# Binary Search Problems - I

# Search an element in an Infinite Sorted Array

# Find square root of an Integer

# Find the median of two sorted Arrays

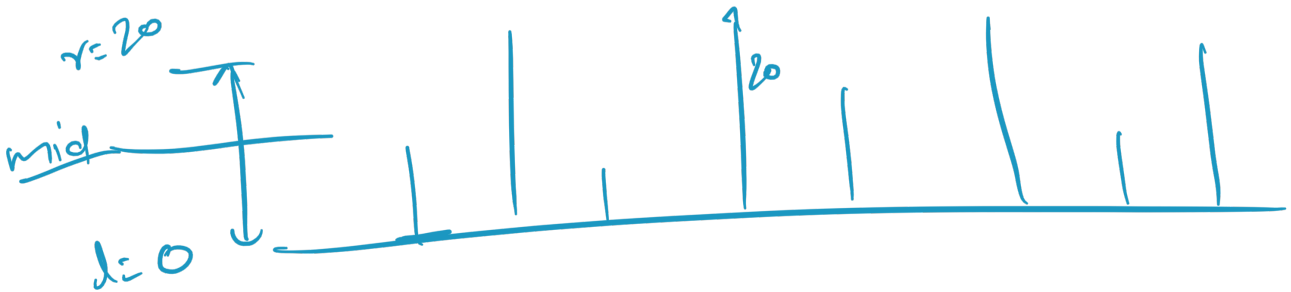
# Practice Problems

1. https://www.interviewbit.com/courses/programming/binary-search

# Binary Search Problems - II

# Tree Cutter Problem

Given an array of length ‘N’, where each element denotes the height of a tree all placed in one row. Your task is to chop down trees so that you can get at least ‘k’ length of wood in total. Determine the maximum height at which a blade should be placed so that it can cut the tree above that height to get at least ‘k’ length of wood in total.



# Books Allocation Problem

Given an array of integers A of size N and an integer B.

College library has N bags,the ith book has A[i] number of pages.

You have to allocate books to B number of students so that maximum number of pages alloted to a student is minimum.

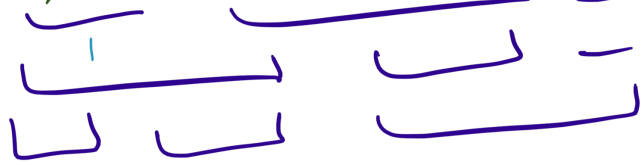
A book will be allocated to exactly one student.
Each student has to be allocated at least one book.
Allotment should be in contiguous order, for example: A student cannot be allocated book 1 and book 3, skipping book 2.


Calculate and return that minimum possible number.

NOTE: Return -1 if a valid assignment is not possible.

a[] = { 4, 3, 6, 1, 4, 2 }

k = 3



l = 1

r = 19

m = 10

# Aggressive Cows Problem

Given an array of length ‘N’, where each element denotes the position of a stall. Now you have ‘N’ stalls and an integer ‘K’ which denotes the number of cows that are aggressive. To prevent the cows from hurting each other, you need to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. Return the largest minimum distance.

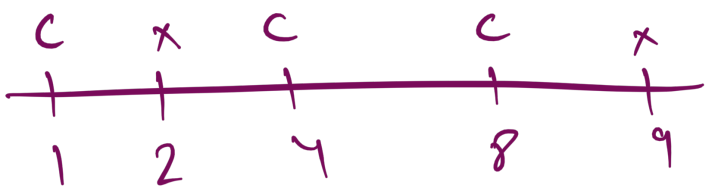
Eg

array: 1,2,4,8,9 & k=3

O/P: 3

Explaination: 1st cow at stall 1 , 2nd cow at stall 4 and 3rd cow at stall 8

K = 3




PC_K

l = 0
r = 9 mid = 4

↓

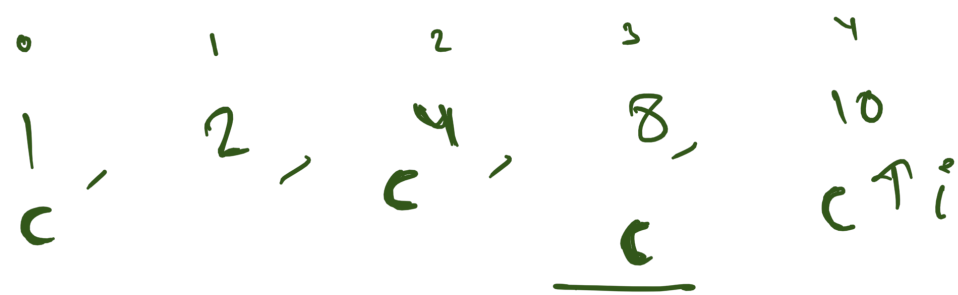
l = 0 mid = 1
r = 3

↓

l = 2 mid = 2
r = 3

↳ l = 3 mid = 3 → l = 4, r = 3 ✗
r = 3

ans = ~~1~~ ~~2~~ 3 ✓

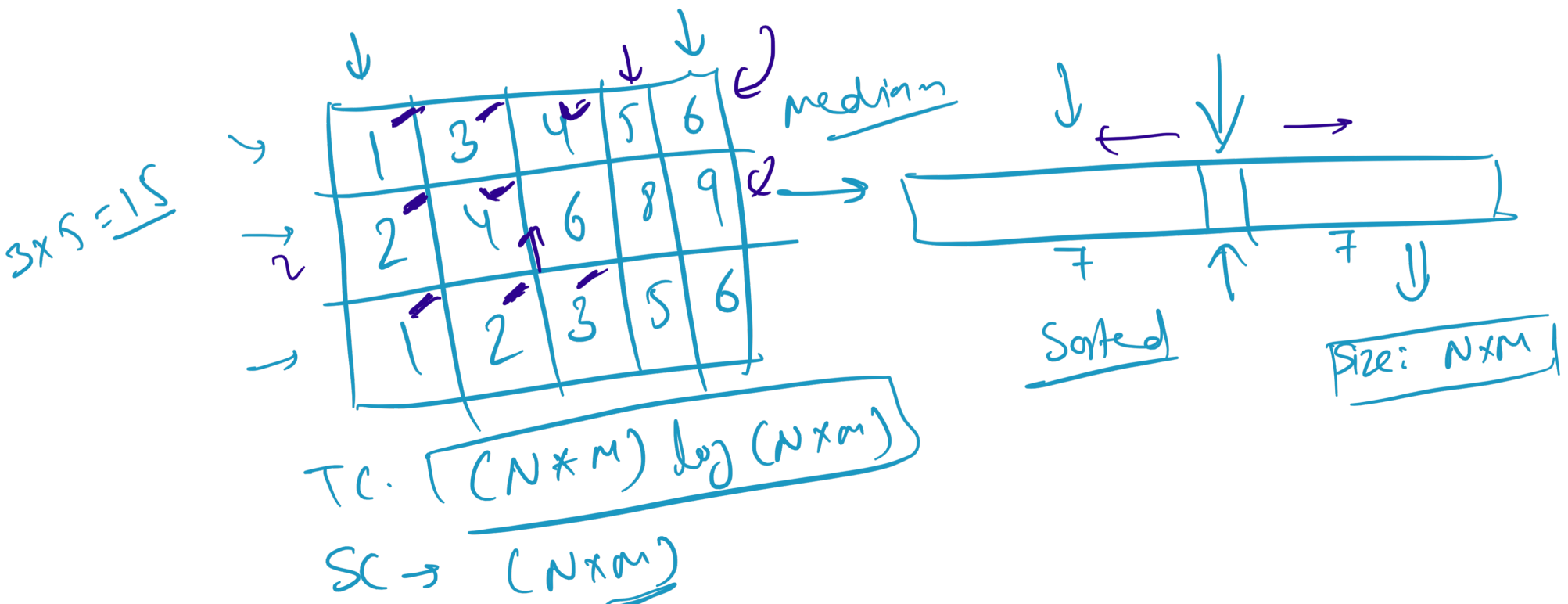



Prev = 10

Cows = 4

(5)

dis = 2



$O(N \times M \log N \times M) +$

l = 1 mid = 5
r = 9

l = 1 mid = 2
r = 4

l = 3 mid = 3
r = 4

l = 4 mid = 4
r = 4

= Arrays.binarySearch(a[], x)

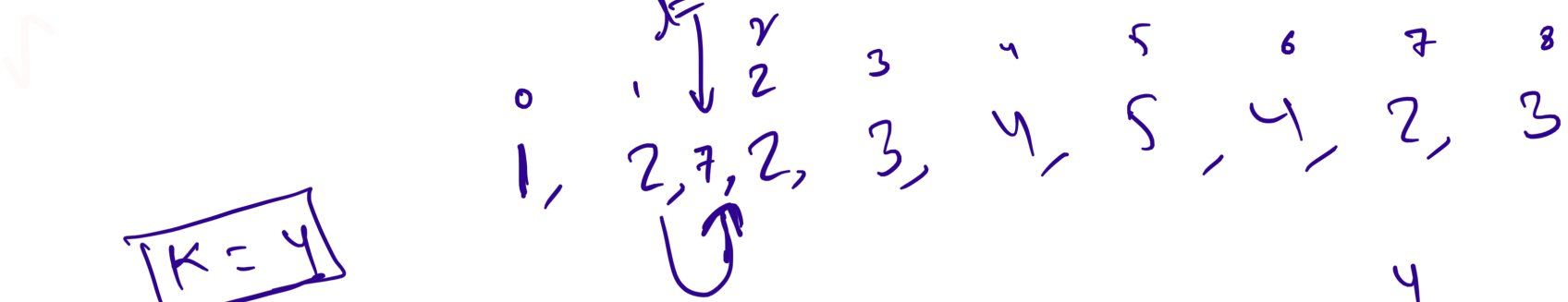
insert position →

[ 1, 2, 2, 3, 3, 3, 4 ]

```
int a[][] = new int [4][7]
int n = a.length;
int m = a[0].length;
```

# Practice Problems

1. Given a matrix of integers A of size N x M in which each row is sorted. Find an return the overall median of the matrix A.
2. Painter's partition Problem.
3. https://www.interviewbit.com/courses/programming/binary-search


0 1 2 3 4 5 6 7 8
1, 2, 7, 2, 3, 4, 5, 4, 2, 3
K = 4
1 → 0
2 → 3
7 → 2
1 4 8 1
4
D.P.
LCS
variations
1 4 5 1 → zeros
a → 1 2 5 2 4 1
b → 1 4 2 5 2 1
1 2 5 2 1
LCS

a → 3 4 1 3 1 5 →

b → 5 1 3 1 4 3

↓↓

| 3 | 1 | 3 |
|---|---|---|

layout

Moore's voting algorithm

1 → N

↓↓

3

↓ ↓ ↓ ↓ ↓

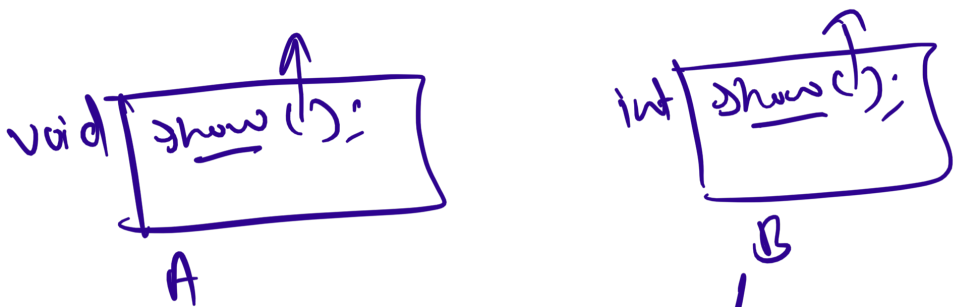
2, 2, 2, 2, 2, 1, 4, 3 → ∅

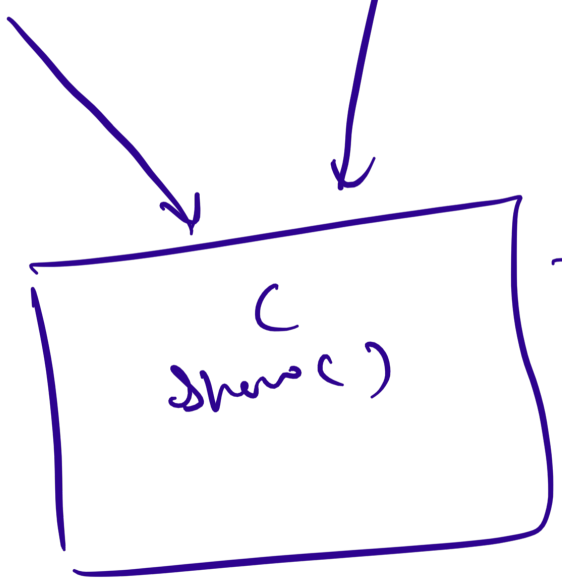
cur = X 2 X 2 3 X → 2

count = X 0 X 0 2 0 → 1

$\frac{7}{2} = 3 > \frac{N}{2}$ 3

# Diamond Problem



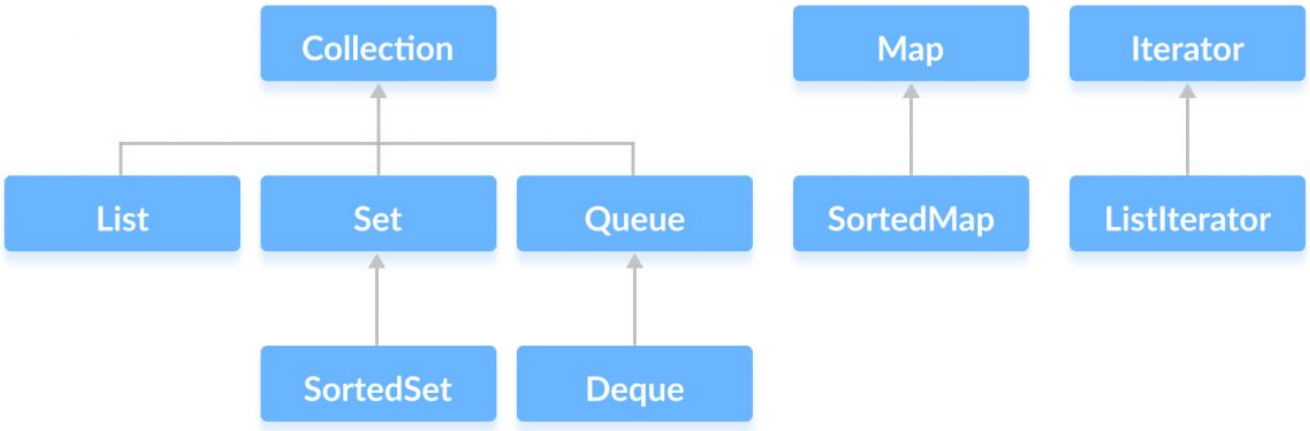




```
class C implements A, B {
    void show() {
    }
    int show() {
    }
}
```

```
obj.show(  )
obj.show();
```

# Collection Framework

# Java Collections Framework



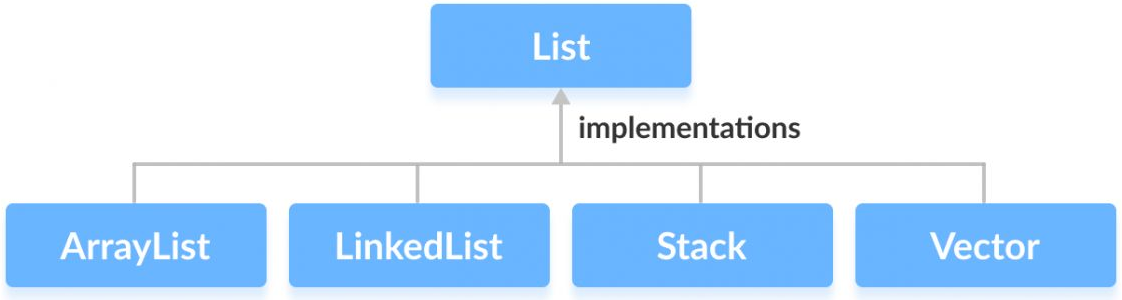
# Abstraction in OOPS

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essential units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

We can achieve Abstraction by these two methods:

1. Abstract Keyword
2. Interfaces

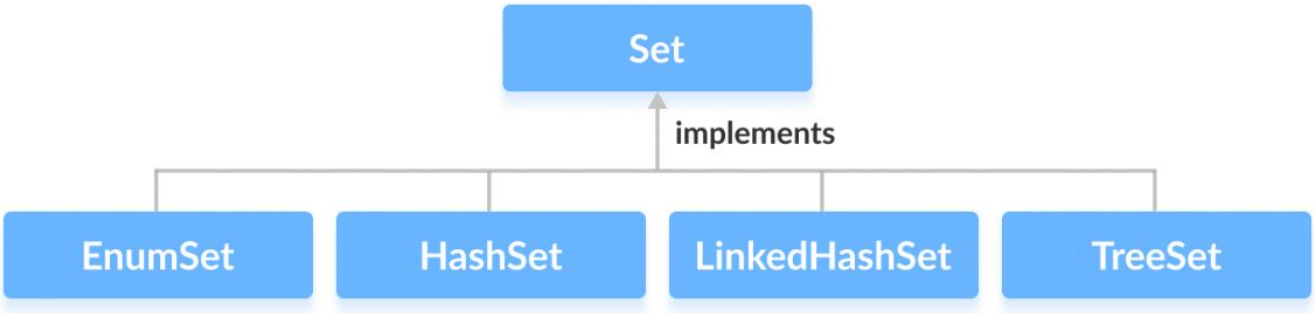
# List Interface



## Methods of List

- `add()` - adds an element to a list
- `addAll()` - adds all elements of one list to another
- `get()` - helps to randomly access elements from lists
- `iterator()` - returns iterator object that can be used to sequentially access elements of lists
- `set()` - changes elements of lists
- `remove()` - removes an element from the list
- `removeAll()` - removes all the elements from the list
- `clear()` - removes all the elements from the list (more efficient than `removeAll()`)
- `size()` - returns the length of lists
- `toArray()` - converts a list into an array
- `contains()` - returns `true` if a list contains specified element

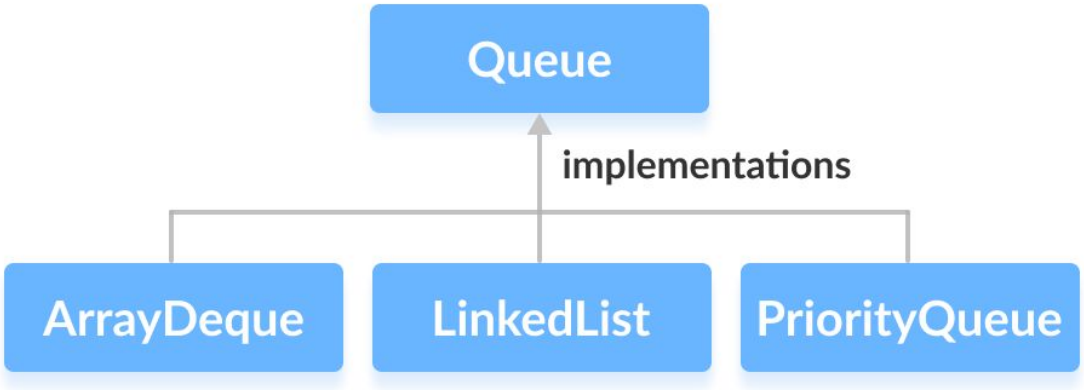
# Set Interface



## Methods of Set

- **add()** - adds the specified element to the set
- **addAll()** - adds all the elements of the specified collection to the set
- **iterator()** - returns an iterator that can be used to access elements of the set sequentially
- **remove()** - removes the specified element from the set
- **removeAll()** - removes all the elements from the set that is present in another specified set
- **retainAll()** - retains all the elements in the set that are also present in another specified set
- **clear()** - removes all the elements from the set
- **size()** - returns the length (number of elements) of the set
- **toArray()** - returns an array containing all the elements of the set
- **contains()** - returns `true` if the set contains the specified element
- **containsAll()** - returns `true` if the set contains all the elements of the specified collection
- **hashCode()** - returns a hash code value (address of the element in the set)

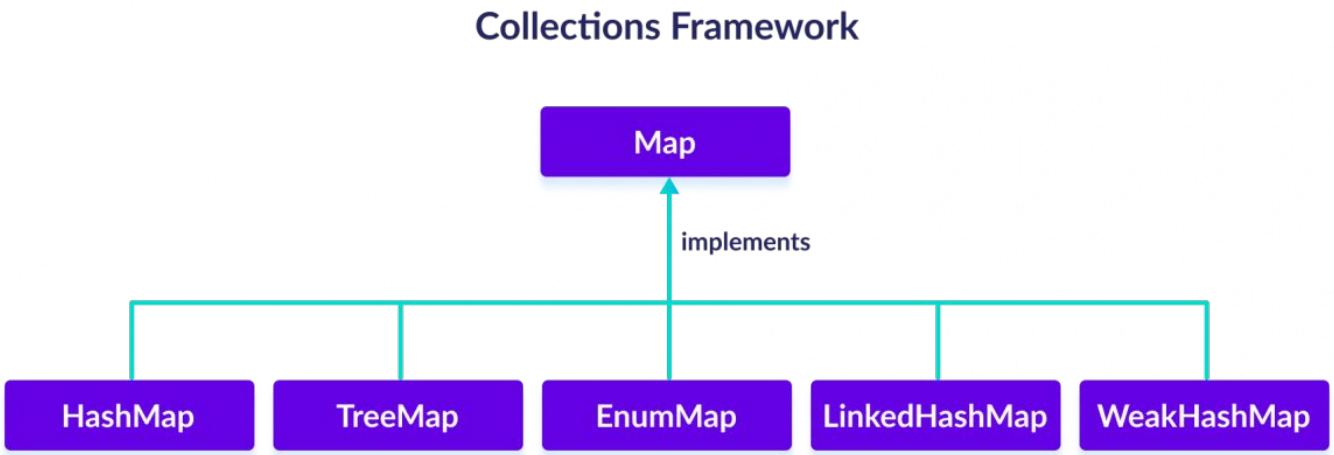
# Queue Interface



## Methods of Queue

- **add()** - Inserts the specified element into the queue. If the task is successful, `add()` returns `true`, if not it throws an exception.
- **offer()** - Inserts the specified element into the queue. If the task is successful, `offer()` returns `true`, if not it returns `false`.
- **element()** - Returns the head of the queue. Throws an exception if the queue is empty.
- **peek()** - Returns the head of the queue. Returns `null` if the queue is empty.
- **remove()** - Returns and removes the head of the queue. Throws an exception if the queue is empty.
- **poll()** - Returns and removes the head of the queue. Returns `null` if the queue is empty.

# Map Interface



## Methods of Map

- **put(K, V)** - Inserts the association of a key `K` and a value `V` into the map. If the key is already present, the new value replaces the old value.
- **putAll()** - Inserts all the entries from the specified map to this map.
- **putIfAbsent(K, V)** - Inserts the association if the key `K` is not already associated with the value `V`.
- **get(K)** - Returns the value associated with the specified key `K`. If the key is not found, it returns `null`.
- **getOrDefault(K, defaultValue)** - Returns the value associated with the specified key `K`. If the key is not found, it returns the `defaultValue`.
- **containsKey(K)** - Checks if the specified key `K` is present in the map or not.
- **containsValue(V)** - Checks if the specified value `V` is present in the map or not.
- **replace(K, V)** - Replace the value of the key `K` with the new specified value `V`.
- **replace(K, oldValue, newValue)** - Replaces the value of the key `K` with the new value `newValue` only if the key `K` is associated with the value `oldValue`.
- **remove(K)** - Removes the entry from the map represented by the key `K`.
- **remove(K, V)** - Removes the entry from the map that has key `K` associated with value `V`.
- **keySet()** - Returns a set of all the keys present in a map.
- **values()** - Returns a set of all the values present in a map.
- **entrySet()** - Returns a set of all the key/value mapping present in a map.

# Hashing & HashMap Basics

# HashSet and HashMap in Java

# hashCode and equals contract

- During the execution of the application, if hashCode() is invoked more than once on the same Object then it must consistently return the same Integer value
- If two Objects are equal, according to the equals(Object) method, then hashCode() method must produce the same Integer on each of the two Objects.
- If two Objects are unequal, according to the equals(Object) method, It is not necessary the Integer value produced by hashCode() method on each of the two Objects will be distinct.

# Count Distinct Elements

# Frequency of Elements in an Array

# Pair with given sum in an Unsorted Array

# Zero Sum Subarray

# Practice Problems

1. Subarray with given Sum.
2. Intersection of Two Arrays.
3. Union of Two Arrays.
4. Find the largest subarray with zero sum.
5. Count distinct elements in every window of size k.