In programming, algorithm is a set of well defined instructions in sequence to solve the problem.

**Qualities of a good algorithm**

1. Input and output should be defined precisely.
2. Each steps in algorithm should be clear and unambiguous.
3. Algorithm should be most effective among many different ways to solve a problem.
4. An algorithm shouldn't have computer code. Instead, the algorithm should be written in such a way that, it can be used in similar programming languages.

**Examples Of Algorithms In Programming**

**Write an algorithm to add two numbers entered by user.**

Step 1: Start

Step 2: Declare variables num1, num2 and sum.

Step 3: Read values num1 and num2.

Step 4: Add num1 and num2 and assign the result to sum.

sum←num1+num2

Step 5: Display sum

Step 6: Stop

**Write an algorithm to find the largest among three different numbers entered by user.**

Step 1: Start

Step 2: Declare variables a,b and c.

Step 3: Read variables a,b and c.

Step 4: If a>b

If a>c

Display a is the largest number.

Else

Display c is the largest number.

Else

If b>c

Display b is the largest number.

Else

Display c is the greatest number.

Step 5: Stop

**Write an algorithm to find all roots of a quadratic equation ax2+bx+c=0.**

Step 1: Start

Step 2: Declare variables a, b, c, D, x1, x2, rp and ip;

Step 3: Calculate discriminant

D←b2-4ac

Step 4: If D≥0

r1←(-b+√D)/2a

r2←(-b-√D)/2a

Display r1 and r2 as roots.

Else

Calculate real part and imaginary part

rp←b/2a

ip←√(-D)/2a

Display rp+j(ip) and rp-j(ip) as roots

Step 5: Stop

**Write an algorithm to find the factorial of a number entered by user.**

Step 1: Start

Step 2: Declare variables n,factorial and i.

Step 3: Initialize variables

factorial←1

i←1

Step 4: Read value of n

Step 5: Repeat the steps until i=n

5.1: factorial←factorial\*i

5.2: i←i+1

Step 6: Display factorial

Step 7: Stop

**Write an algorithm to check whether a number entered by user is prime or not.**

Step 1: Start

Step 2: Declare variables n,i,flag.

Step 3: Initialize variables

flag←1

i←2

Step 4: Read n from user.

Step 5: Repeat the steps until i<(n/2)

5.1 If remainder of n÷i equals 0

flag←0

Go to step 6

5.2 i←i+1

Step 6: If flag=0

Display n is not prime

else

Display n is prime

Step 7: Stop

**Write an algorithm to find the Fibonacci series till term≤1000.**

Step 1: Start

Step 2: Declare variables first\_term,second\_term and temp.

Step 3: Initialize variables first\_term←0 second\_term←1

Step 4: Display first\_term and second\_term

Step 5: Repeat the steps until second\_term≤1000

5.1: temp←second\_term

5.2: second\_term←second\_term+first term

5.3: first\_term←temp

5.4: Display second\_term

Step 6: Stop

Algorithm is not the computer code. Algorithm are just the instructions which gives clear idea to you idea to write the computer code.

# **Ostrich algorithm**

In [computer science](https://en.wikipedia.org/wiki/Computer_science), the **ostrich algorithm** is a strategy of ignoring potential problems on the basis that they may be exceedingly rare. It is named for the [ostrich effect](https://en.wikipedia.org/wiki/Ostrich_effect) which is defined as "to stick one's head in the sand and pretend there is no problem". It is used when it is more cost-effective to allow the problem to occur than to attempt its prevention.

## Use with deadlocks[[edit](https://en.wikipedia.org/w/index.php?title=Ostrich_algorithm&action=edit&section=1)]

This approach may be used in dealing with [deadlocks](https://en.wikipedia.org/wiki/Deadlock) in [concurrent programming](https://en.wikipedia.org/wiki/Concurrent_programming) if they are believed to be very rare and the cost of detection or prevention is high. For example, if each PC deadlocks once per 10 years, the one reboot may be less painful than the restrictions needed to prevent it.[[1]](https://en.wikipedia.org/wiki/Ostrich_algorithm#cite_note-1)

A set of processes is [deadlocked](https://en.wikipedia.org/wiki/Deadlock) if each process in the set is waiting for an event that only another process in the set can cause. Usually the event is release of a currently held resource and none of the processes can run, release resources, and be awakened.[[2]](https://en.wikipedia.org/wiki/Ostrich_algorithm#cite_note-2)

The ostrich algorithm pretends there is no problem and is reasonable to use if deadlocks occur very rarely and the cost of their prevention would be high. The [UNIX](https://en.wikipedia.org/wiki/UNIX) and [Windows](https://en.wikipedia.org/wiki/Windows) operating systems take this approach.[[3]](https://en.wikipedia.org/wiki/Ostrich_algorithm#cite_note-3)

Although using the ostrich algorithm is one of the methods of dealing with [deadlocks](https://en.wikipedia.org/wiki/Deadlock), other effective methods exist such as dynamic avoidance, [banker's algorithm](https://en.wikipedia.org/wiki/Banker%27s_algorithm), detection and recovery, and prevention.[[4]](https://en.wikipedia.org/wiki/Ostrich_algorithm#cite_note-4)

<https://www.youtube.com/watch?v=g5rJwMQtDaY>

# DEADLOCKS

Computer systems are full of resources that can only be used by one process at a time. Common examples include printers, tape drives, and slots in the system’s internal tables. Having two processes simultaneously writing to the printer leads to gibberish. Having two processes using the same file system table slot will invariably lead to a corrupted file system. Consequently, all operating systems have the ability to (temporarily) grant a process exclusive access to certain resources.

For many applications, a process needs exclusive access to not one resource, but several. Suppose, for example, two processes each want to record a scanned document on a CD. Process *A* requests permission to use the scanner and is granted it. Process *B* is programmed differently and requests the CD recorder first and is also granted it. Now *A* asks for the CD recorder, but the request is denied until *B* releases it. Unfortunately, instead of releasing the CD recorder *B* asks for the scanner. At this point both processes are blocked and will remain so forever. This situation is called a **deadlock**.

Deadlocks can also occur across machines. For example, many offices have a local area network with many computers connected to it. Often devices such as scanners, CD recorders, printers, and tape drives are connected to the network as shared resources, available to any user on any machine. If these devices can be reserved remotely (i.e., from the user’s home machine), the same kind of deadlocks can occur as described above. More complicated situations can cause deadlocks involving three, four, or more devices and users.

Deadlocks can occur in a variety of situations besides requesting dedicated I/O devices. In a database system, for example, a program may have to lock several records it is using, to avoid race conditions. If process *A* locks record *R1* and process *B* locks record *R2*, and then each process tries to lock the other one’s record, we also have a deadlock. Thus deadlocks can occur on hardware resources or on software resources.

In this chapter, we will look at deadlocks more closely, see how they arise, and study some ways of preventing or avoiding them. Although this material is about deadlocks in the context of operating systems, they also occur in database systems and many other contexts in computer science, so this material is actually applicable to a wide variety of multiprocess systems. A great deal has been written about deadlocks. Two bibliographies on the subject have appeared in *Operating Systems Review* and should be consulted for references (Newton, 1979; and Zobel, 1983). Although these bibliographies are old, most of the work on deadlocks was done well before 1980, so they are still useful.

## 3.1 RESOURCES

Deadlocks can occur when processes have been granted exclusive access to devices, files, and so forth. To make the discussion of deadlocks as general as possible, we will refer to the objects granted as **resources**. A resource can be a hardware device (e.g., a tape drive) or a piece of information (e.g., a locked record in a database). A computer will normally have many different resources that can be acquired. For some resources, several identical instances may be available, such as three tape drives. When several copies of a resource are available, any one of them can be used to satisfy any request for the resource. In short, a resource is anything that can be used by only a single process at any instant of time.

### 3.1.1 Preemptable and Nonpreemptable Resources

Resources come in two types: preemptable and nonpreemptable. A preemptable resource is one that can be taken away from the process owning it with no ill effects. Memory is an example of a preemptable resource. Consider, for example, a system with 32 MB of user memory, one printer, and two 32-MB processes that each want to print something. Process *A* requests and gets the printer, then starts to compute the values to print. Before it has finished with the computation, it exceeds its time quantum and is swapped out.

Process *B* now runs and tries, unsuccessfully, to acquire the printer. Potentially, we now have a deadlock situation, because *A* has the printer and *B* has the memory, and neither can proceed without the resource held by the other. Fortunately, it is possible to preempt (take away) the memory from *B* by swapping it out and swapping *A* in. Now *A* can run, do its printing, and then release the printer. No deadlock occurs.

A **nonpreemptable resource**, in contrast, is one that cannot be taken away from its current owner without causing the computation to fail. If a process has begun to burn a CD-ROM, suddenly taking the CD recorder away from it and giving it to another process will result in a garbled CD, CD recorders are not preemptable at an arbitrary moment.

In general, deadlocks involve nonpreemptable resources. Potential deadlocks that involve preemptable resources can usually be resolved by reallocating resources from one process to another. Thus our treatment will focus on nonpreemptable resources.

The sequence of events required to use a resource is given below in an abstract form.

1. Request the resource.
2. Use the resource.
3. Release the resource.

If the resource is not available when it is requested, the requesting process is forced to wait. In some operating systems, the process is automatically blocked when a resource request fails, and awakened when it becomes available. In other systems, the request fails with an error code, and it is up to the calling process to wait a little while and try again.

A process whose resource request has just been denied will normally sit in a tight loop requesting the resource, then sleeping, then trying again. Although this process is not blocked, for all intents and purposes, it is as good as blocked, because it cannot do any useful work. In our further treatment, we will assume that when a process is denied a resource request, it is put to sleep.

The exact nature of requesting a resource is highly system dependent. In some systems, a request system call is provided to allow processes to explicitly ask for resources. In others, the only resources that the operating system knows about are special files that only one process can have open at a time. These are opened by the usual open call. If the file is already in use, the caller is blocked until its current owner closes it.

# **7 algorithms and data structures every programmer must know**

In programmers life algorithms and data structures is most important subject if they want to go out in the programming world and make some bucks. Today, We will see what they do and where they are used with simplest examples. This list is prepared keeping in mind their use in competitive programming and current development practices.

## 1. Sort Algorithms

Sorting is the most heavily studied concept in Computer Science. Idea is to arrange the items of a list in a specific order. Though every major programming language has built-in sorting libraries, it comes in handy if you know how they work. Depending upon requirement you may want to use any of these.

* Merge Sort
* Quick Sort
* Bucket Sort
* Heap Sort
* Counting Sort

More importantly one should know [when and where to use](http://stackoverflow.com/questions/1933759/when-is-each-sorting-algorithm-used) them. Some examples where you can find direct application of sorting techniques include:

* Sorting by price, popularity etc in e-commerce websites

## 2. Search Algorithms

**Binary Search** (in linear data structures)

Binary search is used to perform a very efficient search on sorted dataset. The time complexity is O(log2N). Idea is to repeatedly divide in half the portion of the list that could contain the item, until we narrow it down to one possible item. Some applications are:

* When you search for a name of song in a sorted list of songs, it performs binary search and string-matching to quickly return the results.
* Used to debug in git through [git bisect](http://git-scm.com/book/en/v2/Git-Tools-Debugging-with-Git#Binary-Search)

**Depth/Breadth First Search** (in Graph data structures)

DFS and BFS are tree/graph traversing and searching data structures. We wouldn’t go deep into [how DFS/BFS work](https://codeaccepted.wordpress.com/2014/04/08/depth-and-breadth-first-search/) but will see how they are different through following animation.

Applications:

* Used by search engines for web-crawling
* Used in artificial intelligence to build bots, for instance a chess bot
* Finding shortest path between two cities in a map and many other such applications

## 3. Hashing

Hash lookup is currently the most widely used technique to find appropriate data by key or ID. We access data by its index. Previously we relied on Sorting+Binary Search to look for index whereas now we use hashing.

The data structure is referred as Hash-Map or Hash-Table or Dictionary that maps keys to values, efficiently. We can perform value lookups using keys. Idea is to use an appropriate hash function which does the key -> value mapping. Choosing a good hash function depends upon the scenario.

Applications:

* In routers, to store IP address -> Path pair for routing mechanisms
* To perform the check if a value already exists in a list. Linear search would be expensive. We can also use Set data structure for this operation.

## 4. Dynamic Programming

Dynamic programming (DP) is a method for solving a complex problem by breaking it down into simpler subproblems. We solve the subproblems, remember their results and using them we make our way to solve the complex problem, quickly.

*\*writes down “1+1+1+1+1+1+1+1 =” on a sheet of paper\* What’s that equal to?*  
*\*counting\* Eight!*  
*\*writes down another “1+” on the left\* What about that?*  
*\*quickly\* Nine!*  
*How’d you know it was nine so fast?*  
*You just added one more*  
*So you didn’t need to recount because you remembered there were eight! Dynamic Programming is just a fancy way to say ‘remembering stuff to save time later’*

Applications:

* There are many DP [algorithms](http://en.wikiversity.org/wiki/Topic:Dynamic_programming) and [applications](https://www.quora.com/What-are-some-real-world-problems-that-have-been-solved-with-dynamic-programming) but I’d name one and blow you away, Duckworth-Lewis method in cricket.

## 5. Exponentiation by squaring

Say you want to calculate 232. Normally we’d iterate 32 times and find the result. What if I told you it can be done in 5 iterations?

Exponentiation by squaring or [Binary exponentiation](https://www.hackerearth.com/notes/mod-integer-exponentiation-useful-in-competetive-programming/) is a general method for fast computation of large positive integer powers of a number in O(log2N). Not only this, the method is also used for computation of powers of polynomials and square matrices.

Application:

* Calculation of large powers of a number is mostly required in RSA encryption. RSA also uses modular arithmetic along with binary exponentiation.

## 6. String Matching and Parsing

Pattern matching/searching is one of the most important problem in Computer Science. There have been a lot of research on the topic but we’ll enlist only two basic necessities for any programmer.

**KMP Algorithm** (String Matching)

Knuth-Morris-Pratt algorithm is used in cases where we have to match a short pattern in a long string. For instance, when we Ctrl+F a keyword in a document, we perform pattern matching in the whole document.

**Regular Expression** (String Parsing)

Many a times we have to validate a string by parsing over a predefined restriction. It is heavily used in web development for URL parsing and matching.

## 7. Primality Testing Algorithms

There are deterministic and probabilistic ways of determining whether a given number is prime or not. We’ll see both deterministic and probabilistic (nondeterministic) ways.

**Sieve of Eratosthenes (deterministic)**

If we have certain limit on the range of numbers, say determine all primes within range 100 to 1000 then Sieve is a way to go. The length of range is a crucial factor, because we have to allocate certain amount of memory according to range.

**For any number n, incrementally testing upto sqrt(n) (deterministic)**

In case you want to check for few numbers which are sparsely spread over a long range (say 1 to 1012), Sieve won’t be able to allocate enough memory. You can check for each number n by traversing only upto sqrt(n) and perform a divisibility check on n.

**Fermat primality test and Miller–Rabin primality test (both are nondeterministic)**

Both of these are compositeness tests. If a number is proved to be composite, then it sure isn’t a prime number. Miller-Rabin is a more sophisticated one than Fermat’s. Infact, Miller-Rabin also has a deterministic variant, but then its a game of trade between time complexity and accuracy of the algorithm.

Application:

* The single most important use of prime numbers is in Cryptography. More precisely, they are used in encryption and decryption in RSA algorithm which was the very first implementation of Public Key Cryptosystems
* Another use is in Hash functions used in Hash Tables

We’ll discuss some advanced algorithms every competitive programmer should know in the next post. Meanwhile master the above algorithms or share in the comments about what you think every beginner-intermediate programmer should know.

# **Commonly Asked Algorithm Interview Questions | Set 1**

**What is an algorithm?**  
Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output. (Source: [Introduction to Algorithms 3rd Edition by CLRS](http://www.flipkart.com/introduction-algorithms-3rd/p/itmczynzhyhxv2gs?pid=9788120340077&affid=sandeepgfg))

**What is time complexity of Binary Search?**  
Time complexity of binary search is O(Logn). See [Binary Search](http://quiz.geeksforgeeks.org/binary-search/) for more details.

**Can Binary Search be used for linked lists?**  
Since random access is not allowed in linked list, we cannot reach the middle element in O(1) time. Therefore Binary Search is not possible for linked lists. There are other ways though, refer [Skip List](https://www.geeksforgeeks.org/skip-list/) for example.

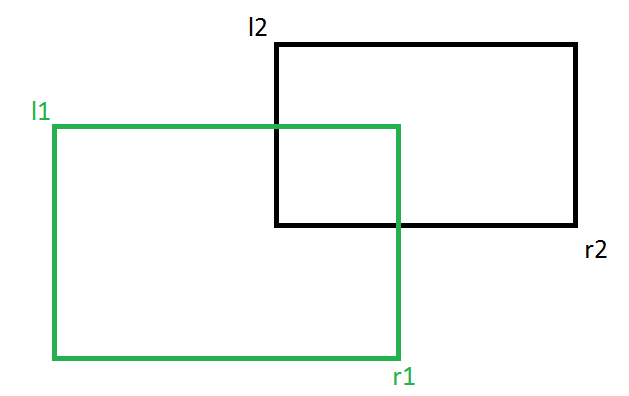
***How to find if two given rectangles overlap?***  
Two rectangles do not overlap if one of the following conditions is true.  
1) One rectangle is above top edge of other rectangle.  
2) One rectangle is on left side of left edge of other rectangle.

See [Find if two rectangles overlap](https://www.geeksforgeeks.org/find-two-rectangles-overlap/) for more details.

# **Find if two rectangles overlap**

Given two rectangles, find if the given two rectangles overlap or not.

Note that a rectangle can be represented by two coordinates, top left and bottom right. So mainly we are given following four coordinates.  
**l1**: Top Left coordinate of first rectangle.  
**r1**: Bottom Right coordinate of first rectangle.  
**l2**: Top Left coordinate of second rectangle.  
**r2**: Bottom Right coordinate of second rectangle.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/rectanglesOverlap.png)

We need to write a function bool doOverlap(l1, r1, l2, r2) that returns true if the two given rectangles overlap.

**Note :**It may be assumed that the rectangles are parallel to the coordinate axis.

One solution is to one by one pick all points of one rectangle and [see if the point lies inside the other rectangle or not](https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/). This can be done using the algorithm discussed [here](https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/).  
Following is a simpler approach. Two rectangles **do not** overlap if one of the following conditions is true.  
**1)** One rectangle is above top edge of other rectangle.  
**2)** One rectangle is on left side of left edge of other rectangle.

We need to check above cases to find out if given rectangles overlap or not. Following is the implementation of the above approach.

|  |
| --- |
| // C# program to check if rectangles overlap  using System;    class GFG  {      class Point      {          public int x, y;      }        // Returns true if two rectangles (l1, r1)      // and (l2, r2) overlap      static bool doOverlap(Point l1, Point r1,                            Point l2, Point r2)      {          // If one rectangle is on left side of other          if (l1.x > r2.x || l2.x > r1.x)          {              return false;          }            // If one rectangle is above other          if (l1.y < r2.y || l2.y < r1.y)          {              return false;          }          return true;      }        // Driver Code      public static void Main()      {          Point l1 = new Point(), r1 = new Point(),                  l2 = new Point(), r2 = new Point();          l1.x = 0;l1.y = 10; r1.x = 10;r1.y = 0;          l2. = 5;l2.y = 5; r2.x = 15;r2.y = 0;          if (doOverlap(l1, r1, l2, r2))          {              Console.WriteLine("Rectangles Overlap");          } else          {              Console.WriteLine("Rectangles Don't Overlap");          }      }  }    // This code is contributed by  // Rajput-Ji |

**Output:**

Rectangles Overlap

***How to find angle between hour and minute hands at a given time?***  
The idea is to take a reference point as 12. Find the angle moved by hour and minute hands, subtract the two angles to find the angle between them.

See [angle between hour hand and minute hand](https://www.geeksforgeeks.org/calculate-angle-hour-hand-minute-hand/) for more details

# **Calculate the angle between hour hand and minute hand**

This problem is know as [Clock angle problem](http://en.wikipedia.org/wiki/Clock_angle_problem) where we need to find angle between hands of an analog clock at a given time.

**Examples:**

Input: h = 12:00, m = 30.00

Output: 165 degree

Input: h = 3.00, m = 30.00

Output: 75 degree

The idea is to take 12:00 (h = 12, m = 0) as a reference. Following are detailed steps.

**1)** Calculate the angle made by hour hand with respect to 12:00 in h hours and m minutes.  
**2)** Calculate the angle made by minute hand with respect to 12:00 in h hours and m minutes.  
**3)** The difference between two angles is the angle between two hands.

**How to calculate the two angles with respect to 12:00?**  
The minute hand moves 360 degree in 60 minute(or 6 degree in one minute) and hour hand moves 360 degree in 12 hours(or 0.5 degree in 1 minute). In h hours and m minutes, the minute hand would move (h\*60 + m)\*6 and hour hand would move (h\*60 + m)\*0.5.

|  |
| --- |
| // C# program to find angle between  // hour and minute hands  using System;    class GFG {        // Function to calculate the angle      static int calcAngle(double h, double m)      {          // validate the input          if (h < 0 || m < 0 ||              h > 12 || m > 60)              Console.Write("Wrong input");            if (h == 12)              h = 0;            if (m == 60)              m = 0;            // Calculate the angles moved by hour and          // minute hands with reference to 12:00          int hour\_angle = (int)(0.5 \* (h \* 60 + m));          int minute\_angle = (int)(6 \* m);            // Find the difference between two angles          int angle = Math.Abs(hour\_angle - minute\_angle);            // smaller angle of two possible angles          angle = Math.Min(360 - angle, angle);            return angle;      }        // Driver code      public static void Main ()      {          Console.WriteLine(calcAngle(9, 60));          Console.Write(calcAngle(3, 30));      }  }    // This code is contributed by Nitin Mittal. |

**Output:**

90

75

**Exercise:** Find all times when hour and minute hands get superimposed.

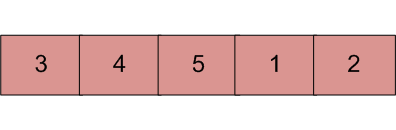
**When does the worst case of QuickSort occur?**  
In [quickSort](http://quiz.geeksforgeeks.org/quick-sort/), we select a pivot element, then partition the given array around the pivot element by placing pivot element at its correct position in sorted array.  
The worst case of quickSort occurs when one part after partition contains all elements and other part is empty. For example, if the input array is sorted and if last or first element is chosen as a pivot, then the worst occurs. See <http://quiz.geeksforgeeks.org/quick-sort/> for more details.

**A sorted array is rotated at some unknown point, how to efficiently search an element in it.**  
A simple approach is linear search, but we can search in O(Logn) time using [Binary Search](http://quiz.geeksforgeeks.org/binary-search/).

See [Search an element in a sorted and pivoted array](https://www.geeksforgeeks.org/search-an-element-in-a-sorted-and-pivoted-array/) for more details.

# **Search an element in a sorted and rotated array**

An element in a sorted array can be found in O(log n) time via [binary search](https://www.geeksforgeeks.org/binary-search/). But suppose we rotate an ascending order sorted array at some pivot unknown to you beforehand. So for instance, 1 2 3 4 5 might become 3 4 5 1 2. Devise a way to find an element in the rotated array in O(log n) time.



Input : arr[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};

key = 3

Output : Found at index 8

Input : arr[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};

key = 30

Output : Not found

Input : arr[] = {30, 40, 50, 10, 20}

key = 10

Output : Found at index 3

**All solutions provided here assume that all elements in array are distinct.**

The idea is to find the pivot point, divide the array in two sub-arrays and call binary search.  
The main idea for finding pivot is – for a sorted (in increasing order) and pivoted array, pivot element is the only element for which next element to it is smaller than it.  
Using above criteria and binary search methodology we can get pivot element in O(logn) time

Input arr[] = {3, 4, 5, 1, 2}

Element to Search = 1

1) Find out pivot point and divide the array in two

sub-arrays. (pivot = 2) /\*Index of 5\*/

2) Now call binary search for one of the two sub-arrays.

(a) **If** element is greater than 0th element then

search in left array

(b) **Else** Search in right array

(1 will go in else as 1 < 0th element(3))

3) **If** element is found in selected sub-array then return index

**Else** return -1.

Below is the implementation of the above approach :

|  |
| --- |
| // C# program to search an element  // in a sorted and pivoted array  using System;    class main {        // Searches an element key in a      // pivoted sorted array arrp[]      // of size n      static int pivotedBinarySearch(int[] arr,                                  int n, int key)      {          int pivot = findPivot(arr, 0, n - 1);            // If we didn't find a pivot, then          // array is not rotated at all          if (pivot == -1)              return binarySearch(arr, 0, n - 1, key);            // If we found a pivot, then first          // compare with pivot and then          // search in two subarrays around pivot          if (arr[pivot] == key)              return pivot;            if (arr[0] <= key)              return binarySearch(arr, 0, pivot - 1, key);            return binarySearch(arr, pivot + 1, n - 1, key);      }        /\* Function to get pivot. For array      3, 4, 5, 6, 1, 2 it returns      3 (index of 6) \*/      static int findPivot(int[] arr, int low, int high)      {          // base cases          if (high < low)              return -1;          if (high == low)              return low;            /\* low + (high - low)/2; \*/          int mid = (low + high) / 2;            if (mid < high && arr[mid] > arr[mid + 1])              return mid;            if (mid > low && arr[mid] < arr[mid - 1])              return (mid - 1);            if (arr[low] >= arr[mid])              return findPivot(arr, low, mid - 1);            return findPivot(arr, mid + 1, high);      }        /\* Standard Binary Search function \*/      static int binarySearch(int[] arr, int low,                              int high, int key)      {          if (high < low)              return -1;            /\* low + (high - low)/2; \*/          int mid = (low + high) / 2;            if (key == arr[mid])              return mid;          if (key > arr[mid])              return binarySearch(arr, (mid + 1), high, key);            return binarySearch(arr, low, (mid - 1), key);      }        // Driver Code      public static void Main()      {          // Let us search 3 in below array          int[] arr1 = { 5, 6, 7, 8, 9, 10, 1, 2, 3 };          int n = arr1.Length;          int key = 3;          Console.Write("Index of the element is : "                      + pivotedBinarySearch(arr1, n, key));      }  }    // This code is contributed by vt\_m. |

Output:

Index of the element is : 8

Time Complexity O(logn). Thanks to Ajay Mishra for initial solution.

**Improved Solution:**  
We can search an element in one pass of Binary Search. The idea is to search

1) Find middle point mid = (l + h)/2

2) **If** key is present at middle point, return mid.

3) **Else If** arr[l..mid] is sorted

a) **If** key to be searched lies in range from arr[l]

to arr[mid], recur for arr[l..mid].

b) **Else** recur for arr[mid+1..h]

4) **Else** (arr[mid+1..h] must be sorted)

a) **If** key to be searched lies in range from arr[mid+1]

to arr[h], recur for arr[mid+1..h].

b) **Else** recur for arr[l..mid]

Below is the implementation of above idea :

|  |
| --- |
| /\* C# program to search an element in  sorted and rotated array using  single pass of Binary Search\*/  using System;    class GFG {        // Returns index of key in arr[l..h]      // if key is present, otherwise      // returns -1      static int search(int []arr, int l, int h,                                         int key)      {          if (l > h)              return -1;            int mid = (l + h) / 2;            if (arr[mid] == key)              return mid;            /\* If arr[l...mid] is sorted \*/          if (arr[l] <= arr[mid])          {                /\* As this subarray is sorted, we              can quickly check if key lies in              half or other half \*/              if (key >= arr[l] && key <= arr[mid])              return search(arr, l, mid - 1, key);                return search(arr, mid + 1, h, key);          }            /\* If arr[l..mid] is not sorted,          then arr[mid... r] must be sorted\*/          if (key >= arr[mid] && key <= arr[h])              return search(arr, mid + 1, h, key);            return search(arr, l, mid - 1, key);      }        // main function      public static void Main()      {          int []arr = {4, 5, 6, 7, 8, 9, 1, 2, 3};          int n = arr.Length;          int key = 6;          int i = search(arr, 0, n - 1, key);            if (i != -1)              Console.WriteLine("Index: " + i);          else              Console.WriteLine("Key not found");      }  }    // This code is contributed by anuj\_67. |

Output:

Index: 2

Thanks to Gaurav Ahirwar for suggesting above solution.

**How to handle duplicates?**  
It doesn’t look possible to search in O(Logn) time in all cases when duplicates are allowed. For example consider searching 0 in {2, 2, 2, 2, 2, 2, 2, 2, 0, 2} and {2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}. It doesn’t look possible to decide whether to recur for left half or right half by doing constant number of comparisons at the middle.

Other variations of this problem like [find the minimum element or maximum element in a sorted and rotated array](https://www.geeksforgeeks.org/find-minimum-element-in-a-sorted-and-rotated-array/).

# **Find the Missing Number**

You are given a list of n-1 integers and these integers are in the range of 1 to n. There are no duplicates in the list. One of the integers is missing in the list. Write an efficient code to find the missing integer.

**Example :**

***Input****: arr[] = {1, 2, 4,, 6, 3, 7, 8}****Output****: 5*

***Input****: arr[] = {1, 2, 3, 5}****Output****: 4*

**METHOD 1(Use sum formula)**  
**Algorithm**:

1. Get the sum of numbers which is total = n\*(n+1)/2

2. Subtract all the numbers from sum and

you will get the missing number

|  |
| --- |
| // C# program to find missing Number  using System;    class GFG {      // Function to ind missing number      static int getMissingNo(int[] a, int n)      {          int total = (n + 1) \* (n + 2) / 2;            for (int i = 0; i < n; i++)              total -= a[i];            return total;      }        /\* program to test above function \*/      public static void Main()      {          int[] a = { 1, 2, 4, 5, 6 };          int miss = getMissingNo(a, 5);          Console.Write(miss);      }  }    // This code is contributed by Sam007\_ |

**Output :**

3

**Time Complexity:** O(n)

There can be overflow if n is large. In order to avoid Integer Overflow, we can pick one number from known numbers and subtract one number from given numbers. This way we won’t have Integer Overflow ever. Thanks to Sahil Rally for suggesting this improvement.

Here is the implementation of the same

|  |
| --- |
| using System;    class GFG  {    // a represents the array  // n : Number of elements in array a  static int getMissingNo(int[] a, int n)  {      int i, total = 1;        for ( i = 2; i <= (n + 1); i++)      {          total += i;          total -= a[i - 2];      }      return total;  }    // Driver Code  public static void Main()  {      int[] arr = {1, 2, 3, 5};      Console.Write(getMissingNo(arr, (arr.Length)));        // Console.Write(getMissingNo(arr, 4));  }  }    // This code is contributed by SoumikMondal |

**METHOD 2(Use XOR)**

1) XOR all the array elements, let the result of XOR be X1.

2) XOR all numbers from 1 to n, let XOR be X2.

3) XOR of X1 and X2 gives the missing number.

|  |
| --- |
| // C# program to find missing Number  // using xor  using System;    class GFG {      // Function to find missing number      static int getMissingNo(int[] a, int n)      {          int x1 = a[0];          int x2 = 1;            /\* For xor of all the elements          in array \*/          for (int i = 1; i < n; i++)              x1 = x1 ^ a[i];            /\* For xor of all the elements          from 1 to n+1 \*/          for (int i = 2; i <= n + 1; i++)              x2 = x2 ^ i;            return (x1 ^ x2);      }        /\* driver program to test above function \*/      public static void Main()      {          int[] a = { 1, 2, 4, 5, 6 };          int miss = getMissingNo(a, 5);          Console.Write(miss);      }  }    // This code is contributed by Sam007\_ |

**Output :**

3

**Given a big string of characters, how to efficiently find the first unique character in it?**  
The efficient solution is to use character as an index in a count array. Traverse the given string and store index of first occurrence of every character, also store count of occurrences. Then traverse the count array and find the smallest index with count as 1.

See [find the first unique character](https://www.geeksforgeeks.org/given-a-string-find-its-first-non-repeating-character/) for more details.

# **Given a string, find its first non-repeating character**

Given a string, find the first non-repeating character in it. For example, if the input string is “GeeksforGeeks”, then output should be ‘f’ and if input string is “GeeksQuiz”, then output should be ‘G’.

  
We can use string characters as index and build a count array. Following is the algorithm.

1) Scan the string from left to right and construct the count array.

2) Again, scan the string from left to right and check for count of each

character, if you find an element who's count is 1, return it.

**Example:**

Input string: str = geeksforgeeks

1: Construct character count array from the input string.

....

count['e'] = 4

count['f'] = 1

count['g'] = 2

count['k'] = 2

……

2: Get the first character who's count is 1 ('f').

**Implementation:**

|  |
| --- |
| // C# program to find first non-repeating character  using System;  using System.Globalization;    class GFG {        static int NO\_OF\_CHARS = 256;      static char []count = new char[NO\_OF\_CHARS];        /\* calculate count of characters      in the passed string \*/      static void getCharCountArray(string str)      {          for (int i = 0; i < str.Length; i++)              count[str[i]]++;      }        /\* The method returns index of first non-repeating      character in a string. If all characters are      repeating then returns -1 \*/      static int firstNonRepeating(string str)      {          getCharCountArray(str);          int index = -1, i;            for (i = 0; i < str.Length; i++)          {              if (count[str[i]] == 1)              {                  index = i;                  break;              }          }        return index;      }        // Driver code      public static void Main()      {          string str = "geeksforgeeks";          int index = firstNonRepeating(str);            Console.WriteLine(index == -1 ? "Either " +          "all characters are repeating or string " +          "is empty" : "First non-repeating character"          + " is " + str[index]);      }  }    // This code is contributed by Sam007 |

**Output:**

First non-repeating character is f

**Can we do it by traversing the string only once?**  
The above approach takes O(n) time, but in practice it can be improved. The first part of the algorithm runs through the string to construct the count array (in O(n) time). This is reasonable. But the second part about running through the string again just to find the first non-repeater is not good in practice. In real situations, your string is expected to be much larger than your alphabet. Take DNA sequences for example: they could be millions of letters long with an alphabet of just 4 letters. What happens if the non-repeater is at the end of the string? Then we would have to scan for a long time (again).  
We can augment the count array by storing not just counts but also the index of the first time you encountered the character e.g. (3, 26) for ‘a’ meaning that ‘a’ got counted 3 times and the first time it was seen is at position 26. So when it comes to finding the first non-repeater, we just have to scan the count array, instead of the string. Thanks to Ben for suggesting this approach.

Following is implementation of the extended approach that traverses the input string only once.

|  |
| --- |
| // C# program to find first  // non-repeating character  // Note : hashmap is used  using System;  using System.Collections.Generic;    class CountIndex  {      public int count, index;        // constructor for first occurrence      public CountIndex(int index)      {          this.count = 1;          this.index = index;      }        // method for updating count      public virtual void incCount()      {          this.count++;      }  }    class GFG  {      public const int NO\_OF\_CHARS = 256;        public static Dictionary<char,                    CountIndex> hm = new Dictionary<char,                                         CountIndex>(NO\_OF\_CHARS);        /\* calculate count of characters      in the passed string \*/      public static void getCharCountArray(string str)      {          for (int i = 0; i < str.Length; i++)          {              // If character already occurred,              if (hm.ContainsKey(str[i]))              {                  // updating count                  hm[str[i]].incCount();              }                // If it's first occurrence, then              // store the index and count = 1              else              {                  hm[str[i]] = new CountIndex(i);              }            }      }        /\* The method returns index of first      non-repeating character in a string.      If all characters are repeating then      returns -1 \*/      internal static int firstNonRepeating(string str)      {          getCharCountArray(str);          int result = int.MaxValue, i;            for (i = 0; i < str.Length; i++)          {              // If this character occurs only              // once and appears before the              // current result, then update the result              if (hm[str[i]].count == 1 &&                  result > hm[str[i]].index)              {                  result = hm[str[i]].index;              }            }        return result;      }        // Driver Code      public static void Main(string[] args)      {          string str = "geeksforgeeks";          int index = firstNonRepeating(str);            Console.WriteLine(index == int.MaxValue ?                            "Either all characters are repeating " +                            " or string is empty" :                            "First non-repeating character is " +                                                     str[index]);      }  }    // This code is contributed by Shrikant13 |

**Output:**

First non-repeating character is f

Related Problem :[K’th Non-repeating Character](https://www.geeksforgeeks.org/kth-non-repeating-character/)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

**How to count inversions in a sorted array?**  
Two elements arr[i] and arr[j] in an array arr[] form an inversion if a[i] > a[j] and i < j. How to count all inversions in an unsorted array.

See [Count Inversions in an array](https://www.geeksforgeeks.org/counting-inversions/) for all approaches.

# **Count Inversions in an array | Set 1 (Using Merge Sort)**

Inversion Count for an array indicates – how far (or close) the array is from being sorted. If array is already sorted then inversion count is 0. If array is sorted in reverse order that inversion count is the maximum.  
Formally speaking, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j

**Example:**  
The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).

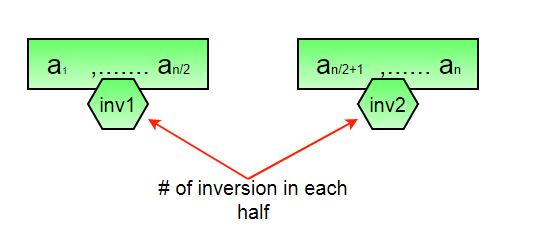
**METHOD 1 (Simple)**  
For each element, count number of elements which are on right side of it and are smaller than it.

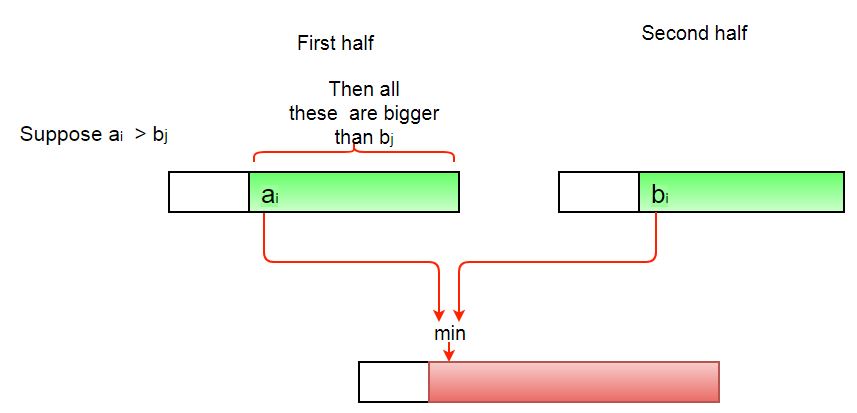
|  |
| --- |
| // C# program to count inversions  // in an array  using System;  using System.Collections.Generic;    class GFG {        static int[] arr = new int[] { 1, 20, 6, 4, 5 };        static int getInvCount(int n)      {          int inv\_count = 0;            for (int i = 0; i < n - 1; i++)              for (int j = i + 1; j < n; j++)                  if (arr[i] > arr[j])                      inv\_count++;            return inv\_count;      }        // Driver code      public static void Main()      {          Console.WriteLine("Number of "                            + "inversions are "                            + getInvCount(arr.Length));      }  }    // This code is contributed by Sam007 |

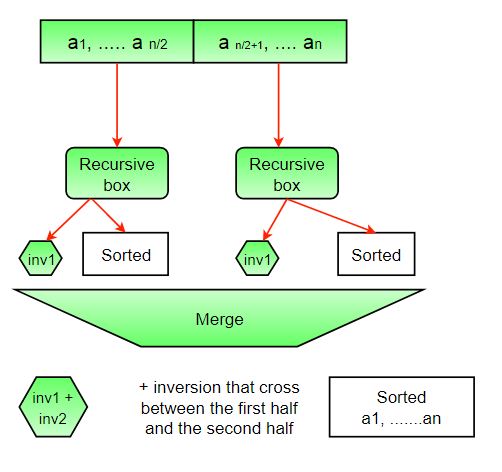
**Output:**

Number of inversions are 5

**Time Complexity:** O(n^2)  
**METHOD 2(Enhance Merge Sort)**  
Suppose we know the number of inversions in the left half and right half of the array (let be inv1 and inv2), what kinds of inversions are not accounted for in Inv1 + Inv2? The answer is – the inversions we have to count during the merge step. Therefore, to get number of inversions, we need to add number of inversions in left subarray, right subarray and merge().

  
**How to get number of inversions in merge()?**  
In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge(), if a[i] is greater than a[j], then there are (mid – i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than a[j]



**The complete picture:**  


**Implementation:**

|  |
| --- |
| // C# implementation of counting the  // inversion using merge sort    using System;  public class Test {        /\* This method sorts the input array and returns the         number of inversions in the array \*/      static int mergeSort(int[] arr, int array\_size)      {          int[] temp = new int[array\_size];          return \_mergeSort(arr, temp, 0, array\_size - 1);      }        /\* An auxiliary recursive method that sorts the input array and        returns the number of inversions in the array. \*/      static int \_mergeSort(int[] arr, int[] temp, int left, int right)      {          int mid, inv\_count = 0;          if (right > left) {              /\* Divide the array into two parts and call \_mergeSortAndCountInv()             for each of the parts \*/              mid = (right + left) / 2;                /\* Inversion count will be the sum of inversions in left-part, right-part            and number of inversions in merging \*/              inv\_count += \_mergeSort(arr, temp, left, mid);              inv\_count += \_mergeSort(arr, temp, mid + 1, right);                /\*Merge the two parts\*/              inv\_count += merge(arr, temp, left, mid + 1, right);          }          return inv\_count;      }        /\* This method merges two sorted arrays and returns inversion count in         the arrays.\*/      static int merge(int[] arr, int[] temp, int left, int mid, int right)      {          int i, j, k;          int inv\_count = 0;            i = left; /\* i is index for left subarray\*/          j = mid; /\* j is index for right subarray\*/          k = left; /\* k is index for resultant merged subarray\*/          while ((i <= mid - 1) && (j <= right)) {              if (arr[i] <= arr[j]) {                  temp[k++] = arr[i++];              }              else {                  temp[k++] = arr[j++];                    /\*this is tricky -- see above explanation/diagram for merge()\*/                  inv\_count = inv\_count + (mid - i);              }          }            /\* Copy the remaining elements of left subarray         (if there are any) to temp\*/          while (i <= mid - 1)              temp[k++] = arr[i++];            /\* Copy the remaining elements of right subarray         (if there are any) to temp\*/          while (j <= right)              temp[k++] = arr[j++];            /\*Copy back the merged elements to original array\*/          for (i = left; i <= right; i++)              arr[i] = temp[i];            return inv\_count;      }        // Driver method to test the above function      public static void Main()      {          int[] arr = new int[] { 1, 20, 6, 4, 5 };          Console.Write("Number of inversions are " + mergeSort(arr, 5));      }  }  // This code is contributed by Rajput-Ji |

**Output:**

Number of inversions are 5

**Time Complexity:**O(N log N)  
**Algorithmic Paradigm:** Divide and Conquer

Note that above code modifies (or sorts) the input array. If we want to count only inversions then we need to create a copy of original array and call mergeSort() on copy.

***Given a big array, how to efficiently find k’th largest element in it?***  
There can be many solutions for this. The best solution is to use min heap. We Build a Min Heap MH of the first k elements. For each element, after the kth element (arr[k] to arr[n-1]), compare it with root of MH, if the element is greater than the root then make it root and call heapify for MH, Else ignore it. Finally, MH has k largest elements and root of the MH is the kth largest element.

See [k largest(or smallest) elements](https://www.geeksforgeeks.org/k-largestor-smallest-elements-in-an-array/) for more details.

# **k largest(or smallest) elements in an array | added Min Heap method**

**Question:**Write an efficient program for printing k largest elements in an array. Elements in array can be in any order.

For example, if given array is [1, 23, 12, 9, 30, 2, 50] and you are asked for the largest 3 elements i.e., k = 3 then your program should print 50, 30 and 23.

**Method 1 (Use Bubble k times)**  
Thanks to Shailendra for suggesting this approach.  
1) Modify [Bubble Sort](https://www.geeksforgeeks.org/bubble-sort/) to run the outer loop at most k times.  
2) Print the last k elements of the array obtained in step 1.

Time Complexity: O(nk)

Like Bubble sort, other sorting algorithms like [Selection Sort](http://en.wikipedia.org/wiki/Selection_sort) can also be modified to get the k largest elements.

**Method 2 (Use temporary array)**  
K largest elements from arr[0..n-1]

1) Store the first k elements in a temporary array temp[0..k-1].  
2) Find the smallest element in temp[], let the smallest element be min.  
3-a) For each element x in arr[k] to arr[n-1]. **O(n-k)**  
If x is greater than the min then remove min from temp[] and insert x.  
3-b)Then, determine the new min from temp[]. **O(k)**  
4) Print final k elements of temp[]

Time Complexity: O((n-k)\*k). If we want the output sorted then O((n-k)\*k + klogk)

Thanks to nesamani1822 for suggesting this method.

**Method 3(Use Sorting)**  
1) Sort the elements in descending order in O(nLogn)  
2) Print the first k numbers of the sorted array O(k).  
Following is the implementation of above.

|  |
| --- |
| // C# code for k largest elements in an array  using System;    class GFG {      public static void kLargest(int[] arr, int k)      {          // Sort the given array arr in reverse order          // This method doesn't work with primitive data          // types. So, instead of int, Integer type          // array will be used          Array.Sort(arr);          Array.Reverse(arr);            // Print the first kth largest elements          for (int i = 0; i < k; i++)              Console.Write(arr[i] + " ");      }        // Driver code      public static void Main(String[] args)      {          int[] arr = new int[] { 1, 23, 12, 9,                                  30, 2, 50 };          int k = 3;          kLargest(arr, k);      }  }    // This code contributed by Rajput-Ji |

**Output:**

50 30 23

**Time complexity:** O(nlogn)

**Method 4 (Use Max Heap)**  
1) Build a Max Heap tree in O(n)  
2) Use Extract Max k times to get k maximum elements from the Max Heap O(klogn)

**Time complexity:** O(n + klogn)

**Method 5(Use Oder Statistics)**  
1) Use order statistic algorithm to find the kth largest element. Please [see the topic selection in worst-case linear time](https://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array-set-3-worst-case-linear-time/)O(n)  
2) Use [QuickSort](https://www.geeksforgeeks.org/quick-sort/)Partition algorithm to partition around the kth largest number O(n).  
3) Sort the k-1 elements (elements greater than the kth largest element) O(kLogk). This step is needed only if sorted output is required.

**Time complexity:** O(n) if we don’t need the sorted output, otherwise O(n+kLogk)

Thanks to Shilpi for suggesting the first two approaches.

**Method 6 (Use Min Heap)**  
This method is mainly an optimization of method 1. Instead of using temp[] array, use Min Heap.

1) Build a Min Heap MH of the first k elements (arr[0] to arr[k-1]) of the given array. O(k)

2) For each element, after the kth element (arr[k] to arr[n-1]), compare it with root of MH.  
……a) If the element is greater than the root then make it root and call [heapify](https://www.geeksforgeeks.org/binary-heap/)for MH  
……b) Else ignore it.  
// The step 2 is O((n-k)\*logk)

3) Finally, MH has k largest elements and root of the MH is the kth largest element.

Time Complexity: O(k + (n-k)Logk) without sorted output. If sorted output is needed then O(k + (n-k)Logk + kLogk)

All of the above methods can also be used to find the kth largest (or smallest) element.

|  |
| --- |
| #include <iostream>  using namespace std;    // Swap function to interchange  // the value of variables x and y  int swap(int& x, int& y)  {      int temp = x;      x = y;      y = temp;  }    // Min Heap Class  // arr holds reference to an integer  // array size indicate the number of  // elements in Min Heap  class MinHeap {        int size;      int\* arr;    public:      // Constructor to initialize the size and arr      MinHeap(int size, int input[]);        // Min Heapify function, that assumes that      // 2\*i+1 and 2\*i+2 are min heap and fix the      // heap property for i.      void heapify(int i);        // Build the min heap, by calling heapify      // for all non-leaf nodes.      void buildHeap();  };    // Constructor to initialize data  // members and creating mean heap  MinHeap::MinHeap(int size, int input[])  {      // Initializing arr and size        this->size = size;      this->arr = input;        // Building the Min Heap      buildHeap();  }    // Min Heapify function, that assumes  // 2\*i+1 and 2\*i+2 are min heap and  // fix min heap property for i    void MinHeap::heapify(int i)  {      // If Leaf Node, Simply return      if (i >= size / 2)          return;        // variable to store the smallest element      // index out of i, 2\*i+1 and 2\*i+2      int smallest;        // Index of left node      int left = 2 \* i + 1;        // Index of right node      int right = 2 \* i + 2;        // Select minimum from left node and      // current node i, and store the minimum      // index in smallest variable      smallest = arr[left] < arr[i] ? left : i;        // If right child exist, compare and      // update the smallest variable      if (right < size)          smallest = arr[right] < arr[smallest]                               ? right : smallest;        // If Node i violates the min heap      // property, swap  current node i with      // smallest to fix the min-heap property      // and recursively call heapify for node smallest.      if (smallest != i) {          swap(arr[i], arr[smallest]);          heapify(smallest);      }  }    // Build Min Heap  void MinHeap::buildHeap()  {      // Calling Heapify for all non leaf nodes      for (int i = size / 2 - 1; i >= 0; i--) {          heapify(i);      }  }    void FirstKelements(int arr[],int size,int k){      // Creating Min Heap for given      // array with only k elements      MinHeap\* m = new MinHeap(k, arr);        // Loop For each element in array      // after the kth element      for (int i = k; i < size; i++) {            // if current element is smaller          // than minimum element, do nothing          // and continue to next element          if (arr[0] > arr[i])              continue;            // Otherwise Change minimum element to          // current element, and call heapify to          // restore the heap property          else {              arr[0] = arr[i];              m->heapify(0);          }      }      // Now min heap contains k maximum      // elements, Iterate and print      for (int i = 0; i < k; i++) {          cout << arr[i] << " ";      }  }  // Driver Program  int main()  {        int arr[] = { 11, 3, 2, 1, 15, 5, 4,                             45, 88, 96, 50, 45 };        int size = sizeof(arr) / sizeof(arr[0]);        // Size of Min Heap      int k = 3;        FirstKelements(arr,size,k);        return 0;  }  // This code is contributed by Ankur Goel |

**Output:**

50 88 96

Please write comments if you find any of the above explanations/algorithms incorrect, or find better ways to solve the same problem.

**Given an array of size n with range of numbers from 1 to n+1. The array doesn’t contain any duplicate, one number is missing, find the missing number.**  
There can be many ways to solve it. The best among is to use XOR.

See [Find the missing number](https://www.geeksforgeeks.org/find-the-missing-number/) for details.

# **Find the Missing Number**

You are given a list of n-1 integers and these integers are in the range of 1 to n. There are no duplicates in the list. One of the integers is missing in the list. Write an efficient code to find the missing integer.

**Example :**

***Input****: arr[] = {1, 2, 4,, 6, 3, 7, 8}****Output****: 5*

***Input****: arr[] = {1, 2, 3, 5}****Output****: 4*

**METHOD 1(Use sum formula)**

**Algorithm**:

1. Get the sum of numbers which is total = n\*(n+1)/2

2. Subtract all the numbers from sum and

you will get the missing number

|  |
| --- |
| // C# program to find missing Number  using System;    class GFG {      // Function to ind missing number      static int getMissingNo(int[] a, int n)      {          int total = (n + 1) \* (n + 2) / 2;            for (int i = 0; i < n; i++)              total -= a[i];            return total;      }        /\* program to test above function \*/      public static void Main()      {          int[] a = { 1, 2, 4, 5, 6 };          int miss = getMissingNo(a, 5);          Console.Write(miss);      }  }    // This code is contributed by Sam007\_ |

**Output :**

3

**Time Complexity:** O(n)

There can be overflow if n is large. In order to avoid Integer Overflow, we can pick one number from known numbers and subtract one number from given numbers. This way we won’t have Integer Overflow ever. Thanks to Sahil Rally for suggesting this improvement.

Here is the implementation of the same

|  |
| --- |
| using System;    class GFG  {    // a represents the array  // n : Number of elements in array a  static int getMissingNo(int[] a, int n)  {      int i, total = 1;        for ( i = 2; i <= (n + 1); i++)      {          total += i;          total -= a[i - 2];      }      return total;  }    // Driver Code  public static void Main()  {      int[] arr = {1, 2, 3, 5};      Console.Write(getMissingNo(arr, (arr.Length)));        // Console.Write(getMissingNo(arr, 4));  }  }    // This code is contributed by SoumikMondal |

**METHOD 2(Use XOR)**

1) XOR all the array elements, let the result of XOR be X1.

2) XOR all numbers from 1 to n, let XOR be X2.

3) XOR of X1 and X2 gives the missing number.

|  |
| --- |
| // C# program to find missing Number  // using xor  using System;    class GFG {      // Function to find missing number      static int getMissingNo(int[] a, int n)      {          int x1 = a[0];          int x2 = 1;            /\* For xor of all the elements          in array \*/          for (int i = 1; i < n; i++)              x1 = x1 ^ a[i];            /\* For xor of all the elements          from 1 to n+1 \*/          for (int i = 2; i <= n + 1; i++)              x2 = x2 ^ i;            return (x1 ^ x2);      }        /\* driver program to test above function \*/      public static void Main()      {          int[] a = { 1, 2, 4, 5, 6 };          int miss = getMissingNo(a, 5);          Console.Write(miss);      }  }    // This code is contributed by Sam007\_ |

**Output :**

3

There are many variations of this problem like [find the two repeating numbers](https://www.geeksforgeeks.org/find-the-two-repeating-elements-in-a-given-array/),

# **Find the two repeating elements in a given array**

You are 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. Find the two repeating numbers.

For example, array = {4, 2, 4, 5, 2, 3, 1} and n = 5

The above array has n + 2 = 7 elements with all elements occurring once except 2 and 4 which occur twice. So the output should be 4 2.

**Method 1 (Basic)**  
Use two loops. In the outer loop, pick elements one by one and count the number of occurrences of the picked element in the inner loop.

This method doesn’t use the other useful data provided in questions like range of numbers is between 1 to n and there are only two repeating elements.

|  |
| --- |
| using System;    class GFG  {        static void printRepeating(int []arr, int size)      {          int i, j;            Console.Write("Repeated Elements are :");          for (i = 0; i < size; i++)          {              for (j = i + 1; j < size; j++)              {                  if (arr[i] == arr[j])                      Console.Write(arr[i] + " ");              }          }      }      // driver code      public static void Main()      {          int []arr = {4, 2, 4, 5, 2, 3, 1};          int arr\_size = arr.Length;            printRepeating(arr, arr\_size);      }  }    // This code is contributed by Sam007 |

**Output :**

Repeating elements are 4 2

Time Complexity: O(n\*n)  
Auxiliary Space: O(1)

**Method 2 (Use Count array)**  
Traverse the array once. While traversing, keep track of count of all elements in the array using a temp array count[] of size n, when you see an element whose count is already set, print it as duplicate.

This method uses the range given in the question to restrict the size of count[], but doesn’t use the data that there are only two repeating elements.

|  |
| --- |
| // C# program to Find the two  // repeating elements in a given array  using System;    class GFG  {        static void printRepeating(int []arr,                                      int size)      {          int []count = new int[size];          int i;            Console.Write("Repeated elements are: ");          for (i = 0; i < size; i++)          {              if (count[arr[i]] == 1)                  Console.Write(arr[i] + " ");              else                  count[arr[i]]++;          }      }        // driver code      public static void Main()      {          int []arr = {4, 2, 4, 5, 2, 3, 1};          int arr\_size = arr.Length;            printRepeating(arr, arr\_size);      }  }    //This code is contributed by Sam007 |

**Output :**

Repeating elements are 4 2

**Time Complexity:** O(n)  
**Auxiliary Space:** O(n)

**Method 3 (Make two equations)**  
Let the numbers which are being repeated are X and Y. We make two equations for X and Y and the simple task left is to solve the two equations.  
We know the sum of integers from 1 to n is n(n+1)/2 and product is n!. We calculate the sum of input array, when this sum is subtracted from n(n+1)/2, we get X + Y because X and Y are the two numbers missing from set [1..n]. Similarly calculate product of input array, when this product is divided from n!, we get X\*Y. Given sum and product of X and Y, we can find easily out X and Y.

Let summation of all numbers in array be S and product be P

X + Y = S – n(n+1)/2  
XY = P/n!

Using above two equations, we can find out X and Y. For array = 4 2 4 5 2 3 1, we get S = 21 and P as 960.

X + Y = 21 – 15 = 6

XY = 960/5! = 8

X – Y = sqrt((X+Y)^2 – 4\*XY) = sqrt(4) = 2

Using below two equations, we easily get X = (6 + 2)/2 and Y = (6-2)/2  
X + Y = 6  
X – Y = 2

Thanks to geek4u for suggesting this method. As pointed by Beginer , there can be addition and multiplication overflow problem with this approach.  
The methods 3 and 4 use all useful information given in the question

|  |
| --- |
| using System;    class GFG  {        static void printRepeating(int []arr, int size)      {          /\* S is for sum of elements in arr[] \*/          int S = 0;            /\* P is for product of elements in arr[] \*/          int P = 1;            /\* x and y are two repeating elements \*/          int x, y;            /\* D is for difference of x and y, i.e., x-y\*/          int D;            int n = size - 2, i;            /\* Calculate Sum and Product           of all elements in arr[] \*/          for (i = 0; i < size; i++)          {              S = S + arr[i];              P = P \* arr[i];          }            /\* S is x + y now \*/          S = S - n \* (n + 1) / 2;            /\* P is x\*y now \*/          P = P / fact(n);            /\* D is x - y now \*/          D = (int) Math.Sqrt(S \* S - 4 \* P);              x = (D + S) / 2;          y = (S - D) / 2;            Console.WriteLine("The two" +                  " repeating elements are :");          Console.Write(x + " " + y);      }        static int fact(int n)      {          return (n == 0) ? 1 : n \* fact(n - 1);      }        // driver code      public static void Main() {          int []arr = {4, 2, 4, 5, 2, 3, 1};          int arr\_size = arr.Length;            printRepeating(arr, arr\_size);      }  }  // This code is contributed by Sam007 |

**Output :**

Repeating elements are 4 2

**Time Complexity:** O(n)  
**Auxiliary Space:** O(1)

**Method 4 (Use XOR)**  
Thanks to neophyte for suggesting this method.  
The approach used here is similar to method 2 of [this post](https://www.geeksforgeeks.org/find-two-non-repeating-elements-in-an-array-of-repeating-elements/).  
Let the repeating numbers be X and Y, if we xor all the elements in the array and all integers from 1 to n, then the result is X xor Y.  
The 1’s in binary representation of X xor Y is corresponding to the different bits between X and Y. Suppose that the kth bit of X xor Y is 1, we can xor all the elements in the array and all integers from 1 to n, whose kth bits are 1. The result will be one of X and Y.

|  |
| --- |
| using System;    class GFG  {      static void printRepeating(int []arr, int size)      {          /\* Will hold xor of all elements \*/          int xor = arr[0];            /\* Will have only single set bit of xor \*/          int set\_bit\_no;            int i;          int n = size - 2;          int x = 0, y = 0;            /\* Get the xor of all elements           in arr[] and {1, 2 .. n} \*/          for (i = 1; i < size; i++)              xor ^= arr[i];            for (i = 1; i <= n; i++)              xor ^= i;            /\* Get the rightmost set bit in set\_bit\_no \*/          set\_bit\_no = (xor & ~(xor - 1));            /\* Now divide elements in two sets by           comparing rightmost set bit of xor with bit           at same position in each element. \*/          for (i = 0; i < size; i++) {              int a = arr[i] & set\_bit\_no;                if (a != 0)                  /\* XOR of first set in arr[] \*/                  x = x ^ arr[i];              else                  /\* XOR of second set in arr[] \*/                  y = y ^ arr[i];          }          for (i = 1; i <= n; i++)          {              int a = i & set\_bit\_no;                if (a != 0)                    /\* XOR of first set in                   arr[] and {1, 2, ...n }\*/                  x = x ^ i;              else                    /\* XOR of second set in                   arr[] and {1, 2, ...n } \*/                  y = y ^ i;          }            Console.WriteLine("The two" +              " reppeated elements are :");          Console.Write(x + " " + y);      }        /\* Driver program to test the above function \*/      public static void Main()      {          int []arr = {4, 2, 4, 5, 2, 3, 1};          int arr\_size = arr.Length;            printRepeating(arr, arr\_size);      }  }    // This code is contributed by Sam007 |

**Output :**

The two repeating elements are 4 2

**Method 5 (Use array elements as index)**  
Thanks to Manish K. Aasawat for suggesting this method.

Traverse the array. Do following for every index i of A[].

{

check for sign of A[abs(A[i])] ;

if positive then

make it negative by A[abs(A[i])]=-A[abs(A[i])];

else // i.e., A[abs(A[i])] is negative

this element (ith element of list) is a repetition

}

Example: A[] = {1, 1, 2, 3, 2}

i=0;

Check sign of A[abs(A[0])] which is A[1]. A[1] is positive, so make it negative.

Array now becomes {1, -1, 2, 3, 2}

i=1;

Check sign of A[abs(A[1])] which is A[1]. A[1] is negative, so A[1] is a repetition.

i=2;

Check sign of A[abs(A[2])] which is A[2]. A[2] is positive, so make it negative. '

Array now becomes {1, -1, -2, 3, 2}

i=3;

Check sign of A[abs(A[3])] which is A[3]. A[3] is positive, so make it negative.

Array now becomes {1, -1, -2, -3, 2}

i=4;

Check sign of A[abs(A[4])] which is A[2]. A[2] is negative, so A[4] is a repetition.

Note that this method modifies the original array and may not be a recommended method if we are not allowed to modify the array.

|  |
| --- |
| // C# code for Find the two repeating  // elements in a given array   using System;    class GFG  {      static void printRepeating(int []arr, int size)      {          int i;          Console.Write("The repeating elements are : ");            for(i = 0; i < size; i++)          {              if(arr[Math.Abs(arr[i])] > 0)                  arr[Math.Abs(arr[i])] = -arr[Math.Abs(arr[i])];              else                  Console.Write(Math.Abs(arr[i]) + " ");          }      }        /\* Driver program to test the above function \*/      public static void Main()      {          int []arr = {4, 2, 4, 5, 2, 3, 1};          int arr\_size = arr.Length;            printRepeating(arr, arr\_size);      }  }    // This code is contributed by Sam007 |

**Output :**

The repeating elements are 4 2

Please write comments if you find the above codes/algorithms incorrect, or find better ways to solve the same problem.

See [find a missing and a repeating number](https://www.geeksforgeeks.org/find-a-repeating-and-a-missing-number/), etc.

# **Find the repeating and the missing | Added 3 new methods**

Given an unsorted array of size n. Array elements are in the range from 1 to n. One number from set {1, 2, …n} is missing and one number occurs twice in the array. Find these two numbers.

**Examples:**

**Input:** arr[] = {3, 1, 3}

**Output:** Missing = 2, Repeating = 3

**Explanation:** In the array,

2 is missing and 3 occurs twice

**Input:** arr[] = {4, 3, 6, 2, 1, 1}

**Output:** Missing = 5, Repeating = 1

**Below are various methods to solve the problems:**

* **Method 1 (Use Sorting)**

**Approach:**

* 1. Sort the input array.
  2. Traverse the array and check for missing and repeating.

**Time Complexity:** O(nLogn)

Thanks to **LoneShadow** for suggesting this method.

* **Method 2 (Use count array)**

**Approach:**

* 1. Create a temp array temp[] of size n with all initial values as 0.
  2. Traverse the input array arr[], and do following for each arr[i]
     + if(temp[arr[i]] == 0) temp[arr[i]] = 1;
     + if(temp[arr[i]] == 1) output “arr[i]” //repeating
  3. Traverse temp[] and output the array element having value as 0 (This is the missing element)

**Time Complexity:** O(n)  
**Auxiliary Space:** O(n)

* **Method 3 (Use elements as Index and mark the visited places)**

**Approach:**  
Traverse the array. While traversing, use the absolute value of every element as an index and make the value at this index as negative to mark it visited. If something is already marked negative then this is the repeating element. To find missing, traverse the array again and look for a positive value.

|  |
| --- |
| // C# program to Find the repeating  // and missing elements    using System;    class GFG {      static void printTwoElements(int[] arr, int size)      {          int i;          Console.Write("The repeating element is ");            for (i = 0; i < size; i++) {              int abs\_val = Math.Abs(arr[i]);              if (arr[abs\_val - 1] > 0)                  arr[abs\_val - 1] = -arr[abs\_val - 1];              else                  Console.WriteLine(abs\_val);          }            Console.Write("And the missing element is ");          for (i = 0; i < size; i++) {              if (arr[i] > 0)                  Console.WriteLine(i + 1);          }      }        // Driver program      public static void Main()      {          int[] arr = { 7, 3, 4, 5, 5, 6, 2 };          int n = arr.Length;          printTwoElements(arr, n);      }  }  // This code is contributed by Sam007 |

**Output:**

The repeating element is 5

and the missing element is 1

**Time Complexity:** O(n)

Thanks to **Manish Mishra** for suggesting this method.

* **Method 4 (Make two equations)**

**Approach:**

* 1. Let x be the missing and y be the repeating element.
  2. Get the sum of all numbers using formula **S = n(n+1)/2 – x + y**
  3. Get product of all numbers using formula **P = 1\*2\*3\*…\*n \* y / x**
  4. The above two steps give us two equations, we can solve the equations and get the values of x and y.

**Time Complexity:** O(n)

Thanks to **disappearedng** for suggesting this solution.

**Note:** This method can cause arithmetic overflow as we calculate product and sum of all array elements.

* **Method 5 (Use XOR)**

**Approach:**

* 1. Let x and y be the desired output elements.
  2. Calculate XOR of all the array elements.

***xor1 = arr[0]^arr[1]^arr[2]…..arr[n-1]***

* 1. XOR the result with all numbers from 1 to n

***xor1 = xor1^1^2^…..^n***

* 1. In the result xor1, all elements would nullify each other except x and y. All the bits that are set in xor1 will be set in either x or y. So if we take any set bit (We have chosen the rightmost set bit in code) of xor1 and divide the elements of the array in two sets – one set of elements with same bit set and other set with same bit not set. By doing so, we will get x in one set and y in another set. Now if we do XOR of all the elements in first set, we will get x, and by doing same in other set we will get y..

Below is the implementation of the above approach:

|  |
| --- |
| // C# program to Find the repeating  // and missing elements    using System;    class GFG {      static int x, y;        static void getTwoElements(int[] arr, int n)      {          /\* Will hold xor of all elements          and numbers from 1 to n \*/          int xor1;            /\* Will have only single set bit of xor1 \*/          int set\_bit\_no;            int i;          x = 0;          y = 0;            xor1 = arr[0];            /\* Get the xor of all array elements \*/          for (i = 1; i < n; i++)              xor1 = xor1 ^ arr[i];            /\* XOR the previous result with numbers from          1 to n\*/          for (i = 1; i <= n; i++)              xor1 = xor1 ^ i;            /\* Get the rightmost set bit in set\_bit\_no \*/          set\_bit\_no = xor1 & ~(xor1 - 1);            /\* Now divide elements in two sets by comparing          rightmost set bit of xor1 with bit at same          position in each element. Also, get XORs of two          sets. The two XORs are the output elements.The          following two for loops serve the purpose \*/          for (i = 0; i < n; i++) {              if ((arr[i] & set\_bit\_no) != 0)                    /\* arr[i] belongs to first set \*/                  x = x ^ arr[i];                else                    /\* arr[i] belongs to second set\*/                  y = y ^ arr[i];          }          for (i = 1; i <= n; i++) {              if ((i & set\_bit\_no) != 0)                    /\* i belongs to first set \*/                  x = x ^ i;                else                    /\* i belongs to second set\*/                  y = y ^ i;          }            /\* \*x and \*y hold the desired output elements \*/      }        // Driver program      public static void Main()      {          int[] arr = { 1, 3, 4, 5, 1, 6, 2 };            int n = arr.Length;          getTwoElements(arr, n);          Console.Write(" The missing element is "                        + x + "and the "                        + "repeating number is "                        + y);      }  }    // This code is contributed by Sam007 |

**Output:**

The missing element is 7 and the repeating number is 5

**Time Complexity:** O(n)

This method doesn’t cause overflow, but it doesn’t tell which one occurs twice and which one is missing. We can add one more step that checks which one is missing and which one is repeating. This can be easily done in O(n) time.

* **Method 6 (Use a Map)**

**Approach:**  
This method involves creating a Hashtable with the help of Map. In this, the elements are mapped to their natural index. In this process, if an element is mapped twice, then it is the repeating element. And if an element’s mapping is not there, then it is the missing element.

Below is the implementation of the above approach:

|  |
| --- |
| // Java program to find the  // repeating and missing elements  // using Maps    import java.util.\*;    public class Test1 {        public static void main(String[] args)      {            int[] arr = { 4, 3, 6, 2, 1, 1 };            Map<Integer, Boolean> numberMap              = new HashMap<>();            int max = arr.length;            for (Integer i : arr) {                if (numberMap.get(i) == null) {                  numberMap.put(i, true);              }              else {                  System.out.println("Repeating = " + i);              }          }          for (int i = 1; i <= max; i++) {              if (numberMap.get(i) == null) {                  System.out.println("Missing = " + i);              }          }      }  } |

**Output:**

Repeating = 1

Missing = 5

* **Method 7 (Make two equations using sum and sum of squares)**

**Approach:**

* 1. Let x be the missing and y be the repeating element.
  2. Let N is the size of array.
  3. Get the sum of all numbers using formula **S = N(N+1)/2**
  4. Get product of all numbers using formula **Sum\_Sq = N(N+1)(2N+1)/6**
  5. Iterate through a loop from i=1….N
  6. **S -= A[i]**
  7. **Sum\_Sq -= (A[i]\*A[i])**
  8. It will give two equations

x-y = S – (1)  
x^2 – y^2 = Sum\_sq  
x+ y = (Sum\_sq/S) – (2)

**Time Complexity:** O(n)

|  |
| --- |
| #include <bits/stdc++.h>    using namespace std;    vector<int>repeatedNumber(const vector<int> &A) {      long long int len = A.size();      long long int Sum\_N = (len \* (len+1) ) /2, Sum\_NSq = (len \* (len +1) \*(2\*len +1) )/6;      long long int missingNumber=0, repeating=0;        for(int i=0;i<A.size(); i++){         Sum\_N -= (long long int)A[i];         Sum\_NSq -= (long long int)A[i]\*(long long int)A[i];      }        missingNumber = (Sum\_N + Sum\_NSq/Sum\_N)/2;      repeating= missingNumber - Sum\_N;      vector <int> ans;      ans.push\_back(repeating);      ans.push\_back(missingNumber);      return ans;    }      int main(void){          std::vector<int> v = {4, 3, 6, 2, 1, 6,7};      vector<int> res = repeatedNumber(v);      for(int x: res){          cout<< x<<"  ";      }      cout<<endl;  } |

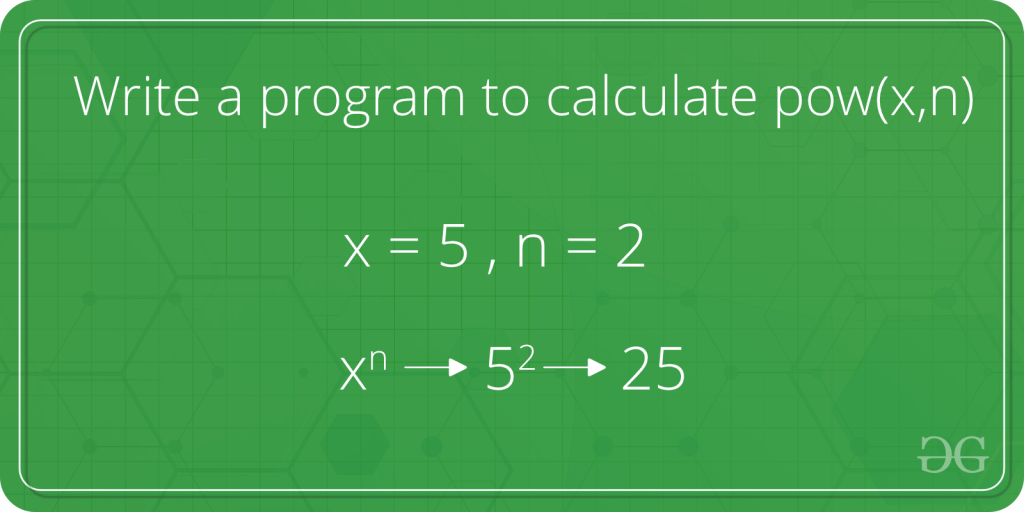
Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

**How to write an efficient method to calculate x raise to the power n?**  
The idea is to use [divide an conquer](https://www.geeksforgeeks.org/divide-and-conquer-set-1-find-closest-pair-of-points/) here to do it in O(Logn) time.

See [Write a C program to calculate pow(x,n)](https://www.geeksforgeeks.org/write-a-c-program-to-calculate-powxn/) for more details.

# **Write a program to calculate pow(x,n)**

Given two integers x and n, write a function to compute xn. We may assume that x and n are small and overflow doesn’t happen.



**Examples :**

Input : x = 2, n = 3

Output : 8

Input : x = 7, n = 2

Output : 49

**Below solution divides the problem into subproblems of size y/2 and call the subproblems recursively.**

|  |
| --- |
| using System;    public class GFG {        // Function to calculate x raised to the power y      static int power(int x, int y)      {          if (y == 0)              return 1;          else if (y % 2 == 0)              return power(x, y / 2) \* power(x, y / 2);          else              return x \* power(x, y / 2) \* power(x, y / 2);      }        // Program to test function power      public static void Main()      {          int x = 2;          int y = 3;            Console.Write(power(x, y));      }  }    // This code is contributed by shiv\_bhakt. |

**Output :**

8

**Time Complexity:**O(n)  
**Space Complexity:** O(1)  
**Algorithmic Paradigm:**Divide and conquer.

Above function can be optimized to O(logn) by calculating power(x, y/2) only once and storing it.

|  |
| --- |
| /\* Function to calculate x raised to the power y in O(logn)\*/  int power(int x, unsigned int y)  {      int temp;      if( y == 0)          return 1;      temp = power(x, y/2);      if (y%2 == 0)          return temp\*temp;      else          return x\*temp\*temp;  } |

**Time Complexity of optimized solution:** O(logn)  
Let us extend the pow function to work for negative y and float x.

|  |
| --- |
| // C# code for extended version of power function  // that can work for float x and negative y    using System;    public class GFG{        static float power(float x, int y)      {          float temp;            if( y == 0)              return 1;          temp = power(x, y/2);            if (y % 2 == 0)              return temp \* temp;          else          {              if(y > 0)                  return x \* temp \* temp;              else                  return (temp \* temp) / x;          }      }        // Program to test function power      public static void Main()      {          float x = 2;          int y = -3;            Console.Write(power(x, y));      }  }    // This code is contributed by shiv\_bhakt. |

**Output :**

0.125000

[**Write an iterative O(Log y) function for pow(x, y)**](https://www.geeksforgeeks.org/write-an-iterative-olog-y-function-for-powx-y/)[**Modular Exponentiation (Power in Modular Arithmetic)**](https://www.geeksforgeeks.org/modular-exponentiation-power-in-modular-arithmetic/)

If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](http://www.contribute.geeksforgeeks.org/) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

**Given an input string and a dictionary of words, find out if the input string can be segmented into a space-separated sequence of dictionary words.**  
The idea is to use Dynamic Programming.

See [Word Break Problem](https://www.geeksforgeeks.org/dynamic-programming-set-32-word-break-problem/) for more details.

**Given a row of n coins of values v1 . . . vn, where n is even. We play a game against an opponent by alternating turns. In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin. Determine the maximum possible amount of money we can definitely win if we move first.**  
This is also a Dynamic Programming Question.

See [Optimal Strategy for a Game](https://www.geeksforgeeks.org/dynamic-programming-set-31-optimal-strategy-for-a-game/)for more details.

***You are given an array of sorted words in an arbitrary language, you need to find order (or precedence of characters) in the language.*** For example if the given arrays is {“baa”, “abcd”, “abca”, “cab”, “cad”}, then order of characters is ‘b’, ‘d’, ‘a’, ‘c’. Note that words are sorted and in the given language “baa” comes before “abcd”, therefore ‘b’ is before ‘a’ in output. Similarly we can find other orders.  
This can be solved using two steps: First create a graph by processing given set of words, then do [topological sorting](https://www.geeksforgeeks.org/topological-sorting/) of the created graph,

See [this](https://www.geeksforgeeks.org/given-sorted-dictionary-find-precedence-characters/)for more details.

# **Given a sorted dictionary of an alien language, find order of characters**

Given a sorted dictionary (array of words) of an alien language, find order of characters in the language.

**Examples:**

Input: words[] = {"baa", "abcd", "abca", "cab", "cad"}

Output: Order of characters is 'b', 'd', 'a', 'c'

Note that words are sorted and in the given language "baa"

comes before "abcd", therefore 'b' is before 'a' in output.

Similarly we can find other orders.

Input: words[] = {"caa", "aaa", "aab"}

Output: Order of characters is 'c', 'a', 'b'

The idea is to create a graph of characters and then find [topological sorting](https://www.geeksforgeeks.org/topological-sorting/) of the created graph. Following are the detailed steps.

1) Create a graph g with number of vertices equal to the size of alphabet in the given alien language. For example, if the alphabet size is 5, then there can be 5 characters in words. Initially there are no edges in graph.

2) Do following for every pair of adjacent words in given sorted array.  
…..a) Let the current pair of words be word1 and word2. One by one compare characters of both words and find the first mismatching characters.  
…..b) Create an edge in g from mismatching character of word1 to that of word2.

3) Print [topological sorting](https://www.geeksforgeeks.org/topological-sorting/) of the above created graph.

Following is the implementation of the above algorithm.

|  |
| --- |
| // A Java program to order of  // characters in an alien language  import java.util.\*;    // Class to represent a graph  class Graph  {        // An array representing the graph as an adjacency list      private final LinkedList<Integer>[] adjacencyList;        Graph(int nVertices)      {          adjacencyList = new LinkedList[nVertices];          for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++)          {              adjacencyList[vertexIndex] = new LinkedList<>();          }      }        // function to add an edge to graph      void addEdge(int startVertex, int endVertex)      {          adjacencyList[startVertex].add(endVertex);      }        private int getNoOfVertices()      {          return adjacencyList.length;      }        // A recursive function used by topologicalSort      private void topologicalSortUtil(int currentVertex, boolean[] visited,                                       Stack<Integer> stack)      {          // Mark the current node as visited.          visited[currentVertex] = true;            // Recur for all the vertices adjacent to this vertex          for (int adjacentVertex : adjacencyList[currentVertex])          {              if (!visited[adjacentVertex])              {                  topologicalSortUtil(adjacentVertex, visited, stack);              }          }            // Push current vertex to stack which stores result          stack.push(currentVertex);      }        // prints a Topological Sort of the complete graph      void topologicalSort()      {          Stack<Integer> stack = new Stack<>();            // Mark all the vertices as not visited          boolean[] visited = new boolean[getNoOfVertices()];          for (int i = 0; i < getNoOfVertices(); i++)          {              visited[i] = false;          }            // Call the recursive helper function to store Topological          // Sort starting from all vertices one by one          for (int i = 0; i < getNoOfVertices(); i++)          {              if (!visited[i])              {                  topologicalSortUtil(i, visited, stack);              }          }            // Print contents of stack          while (!stack.isEmpty())          {              System.out.print((char)('a' + stack.pop()) + " ");          }      }  }    public class OrderOfCharacters  {      // This function fidns and prints order      // of characer from a sorted array of words.      // alpha is number of possible alphabets      // starting from 'a'. For simplicity, this      // function is written in a way that only      // first 'alpha' characters can be there      // in words array. For example if alpha      //  is 7, then words[] should contain words      // having only 'a', 'b','c' 'd', 'e', 'f', 'g'      private static void printOrder(String[] words, int alpha)      {          // Create a graph with 'aplha' edges          Graph graph = new Graph(alpha);            for (int i = 0; i < words.length - 1; i++)          {              // Take the current two words and find the first mismatching              // character              String word1 = words[i];              String word2 = words[i+1];              for (int j = 0; j < Math.min(word1.length(), word2.length()); j++)              {                  // If we find a mismatching character, then add an edge                  // from character of word1 to that of word2                  if (word1.charAt(j) != word2.charAt(j))                  {                      graph.addEdge(word1.charAt(j) - 'a', word2.charAt(j)- 'a');                      break;                  }              }          }            // Print topological sort of the above created graph          graph.topologicalSort();      }        // Driver program to test above functions      public static void main(String[] args)      {          String[] words = {"caa", "aaa", "aab"};          printOrder(words, 3);      }  }    //Contributed by Harikrishnan Rajan |

Output:

c a b

**Time Complexity:**The first step to create a graph takes O(n + alhpa) time where n is number of given words and alpha is number of characters in given alphabet. The second step is also topological sorting. Note that there would be alpha vertices and at-most (n-1) edges in the graph. The time complexity of [topological sorting](https://www.geeksforgeeks.org/topological-sorting/) is O(V+E) which is O(n + aplha) here. So overall time complexity is O(n + aplha) + O(n + aplha) which is O(n + aplha).

**Exercise:**  
The above code doesn’t work when the input is not valid. For example {“aba”, “bba”, “aaa”} is not valid, because from first two words, we can deduce ‘a’ should appear before ‘b’, but from last two words, we can deduce ‘b’ should appear before ‘a’ which is not possible. Extend the above program to handle invalid inputs and generate the output as “Not valid”.

This article is contributed by **Piyush Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

# **Find Next Sparse Number**

A number is Sparse if there are no two adjacent 1s in its binary representation. For example 5 (binary representation: 101) is sparse, but 6 (binary representation: 110) is not sparse.  
Given a number x, find the smallest Sparse number which greater than or equal to x.

**Examples:**

Input: x = 6

Output: Next Sparse Number is 8

Input: x = 4

Output: Next Sparse Number is 4

Input: x = 38

Output: Next Sparse Number is 40

Input: x = 44

Output: Next Sparse Number is 64

[We strongly recommend that you click here and practice it, before moving on to the solution.](https://practice.geeksforgeeks.org/problem-page.php?pid=391" \t "_blank)

A **Simple Solution**is to do following:

1) Write a utility function isSparse(x) that takes a number

and returns true if x is sparse, else false. This function

can be easily written by traversing the bits of input number.

2) Start from x and do following

while(1)

{

if (isSparse(x))

return x;

else

x++

}

Time complexity of isSparse() is O(Log x). Time complexity of this solution is O(x Log x). The next sparse number can be at most O(x) distance away.

Thanks to **kk\_angel** for suggesting above solution.

An **Efficient Solution** can solve this problem without checking all numbers on by one. Below are steps.

1) Find binary of the given number and store it in a

boolean array.

2) Initialize last\_finalized bit position as 0.

2) Start traversing the binary from least significant bit.

a) If we get two adjacent 1's such that next (or third)

bit is not 1, then

(i) Make all bits after this 1 to last finalized

bit (including last finalized) as 0.

(ii) Update last finalized bit as next bit.

For example, let binary representation be 010100010**11**101, we change it to 01010001**100000** (all bits after highlighted 11 are set to 0). Again two 1’s are adjacent, so change binary representation to 010100**10000000**. This is our final answer.

Below is the implementation of above solution.

|  |
| --- |
| // C# program to find next sparse number  using System;  using System.Collections;      class GFG{  static int nextSparse(int x)  {      // Find binary representation of x and store it in bin.get(].      // bin.get(0] contains least significant bit (LSB), next      // bit is in bin.get(1], and so on.      ArrayList bin = new ArrayList();      while (x != 0)      {          bin.Add(x&1);          x >>= 1;      }        // There my be extra bit in result, so add one extra bit      bin.Add(0);      int n = bin.Count; // Size of binary representation        // The position till which all bits are finalized      int last\_final = 0;        // Start from second bit (next to LSB)      for (int i = 1; i < n-1; i++)      {      // If current bit and its previous bit are 1, but next      // bit is not 1.      if ((int)bin[i] == 1 && (int)bin[i-1] == 1 && (int)bin[i+1] != 1)      {              // Make the next bit 1              bin[i+1]=1;                // Make all bits before current bit as 0 to make              // sure that we get the smallest next number              for (int j = i; j >= last\_final; j--)                  bin[j]=0;                // Store position of the bit set so that this bit              // and bits before it are not changed next time.              last\_final = i + 1;          }      }        // Find decimal equivalent of modified bin.get(]      int ans = 0;      for (int i = 0; i < n; i++)          ans += (int)bin[i]\*(1<<i);      return ans;  }    // Driver program  static void Main()  {      int x = 38;      Console.WriteLine("Next Sparse Number is "+nextSparse(x));  }  }  // This code is contributed by mits |

**Output:**

Next Sparse Number is 40

Time complexity of this solution is O(Log x).