if no such element`
	`treeSet.lower(Integer e);`
	
	`// return greatest element that is <= e, or null if no such element`
	`treeSet.floor(Integer e);`
	
	`// return smallest element that is >= e, or null if no such element`
	`treeSet.ceiling(Integer e);`
	
	`// return smallest element that is > e, or null if no such element`
	`treeSet.higher(Integer e);`
	
	`// return the first element in the treeset (if min set, return minimum element)`
	`treeSet.first();`
	
	`// return the last element in the treeset`
	`treeSet.last();`

Stack	Queue

```java
/**
 * LinkedList can be used
 */
Stack<Integer> stack = new Stack<Integer>();
stack.push(10);
stack.pop();
stack.peek();
stack.isEmpty();
stack.size();
```

```java
Queue<Integer> q = new LinkedList<Integer>();
q.offer(10);
// q.add() is also acceptable
q.poll();
q.peek();
q.isEmpty();
q.size();
```

PriorityQueue

```java
/**
 * Priority Queue: an abstract data type that is similar to a queue,
   and every element has some priority value associated with it.
 * The priority of the elements in a priority queue determines the order in which elements
are served (i.e., the order in which they are removed).
 * If in any case the elements have same priority, they are served as per their ordering in
the queue
 * queue.offer(entry): Inserts the specified element into this priority queue
 * The poll() method returns and removes the element at the front end of the container.
 * Enqueing & Dequeing (offer, poll, remove() and add) → O(log n)
 * remove(Object) & contains(Object) → O(n)
 * Retrieval methods (peek, element, and size) → O(1)
 */
// minimum Heap by default
PriorityQueue<Integer> pq = new PriorityQueue<>();
PriorityQueue<Integer> pq = new PriorityQueue<>(Comparator.naturalOrder());

// change to maximum Heap
PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
PriorityQueue<Integer> pq = new PriorityQueue<>((x, y) -> Integer.compare(x, y));

pq.add(10);
pq.poll();
pq.peek();
pq.isEmpty();
pq.size();
class Node implements Comparable<Node>{
    int x;
    int y;
    public Node(int x, int y){
        this.x = x;
        this.y = y;
    }
    @Override
    public int compareTo(Node that){
        return this.x - that.x;                        // ascending order / minimum Heap
        // return that.x - this.x;                     // descending order / maximum Heap
    }
}
PriorityQueue<Node> pq = new PriorityQueue<>();
```

ArrayDeque

```
/**
 * (Array Double Ended Queue, ArrayDeck) growable array that allows us to add or remove an
element from both sides.
 */
import java.util.Deque;
Deque<Integer> dq = new LinkedList<Integer>();    // Deque is usually used to implement
monotone queue
dq.addFirst();  //  dq.offerFirst();
dq.addLast();   //  dq.offerLast();
dq.peekFirst(); //
dq.peekLast();
dq.pollFirst(); //  dq.removeFirst();
dq.pollLast();  //  dq.removeLast();
```

LinkedHashMap

```
/**
 * https://www.geeksforgeeks.org/linkedhashmap-class-in-java/
 * Just like HashMap with an additional feature of maintaining an order of elements inserted
into it.
 * The implementation of the LinkedHashMap is very similar to a doubly-linked list.
 * Is not synchronized:
 * If multiple threads access a linked hash map concurrently, and at least one of the threads
modifies the map structurally, it must be synchronized externally.
 * This is typically accomplished by synchronizing on some object that naturally encapsulates
the map.
 * If no such object exists, the map should be "wrapped" using the
Collections.synchronizedMap method.
 * This is best done at creation time, to prevent accidental unsynchronized access to the
map.
 */
Map<Integer,String> map = new LinkedHashMap<>();
map.put(1, "first");
map.put(2, "second");
map.put(3, "third");
for(Map.Entry<Integer,String> entry : map.entrySet())
    System.out.println(entry.getKey(), entry.getValue());   // print order: 1, 2, 3
```

LinkedHashSet

```
Set<Integer> set = new LinkedHashSet<>();
```

Enum

```
set1 = EnumSet.of(Gfg.QUIZ, Gfg.CONTRIBUTE,  Gfg.LEARN, Gfg.CODE);
set2 = EnumSet.complementOf(set1);
// initially containing all the elements of this type that are not contained in the specified
set
set3 = EnumSet.allOf(Gfg.class);
set4 = EnumSet.range(Gfg.CODE, Gfg.CONTRIBUTE);
```

```
// contains all of the elements in the range defined by the two specified endpoints.
```

Random method

```
Random rand =new Random();    // initialize Random object
int i = rand.nextInt(100);    // generate random number in [0, 100)
float f = rand.nextFloat();   // generate float value in [0, 1)
double d = rand.nextDouble(); // generate double value in [0.0, 1.0)
```

Collections/Object

```
// return an immutable list which contains n copies of given object
Collections.nCopies(100, new Object[]{true});

// Returns the runtime class of this {@code Object}
getClass()

// use it to replace Arrays.asList() when there is only one element
Collections.singletonList()

// returns an unmodifiable view of the specified set. Note that, changes in specified set
will be reflected in unmodifieable set.
Collections.unmodifiableSet(new HashSet<>())

// Also, any modification on unmodifiableSet is not allowed, which triggers exception.
Collections.swap(List, int i, int j);        // swap the ith and jth element in list
```

Lamda expression

```
1. Functional interface: the interface contains exactly one abstract method
   @FunctionalInterface
   public interface Sprint {
      public void sprint(Animal animal);
   }
2. lamda expression
   a -> a.canHop()
   (Animal a) -> { return a.canHop(); }
```

std input/output file read/write

```
import java.io.*;
import java.net.*;
Scanner in = new Scanner(System.in);
int n = in.nextInt();
while(in.hasNext()){
    String str = in.nextLine();
}

String inputfile="in.txt";
String outputfile="out.txt";
try
{
    BufferedReader in = new BufferedReader(new FileReader(inputfile));
    line = in.readLine();
    while (line!=null)
    {
        // do something with line
        line=in.readLine();
    }
    in.close();              // close the file
} catch (IOException e) {e.printStackTrace();}
try {
    BufferedWriter out = new BufferedWriter(new FileWriter(outputfile));
    for(String str : map.keySet()){
         out.write(str + " " + map.get(str));
         out.newLine();
    }
    out.close();           // close the file
} catch (IOException e) { e.printStackTrace(); }
URL wordlist = new URL("http://foo.com/wordlist.txt");
BufferedReader in = new BufferedReader(new InputStreamReader(wordlist.OpenStream()));
String inputLine = null;
List<String> res = new ArrayList<>();
while((inputLine = in.readLine()) != null){
    res.add(inputLine);
}
```

Atomic Class

NavigableMap

- It is an extension of SortedMap which provides convenient navigation methods like lowerKey, floorKey, ceilingKey and higherKey, and along with this popular navigation method. It also provide ways to create a Sub Map from existing Map in Java e.g. headMap whose keys are less than the specified key, tailMap whose keys are greater than the specified key, and a subMap which strictly contains keys which fall between toKey and fromKey.

ConcurrentNavigableMap

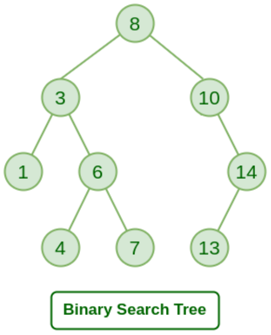
- Provides thread-safe access to map elements along with providing convenient navigation methods.
- It extends from the NavigableMap interface and ConcurrentMap interface.

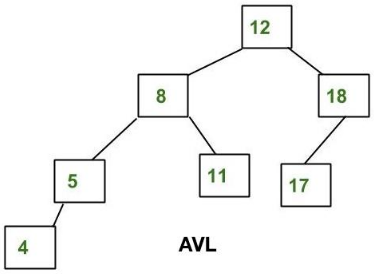
ConcurrentSkipListMap

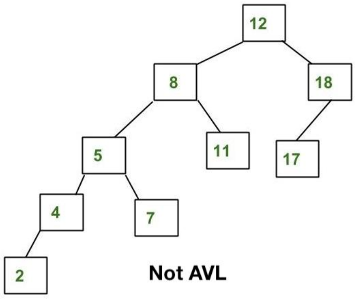
- A scalable implementation of ConcurrentNavigableMap.

AVL Tree (Adelson-Velsky and Landis)

- Self-balancing BST where the difference between heights of left and right subtrees cannot be more than one for all nodes.







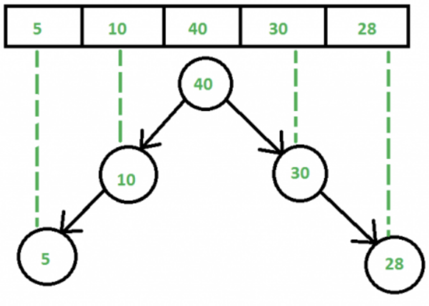
Binary Tree (BTree): A tree data structure where each node has at most 2 children

Red-Black Tree:

- Kind of self-balancing BST where each node has an extra bit, interpreted as the color (red or black).
    - These colors are used to ensure that the tree remains balanced during insertions and deletions.
    - Although the balance of the tree is not perfect, it is good enough to reduce the searching time and maintain it around O(log n) time.
- Rules:
    - Every node has a color either red or black.
    - The root of the tree is always black.
    - There are no two adjacent red nodes (A red node cannot have a red parent or red child).
    - Every path from a node (including root) to any of its descendants NULL nodes has the same number of black nodes.
    - All leaf nodes are black nodes.

Cartesian Tree

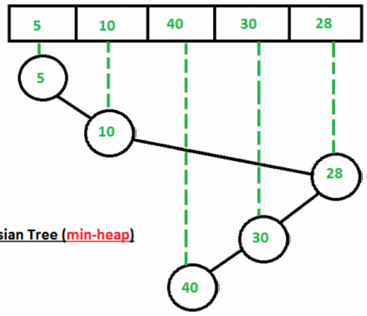
- A tree created from a set of data that obeys the following structural invariants:
    - The tree obeys in the min (or max) heap property – each node is less (or greater) than its children.
    - An inorder traversal of the nodes yields the values in the same order in which they appear in the initial sequence.
- Suppose we have an input array- {5,10,40,30,28}. Then the max-heap Cartesian Tree would be:



A sequence and its corresponding Cartesian tree

Note that this is a max-heap Cartesian Tree.

Similarly a min-heap Cartesian Tree is also possible.



A sequence and its Cartesian Tree (min-heap)

Splay Tree:

KD Tree:

Indexed Priority Queue:

- https://algs4.cs.princeton.edu/24pq/IndexMinPQ.java.html

Data Types

- boolean: 1 bit
- byte: 8
- short, char: 16
- Int, float: 32
- long, double: 64

Characters

- To int
    - s - ‘0’
    - a - ‘a’ == 0;
- Character.isDigit(str.charAt(i))
- Character.isWhitespace(c)
- Character.isLetter();
- chars not in 256 ASCII: !,

Char Array

- To String: char[] a = {‘a’, 'b'};
    1. new String(a);
    2. String.valueOf(a);
    3. String.copyValueOf(a)

Immutability

- An object is immutable when its state doesn't change after it has been initialized.
    - i.e., String is an immutable class, and, once instantiated, the value of a String object never changes.

Sorting in reverse order

- Objects
    - Arrays.sort(array, Collections.reverseOrder());
- Primitive ints: Can’t!
    a. Write your own sorting method
    b. Convert to object

Integers

- to char

```
(char)(s + '0'))
A == (char)(1 + 64))
a == (char)(s + '0'))
```

Lists of Lists

- Merging many lists into one lists

```
        Stream.of(list1, list2, list3,list4)
              .flatMap(Collection::stream)
              .collect(Collectors.toList());
```

Tree

- Traversals:
    - BFS: Node, Left, Right
    - DFS:
        - In-order: Left, Node, Right
        - Post-order: Left, Right, Node
        - Pre-order: Node, Left, Right
- Making the solution iterative will often increase speed.
- Problems:
    - Kth Smallest Element in a BST

Overflows

- Multiply by 1000000007

Math & Bits

```
/**
 * Integer arithmetic always rounds down (towards zero).
 * XOR: Exclusive or, is ^
 * i.e, 8 ^ 6 == 1000 ^ 0110 == 1110 == 14
 * Integer.toBinaryString(x)
 * Gauss' Formula to compute the sum of numbers in a range from 0 to n: n * (n + 1) / 2
 */
Math.pow(double x, double y); // return x^y
Math.round(float a);          // returns the closest int to the argument
Math.abs(int/float/doubld val);
Math.sqrt();
Math.sin(double rad);         // input is rad not angle
Math.PI;
Math.E;

// returns the angle theta from the conversion of rectangular coordinates (x, y) to polar
coordinates (r, theta), where r = sqrt(x^2 + y^2) and theta is in radians.
Math.atan2(double y, double x);

// Compute the number of ones in a bit representation of a number using popCount:
int count = 0;
for (int x = i; x != 0; ++count) {
       x &= x - 1; // zeroing out the least significant nonzero bit
}
return count;

// Sum of to ints using bits
(a | b) + (a & b)

// Set given index bit to 1
n = ((1 << index) | n);

// Set given index bit to 0
n &= ~(1 << num);

// Check if given index bit is set to 1
(n & (1 << i)) == 0)
```

Lambda Functions

Background

- A way to write anonymous functions that can be passed around as variables.
- Can be used to simplify code, reduce redundancy, and make it easier to write functional-style code.
- Warning: The classes for lambda expressions are generated at runtime rather than being loaded from your class path

Syntax

- (parameters) -> expression
  - Or -> { return expression; }

```
BiPredicate<Integer, Integer> comparison = (a, b) -> arr[a] > arr[b]
comparison.test(i, i + 1);
```

Python

Methods

- Enumerate() method adds a counter to an iterable and returns it in a form of enumerating object.

Arrays

Sorting

```
array.sort()
```

Two Dimensional int Arrays

Instantiation

- Creating a new 2D int array:

```
arr = [[0 for col in range(cols)] for row in range(rows)]
```

Iterating

```
for i in range(len(arr)):
    for j in range(len(arr[i])):
        print(arr[i][j])
```

Sorting

- Sorting a 2D int array in ascending order based on the first element of each subarray:

```
arr.sort(key=lambda x: x[0])
```

- Sorting a 2D int array in descending order based on the first element of each subarray:

```
arr.sort(key=lambda x: x[0], reverse=True)
```

- Sorting a 2D int array based on a custom comparator:

```
from operator import itemgetter
my_comparator = itemgetter(0)
arr.sort(key=my_comparator)
```

- Sorts a list of intervals in ascending order based on their start points and in descending order based on their end points.

```
arr.sort(key=lambda x: (x[0], -x[1]))
```

Strings

- substring

x[2:]

List

Declaring

```
list = []
```

Retrieving value

```
list[index]
```

Adding value

```
list.append("orange")
```

Can consist of elements belonging to different data types

Set

- mySet = set()
- mySet.add()

Dictionaries

Creating	dict = {}
	dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
	dict = dict(key1='value1', key2='value2', key3='value3')
	dict = Counter([‘A’, ‘B’, ‘C’, ‘A’, ‘B’, ‘C’])

Accessing vals	my_dict['key1'] my_dict.get('key2')

Iterating over	# Iterating over keys
	for key in my_dict:
	    print(key)
	
	# Iterating over values
	for value in my_dict.values():
	    print(value)
	
	# Iterating over key-value pairs
	for key, value in my_dict.items():
	    print(key, value)

Adding & Updating Elements	# Adding a new key-value pair
	my_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
	my_dict['key4'] = 'value4'
	
	# Updating the value of an existing key
	my_dict['key1'] = 'new_value1'

Removing Elements	# Removing a key-value pair
	my_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
	del my_dict['key2']
	
	# Removing all key-value pairs
	my_dict.clear()

Sorting	# Sorting by key
	sorted_dict = dict(sorted(my_dict.items()))
	
	# Sorting by value
	sorted_dict = dict(sorted(my_dict.items(), key=lambda x: x[1]))

Merging Two	# Using the update() method
	dict1 = {'key1': 'value1', 'key2': 'value2'}
	dict2 = {'key3': 'value3', 'key4': 'value4'}
	dict1.update(dict2)
	
	# Using the ** operator
	dict1 = {'key1': 'value1', 'key2': 'value2'}
	dict2 = {'key3': 'value3', 'key4': 'value4'}
	merged_dict = {**dict1, **dict2}

Stack

Comparable

- x = max(5, 10)

Max & Min

- import sys
  - max_size = sys.maxsize
  - min_size = -sys.maxsize - 1
- Max:
  - float('inf')
  - pow(10, 5)

Mathematical

- import math → (math.pow(9, 3))
- float('inf') is a special float value that represents infinity.
- The random module in Python provides functions for generating random numbers, such as random (a random float between 0 and 1), randint (a random integer between two specified values), and choice (a random choice from a sequence).
- Python supports complex numbers, which can be created using the complex function or by using the j suffix on a numeric literal to represent the imaginary component. For example, 3 + 4j is a complex number with a real part of 3 and an imaginary part of 4.
- The numpy library provides powerful mathematical functions and structures for working with arrays, matrices, and vectors. It includes functions for linear algebra, Fourier analysis, and more.
- Python supports various operators for mathematical operations, such as + for addition, - for subtraction, * for multiplication, / for division, % for modulus, and ** for exponentiation.

Classic Differences Between Python and Java

- Boolean: In Python, they start with capital letters - True and False
- To refer to class variables: use the name of the class instead of the keyword this.
- Methods:
  - If you pass the method the keyword self, i.e. foo(self)
    - To call the method you must use the keyword self, i.e. self.foo()
  - If you don't pass the method the keyword self
    - You use the class name to call the method, i.e. Solution.foo()

Questions & Topics to Review:

What's the difference between & and && in Java?

Classic Problems

Graphs

Cycle Detection	BFS 1. Build an adj list 2. Update in degrees 3. Add all nodes to queue with inDegrees of 0 4. while (queue is not empty): a. decrement in degree for all children b. enqueue child if their inDegree is 0 5. return a. true if nodes visited == V b. otherwise false • Time: O(E log V) • Space: O(V)

Palindrome

- Iteratively InwardOut, OutwardIn

Rotated Array

- Binary Search (find rotation element)

Word Search I

- Backtracking

Word Search II

- Backtracking with Trie

Tracking coordinates or a combination of 2 numbers:

	`int coordinate = row * column_size + column`
	`int row = coordinate / column_size`
	`int column = coordinate % column_size`

Use arrays
Hash coordinates to indexes from 0 to n
	`int prime = n`
	`int N = ((n - 1) * (prime - 1)) - 1`
	`boolean b = n1 < n2`
	`int x = b ? n1 : n2`
	`int y = b ? n2 : n1`
	`int p = (x * prime + y) % (N * prime) - 1`

Map<Point, Val>	

Use a trick	`bool b = n1 < n1`
	`int x = b ? n1 : n2`
	`int y = b ? n2 : n1`
	`int p = (x  <<  16) | y`

Point subclass	`static class Point {`
	`    int x, y;`
	`    public Point (int X, int Y) { x = X; y = Y; }`
	`    public boolean equals(Object o) {`
	`        Point c = (Point) o;`
	`        return c.x == x && c.y == y; }`
	`    public int hashCode() {Objects.hash(x, y);}}`

Two-dimensional boolean array	
Set of strings	

Turn equation string into lists: String[] strings = s.split("(?<=[-+*/=()])|(?=[-+*/=()])");