**Add One To Number**

* [Google](https://www.interviewbit.com/search/?q=Google)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given a non-negative number represented as an array of digits,

add 1 to the number ( increment the number represented by the digits ).

The digits are stored such that the most significant digit is at the head of the list.

**Example:**

If the vector has [1, 2, 3]

the returned vector should be [1, 2, 4]

as 123 + 1 = 124.

***NOTE:****Certain things are intentionally left unclear in this question which you should practice asking the interviewer.  
For example, for this problem, following are some good questions to ask :*

**Q :** Can the input have 0’s before the most significant digit. Or in other words, is 0 1 2 3 a valid input?

***A :****For the purpose of this question,****YES***

**Q :** Can the output have 0’s before the most significant digit? Or in other words, is 0 1 2 4 a valid output?

**A :** For the purpose of this question, **NO**. Even if the input has zeroes before the most significant digit.

**public**ArrayList<Integer>plusOne(ArrayList<Integer>A) {

**int**n = A.size()-1;

int val = A.get(n) + 1;

**int**carry = val / 10;

A.set(n, val % 10);

**for**(**int**i=n-1; i>=0; i--){

**if** (carry == 1){

val = A.get(i) + 1;

carry = val / 10;

A.set(i, val % 10);

} **else**{

**break**;

}

}

**if**(carry==1){

A.add(0,1);

}

**while**(A.size()>0 &&A.get(0)==0){

A.remove(0);

}

**return**A;

}

**Max Sum Contiguous Subarray**

* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Paypal](https://www.interviewbit.com/search/?q=Paypal)
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [LinkedIn](https://www.interviewbit.com/search/?q=LinkedIn)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Goldman Sachs](https://www.interviewbit.com/search/?q=Goldman)

Find the **contiguous** subarray within an array, **A** of length **N** which has the **largest sum**.

**Input Format:**

The first and the only argument contains an integer array, A.

**Output Format:**

Return an integer representing the maximum possible sum of the contiguous subarray.

**Constraints:**

1 <= N <= 1e6

-1000 <= A[i] <= 1000

**For example:**

Input 1:

A = [1, 2, 3, 4, -10]

Output 1:

10

Explanation 1:

The subarray [1, 2, 3, 4] has the maximum possible sum of 10.

Input 2:

A = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Output 2:

6

Explanation 2:

The subarray [4,-1,2,1] has the maximum possible sum of 6.

**publicint**maxSubArray(**final** List<Integer>A) {

**int**meh = 0;

**int**msf = Integer.***MIN\_VALUE***;

**for**(**int**i : A){

meh = meh + i;

**if**(meh<i){

meh = i;

}

**if**(msf<meh){

msf = meh;

}

}

**return**msf;

}

**Largest Number**

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Goldman Sachs](https://www.interviewbit.com/search/?q=Goldman)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given a list of non negative integers, arrange them such that they form the largest number.

**For example:**

Given [3, 30, 34, 5, 9], the largest formed number is 9534330.

*Note: The result may be very large, so you need to return a string instead of an integer.*

Example 1:

Input: nums = [10,2]

Output: "210"

Example 2:

Input: nums = [3,30,34,5,9]

Output: "9534330"

Example 3:

Input: nums = [1]

Output: "1"

Example 4:

Input: nums = [10]

Output: "10"

**class**ArrSort**implements** Comparator<String>{

**publicint**compare(String a,Stringb){

**return** (a+b).compareTo((b+a));

}

}

**publicclass**LargestNumber {

**public** String largestNumber(**finalint**[] A) {

String[] arr=**new** String[A.length];

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

arr[i]=Integer.*toString*(A[i]);

}

Arrays.*sort*(arr,**new**ArrSort());

StringBuilder sb=**new**StringBuilder();

**for**(**int**i=arr.length-1;i>=0;i--){

sb.append(arr[i]);

}

**if**(sb.charAt(0)=='0'){

**return**"0";

}

**return**sb.toString();

}

**publicstaticvoid**main(String[] args) {

System.***out***.println(**new**LargestNumber().largestNumber(**newint**[] {3,30,34,50,5,9}));

}

}

**Rotate Matrix**

* [Google](https://www.interviewbit.com/search/?q=Google)
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

You need to do this in place.

Note that if you end up using an additional array, you will only receive partial score.

**Example:**

If the array is

[

[1, 2],

[3, 4]

]

Then the rotated array becomes:

[

[3, 1],

[4, 2]

]

**publicvoid**rotate(ArrayList<ArrayList<Integer>>a) {

**int**n = a.size();

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

**for**(**int**j=i; j<n; j++){

**int**temp = a.get(i).get(j);

a.get(i).set(j,a.get(j).get(i));

a.get(j).set(i, temp);

}

}

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

**for**(**int**j=0; j<n/2; j++){

**int**temp = a.get(i).get(n-1-j);

a.get(i).set(n-1-j,a.get(i).get(j)); a.get(i).set(j, temp);

}

}

}

**Merge Overlapping Intervals**

* [Google](https://www.interviewbit.com/search/?q=Google)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a collection of intervals, merge all overlapping intervals.

**For example:**

Given [1,3],[2,6],[8,10],[15,18],

return [1,6],[8,10],[15,18].

Make sure the returned intervals are sorted.

**public**ArrayList<Interval>merge(ArrayList<Interval>intervals) {

**if**(intervals.size()<2){

**return**intervals;

}

Collections.*sort*(intervals,**new** Comparator<Interval>(){

**publicint**compare(Interval i1,Interval i2)

{

**return**i1.start-i2.start;

}

});

**int**start = intervals.get(0).start;

**int**end = intervals.get(0).end;

ArrayList<Interval>list = **new**ArrayList<Interval>();

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

**int**newStart = intervals.get(i).start;

**int**newEnd = intervals.get(i).end;

**if**(end>= newStart){

end = Math.*max*(end, newEnd);

}**else**{

Interval interval = **new**Interval(start, end);

list.add(interval);

start = newStart;

end = newEnd;

}

}

Interval interval = **new**Interval(start, end);

list.add(interval);

**return**list;

}

**Set Matrix Zeros**

* [Oracle](https://www.interviewbit.com/search/?q=Oracle)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a matrix, **A** of size **M x N** of **0s and 1s**. If an element is **0**, set its entire row and column to **0**.

**Note:** This will be evaluated on the extra memory used. Try to minimize the space and time complexity.

**Input Format:**

The first and the only argument of input contains a 2-d integer matrix, A, of size M x N.

**Output Format:**

Return a 2-d matrix that satisfies the given conditions.

**Constraints:**

1 <= N, M <= 1000

0 <= A[i][j] <= 1

**Examples:**

Input 1:

[ [1, 0, 1],

[1, 1, 1],

[1, 1, 1] ]

Output 1:

[ [0, 0, 0],

[1, 0, 1],

[1, 0, 1] ]

Input 2:

[ [1, 0, 1],

[1, 1, 1],

[1, 0, 1] ]

Output 2:

[ [0, 0, 0],

[1, 0, 1],

[0, 0, 0] ]

**publicvoid**setZeroes(ArrayList<ArrayList<Integer>>a) {

Set<Integer>cols = **new** HashSet<>();

Set<Integer>rows = **new** HashSet<>();

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

**for**(**int**j = 0; j<a.get(0).size(); j++){

**if**(a.get(i).get(j) == 0){

rows.add(i);

cols.add(j);

}

}

}

**for**(**int**row: rows){

updateRow(a, row);

}

**for**(**int**col: cols){

updateCol(a, col);

}

}

**publicvoid**updateRow(ArrayList<ArrayList<Integer>>a, **int**index){

**for**(**int**j = 0; j<a.get(0).size(); j++)

a.get(index).set(j, 0);

}

**publicvoid**updateCol(ArrayList<ArrayList<Integer>>a, **int**index){

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

a.get(i).set(index,0);

}

**First Missing Integer**

* [Model N](https://www.interviewbit.com/search/?q=Model)
* [InMobi](https://www.interviewbit.com/search/?q=InMobi)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given an unsorted integer array, find the first missing positive integer.

**Example:**

Given [1,2,0] return 3,

[3,4,-1,1] return 2,

[-8, -7, -6] returns 1

Your algorithm should run in O(n) time and use constant space.

**publicint**firstMissingPositive(ArrayList<Integer>A) {

**if**(A.size()==0 || !A.contains(1))

**return**1;

**int**n = A.size();

**if**(n==1)

**return**2;

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

**if**(A.get(i)<=0 || A.get(i) >n)

A.set(i, 1);

}

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

**int**indexVal = Math.*abs*(A.get(i))-1;

**if**(A.get(indexVal)>0){

A.set(indexVal, -1 \* A.get(indexVal));

}

}

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

**if**(A.get(i)>0){

**return**i+1;

}

}

**return**n+1;

}

**Pascal Triangle**

* [Google](https://www.interviewbit.com/search/?q=Google)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given numRows, generate the first numRows of Pascal’s triangle.

Pascal’s triangle : To generate A[C] in row R, sum up A’[C] and A’[C-1] from previous row R - 1.

**Example:**

Given numRows = 5,

Return

[

[1],

[1,1],

[1,2,1],

[1,3,3,1],

[1,4,6,4,1]

]

**public**ArrayList<ArrayList<Integer>>solve(**int**A) {

**int**[][] matrix = **newint**[A][A];

ArrayList<ArrayList<Integer>>result = **new**ArrayList<ArrayList<Integer>>();

**if**(A<=0)

**return**result;

ArrayList<Integer>list = **new**ArrayList<Integer>();

matrix[0][0] = 1;

list.add(1);

result.add(list);

**int**row = 1;

**for**(**int**i=1; i<matrix.length; i++) {

list = **new**ArrayList<Integer>();

**for**(**int**j=0; j<=row; j++) {

**if**(j==0) {

matrix[i][j] = 1;

list.add(1);

}**else** {

**int**val = matrix[i-1][j] + matrix[i-1][j-1];

matrix[i][j] = val;

list.add(val);

}

}

result.add(list);

row++;

}

**return**result;

}

**Spiral Order Matrix II**

* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [JP Morgan](https://www.interviewbit.com/search/?q=JP)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given an integer **A**, generate a square matrix filled with elements from **1** to **A2** in **spiral order**.

**Input Format:**

The first and the only argument contains an integer, A.

**Output Format:**

Return a 2-d matrix of size A x A satisfying the spiral order.

**Constraints:**

1 <= A <= 1000

**Examples:**

Input 1:

A = 3

Output 1:

[ [ 1, 2, 3 ],

[ 8, 9, 4 ],

[ 7, 6, 5 ] ]

Input 2:

4

Output 2:

[ [1, 2, 3, 4],

[12, 13, 14, 5],

[11, 16, 15, 6],

[10, 9, 8, 7] ]

**publicint**[][] generateMatrix(**int**A) {

**int**[][] matrix=**newint**[A][A];

**int**n=1;

**int**top=0,left=0,bottom=A-1,right=A-1;

**while**(n<=A\*A) {

**for**(**int**i=left;i<=right;i++) matrix[top][i]=n++;

top++;

**for**(**int**i=top;i<=bottom;i++) matrix[i][right]=n++;

right--;

**for**(**int**i=right;i>=left;i--) matrix[bottom][i]=n++;

bottom--;

**for**(**int**i=bottom;i>=top;i--) matrix[i][left]=n++;

left++;

}

**return**matrix;

}

**Find Duplicate in Array**

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [VMWare](https://www.interviewbit.com/search/?q=VMWare)
* [Riverbed](https://www.interviewbit.com/search/?q=Riverbed)

Given a read only array of n + 1 integers between 1 and n, find one number that repeats in linear time using less than O(n) space and traversing the stream sequentially O(1) times.

**Sample Input:**

[3 4 1 4 1]

**Sample Output:**

1

If there are multiple possible answers ( like in the sample case above ), output any one.

If there is no duplicate, output -1

**publicint**repeatedNumber(**final** List<Integer>A) {

List<Integer>list = **new**ArrayList<Integer>();

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

**if**(list.get(A.get(i)-1)>0)

**return**A.get(i);

**else**

list.add(i-1, A.get(i));

}

**return**-1;

}

**publicint**repeatedNumber(**finalint**[] a) {

**int**n=a.length;

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

{

**int**temp=Math.*abs*(a[i]);

**int**temp2=a[temp];

**if**(temp2<0)//if this happens it will mean that a[temp] is already visited

**return**temp;//which means that temp is already present in the array

**else**

a[temp]=-a[temp];

}

**return**-1;

}

**Wave Array**

* [Google](https://www.interviewbit.com/search/?q=Google)
* [Adobe](https://www.interviewbit.com/search/?q=Adobe)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given an array of integers, sort the array into a wave like array and return it,  
In other words, arrange the elements into a sequence such that a1 >= a2 <= a3 >= a4 <= a5.....

**Example**

Given [1, 2, 3, 4]

One possible answer : [2, 1, 4, 3]

Another possible answer : [4, 1, 3, 2]

***NOTE :****If there are multiple answers possible, return the one thats lexicographically smallest.  
So, in example case, you will return [2, 1, 4, 3]*

**public**ArrayList<Integer>wave(ArrayList<Integer>A) {

Collections.*sort*(A);

**int**n = A.size();

**for**(**int**i=0; i<n-1; i=i+2){

**int**temp = A.get(i);

A.set(i, A.get(i+1));

A.set(i+1, temp);

}

**return**A;

}

1539. Kth Missing Positive Number

Easy

Given an array arr of positive integers sorted in a **strictly increasing order**, and an integer k.

*Find the*kth*positive integer that is missing from this array.*

**Example 1:**

**Input:**arr = [2,3,4,7,11], k = 5

**Output:** 9

**Explanation:** The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5th missing positive integer is 9.

**Example 2:**

**Input:**arr = [1,2,3,4], k = 2

**Output:** 6

**Explanation:** The missing positive integers are [5,6,7,...]. The 2nd missing positive integer is 6.

**Constraints:**

* 1 <= arr.length<= 1000
* 1 <= arr[i] <= 1000
* 1 <= k <= 1000
* arr[i] <arr[j] for 1 <= i< j <= arr.length

**package**com.data.structure.array;

**publicclass**KthMissingNumber {

**publicint** findKthPositive1(**int**[] arr, **int**k) {

//simple binary search

**int**n = arr.length;

**int**left = 0, right = n-1;

**int**missing = compute(arr[n-1], n);

**while**(left<= right) {

**int**mid = left + (right-left)/2;

missing = compute(arr[mid], mid+1);

**if**(missing>= k) right = mid-1;

**else**left = mid+1;

}

// Right ->-1;

**if**(right == -1) **return**k;

**return**arr[right] + k-compute(arr[right], right+1);

}

**int**compute(**int**actual, **int**expected){

**return**actual - expected;

}

**publicint**findKthPositive(**int**[] arr, **int**k) {

//another binary search. diff need to be discussed in both binary searches

**int**low = 0;

**int**high = arr.length;

**while**(low<high){

**int**mid = (high -low)/2 + low;

**if**(arr[mid] -(mid +1) >= k){

high = mid;

} **else**{

low =mid+1;

}

}

//low is the index of smallest element index in the array in which adding k

//will give us the kth missing element.

**return**low +k;

}

**publicstaticvoid**main(String[] args) {

KthMissingNumberobj=**new**KthMissingNumber();

**int**[] arr=**newint**[] {2,3,4,7,11};

**int**k=5;

obj.findKthPositive(arr, k);

}

}

**713. Subarray Product Less Than K**

Medium

208979Add to ListShare

Your are given an array of positive integers nums.

Count and print the number of (contiguous) subarrays where the product of all the elements in the subarray is less than k.

**Example 1:**

**Input:**nums = [10, 5, 2, 6], k = 100

**Output:** 8

**Explanation:** The 8 subarrays that have product less than 100 are: [10], [5], [2], [6], [10, 5], [5, 2], [2, 6], [5, 2, 6].

Note that [10, 5, 2] is not included as the product of 100 is not strictly less than k.

**publicstatic** List<List<Integer>>findSubarrays(**int**[] arr, **int**target) {

**int**count = 0;

List<List<Integer>>result = **new**ArrayList<>();

**int**product = 1, left = 0;

**for** (**int**right = 0; right<arr.length; right++) {

product \*= arr[right];

**while** (product>= target&&left<arr.length)

product /= arr[left++];

// since the product of all numbers from left to right is less than the target therefore,

// all subarrays from lef to right will have a product less than the target too; to avoid

// duplicates, we will start with a subarray containing only arr[right] and then extend it

List<Integer>tempList = **new** LinkedList<>();

**for** (**int**i = right; i>= left; i--) {

tempList.add(0, arr[i]);

result.add(**new**ArrayList<>(tempList));

}

count+=(right-left+1);

}

System.***out***.println(count);

**return**result;

}

**publicstaticvoid**main(String[] args) {

*findSubarrays*(**newint**[] {10,5,2,6}, 100);

}

**Sales By Match**

Alex works at a clothing store. There is a large pile of socks that must be paired by color for sale. Given an array of integers representing the color of each sock, determine how many pairs of socks with matching colors there are.

For example, there are  n=7 socks with colors ar=[1,2,1,2,1,3,2]. There is one 1 pair of color  and one of color 2 . There are three odd socks left, one of each color. The number of pairs is 2.

**Function Description**

Complete the *sockMerchant* function in the editor below. It must return an integer representing the number of matching pairs of socks that are available.

sockMerchant has the following parameter(s):

* *n*: the number of socks in the pile
* *ar*: the colors of each sock

**Input Format**

The first line contains an integer , the number of socks represented in .  
The second line contains  space-separated integers describing the colors  of the socks in the pile.

**Output Format**

Return the total number of *matching pairs* of socks that Alex can sell.

**Sample Input**

9

10 20 20 10 10 30 50 10 20

**Sample Output**

3

**publicvoid**salesByMatch(**int**n, **int**[] c) {

Set<Integer>colors = **new** HashSet<>();

**int**pairs = 0;

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

**if**(!colors.contains(c[i])) {

colors.add(c[i]);

} **else** {

pairs++;

colors.remove(c[i]);

}

}

System.***out***.println(pairs);

}

**Repeated String**

There is a string, , of lowercase English letters that is repeated infinitely many times. Given an integer, , find and print the number of letter a's in the first  letters of the infinite string.

**Example**

S=abcac

N=10

The substring we consider is abcacabcac, the first  10characters of the infinite string. There are 4 occurrences of a in the substring.

**Function Description**

Complete the *repeatedString* function in the editor below.

repeatedString has the following parameter(s):

* *s:* a string to repeat
* *n:* the number of characters to consider

**Returns**

* *int:* the frequency of a in the substring

**Input Format**

The first line contains a single string, s.  
The second line contains an integer, n.

**Sample Input**

**Sample Input 0**

aba

10

**Sample Output 0**

7

**Explanation 0**  
The first n=10 letters of the infinite string are abaabaabaa. Because there are 7 a's, we 7 return .

**Sample Input 1**

a

1000000000000

**Sample Output 1**

1000000000000

**Explanation 1**  
Because all of the first  n=1000000000000letters of the infinite string are a, we return 1000000000000.

**staticlong**repeatedString(String s, **long**n) {

**long**size = s.length(), repeated = n/size;

**long**left = n - (size \* repeated);

**int**extra = 0;

**int**count = 0;

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

**if**(s.charAt(i) == 'a'){

++count;

}

}

**for**(**int**i = 0; i<left; i++){

**if**(s.charAt(i) == 'a'){

++extra;

}

}

repeated = (repeated \* count) + extra;

**return**repeated;

}

**Counting Valleys**

An avid hiker keeps meticulous records of their hikes. During the last hike that took exactly  **steps**steps, for every step it was noted if it was an *uphill*, **U**, or a *downhill*,**D**  step. Hikes always start and end at sea level, and each step up or down represents a **1** unit change in altitude. We define the following terms:

* A *mountain* is a sequence of consecutive steps *above* sea level, starting with a step *up* from sea level and ending with a step *down* to sea level.
* A *valley* is a sequence of consecutive steps *below* sea level, starting with a step *down* from sea level and ending with a step *up* to sea level.

Given the sequence of *up* and *down* steps during a hike, find and print the number of *valleys* walked through.

**Example**

 Steps = 8 path=[DDUUUUDD]

The hiker first enters a valley 2 units deep. Then they climb out and up onto a mountain**2**  units high. Finally, the hiker returns to sea level and ends the hike.

**Function Description**

Complete the *countingValleys* function in the editor below.

countingValleys has the following parameter(s):

* *int steps*: the number of steps on the hike
* *string path*: a string describing the path

**Returns**

* *int:* the number of valleys traversed

**Input Format**

The first line contains an integer**steps** , the number of steps in the hike.  
The second line contains a single string **path**, of **steps**characters that describe the path.

**Sample Input**

8

UDDDUDUU

**Sample Output**

1

**Explanation**

If we represent \_ as sea level, a step up as /, and a step down as \, the hike can be drawn as:

\_/\ \_

\ /

\/\/

The hiker enters and leaves one valley.

**publicstaticint**countingValleys(**int**steps, String path) {

// Write your code here

**int**v = 0; // # of valleys

**int**lvl = 0; // current level

**for**(**char**c : path.toCharArray()){

**if**(c == 'U') ++lvl;

**if**(c == 'D') --lvl;

// if we just came UP to sea level

**if**(lvl == 0 &&c == 'U')

++v;

}

System.***out***.print(v);

**return**v;

}

**publicstaticvoid**main(String[] args) {

*countingValleys*(8, "DDUUUUDD");

}

**Knap Sack**

**package** com.patterns.dynamic.programming.aditya.verma.knap.sack;

**public** **class** kanpsack {

**int**[][] dp;

**public** **int** getKnapSack(**int**[] w, **int**[] p, **int** c, **int** n) {

dp = **new** **int**[n+1][c+1];

**for**(**int** i=1; i<=n; i++) {

**for**(**int** j=1; j<=c; j++) {

//Max of (Weight is less than equal to capacity, will use the price of the weight plus price of capacity minus weight from previous row) or (Previous price on same capacity)

**if**(w[i-1]<=j) {

dp[i][j] = Math.*max*( p[i-1] + dp[i-1] [j-w[i-1]], dp[i-1][j]);

}**else**

dp[i][j]= dp[i-1][j];

}

}

**return** dp[n][c];

}

**public** **static** **void** main(String[] args) {

**int**[] w = {3,4,6,5};

**int**[] p = {2,3,1,4};

**int** capacity = 8;

kanpsack testDP = **new** kanpsack();

System.***out***.println("getKnapSackUsingBottomUp " +testDP.getKnapSack(w, p, capacity, w.length));

}

}

**Subset Sum**

**boolean**[][] b;

**public** **boolean** getSubsetSumUsingBottomUp(**int**[] arr, **int** sum, **int** n) {

b = **new** **boolean**[n+1][sum+1];

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

**for**(**int** j=0; j<=sum; j++) {

**if**(j==0)

b[i][j] = **true**;

**else** **if**(i==0)

b[i][j] = **false**;

}

}

**for**(**int** i=1; i<=n; i++) {

**for**(**int** j=1; j<=sum; j++) {

**if**(arr[i-1]<=j) {

b[i][j] = b[i-1] [j-arr[i-1]] || b[i-1][j];

}**else**

b[i][j]= b[i-1][j];

}

}

**return** b[n][sum];

}

**Equal Sum Subset**

**public** **boolean** canPartition(**int**[] nums) {

**int** sum = 0;

**for**(**int** val:nums)

sum+=val;

**if**(sum%2==0) {

**return** getEqualSubsetSumUsingDP(nums, sum/2, nums.length);

}

**return** **false**;

}

**public** **boolean** getEqualSubsetSumUsingDP(**int**[] arr, **int** sum, **int** n) {

**boolean**[][] b = **new** **boolean**[n+1][sum+1];

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

**for**(**int** j=0; j<=sum; j++) {

**if**(j==0)

b[i][j] = **true**;

**else** **if**(i==0)

b[i][j] = **false**;

}

}

**for**(**int** i=1; i<=n; i++) {

**for**(**int** j=1; j<=sum; j++) {

**if**(arr[i-1]<=j) {

b[i][j] = b[i-1] [j-arr[i-1]] || b[i-1][j];

}**else**

b[i][j]= b[i-1][j];

}

}

**return** b[n][sum];

}

**Count of Subset Sum**

**public** **int** getCountSubsetSumUsingDP(**int**[] arr, **int** sum, **int** n) {

**int**[][] b = **new** **int**[arr.length+1][sum+1];

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

**for**(**int** j=0; j<=sum; j++) {

**if**(j==0)

b[i][j] = 1;

**else** **if**(i==0)

b[i][j] = 0;

}

}

**for**(**int** i=1; i<=n; i++) {

**for**(**int** j=1; j<=sum; j++) {

**if**(arr[i-1]<=j) {

b[i][j] = b[i-1] [j-arr[i-1]] + b[i-1][j];

}**else**

b[i][j]= b[i-1][j];

}

}

printSelectedElements(b, arr, sum);

**return** b[n][sum];

}

**Equal Subset Sum Diff**

**public** **boolean**[][] b = **null**;

**public** **int** lastStoneWeightII(**int**[] stones) {

**int** sum = 0;

**for**(**int** val : stones)

sum+=val;

getMinimumSubsetSumDiffUsingDP(stones, sum, stones.length);

**int** minSubsetSumDiff = Integer.***MAX\_VALUE***;

**int** max = sum/2;

**for**(**int** j=0; j<=max; j++) {

**if**(b[b.length-1][j]) {

minSubsetSumDiff = Math.*min*(minSubsetSumDiff, sum-(2\*j));

}

}

**return** minSubsetSumDiff;

}

**public** **boolean** getMinimumSubsetSumDiffUsingDP(**int**[] arr, **int** sum, **int** n) {

b = **new** **boolean**[arr.length+1][sum+1];

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

**for**(**int** j=0; j<=sum; j++) {

**if**(j==0)

b[i][j] = **true**;

**else** **if**(i==0)

b[i][j] = **false**;

}

}

**for**(**int** i=1; i<=n; i++) {

**for**(**int** j=1; j<=sum; j++) {

**if**(arr[i-1]<=j) {

b[i][j] = b[i-1] [j-arr[i-1]] || b[i-1][j];

}**else**

b[i][j]= b[i-1][j];

}

}

**return** b[n][sum];

}

## 1) Java char to int Example: Get ASCII value

Let's see the simple code to convert char to int in java.

**public** **class** CharToIntExample1{

**public** **static** **void** main(String args[]){

**char** c='a';

**char** c2='1';

**int** a=c;

**int** b=c2;

System.out.println(a);

System.out.println(b);

}

}

**Output:**

97

49

## 2) Java char to int Example: Character.getNumericValue()

Let's see the simple code to convert char to int in java using Character.getNumericValue(char) method which returns an integer value.

**public** **class** CharToIntExample2{

**public** **static** **void** main(String args[]){

**char** c='1';

**int** a=Character.getNumericValue(c);

System.out.println(a);

}

}

**Output:**

1

## 3) Java char to int Example: String.valueOf()

Let's see another example which returns integer value of specified char value using String.valueOf(char) method.

**public** **class** CharToIntExample3{

**public** **static** **void** main(String args[]){

**char** c='1';

**int** a=Integer.parseInt(String.valueOf(c));

System.out.println(a);

}

}

**Output:**

1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Decimal | Char |  | Decimal | Char |  | Decimal | Char |
| 48 | 0 |  | 65 | A |  | 97 | a |
| 49 | 1 |  | 66 | B |  | 98 | B |
| 50 | 2 |  | 67 | C |  | 99 | C |
| 51 | 3 |  | 68 | D |  | 100 | D |
| 52 | 4 |  | 69 | E |  | 101 | E |
| 53 | 5 |  | 70 | F |  | 102 | F |
| 54 | 6 |  | 71 | G |  | 103 | G |
| 55 | 7 |  | 72 | H |  | 104 | H |
| 56 | 8 |  | 73 | I |  | 105 | I |
| 57 | 9 |  | 74 | J |  | 106 | J |
|  |  |  | 75 | K |  | 107 | K |
|  |  |  | 76 | L |  | 108 | L |
|  |  |  | 77 | M |  | 109 | M |
|  |  |  | 78 | N |  | 110 | N |
|  |  |  | 79 | O |  | 111 | O |
|  |  |  | 80 | P |  | 112 | P |
|  |  |  | 81 | Q |  | 113 | Q |
|  |  |  | 82 | R |  | 114 | R |
|  |  |  | 83 | S |  | 115 | S |
|  |  |  | 84 | T |  | 116 | T |
|  |  |  | 85 | U |  | 117 | U |
|  |  |  | 86 | V |  | 118 | V |
|  |  |  | 87 | W |  | 119 | W |
|  |  |  | 88 | X |  | 120 | X |
|  |  |  | 89 | Y |  | 121 | Y |
|  |  |  | 90 | Z |  | 122 | Z |

Convert Decimal to Binary

**public** **class** DecimalToBinaryExample{

**public** **static** **void** main(String args[]){

System.out.println(Integer.toBinaryString(10));

System.out.println(Integer.toBinaryString(21));

System.out.println(Integer.toBinaryString(31));

}

}

**Output:**

1010

10101

11111

**public** **class** DecimalToBinaryExample {

**public** **static** **void** toBinary(**int** decimal){

**int** binary[] = **new** **int**[40];

**int** index = 0;

**while**(decimal > 0){

binary[index++] = decimal%2;

decimal = decimal/2;

}

**for**(**int** i = index-1;i >= 0;i--){

System.***out***.print(binary[i]);

}

System.***out***.println();//new line

}

**public** **static** **void** main(String args[]){

System.***out***.println("Decimal of 10 is: ");

*toBinary*(10);

System.***out***.println("Decimal of 21 is: ");

*toBinary*(21);

System.***out***.println("Decimal of 31 is: ");

*toBinary*(31);

}

}

Decimal of 10 is:

1010

Decimal of 21 is:

10101

Decimal of 31 is:

11111

Convert Binary to Decimal

**public** **void** convertBinaryToDecimal(String binaryString) {

**int** decimal=Integer.*parseInt*(binaryString,2);

System.***out***.println(decimal);

}

**public** **void** convertBinaryToDecimalCustom1(String binaryString) {

**int** decimal = 0;

**char** ch [] = binaryString.toCharArray();

**int** pow=0;

**for**(**int** index=ch.length-1; index>=0; index--) {

decimal+= Math.*pow*(2, pow++) \* (ch[index]-48);

}

System.***out***.println("Decimal of binary "+ binaryString + " is " +decimal);

}

**public** **void** convertBinaryToDecimalCustom2(String binaryString) {

**int** decimal = 0;

**for**(**char** ch: binaryString.toCharArray()) {

decimal = decimal \* 2 + (ch - 48);

}

System.***out***.println("Decimal of binary "+ binaryString + " is " +decimal);

}

**public** **static** **void** main(String[] args) {

**new** BinaryToDecimal().convertBinaryToDecimal("1010");

**new** BinaryToDecimal().convertBinaryToDecimalCustom1("1010");

**new** BinaryToDecimal().convertBinaryToDecimalCustom2("1110");

}