Links:

<http://algorithms.tutorialhorizon.com/implement-queue-using-stacks/>

<http://www.careerride.com/Interview-Questions-Data-Structures-Stack-queue.aspx>

Data structures stack and queue interview questions - stack and queue interview

Stack is a data structure that follows Last in First out strategy.  
**Stack Operations:-**  
Push – Pushes (inserts) the element in the stack. The location is specified by the pointer.  
Pop – Pulls (removes) the element out of the stack. The location is specified by the pointer  
Swap: - the two top most elements of the stack can be swapped  
Peek: - Returns the top element on the stack but does not remove it from the stack  
Rotate:- the topmost (n) items can be moved on the stack in a rotating fashion  
A stack has a fixed location in the memory. When a data element is pushed in the stack, the pointer points to the current element

## Describe queue operation.

Queue is a data structure that follows First in First out strategy.  
**Queue Operations:**  
Push – Inserts the element in the queue at the end.  
Pop – removes the element out of the queue from the front  
Size – Returns the size of the queue  
Front – Returns the first element of the queue.   
Empty – to find if the queue is empty.

## Explain stacks and queues in detail.

A stack is a data structure based on the principle Last In First Out. Stack is container to hold nodes and has two operations - push and pop. Push operation is to add nodes into the stack and pop operation is to delete nodes from the stack and returns the top most node.  
  
A queue is a data structure based on the principle First in First Out. The nodes are kept in an order. A node is inserted from the rear of the queue and a node is deleted from the front. The first element inserted in the first element first to delete.

=============== Data structures stack================

Implement Stack Using Linked List

BY [SJ](http://algorithms.tutorialhorizon.com/author/sumitjain/) · MARCH 19, 2016

**Objec­tive:**Write an algo­rithm to imple­ment Stack using Linked List.

If you do not know about then for starters its abstract data type in which fol­lows the prin­ci­ple of LIFO (Last-In-First-Out) which means the data goes in last comes out first to read about in detail please read this link [Stack](https://en.wikipedia.org/wiki/Stack_(abstract_data_type))

**Approach:**

Solu­tion is quite sim­ple, Ear­lier we have seen an arti­cle  “[Linked List Imple­men­ta­tion](http://algorithms.tutorialhorizon.com/singly-linked-list-implementation/)”, we need to make some changes to make it work as Stack.

Stack Oper­a­tions:

**Push() :** Insert the ele­ment into linked list at the begin­ning and increase the size of the list. O(1) operation.

**Pop() :** Return the ele­ment first node from the linked list and move the head pointer to the sec­ond node. Decrease the size of the list. O(1) operation.

**get­Size():** Return the size of linked list.

**dis­playStack():** Print the linked list.

**Com­plete Code:**

|  |  |
| --- | --- |
|  | public class StackUsingLinkedList { |
|  | Node head= null; |
|  | int size =0; |
|  |  |
|  | public void push(int data){ |
|  | Node x = new Node(data); |
|  | if(getSize()==0){ |
|  | head = x; |
|  | }else{ |
|  | //add the Node at the start of a Linked List |
|  | Node temp = head; |
|  | x.next = temp; |
|  | head = x; |
|  | } |
|  | System.out.println("Element "+ data + " is pushed into Stack"); |
|  | size++; |
|  | } |
|  |  |
|  | public int pop(){ |
|  | if(getSize()==0){ |
|  | System.out.println("Stack is Empty"); |
|  | return -1; |
|  | }else{ |
|  | Node temp = head; |
|  | head = head.next; |
|  | size--; |
|  | return temp.data; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | public void printStack(){ |
|  | Node curr = head; |
|  | while(curr!=null){ |
|  | System.out.print(curr.data + " "); |
|  | curr = curr.next; |
|  | } |
|  | System.out.println(); |
|  | } |
|  | public int getSize(){ |
|  | return size; |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | StackUsingLinkedList stck = new StackUsingLinkedList(); |
|  | stck.push(1); |
|  | stck.push(2); |
|  | stck.push(4); |
|  | stck.printStack(); |
|  | System.out.println("Pop out element " + stck.pop()); |
|  | stck.printStack(); |
|  | } |
|  | } |
|  | class Node{ |
|  | int data; |
|  | Node next; |
|  | public Node(int data){ |
|  | this.data = data; |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/c0439fb8fb4ff95af5bc/raw/6b0b03e7dfe69d5374e7e5fa75ee8cb75cf1bd7a/StackUsingLinkedList.java)[StackUsingLinkedList.java](https://gist.github.com/thmain/c0439fb8fb4ff95af5bc" \l "file-stackusinglinkedlist-java)** hosted with  by **[GitHub](https://github.com/)**

**Out­put:**

Element 1 is pushed into Stack

Element 2 is pushed into Stack

Element 4 is pushed into Stack

4 2 1

Pop out element 4

2 1

**Note**: Also read, [Track the Max­i­mum Ele­ment in a Stack.](http://algorithms.tutorialhorizon.com/track-the-maximum-element-in-a-stack/) This prob­lem was asked in Yahoo in Soft­ware Engi­neer position.

# Track the Maximum Element in a Stack.

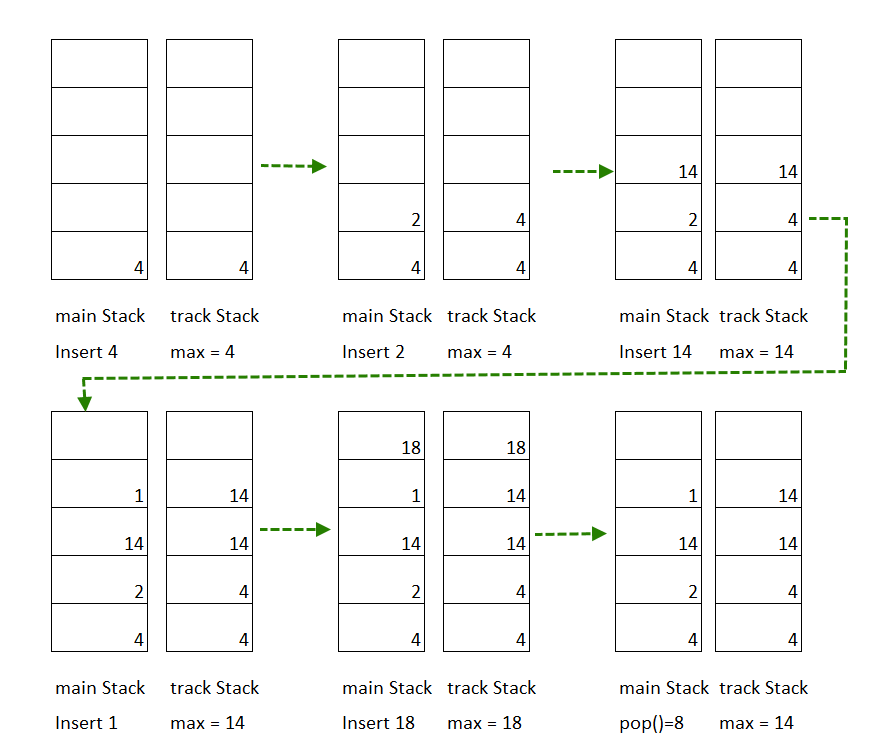
BY [SJ](http://algorithms.tutorialhorizon.com/author/sumitjain/) · FEBRUARY 16, 2015

**Objec­tive:**In a Stack, keep track of max­i­mum value in it. It might be the top ele­ment in the stack but once it is poped out, the max­i­mum value should be from the rest of the ele­ments in the stack.

**Approach:**

* Cre­ate another another Stack(call it as **track**) which will keep track of max­i­mum in the given Stack(call it as **main**).
* When you insert an ele­ment in the **main stack** for the first time ( means it is empty), insert it in the **track Stack** as well.
* Now onwards when you insert a new element(say it is **x**) in the **main Stack**, peek() the ele­ment from the track Stack ( say it is ‘**a**’). Com­pare **x and a** and which ever is greater, insert it into **track Stack**.
* When you pop the ele­ment from the **main stack**, pop from the **track Stack** as well
* So to get to know the **max­i­mum** ele­ment in the **main Stack**, **peek the ele­ment in the track Stack**. . See Exam­ple below.

Thanks [Gau­rav Dey](https://www.facebook.com/gaurav.dey.7?fref=ts" \o "Gaurav) for sug­gest­ing this solution.

[](http://algorithms.tutorialhorizon.com/files/2015/02/Track-the-Maximum-Element-in-a-Stack.png)

*Track-the-Maximum-Element-in-a-Stack*

**Com­plete Code:**

|  |  |
| --- | --- |
|  | import java.util.Stack; |
|  |  |
|  | public class TrackMaxInStack { |
|  |  |
|  | // objective here is to keep track of maximum value in a stack of integers |
|  | // create another another Stack which will keep track of maximum |
|  | Stack<Integer> main = new Stack<>(); |
|  | Stack<Integer> track = new Stack<>(); |
|  |  |
|  | public void insert(int x) { |
|  | if (main.isEmpty()) { // if stack is empty, insert the number in both |
|  | // stacks |
|  | main.add(x); |
|  | track.add(x); |
|  | } else { |
|  | // check if number in Stack(track) is bigger than x |
|  | // which ever is bigger, insert it into Stack |
|  |  |
|  | int a = track.peek(); |
|  | track.add(Math.max(a, x)); |
|  | main.add(x); // insert it into main stack. |
|  | } |
|  | } |
|  |  |
|  | public int remove() { |
|  | if (!main.isEmpty()) { // pop the top elements |
|  | track.pop(); |
|  | return main.pop(); |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  | public int getMax() { |
|  | return track.peek(); |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | TrackMaxInStack i = new TrackMaxInStack(); |
|  | i.insert(4); |
|  | i.insert(2); |
|  | i.insert(14); |
|  | i.insert(1); |
|  | i.insert(18); |
|  | System.out.println("Max Element is " + i.getMax()); |
|  | System.out.println("Removing Element " + i.remove()); |
|  | System.out.println("Max Element is " + i.getMax()); |
|  | } |
|  |  |
|  | } |

**[view raw](https://gist.github.com/thmain/958f49d6b9dee14f130f/raw/c94a0114b1974f4e67a6f781733142deaea313bc/TrackMaxInStack.java)[TrackMaxInStack.java](https://gist.github.com/thmain/958f49d6b9dee14f130f" \l "file-trackmaxinstack-java)** hosted with  by **[GitHub](https://github.com/)**

**Out­put**:

Max Element is 18

Removing Element 18

Max Element is 14

Find the Kth Smallest/Largest Element in an Array

BY [SJ](http://algorithms.tutorialhorizon.com/author/sumitjain/) · MAY 10, 2015

**Objec­tive**: Given an array of inte­gers. find the Kth Smallest/largest ele­ment in the array.

**Example**:

int[] A = { 1, 2, 10, 20, 40, 32, 44, 51, 6 };

K=4. 4th smallest element in given array: 10

**Approach**: (Kth Small­est Element)

* Use min-Heap. ([Click here to read about Pri­or­ity Queue](http://algorithms.tutorialhorizon.com/priority-queue-implementation/" \o "Priority Queue Implementation" \t "_blank)).
* Insert all the ele­ments in the Pri­or­ity Queue.
* Extract K ele­ments from the pri­or­ity queue. The last ele­ment (kth) extracted with be the kth small­est ele­ment in the array.

**Com­plete Code:**

|  |  |
| --- | --- |
|  | import java.util.PriorityQueue; |
|  |  |
|  | public class KthSmallestElementInArray { |
|  |  |
|  | public static int find(int [] A, int k){ |
|  | PriorityQueue<Integer> pq = new PriorityQueue<Integer>(); |
|  | for(int i=0;i<A.length;i++){ |
|  | pq.offer(A[i]); |
|  | } |
|  | int n = -1; |
|  | while(k>0){ |
|  | n = pq.poll(); |
|  | k--; |
|  | } |
|  | return n; |
|  | } |
|  | public static void main(String[] args) { |
|  | int[] A = { 1, 2, 10, 20, 40, 32, 44, 51, 6 }; |
|  | int k = 4; |
|  | System.out.println("4th smallest element:" + find(A,4)); |
|  |  |
|  | } |
|  |  |
|  | } |

**[view raw](https://gist.github.com/SumitJainUTD/4e5f7374237d33c64c36/raw/de83ad12965a093eca032a74a84b26958469353d/KthSmallestElementInArray.java)[KthSmallestElementInArray.java](https://gist.github.com/SumitJainUTD/4e5f7374237d33c64c36" \l "file-kthsmallestelementinarray-java)** hosted with  by **[GitHub](https://github.com/)**

Output:

4th smallest element:10

**Note:**For kth largest ele­ment, imple­ment pri­or­ity queue for max-Heap.

Find the Second Largest Element in an Array

BY [SJ](http://algorithms.tutorialhorizon.com/author/sumitjain/) · MAY 9, 2015

**Objec­tive**: Given an array of inte­gers. find the sec­ond largest ele­ment in the array.

**Exam­ple**:

int[] A = { 1, 2, 10, 20, 40, 32, 44, 51, 6 };

Second largest Element : 44

**Approach:**

* Keep track of largest ele­ment and when ever you change the value of largest ele­ment, store its cur­rent value to another vari­able, call it as sec­ond largest element.
* If you are not updat­ing the largest ele­ment then check if sec­ond largest ele­ment is less than the cur­rent ele­ment, if yes then update it.

**Com­plete Code:**

|  |  |
| --- | --- |
|  | public class SecondLargestElement { |
|  |  |
|  | public static int findSecond(int[] A) { |
|  | int fstNo = A[0]; |
|  | int sndNo = -1; |
|  | for (int i = 1; i < A.length; i++) { |
|  | if (fstNo < A[i]) { |
|  | sndNo = fstNo;// store the first largest no value to second |
|  | // largest |
|  | fstNo = A[i]; |
|  | } else if (sndNo < A[i]) { |
|  | sndNo = A[i]; |
|  | } |
|  | } |
|  | return sndNo; |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | int[] A = { 1, 2, 10, 20, 40, 32, 44, 51, 6 }; |
|  | System.out.println("Second largest Element : " + findSecond(A)); |
|  |  |
|  | } |
|  |  |
|  | } |

**[view raw](https://gist.github.com/SumitJainUTD/96445db5b06952463fc5/raw/e3d9930e9625d9fdb0089819f12fa3e670eeff17/SecondLargestElement.java)[SecondLargestElement.java](https://gist.github.com/SumitJainUTD/96445db5b06952463fc5" \l "file-secondlargestelement-java)** hosted with  by **[GitHub](https://github.com/)**

Second largest Element : 44

Find the first repeated element in an array by its index

BY [SJ](http://algorithms.tutorialhorizon.com/author/sumitjain/) · NOVEMBER 10, 2014

**Objec­tive:** Given an array of inte­gers, find out the first repeated ele­ment. First repeated ele­ment means the ele­ment occurs atleast twice and has small­est index.

**Input:** An Array

**Out­put:**The first repeated element

**Exam­ples** :

Array {1,1,3,4,6,7,9} first repeated Number : 1

Array {7,2,2,3,7} first repeated Number : 7

Array {5,3,3,3,3,3,3,3,4,4,4,5,3} first repeated Number : 5

**Approach:**

**Naive Solu­tion :**

Use two for loops. Time Com­plex­ity O(N2).

**Bet­ter Solu­tion: Using Hast­Set, Time Com­plex­ity O(N).**

* Take a vari­able say *index = –1.*
* Ini­tial­ize a HashSet.
* Nav­i­gate the array from right to left(backwards) tak­ing one ele­ment at a time
* if Hash­Set doesnt con­tain ele­ment, add it
* if Hash­Set con­tains then update the index with cur­rent index in navigation.
* At the end index will be updated by the ele­ment which is repeated and has the low­est index.

**Com­plete Code:**

|  |  |
| --- | --- |
|  | import java.util.HashSet; |
|  |  |
|  | public class FirstRepeatingelement { |
|  | public int find(int [] arrA){ |
|  | int index = -1; |
|  | HashSet<Integer> hs = new HashSet<>(); |
|  | for(int i = arrA.length-1;i>=0;i--){ |
|  | if(hs.contains(arrA[i])){ |
|  | index = i; |
|  | }else{ |
|  | hs.add(arrA[i]); |
|  | } |
|  | } |
|  | return arrA[index]; |
|  | } |
|  | public static void main(String args[]){ |
|  | int [] a = {1,2,5,7,5,3,10,2}; |
|  | FirstRepeatingelement f = new FirstRepeatingelement(); |
|  | System.out.println"{1,2,5,7,5,3,10,2}"); |
|  | System.out.println("first repeated element by index is : " + f.find(a)); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/49307528e2f5bd173800/raw/854db2278d7dd9c513ae4de1d197247966cd8ab7/FirstRepeatingelement.java)[FirstRepeatingelement.java](https://gist.github.com/thmain/49307528e2f5bd173800" \l "file-firstrepeatingelement-java)** hosted with  by **[GitHub](https://github.com/)**

**Out­put:**

{1,2,5,7,5,3,10,2}

first repeated element by index is : 2

Find the element which appears maximum number of times in the array.

BY [SJ](http://algorithms.tutorialhorizon.com/author/sumitjain/) · JUNE 26, 2017

**Objec­tive:**Given an array of inte­gers, write a algo­rithm to find the ele­ment which appears max­i­mum num­ber of times in the array.

**Exam­ple:**

**int** [] arrA = {4, 1, 5, 2, 1, 5, 9, 8, 6, 5, 3, 2, 4, 7};

**Output**: Element repeating maximum no of times: 5, maximum count: 3

**Approach:**

**Naive approach:**Use 2 loops. Check each ele­ment in the array with all other ele­ments in the array and keep track of its count and also main­tain the max counter which track the ele­ment repeats the max­i­mum num­ber of time.

Time Com­plex­ity : O(n^2) Space Com­plex­ity: O(1)

**Code:**

|  |  |
| --- | --- |
|  | public class MaxRepeatingBruteForce { |
|  | public void MaxRepeatingElement(int [] arrA){ |
|  | int maxCounter = 0; |
|  | int element=0; |
|  | for (int i = 0; i <arrA.length ; i++) { |
|  | int counter =1; |
|  | for (int j = i+1; j <arrA.length ; j++) { |
|  | if(arrA[i]==arrA[j]){ |
|  | counter++; |
|  | } |
|  | } |
|  | if(maxCounter<counter){ |
|  | maxCounter=counter; |
|  | element = arrA[i]; |
|  | } |
|  | } |
|  | System.out.println("Element repeating maximum no of times: " + element + ", maximum count: " + maxCounter); |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | int [] arrA = {4, 1, 5, 2, 1, 5, 9, 8, 6, 5, 3, 2, 4, 7}; |
|  | MaxRepeatingBruteForce m = new MaxRepeatingBruteForce(); |
|  | m.MaxRepeatingElement(arrA); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/fb8a5d010c8e0463220cc0e5ad4801a3/raw/c1d4a6eec11ef3d756d3ed1d6b370c4a8fe7b85b/MaxRepeatingBruteForce.java)[MaxRepeatingBruteForce.java](https://gist.github.com/thmain/fb8a5d010c8e0463220cc0e5ad4801a3" \l "file-maxrepeatingbruteforce-java)** hosted with  by **[GitHub](https://github.com/)**

**Sort­ing approach:** Sort the array, this will bring all the dupli­cates together if present. Now nav­i­gate the array and keep track of its count and also main­tain the max counter which track the ele­ment repeats the max­i­mum num­ber of time.

Time Com­plex­ity : O(nlogn) Space Com­plex­ity: O(n) by using merge sort.

**Code:**

|  |  |
| --- | --- |
|  | import java.util.Arrays; |
|  |  |
|  | public class MaxRepeatingUsingSorting { |
|  | public void maxRepeatingElementUsingSorting(int [] arrA){ |
|  | if(arrA.length<1){ |
|  | System.out.println("Inavlid Array"); |
|  | return; |
|  | } |
|  | Arrays.sort(arrA); |
|  | int count=1; |
|  | int maxCount=1; |
|  | int currentElement = arrA[0]; |
|  | int maxCountElement =arrA[0]; |
|  | for (int i = 1; i <arrA.length ; i++) { |
|  | if(currentElement==arrA[i]){ |
|  | count++; |
|  | }else{ |
|  | if(count>maxCount){ |
|  | maxCount = count; |
|  | maxCountElement = currentElement; |
|  | } |
|  | currentElement = arrA[i]; |
|  | count = 1; |
|  | } |
|  | } |
|  | System.out.println("Element repeating maximum no of times: " + maxCountElement + ", maximum count: " + maxCount); |
|  | } |
|  | public static void main(String[] args) { |
|  | int [] arrA = {4, 1, 5, 2, 1, 5, 9, 8, 6, 5, 3, 2, 4, 7}; |
|  | MaxRepeatingUsingSorting m = new MaxRepeatingUsingSorting(); |
|  | m.maxRepeatingElementUsingSorting(arrA); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/941a69e7b426564f90499cb86a7904d3/raw/8bba65e58181591a1032df2f3cd275ee2fd46f10/MaxRepeatingUsingSorting.java)[MaxRepeatingUsingSorting.java](https://gist.github.com/thmain/941a69e7b426564f90499cb86a7904d3" \l "file-maxrepeatingusingsorting-java)** hosted with  by **[GitHub](https://github.com/)**

**Bet­ter Solu­tion :** Use Hashmap. Store the count of each ele­ment of array in a hash table and later check in Hash map which ele­ment has the max­i­mum count — [Find dupli­cates in an given array](http://algorithms.tutorialhorizon.com/find-duplicates-in-an-given-array-in-on-time-and-o1-extra-space/)

Time Com­plex­ity : O(n) and Space Com­plex­ity: O(n).

**Code:**

|  |  |
| --- | --- |
|  | import java.util.HashMap; |
|  | import java.util.Iterator; |
|  | import java.util.Map; |
|  |  |
|  | public class MaxRepeatingUsingHashMap { |
|  | public void maxRepeatingElementUsingMap(int [] arrA){ |
|  | //Will store each character and it's count |
|  | HashMap<Integer, Integer> map = new HashMap<>(); |
|  | for (int i = 0; i <arrA.length; i++) { |
|  | if(map.containsKey(arrA[i])){ |
|  | map.put(arrA[i],map.get(arrA[i])+1); |
|  | }else{ |
|  | map.put(arrA[i], 1); |
|  | } |
|  | } |
|  |  |
|  | //traverse the map and track the element which has max count |
|  | Iterator entries = map.entrySet().iterator(); |
|  | int maxCount = 0; |
|  | int element =arrA[0]; |
|  | while(entries.hasNext()){ |
|  | Map.Entry entry = (Map.Entry) entries.next(); |
|  | int count = (Integer)entry.getValue(); |
|  | if(maxCount<count){ |
|  | maxCount = count; |
|  | element = (Integer)entry.getKey(); |
|  | } |
|  | } |
|  | System.out.println("Element repeating maximum no of times: " + element + ", maximum count: " + maxCount); |
|  | } |
|  | public static void main(String[] args) { |
|  | int [] arrA = {4, 1, 5, 2, 1, 5, 9, 8, 6, 5, 3, 2, 4, 7}; |
|  | MaxRepeatingUsingHashMap m = new MaxRepeatingUsingHashMap(); |
|  | m.maxRepeatingElementUsingMap(arrA); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/cfb6ef600cea630da64cfe8a773662b3/raw/727eef0102ac93c5514d92a20d80a1875c5110db/MaxRepeatingUsingHashMap.java)[MaxRepeatingUsingHashMap.java](https://gist.github.com/thmain/cfb6ef600cea630da64cfe8a773662b3" \l "file-maxrepeatingusinghashmap-java)** hosted with  by **[GitHub](https://github.com/)**

**Bet­ter Solu­tion** (Con­di­tional) : **O(n) time and O(1) extra space.**

* This solu­tion works only if array has pos­i­tive inte­gers and all the ele­ments in the array are in range from 0 to n-1 where n is the size of the array.
* Nav­i­gate the array.
* Update the array as for ith index :- arrA[arrA[i]% n] = arrA[arrA[i]% n] + n;
* Now nav­i­gate the updated array and check which index has the max­i­mum value, that index num­ber is the ele­ment which has the max­i­mum occur­rence in the array.
* See the pic­ture below for more explanation.

Sim­i­lar approach used in prob­lem : [if array has all con­sec­u­tive numbers.](http://algorithms.tutorialhorizon.com/check-if-array-is-consecutive-integers/)

**Code:**

|  |  |
| --- | --- |
|  | public class MaxRepeatingElement { |
|  | public void MaxRepeatingElementInPlace(int [] arrA){ |
|  | int size = arrA.length; |
|  | int maxCount=0; |
|  | int maxIndex=0; |
|  | for (int i = 0; i <size ; i++) { |
|  | //get the index to be updated |
|  | int index = arrA[i]% size; |
|  | arrA[index] = arrA[index] + size; |
|  | } |
|  | for (int i = 0; i <size ; i++) { |
|  | if(arrA[i]/size>maxCount){ |
|  | maxCount=arrA[i]/size; |
|  | maxIndex=i; |
|  | } |
|  | } |
|  | System.out.println("Element repeating maximum no of times: " + maxIndex + ", maximum count: " + maxCount); |
|  | } |
|  | public static void main(String[] args) { |
|  | int [] arrA = {4, 1, 5, 2, 1, 5, 9, 8, 6, 5, 3, 2, 4, 7}; |
|  | MaxRepeatingElement m = new MaxRepeatingElement(); |
|  | m.MaxRepeatingElementInPlace(arrA); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/1cea9f46da8b345e71bff559fea24980/raw/11ed279156d8a5023aa0aae54f3e4e428360a1f9/MaxRepeatingElement.java)[MaxRepeatingElement.java](https://gist.github.com/thmain/1cea9f46da8b345e71bff559fea24980" \l "file-maxrepeatingelement-java)** hosted with  by **[GitHub](https://github.com/)**

**Out­put:**

Element repeating maximum no of times: 5, maximum count: 3

Find the two repeating elements in a given array | 6 Approaches

BY [SJ](http://algorithms.tutorialhorizon.com/author/sumitjain/) · JULY 24, 2017

**Objec­tive:**Given an array of n+2 ele­ments. All ele­ments of the array are in range 1 to n and all ele­ments occur once except two num­bers which occur twice. Write an algo­rithm to find the two repeat­ing numbers.

**Exam­ple:**

int [] A = {1,4,5,6,3,2,5,2};

int n = 6;

**Output**: Two Repeated elements are: 2 and 5

**Approach 1:**

**Naive**: This prob­lem can be eas­ily solved using two nested loops. Take each ele­ment at a time and com­pare it with all the other ele­ments and if it appears twice, print it. This solu­tion will work even if all the num­bers are not in the range of 1 to n.

Time com­plex­ity is O(N^2).

**Code**:

|  |  |
| --- | --- |
|  | public class TwoRepeatingBruteForce { |
|  | //this solution will work even if all the numbers are not in the range of 1 to n |
|  | public static void twoRepeating(int [] A){ |
|  | System.out.println("Repeated Elements: "); |
|  | for (int i = 0; i <A.length ; i++) { |
|  | for (int j = i+1; j <A.length ; j++) { |
|  | if(A[i]==A[j]){ |
|  | System.out.print(A[i] + " "); |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | int [] A = {1,5,2,4,8,9,3,1,4,0}; |
|  | twoRepeating(A); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/9c1b3608798569d449f04b7d9a767c55/raw/7cc7b8a1ee78fbbe6c2571a5fad5173a59f4365c/TwoRepeatingBruteForce.java)[TwoRepeatingBruteForce.java](https://gist.github.com/thmain/9c1b3608798569d449f04b7d9a767c55" \l "file-tworepeatingbruteforce-java)** hosted with  by **[GitHub](https://github.com/)**

**Approach 2:**

**Using Hash Map**

* This solu­tion will work even if all the num­bers are not in the range of 1 to n.
* Keep the count of each ele­ment in the Hash Map.
* Print the ele­ments which has count = 2.

Time Com­plex­ity: O(N),  Space Com­plex­ity: O(N)

**Code**:

|  |  |
| --- | --- |
|  | import java.util.HashMap; |
|  | import java.util.Iterator; |
|  | import java.util.Set; |
|  |  |
|  | public class TwoRepeatingHashMap { |
|  | public static void twoElements(int [] A){ |
|  | HashMap<Integer, Integer> map = new HashMap<>(); |
|  | for (int i = 0; i <A.length ; i++) { |
|  | if(map.containsKey(A[i])){ |
|  | int count = map.get(A[i]); |
|  | map.put(A[i],++count); |
|  | }else |
|  | map.put(A[i],1); |
|  |  |
|  | } |
|  | System.out.print("Repeated Elements are : "); |
|  | Set set = map.keySet(); |
|  | Iterator<Integer> iterator = set.iterator(); |
|  | while(iterator.hasNext()){ |
|  | int key = iterator.next(); |
|  | if(map.get(key)==2){ |
|  | System.out.print(key + " "); |
|  | } |
|  | } |
|  | } |
|  | public static void main(String[] args) { |
|  | int [] A = {1,5,2,4,8,9,3,1,4,0}; |
|  | twoElements(A); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/eeb918402086a00ec8cb353101062f1d/raw/28d49c0d2516b6da95f9a2ea1122d3572186b865/TwoRepeatingHashMap.java)[TwoRepeatingHashMap.java](https://gist.github.com/thmain/eeb918402086a00ec8cb353101062f1d" \l "file-tworepeatinghashmap-java)** hosted with  by **[GitHub](https://github.com/)**

**Approach 3:**

**Using Math’s Formula:**

* This solu­tion will work only if all the num­bers are in the range of 1 to n and are >0
* Let’s say two repeated ele­ments are a, b
* Sum of 1 to n ele­ments = S, Sum of all array ele­ments = X, so a + b = X-S
* Prod­uct of 1 to n ele­ments = n!, Prod­uct of all array ele­ments = Y, so a \* b = Y/n!
* Now we have 2 equa­tions and 2 unknowns , we can solve to get a and b
* We know that a — b = sqrt( (a + b)^2 — 4ab )

**Exam­ple**:

1. int [] A = {1,4,5,6,3,2,5,2};

int n = 6;

2. S = n\*(n+1)/2 = 6 \* 7/2 = 21

3. X (sum of all array elements) = 28

4. a + b = 28 – 21 = 7

5. Product of 1 to 6 = !6 = 720

6. Y (Product of all array elements) = 7200

7. a \* b = 7200/720 = 10

8. So now, a + b = 7 and a \* b = 10

9. a - b = sqrt( (a + b)^2 - 4ab )

10. a – b = sqrt(7\*7 – 4\*10) = sqrt(49-40) = sqrt(9) = 3

11. a = (7 + 3)/2 = 5

12. b = 7-5 = 2

13. Elements are 5, 2

Time Com­plex­ity: O(N),  Space Com­plex­ity: O(1)

**Code**:

|  |  |
| --- | --- |
|  | public class TwoRepeatingByFormula { |
|  | public static void twoElements(int [] A, int n){ |
|  | int a,b; |
|  | int X =0; |
|  | int Y =1; |
|  | int S = n\*(n+1)/2; |
|  | int fact = factorial(n); |
|  | for (int i = 0; i <A.length ; i++) { |
|  | X += A[i]; |
|  | Y \*= A[i]; |
|  | } |
|  | int sum = X - S; |
|  | int product = Y/fact; |
|  | int subtract = (int)Math.sqrt(sum\*sum - 4\*product); |
|  |  |
|  | a = (sum + subtract)/2; |
|  | b = sum - a; |
|  |  |
|  | System.out.println("Two Repeating Elements are: " + a + " and " + b); |
|  |  |
|  | } |
|  | static int factorial(int n){ |
|  | if(n==0) |
|  | return 1; |
|  | Else |
|  | return n\*factorial(n-1); |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | int [] A = {1,4,5,6,3,2,5,2}; |
|  | int n = 6; |
|  | twoElements(A, n); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/af9c0870ed342ea8524f1d282818cedb/raw/99e6b9b398f7d2127daccf6b98eebf193fe493b4/TwoRepeatingByFormula.java)[TwoRepeatingByFormula.java](https://gist.github.com/thmain/af9c0870ed342ea8524f1d282818cedb" \l "file-tworepeatingbyformula-java)** hosted with  by **[GitHub](https://github.com/)**

**Approach 4:**

**Array ele­ment as index**

* This solu­tion works only if array has pos­i­tive inte­gers and all the ele­ments in the array are in range from 1 to n.
* Nav­i­gate the array.
* Update the array as for ith index :- A[abs(A[i])] = A[abs(A[i])] \* –1;
* (If it already not negative).
* If A[x] is already neg­a­tive, then it means we are vis­it­ing it sec­ond time, means it is repeated.
* Sim­i­lar approach used in prob­lem : [If array has all con­sec­u­tive numbers.](http://algorithms.tutorialhorizon.com/check-if-array-is-consecutive-integers/)

Time Com­plex­ity: O(N),  Space Com­plex­ity: O(1)

**Code**:

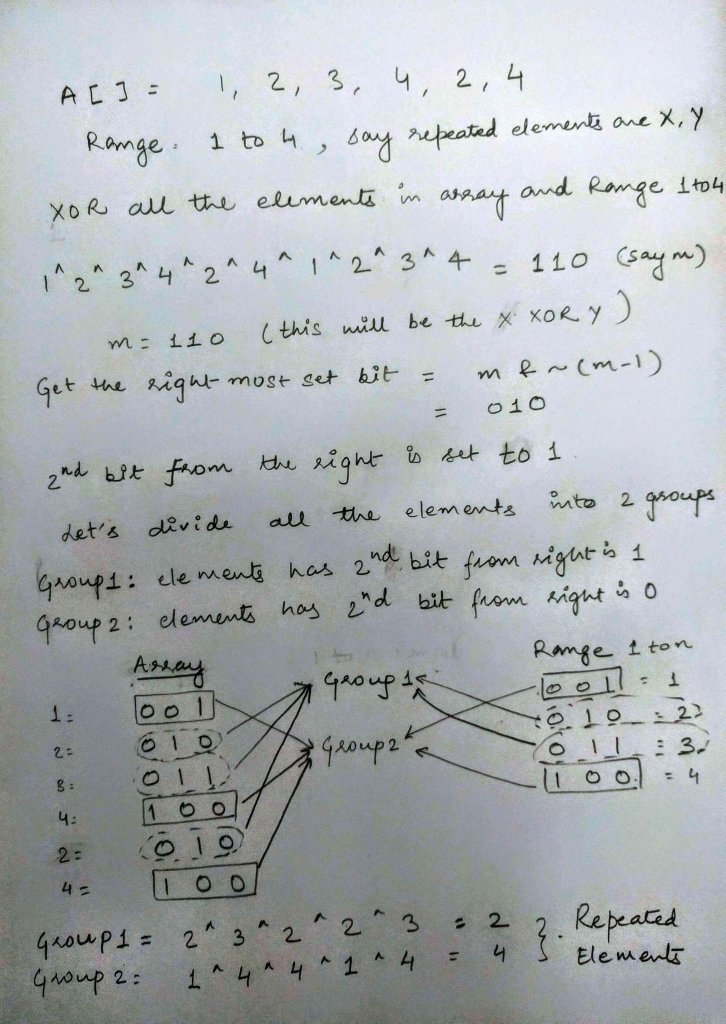
|  |  |
| --- | --- |
|  | public class TwoRepeatingElementsByIndex { |
|  | //this solution will work only if all the numbers are in the range of 1 to n and are >0 |
|  | //navigate the array if number is x then multiply the A[x] by -1. |
|  | //If A[x] is already negative, then it means we are visiting it second time, means it is repeated. |
|  | public static void twoRepeating(int [] A, int n){ |
|  | System.out.print("Repeated Elements are: "); |
|  | for (int i = 0; i <A.length ; i++) { |
|  | if(A[Math.abs(A[i])]<0) |
|  | System.out.print(Math.abs(A[i]) + " "); |
|  | Else |
|  | A[Math.abs(A[i])] = A[Math.abs(A[i])] \* -1; |
|  | } |
|  | } |
|  | public static void main(String[] args) { |
|  | int [] A = {1,4,5,6,3,2,5,2}; |
|  | int n = 6; |
|  | twoRepeating(A, n); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/382e20b5ece73b2cd04a7935ee20037d/raw/6f91587210405b41a26c53fc8f2ae135cd34ab7d/TwoRepeatingElementsByIndex.java)[TwoRepeatingElementsByIndex.java](https://gist.github.com/thmain/382e20b5ece73b2cd04a7935ee20037d" \l "file-tworepeatingelementsbyindex-java)** hosted with  by **[GitHub](https://github.com/)**

**Approach 5:**

**Using XOR**

* This solu­tion works only if array has pos­i­tive inte­gers and all the ele­ments in the array are in range from 1 to n.
* As we know A XOR A = 0. We have n + 2 ele­ments in array with 2 repeated ele­ments (say repeated ele­ments are *X* and *Y*) and we know the range of ele­ments are from 1 to n.
* XOR all the num­bers in array num­bers from 1 to n. Result be *X XOR Y*.
* 1 XOR 1 =  0 and 1 XOR 0 = 1 with this logic in the result of *X XOR Y* if any kth bit is set to 1  implies either kth bit is 1 either in X or in Y not in both.
* Use the above step to divide all the ele­ments in array and from 1 to n into 2 groups, one group which has the ele­ments for which the kth bit is set to 1 and sec­ond group which has the ele­ments for which the kth bit is 0.
* Let’s have that kth bit as right most set bit (Read [how to find right most set bit](http://algorithms.tutorialhorizon.com/find-the-right-most-set-bit-of-a-number/))
* Now we can claim that these two groups are respon­si­ble to pro­duce X and Y.
* Group –1: XOR all the ele­ments whose kth bit is 1 will pro­duce either X or Y.
* Group –2: XOR all the ele­ments whose kth bit is 0 will pro­duce either X or Y.
* See the dia­gram below for more understanding.(Click on the dia­gram to see it larger)

[](http://algorithms.tutorialhorizon.com/files/2017/07/Two-repeated-elements-2.jpg)

Time Com­plex­ity: O(N),  Space Com­plex­ity: O(1)

**Code**:

|  |  |
| --- | --- |
|  | public class TwoRepeatingXOR { |
|  | public static void twoRepeating(int [] A, int n){ |
|  | int XOR = A[0]; |
|  | int right\_most\_bit, X=0, Y=0, size = A.length; |
|  |  |
|  | for (int i = 1; i <=n ; i++) |
|  | XOR ^= i; |
|  |  |
|  | for (int i = 0; i <size ; i++) |
|  | XOR ^= A[i]; |
|  |  |
|  | //Now XOR contains the X XOR Y |
|  | //get the right most bit number |
|  | right\_most\_bit = XOR & ~(XOR-1); |
|  | //divide the elements into 2 groups based on the right most set bit |
|  | for (int i = 0; i <size ; i++) { |
|  | if((A[i] & right\_most\_bit)!=0) |
|  | X = X^A[i]; |
|  | Else |
|  | Y = Y^A[i]; |
|  | } |
|  | for (int i = 1; i <=n ; i++) { |
|  | if((i&right\_most\_bit)!=0) |
|  | X = X^i; |
|  | Else |
|  | Y = Y^i; |
|  | } |
|  | System.out.println("Two Repeated elements are: " + X + " and " + Y); |
|  | } |
|  | public static void main(String[] args) { |
|  | int [] A = {1,4,5,6,3,2,5,2}; |
|  | int n = 6; |
|  | twoRepeating(A, n); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/thmain/1eb30d8deb372c807eda0ec982fbdbda/raw/3b2f9d7781779eae95eafd1d1cba68ab24426217/TwoRepeatingXOR.java)[TwoRepeatingXOR.java](https://gist.github.com/thmain/1eb30d8deb372c807eda0ec982fbdbda" \l "file-tworepeatingxor-java)** hosted with  by **[GitHub](https://github.com/)**

**Out­put:**

Two Repeated ele­ments are: 2 and 5

**Approach 6:**

**Using Sort­ing**

* This solu­tion will work even if all the num­bers are not in the range of 1 to n.
* Sort the array, this will bring all the repeated ele­ments together.
* Now tra­verse the array and com­pare the adja­cent ele­ments and print them if they are same.

Time Com­plex­ity: O(nlogn),  Space Com­plex­ity: O(1)

**Code**:

|  |  |
| --- | --- |
|  | public class TwoRepeatingSorting { |
|  |  |
|  | public static void twoRepeating(int [] A){ |
|  | Arrays.sort(A); |
|  | System.out.print("Repeated Elements are: "); |
|  | for (int i = 0; i <A.length-1 ; i++) { |
|  | if(A[i]==A[i+1]) |
|  | System.out.print(A[i] + " "); |
|  | } |
|  | } |
|  | public static void main(String[] args) { |
|  | int [] A = {1,4,5,6,3,2,5,2}; |
|  | twoRepeating(A); |
|  | } |
|  | } |

Priority Queue Implementation

## Java Priority Queue

Welcome to Priority Queue in Java tutorial. We know that Queue follows **F**irst-**I**n-**F**irst-**O**ut model but sometimes we need to process the objects in the queue based on the priority. That is when Java PriorityQueue is used.

For example, let’s say we have an application that generates stocks reports for daily trading session. This application processes a lot of data and takes time to process it. So customers are sending request to the application that is actually getting queued but we want to process premium customers first and standard customers after them. So in this case **PriorityQueue** implementation in java can be really helpful.

Java Priority Queue doesn’t allow null values and we can’t create PriorityQueue of Objects that are non-comparable. We use [java Comparable and Comparator](https://www.journaldev.com/780/comparable-and-comparator-in-java-example) for sorting Objects and Priority Queue use them for priority processing of it’s elements.

PriorityQueue is **not thread safe**, so java provides PriorityBlockingQueue class that implements the [BlockingQueue interface](https://www.journaldev.com/1034/java-blockingqueue-example) to use in [java multithreading](https://www.journaldev.com/1079/multithreading-in-java) environment.

BY [SJ](http://algorithms.tutorialhorizon.com/author/sumitjain/) · MAY 10, 2015

Ear­lier in we have seen [Min-Heap and Max-Heap Imple­men­ta­tion](http://algorithms.tutorialhorizon.com/binary-min-max-heap/" \o "Binary Min — Max Heap" \t "_blank). Pri­or­ity Queue is its built-in imple­men­ta­tion in Java.

In this arti­cle we will see how to per­form Min-Heap and Max-Heap using [Pri­or­ity Queue](https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html" \o "Priority Queue" \t "_blank).

**Brief:**

A pri­or­ity queue is an abstract data type where each ele­ment has a “pri­or­ity” assigned to it. So the ele­ment with the higher pri­or­ity is served before the other ele­ments. [Click here to know in detail about max-Heap and min-Heap](http://algorithms.tutorialhorizon.com/binary-min-max-heap/" \o "Binary Min — Max Heap" \t "_blank). (Source : [Wiki](http://en.wikipedia.org/wiki/Priority_queue" \o "Priority Queue" \t "_blank))

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Method** | **Descrip­tion** |
| boolean | offer(E e) | Inserts the spec­i­fied ele­ment into this pri­or­ity queue. |
| E | peek() | Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |
| E | poll() | Retrieves and removes the head of this queue, or returns null if this queue is empty. |
| int | size() | Returns the num­ber of ele­ments in this collection. |
| void | clear() | Removes all of the ele­ments from this pri­or­ity queue. |
| boolean | contains(Object o) | Returns true if this queue con­tains the spec­i­fied element. |
| Iterator<E> | iter­a­tor() | Returns an iter­a­tor over the ele­ments in this queue. |
| boolean | remove(Object o) | Removes a sin­gle instance of the spec­i­fied ele­ment from this queue, if it is present. |
| Com­para­tor<? super E> | com­para­tor() | Returns the com­para­tor used to order the ele­ments in this queue, or null if this queue is sorted accord­ing to the nat­ural order­ing of its elements. |

**Min-Heap using Pri­or­ity Queue:**

|  |  |
| --- | --- |
|  | import java.util.PriorityQueue; |
|  |  |
|  | public class MinHeap\_PQ { |
|  | PriorityQueue<Integer> pq; |
|  |  |
|  | public MinHeap\_PQ() { |
|  | pq = new PriorityQueue<Integer>(); |
|  | } |
|  |  |
|  | public void insert(int[] x) { |
|  | for (int i = 0; i < x.length; i++) { |
|  | pq.offer(x[i]); |
|  | } |
|  | } |
|  |  |
|  | public int peek() { |
|  | return pq.peek(); |
|  | } |
|  |  |
|  | public int extractMin() { |
|  | return pq.poll(); |
|  | } |
|  |  |
|  | public int getSize() { |
|  | return pq.size(); |
|  | } |
|  |  |
|  | public void print() { |
|  | System.out.println(pq); |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | int[] arrA = { 1, 6, 2, 9, 4, 3, 8 }; |
|  | MinHeap\_PQ i = new MinHeap\_PQ(); |
|  | i.insert(arrA); |
|  | i.print(); |
|  | System.out.println("Min Element in the Priority Queue: " |
|  | + i.extractMin()); |
|  | System.out.println("Min Element in the Priority Queue: " |
|  | + i.extractMin()); |
|  | System.out.println("Min Element in the Priority Queue: " |
|  | + i.extractMin()); |
|  | System.out.println("Priority Queue Size: " + i.getSize()); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/SumitJainUTD/3bf7c8b8129b46935ae2/raw/5f1d4f8a2dec5b679862f7ed11d572b967454bb8/MinHeap_PQ.java)[MinHeap\_PQ.java](https://gist.github.com/SumitJainUTD/3bf7c8b8129b46935ae2" \l "file-minheap_pq-java)** hosted with  by **[GitHub](https://github.com/)**

Output:

[1, 4, 2, 9, 6, 3, 8]

Min Element in the Priority Queue: 1

Min Element in the Priority Queue: 2

Min Element in the Priority Queue: 3

Priority Queue Size: 4

**Max-Heap using Pri­or­ity Queue:**

This gets bit tricky here. By default the Pri­or­ity Queue works as min-Heap. To imple­ment the max-Heap we need to change the way pri­or­ity queue works inter­nally by over­rid­ing the ***Com­para­tor***.

|  |  |
| --- | --- |
|  | import java.util.Comparator; |
|  | import java.util.PriorityQueue; |
|  |  |
|  | public class MaxHeap\_PQ { |
|  |  |
|  | PriorityQueue<Integer> pq; |
|  |  |
|  | public MaxHeap\_PQ() { |
|  | pq = new PriorityQueue<Integer>(10, new Comparator<Integer>() { |
|  |  |
|  | @Override |
|  | public int compare(Integer o1, Integer o2) { |
|  | // TODO Auto-generated method stub |
|  | return o2 - o1; |
|  | } |
|  | }); |
|  | } |
|  |  |
|  | public void insert(int[] x) { |
|  | for (int i = 0; i < x.length; i++) { |
|  | pq.offer(x[i]); |
|  | } |
|  | } |
|  |  |
|  | public int extractMax() { |
|  | return pq.poll(); |
|  | } |
|  |  |
|  | public void display() { |
|  | System.out.println(pq); |
|  | } |
|  |  |
|  | public int getSize() { |
|  | return pq.size(); |
|  | } |
|  |  |
|  | public void print() { |
|  | System.out.println(pq); |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | int[] arrA = { 1, 6, 2, 9, 4, 3, 8 }; |
|  | MaxHeap\_PQ i = new MaxHeap\_PQ(); |
|  | i.insert(arrA); |
|  | i.print(); |
|  | System.out.println("Max Element in the Priority Queue: " |
|  | + i.extractMax()); |
|  | System.out.println("Max Element in the Priority Queue: " |
|  | + i.extractMax()); |
|  | System.out.println("Max Element in the Priority Queue: " |
|  | + i.extractMax()); |
|  | System.out.println("Priority Queue Size: " + i.getSize()); |
|  | } |
|  | } |

**[view raw](https://gist.github.com/SumitJainUTD/444fa44b2f090244760c/raw/8ff51a28ad7101c41c4b2d209a70c4b59f59d8ba/MaxHeap_PQ.java)[MaxHeap\_PQ.java](https://gist.github.com/SumitJainUTD/444fa44b2f090244760c" \l "file-maxheap_pq-java)** hosted with  by **[GitHub](https://github.com/)**

Output:

[9, 6, 8, 1, 4, 2, 3]

Max Element in the Priority Queue: 9

Max Element in the Priority Queue: 8

Max Element in the Priority Queue: 6

Priority Queue Size: 4

Find the first non repeating character in a given string

BY [SJ](http://algorithms.tutorialhorizon.com/author/sumitjain/) · MAY 27, 2017

**Objec­tive:**Given a string, write an algo­rithm to find the first non repeat­ing char­ac­ter in it.

**Exam­ple:**

String input = " tutorial horizon"

Output: 'u'

String input = "aabbccadd"

Output: No non-repeating character found.

**Approach:**

**Naive approach**: This prob­lem can be eas­ily solved using two nested loops. Take each char­ac­ter from the outer loop and check the char­ac­ter in rest of the string using inner loop and check if that char­ac­ter appears again, if yes then con­tinue else return that char­ac­ter.  Time com­plex­ity is O(N^2).

**Bet­ter approach**: Using extra space

* Iter­ate the string from left to right.
* Count the occur­rence of each char­ac­ter and store it in a map.
* Iter­ate the string again from left to right and check if the char­ac­ter has count = 1 one in the map cre­ated in the pre­vi­ous step, if yes then return that character.
* If none of the char­ac­ter has count = 1 in map, return null.

**Com­plete Code:**

|  |  |
| --- | --- |
|  | import java.util.HashMap; |
|  |  |
|  | public class FirstNonRepeatingCharacter { |
|  | public static Character getCharacter(String input){ |
|  | //remove all the spaces |
|  | input = input.replaceAll(" ", ""); |
|  | Character nonRptChar = null; |
|  | //Will store each character and it's count |
|  | HashMap<Character, Integer> map = new HashMap<Character, Integer>(); |
|  | for (int i = 0; i <input.length(); i++) { |
|  | Character chr = input.charAt(i); |
|  | if(map.containsKey(chr)){ |
|  | map.put(chr,map.get(chr)+1); |
|  | }else{ |
|  | map.put(chr, 1); |
|  | } |
|  | } |
|  | //Iterate the string and return the character for which the count is 1 in map |
|  | for (int i = 0; i <input.length() ; i++) { |
|  | if(map.get(input.charAt(i))==1){ |
|  | nonRptChar = input.charAt(i); |
|  | break; |
|  | } |
|  | } |
|  | return nonRptChar; |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | String input = "tutorial horizon"; |
|  | Character result = getCharacter(input); |
|  | if(result!=null){ |
|  | System.out.println("First Non Repeating Character in '"+input+"' is: " + result); |
|  | }else{ |
|  | System.out.println("No Non Repeating Character found"); |
|  | } |
|  | } |
|  | } |
|  |  |

**[view raw](https://gist.github.com/thmain/36411600b2db2ac5e3b2a681686dc1f4/raw/a5ffa9016c754f1b4a1712472f02390d784cb7e9/FirstNonRepeatingCharacter.java)[FirstNonRepeatingCharacter.java](https://gist.github.com/thmain/36411600b2db2ac5e3b2a681686dc1f4" \l "file-firstnonrepeatingcharacter-java)** hosted with  by **[GitHub](https://github.com/)**

**Output:**

First Non Repeating Character in 'tutorial horizon' is: u