**Array and Integer related Logical Questions and Answers for Goldman Sachs**

**Q1 Rotate a array by N. N can be smaller of greater than the array length.**

**e.g {0,1,2,4,5,6,7} N =4 should return {5,6,7,4,0,1,2}.**

**Ans:**

Just add all elements in queue once. Second step is to do deque and queue the same element n times third step is to just de queue all elements and put them back on array

**Logic**

N = N % len  
reverse (len-N, len-1),

reverse(0, len-1),

Reverse(N,len-1).

**Q2. Given an unsorted array of integers find a minimum number which is not present in array.**

**e.g -1 ,4, 5, -23,24 is array then answer should be -22.??**

**Ans:** Take a boolean array of size n say bool[]  
Iterate over the number array once and find the minimum say it is k  
Set bool[0] (Think it represents the minimum number in the array) to true and rest to false.  
Iterate over the number array again if num[i]-k < n then set bool[num[i]-k] true.  
Check index of first false index in bool array. Say the index is j.  
The number is j+k

public int minNonArrayNum(num[]){

int size = num.length;

boolean bool[] = new boolean[size];

bool[0] = true;

int minimum = num[0];

for (int i = 0 ; i < size ; i++) {

if (num[i] < minimum) {

minimum = num[i];

}

}

for (int i = 0 ; i < size ; i++) {

if (num[i]-minimum < size) {

bool[num[i]-minimum] = true;

}

}

for (int i = 1 ; i < size ; i++) {

if (!bool[i]) {

return i+minimum;

}

}

}

**Or**

Another solution would be to use a Min heap. Store all the values in the min heap and pop the first value and then return one value less than that?  
This would be O(n) space and time complexity..

**Q3: given an array with elements check if just by exchanging two elements of the array we get a sorted array. ??**

Time restriction:

O(NlogN)

space restriction: 2N.

**Ans:**

1) Copy the original array  
2) Sort the copy  
3) loop through the arrays and check how many corresponding elements don't match   
  
O(NlogN) - space 2N

Make a duplicate array , sort duplicate array and compare with original array element by elements , if there are only two mismatch , then it is possible otherwise not.

Or

You can use quick sort to sort the array but during sorting if you need to do more than one swap just return false. Use pivot = N /2  
This way you don't need additional space and most of the time don't need to sort entire array.

Code:

public boolean needMoreThanOneSwap(int[] a){

boolean swappedOnce = false;

int k = -1;

for (int i = 0; i < a.length; i++) {

if(i < a.length -1){

//its not last element yet..

if(a[i] > a[i+1]){

if(k == -1){

//record this index.. because we need to swap it later

// if we hit this condition again.

k = i;

continue;

}

//need swap

if(!swappedOnce){

//swap

int temp = a[k];

a[k] = a[i+1];

a[i+1] = temp;

swappedOnce = true;

k = -1;

}else{

//since we have already swapped once we don't want to swap again.. as

// this means the array cannot be sorted by one swap

return true;

}

}

}

}

if(k != -1){

//reaching here means we found array not sorted only at one index..and we couldn't swap it with.

return true;

}

return false;

}

**Q4: Find the longest running positive sequence in an array -**

**Eg - [1,2,-3,2,3,4,-6,1,2,3,4,5,-8,5,6]**

**It should return 5, with start index : 8??**

**Ans:** I would propose the following solution:   
(i) iterate through the elements of the array   
(ii) keep track of the longest positive sequence   
  
A sample code is shown below:

public static void findPosSeq(int a[]) {

int N = a.length;

int maxIdx = -1;

int maxLen = -1;

int len = -1;

int idx = -1;;

boolean flag = false;

for (int k = 0; k < N; k++) {

if (a[k] > 0) {

if (flag == true) {

len++;

}

else {

len = 1; // new sequence

idx = k;

flag = true;

}

}

else {

flag = false;

if (len > maxLen) {

maxLen = len;

maxIdx = idx;

}

}

}

if (maxLen > 0)

System.out.println("Length "+maxLen+", starting index "+maxIdx);

else

System.out.println("No positive sequence detected.")

}

The algoritnm is O(N) time and O(1) space complexity. After modification it should work also with stream.

Other solution

int longestPositiveSequence(int A[], int n)

{

int ans = 0, cnt = 0;

for (int i = 0; i < n; ++i)

{

if (A[i] > 0) ++cnt;

else

{

ans = max(ans, cnt); cnt = 0;

}

}

return ans;

}

**Q5: I have an two arrays int[] 1 = {2,5,8,9}; and int[] 2={6,3,4,7,1};**

**I need to merge this two array in third array int[] 3 = new int[1.Length + 2.Length]; and give the output in sorted form.**

**Also I need to provide and optimized code with minimal complexity...**

**Output: {1,2,3,4,5,6,7,8,9}**

**Ans:** There is a sorting algorithm called external merge sort that can help you with this. Essentially in the beginning you sort the two smaller arrays using something like quicksort, so you have 1 = {2,5,8,9} and 2= {1,3,4,6,7}. Then you check the first element of both arrays, and put the smaller in the third array: min(1[0], 2[0]), so now you have 3 = {1}. Then you "advance" to the second element in the array that you used, and now compare the second element of that array and the first element of the other array, so you check min(1[0], 2[1]). So now array 3 looks like 3 = {1,2}.

Or

Break the two arrays into individual elements and then use merge sort to get combined sorted array.

Code

class MergeArray {

private void heapSort(int[] arr) {

...

}

public int[] mergeIntoSortedArray(int[] arr1, int[] arr2) {

sorting(arr1);

sorting(arr2);

int[] out = new int[arr1.length + arr2.length];

int j = 0;

int k = 0;

int i = 0;

while (j < arr1.length && k < arr2.length) {

if (arr1[j] < arr2[k]) {

out[i++] = arr1[j++];

} else {

out[i++] = arr2[k++];

}

}

while (j < arr1.length) {

out[i++] = arr1[j++];

}

while (k < arr2.length) {

out[i++] = arr2[k++];

}

return out;

}

}

public class MergeTwoArrayIntoSortedArray {

public static void main(String[] args) {

int[] arr1 = new int[] {2,5,8,9};

int[] arr2 = new int[] {6,3,4,7,1};

MergeArray mMergeArray = new MergeArray();

System.out.println("MERGED SORTED ARRAY: "

+ Arrays.toString(mMergeArray.mergeIntoSortedArray(arr1, arr2)));

}

}

Or

public static void main(String []args)

{

int a[] = {2,5,8,9};

int b[] = {6,3,4,7,1};

int c[] = new int[a.length+b.length];

//copy both a,b to c

int i=0;

for(int j=0;j<a.length;j++)

c[i++] = a[j];

for(int j=0;j<b.length;j++)

c[i++] = b[j];

//now merge sort

Arrays.sort(c);

System.out.println(Arrays.toString(c));

}

**Q6: Given a number A, find the smallest number which has only 1s and 0s as its digits which divisible by the number A. For example: if the given number A is 4, the smallest number with 1s and 0s is which is divisible by 4 is 100.??**

Ans: Implement BFS  
Follow the below steps  
1. verify given number is 1 if yes then return true.  
2. If if it not then put 1 into queue  
3. In while (true) pop queue item, Lets say it is "X"  
4. if(INT\_MAX/10 < X) return false  
5. Multiply X by 10 mod by given number if remainder is zero then return true  
6. else add the number to queue.  
7. if((INT\_MAX-1)/10 > X) return false  
8. Multiply X by 10 and add 1 mod by given number if remainder is zero then return true  
9. else add the number to queue.

or

public int findDivisibleNum(int k) {

Set<Integer> numList = new ConcurrentSkipListSet<>();

numList.add(0);

numList.add(1);

for(int i = 10; i < Integer.MAX\_VALUE; i = i \* 10) {

HashSet<Integer> newList = new LinkedHashSet<>();

for(Integer num : numList) {

newList.add(num + i);

}

for(Integer num : newList) {

if(num != 0 && num % k == 0) {

return num;

}

}

numList.addAll(newList);

}

return -1;

}

**Q7: Given a array of positive integers, you have to find the smallest positive integer that can not be formed from the sum of numbers from array.??**

**Ans:**

First sort them.   
Then calculate cumulative sum.   
If some element a[i] is greater than sum + 1 then there is a gap between them.

So number sum + 1 cannot be formed

Or

iterate each number at array, let x is a[i], lets do swap(a[x], x) (we should use swap if x less then length of array else continue)   
after that lets look throw our array and if a[x] != x return x

int findNumber (vector <int> a)

{

for (int i = 0; i < a.size(); i++) {

int x = a[i];

if (x > a.size()) continue;

swap(a[x - 1], x);

}

for (int i = 0; i < a.size(); i++) {

if (a[i] != i + 1) return i + 1;

}

return a.size();

}

**Q8: Given a sorted array with some sequenced numbers and some non-sequenced numbers. Write an algorithm that takes this array as an input and returns a list of {start, end} of all consecutive numbers. Consecutive numbers have difference of 1 only.**

**E.g. of array:**

**[4, 5, 6, 7, 8, 9, 12, 15, 16, 17, 18, 20, 22, 23, 24, 27]**

public class Range

{

private int begin;

private int end;

public int begin { get; set; }

public int end { get; set; }

}

**Ans:** one important thing to consider here is that array is sorted. We need to follow binary search approach here to have the complexity between O(logn) and O(n).   
For the input given in the question, we have a sequence from a[0] to a[5]. We can find the sequence using the comparison (a[low]-a[high] == low-high)

A solution with binary search

public class FindSets {

public static void main(String[] args) {

int[] array = { 4, 5, 6, 7, 8, 9, 12, 15, 16, 17, 18, 20, 22, 23, 24,

27 };

int i = 0;

int size = array.length;

StringBuilder sb = new StringBuilder();

while (i < size - 1) {

int end = findEnd(i, size - 1, array);

if (end != i) {

sb.append("" + i + ":" + end);

i = end + 1;

} else

i++;

}

System.out.println("O/P:" + sb.toString());

}

private static int findEnd(int i, int size, int[] array) {

int low = i, high = size;

int mid;

while (low < high) {

mid = (low + high) / 2;

if (mid - low == array[mid] - array[low]) {

if (mid - low + 1 < array[mid + 1] - array[low])

return mid;

else

low = mid + 1;

} else

high = mid - 1;

}

return low;

}

}

Or

// [4, 5, 6, 7, 8, 9, 12, 15, 16, 17, 18, 20, 22, 23, 24, 27]

private void accumulateSeqNos(int[] data){

List<String> result = new ArrayList<String>();

for(int i=0; i<data.length; i++){

int head = data[i];

while(i+1 < data.length){

// Loop to find the range of consecutive

if((data[i] + 1 != data[++i])){

i--;

break;

}

}

if(head != data[i]){

result.add("{" + head + " : " + data[i] + "}");

}

}

for(int i=0; i<result.size(); i++){

System.out.println(result.get(i));

}

}

**Q9: You're given a very large array of char's. Write a method to remove duplicates in the array, in place. Optimize for space complexity, not time complexity.**

Ans: Sort array lexicographically with mergesort for O(nlogn) then remove duplicate neighbors for O(n).

It is better to use QuickSort, since MergeSort use additional memory, but QuickSort is in-place sort algorithm.

Or

O(n) complexity. Uses extra boolean array.

private static char[] removeDuplicates(char[] data){

if (data == null || data.length < 2)

return data;

boolean[] entries = new boolean[255];

int tail = 1;

for (int head = 1; head < data.length ; ++head){

if (data[head] == data[tail] || entries[data[head]]) {

continue;

}

data[++tail] = data[head];

entries[data[head]] = true;

}

return Arrays.copyOfRange(data , 0 , tail);

**Q10: You're given an array of integers(eg [3,4,7,1,2,9,8]) Find the index of values that satisfy A+B = C + D, where A,B,C & D are integers values in the array.**

Eg: Given [3,4,7,1,2,9,8] array

The following

3+7 = 1+ 9 satisfies A+B=C+D

So print (0,2,3,5)??

Ans:

1. Sort the Array   
2. Take two pointers left (start of the array) and right (end of the array)   
3. Find the sum of the left and right value in the array   
4. Look up iteratively the sum within the subarray , if sum is found print it   
4.a if sum is greater value then increment left side ( i ) or else reduce the (j) value  
  
-- This solution do not use any extra space , no HashMap

public static void indexSumPair(int arr[])

{

//Return the array if Size less than 4

if(arr.length < 4)

{ return; }

// Sort the array

Arrays.sort(arr);

// Take two pointer left and right

int left =0;

int right = arr.length-1;

// decrease the right pointer on each pass \* array is sorted

for(;left<right;right--)

{

// Take sum of left and right

int outSum = arr[left] + arr[right];

int i = left;

int j = right -1;

// Apply Logic of Pairs in Array of Integers whose Sum is equal to a given Number

// look up the sum within the subarray

while(i<j)

{

if( arr[i]+arr[j] == outSum)

{

System.out.println( " ("+arr[i]+" , "+arr[j]+ ") = "+" ("+arr[left]+" , "+arr[right]+")" );

i++;

j--;

}

//

else if( arr[i]+arr[j] < outSum)

{

i++;

}

else if( arr[i]+arr[j] > outSum)

{

j--;

}

}

}

Or

**There are two ways this can be solved:**1 - use 4-depth loops to iterate through all combinations of pairs. This is O(1) of size but O(n^4) of time   
2 - use hashtable of lists of pairs to cache all sums as they are met and use 2-depth loops to find out all pairs of numbers. Then for each sum print out all pairs permutations.   
This uses O(n^2) of size but O(n^2) of time. Actually this reduces to 2n^2 in case all numbers are equal but this it's O(n) complexity is not changed though.

Or

a general solution will b of the complexity O(n^3).   
1. Make a structure like   
struct ds   
{   
int value;   
int index;   
};   
  
2. Sort according to values. // This process is nlogn   
  
3. Iterate trough all pair (A,B) // This process is O(n^2)   
a).compute sum = A+B   
b).search two indexes(i,j) in sorted array such that :   
Array[i]+array[j]=sum // ths process is O(n) in sorted array   
So complexity is O(n^3).   
This can be improved using hashing.

**Q11:** **Given an array and a number, find two integers that sums to the given number.??**

**Ans:** 1. HashMap and use the numbers as keys   
2. Now scan the HashMap and check existence of (Sum - Key) keys in HashMap   
3. Return all pairs that satisfy 2   
Thats O(n) time and O(n) space

Code

public static void main(String[] args) {

int number = 10;

int[] array = new int[]{2,3,5,7,8,10};

HashMap<Integer,Integer> tempHash = new HashMap<Integer, Integer>();

for(int i =0 ;i<array.length;i++)

{

tempHash.put(array[i], 0);

}

for (int i : array)

{

tempHash.remove(i);

if (tempHash.containsKey(number-i))

{

System.out.println("" + i + "," + (number-i));

}

}

}

Or

public static void sumFindingOfTwoNo(Integer[] array, Integer sum) {

HashSet<Integer> sets = new HashSet<Integer>(Arrays.asList(array));

for(Integer no : sets) {

if(sets.contains(sum-no)) {

System.out.println(no+","+(sum-no));

}

}

}