

Contents

1 A Better Way To Approach Competitive Programming 13	Source
 15
2 AKS Primality Test 16	Source 24
3 Algorithm Library C++ Magicians STL Algorithm 25	Source
 31
4 Array algorithms in C++ STL (all_of, any_of, none_of, copy_n and iota) 32	Source
 35
5 Array range queries over range queries 36	Source
 45
6 Array with GCD of any of its subset belongs to the given array 46	Source
 52
7 BFS using STL for competitive coding 53	Source
 56
8 Bit Tricks for Competitive Programming 57	Source
 60
9 Bits manipulation (Important tactics) 61	Source
 64
10 Bitwise Hacks for Competitive Programming 65	Source
 70
11 Burst Balloon to maximize coins 71	Source
 72
12 C qsort() vs C++ sort() 73	Source
 76
13 C++ tricks for competitive programming (for C++ 11) 77	Source
 80

14 C++: Methods of code shortening in competitive programming 81	Source
 85
15 Check if a M-th fibonacci number divides N-th fibonacci number 86	Source

.....	90
16 Check if any permutation of a large number is divisible by 8 91	Source 99
17 Check if concatenation of two strings is balanced or not 100	Source 107
18 Check if it is possible to convert one string into another with given constraints 108	Source 110
19 Check if the large number formed is divisible by 41 or not 111	Source 118
20 Check if the last element of array is even or odd after performing a operation p times 119	Source 124
21 Cin-Cout vs Scanf-Printf 125	Source 128
22 Cleaning the room 129	Source 132
23 Combinatorics on ordered trees 133	Source 143
24 Common mistakes to be avoided in Competitive Programming in C++ Beginners 144	Source 151
25 Competitive Programming: Conquering a given problem 152	Source 154
26 Container with Most Water 155	Source 160
27 Count inversions of size k in a given array 161	Source 167
28 Count number of primes in an array 168	Source 170
29 Count number of right triangles possible with a given perimeter 171	Source 173
<div style="text-align: right;">2</div> <div style="text-align: right;"><i>Contents</i></div>	
30 Count strings with consonants and vowels at alternate position 174	Source 176
31 Count unique subsequences of length K 177	Source 180

32 Data Type Ranges and their macros in C++ 181	Source	183
33 Difference between the summation of numbers whose frequency of all digits are same and different 184	Source	186
34 Digit DP Introduction 187	Source	192
35 Distinct Prime Factors of Array Product 193	Source	196
36 Dynamic Disjoint Set Data Structure for large range values 197	Source	202
37 Dynamic Programming Wildcard Pattern Matching Linear Time and Constant Space 203	Source	209
38 Element which occurs consecutively in a given subarray more than or equal to K times 210	Source	213
39 Euler tour of Binary Tree 214	Source	217
40 Extended Mo's Algorithm with ff O(1) time complexity 218	Source	229
41 Fast I/O for Competitive Programming 230	Source	233
42 Fast I/O in Java in Competitive Programming 234	Source	241
43 Find $(a^b) \% m$ where 'b' is very large 242	Source	248
44 Find Nth term (A matrix exponentiation example) 249	Source	252
45 Find if it is possible to reach the end through given transitions 253	Source	255

46 Find if neat arrangement of cups and shelves can be made 256	Source	263
47 Find the Largest Cube formed by Deleting minimum Digits from a number 264	Source	267
48 Find the arrangement of queue at given time 268	Source	

.....	273
49 Find the minimum time after which one can exchange notes 274	Source 280
50 Find the number of operations required to make all array elements Equal281	Source 282
51 Find two numbers from their sum and XOR 283	Source 288
52 First occurrence of a digit in a given fraction 289	Source 294
53 Formatted output in Java 295	Source ... 298
54 Frequency Measuring Techniques for Competitive Programming 299	Source 306
55 Generating Test Cases (generate() and generate_n() in C++) 307	Source 309
56 Graph implementation using STL for competitive programming Set 1 (DFS of Unweighted and Undirected) 310	Source 312
57 Graph implementation using STL for competitive programming Set 2 (Weighted graph) 313	Source 315
58 How can competitive programming help you get a job? 316	Source 318
59 How to become a master in competitive programming? 319	Source 320
60 How to begin with Competitive Programming? 321	Source 327
61 How to get rid of Java TLE problem 328	Source 333

62 How to overcome Time Limit Exceed(TLE)? 334	Source 335
63 How to prepare for ACM – ICPC? 336	Source 341
64 How to prepare for Facebook Hacker Cup? 342	Source

.....	346
65 How to prepare for Google Asia Pacific University (APAC) Test ?	347 Source
.....	350
66 How to read Competitive Programming Questions?	351 Source
.....	354
67 How to read content of GeeksforGeeks in an organized way?	355 Source
.....	357
68 Inclusion Exclusion principle and programming applications	358 Source
.....	365
69 Input/Output from external file in C/C++, Java and Python for Com petitive Programming	366 Source
.....	368
70 Input/Output from external file in C/C++, Java and Python for Com petitive Programming Set 2	369 Source
.....	371
71 Introduction to Programming Languages	372 Source
.....	374
72 Java tricks for competitive programming (for Java 8)	375 Source
.....	379
73 Knowing the complexity in competitive programming	380 Source
.....	381
74 LCA for general or n-ary trees (Sparse Matrix DP approach < $O(n \log n)$, $O(\log n)$>)	382 Source
.....	388
75 Largest connected component on a grid	389 Source
.....	399
76 Largest number with one swap allowed	400 Source
.....	405
77 Longest substring having K distinct vowels	406 Source
.....	411

78 Making elements of two arrays same with minimum increment/decrement	412
Source	416
79 Matrix Exponentiation	417 Source
.....	425
80 Maximize the bitwise OR of an array	426 Source
.....	432

81 Maximize the number of segments of length p, q and r 433	Source	438
82 Maximize the total profit of all the persons 439	Source	441
83 Maximum Possible Product in Array after performing given Operations 442	Source	445
84 Maximum difference between groups of size two 446	Source	448
85 Maximum elements that can be made equal with k updates 449	Source	456
86 Maximum number of customers that can be satisfied with given quantity457	Source	460
87 Maximum sum increasing subsequence from a prefix and a given element after prefix is must 461	Source	466
88 Minimum adjacent swaps to move maximum and minimum to corners 467	Source	473
89 Minimum cost path from source node to destination node via an inter mediate node 474	Source	478
90 Minimum difference between groups of size two 479	Source	481
91 Minimum digits to remove to make a number Perfect Square 482	Source	491
92 Minimum number of elements to be removed to make XOR maximum 492	Source	494
93 Minimum number using set bits of a given number 495	Source	498

94 Minimum operations required to make all the elements distinct in an array 499	Source	500
95 Minimum steps to reach end from start by performing multiplication and mod operations with array elements 501	Source	503
96 Minimum total cost incurred to reach the last station 504	Source	

.....	511
97 Modulo 10^9+7 (1000000007) 512 Source	515
98 Modulo power for large numbers represented as strings 516 Source	518
99 Multi Source Shortest Path in Unweighted Graph 519 Source	526
100 Multistage Graph (Shortest Path) 527 Source	530
101 NZEC error in Python 531 Source	533
102 Nth non-Square number 534 Source	537
103 Number of Larger Elements on right side in a string 538 Source	541
104 Number of Transpositions in a Permutation 542 Source	548
105 Number of days until all chocolates become unhealthy 549 Source	555
106 Number of digits in N factorial to the power N 556 Source	559
107 Number of digits in the nth number made of given four digits 560 Source	563
108 Number of distinct prime factors of first n natural numbers 564 Source	572
109 Number of elements greater than K in the range L to R using Fenwick Tree (Offline queries) 573 Source	578

110 Number of horizontal or vertical line segments to connect 3 points 579 Source	586
111 Number of integral solutions for equation $x = b * (\text{sum of digits}(x)^a) + c$ 587 Source	591
112 Number of odd and even results for every value of x in range [min, max] after performing N steps 592 Source	603

113 Number of prime pairs in an array 604	Source	606
114 Number of quadrilaterals possible from the given points 607	Source	615
115 Number of terms in Geometric Series with given conditions 616	Source	619
116 Number of ways to change the XOR of two numbers by swapping the bits 620	Source	627
117 Ordered Set and GNU C++ PBDS 628	Source	631
118 Pair of arrays with equal sum after removing exactly one element from each 632	Source	635
119 Pair with minimum absolute difference after solving each query 636	Source	640
120 Pairs involved in Balanced Parentheses 641	Source	645
121 Pairs with GCD equal to one in the given range 646	Source	649
122 Palindromic Tree Introduction & Implementation 650	Source	663
123 Passing the Assignment 664	Source	671
124 Permutation of a string with maximum number of characters greater than its adjacent characters 672	Source	675
125 Possible timings 676		8

Contents

	Source	680
126 Practice for cracking any coding interview 681	Source	693
127 Prefix Sum Array – Implementation and Applications in Competitive Programming 694	Source	700
128 Print all subsequences of a string Iterative Method 701	Source	706
129 Program to find last two digits of 2^n 707	Source	

.....	722
130 Project Euler 723	Source 728
131 Python Input Methods for Competitive Programming 729	Source 732
132 Python Tricks for Competitive Coding 733	Source 735
133 Python in Competitive Programming 736	Source 738
134 Queries to find distance between two nodes of a Binary tree 739	Source 741
135 Queries to find distance between two nodes of a Binary tree – O(logn) method 742	Source 750
136 Queries to find maximum product pair in range with updates 751	Source 755
137 Querying the number of distinct colors in a subtree of a colored tree using BIT 756	Source 762
138 Quick ways to check for Prime and find next Prime in Java 763	Source 765
139 Range Minimum Query (Square Root Decomposition and Sparse Table)766	Source 774
140 Remove the forbidden strings 775	Source 782
141 Represent a number as sum of minimum possible psuedobinary numbers783	9

Contents

Source	789
142 Searching in a map using std::map functions in C++ 790	Source 796
143 Sequence Alignment problem 797	Source 808
144 Smallest number with sum of digits as N and divisible by 10^N 809	Source 814
145 Some important shortcuts in Competitive Programming 815	Source 816
146 Sqrt (or Square Root) Decomposition Technique Set 1 (Introduction) 817	Source 823

147 Sqrt (or Square Root) Decomposition Set 2 (LCA of Tree in $O(\sqrt{\text{height}})$ time) 824 Source	834
148 String transformation using XOR and OR 835 Source	842
149 Sum of all prime divisors of all the numbers in range L-R 843 Source	850
150 Sum of decimal equivalent of all possible pairs of Binary representation of a Number 851 Source	854
151 Sum of elements in range L-R where first half and second half is filled with odd and even numbers 855 Source	866
152 Sum of elements of all partitions of number such that no element is less than K 867 Source	872
153 Sum of $f(a[i], a[j])$ over all pairs in an array of n integers 873 Source	876
154 Sum of range in a series of first odd then even natural numbers 877 Source	883
155 Test Case Generation Set 1 (Random Numbers, Arrays and Matrices) 884 Source	888
156 Test Case Generation Set 2 (Random Characters, Strings and Arrays of Random Strings) 889	

10
Contents

Source	892
157 Test Case Generation Set 3 (Unweighted and Weighted Trees) 893 Source	900
158 Test Case Generation Set 4 (Random directed / undirected weighted and unweighted Graphs) 901 Source	909
159 Test Case Generation Set 5 (Generating random Sorted Arrays and Palindromes) 910 Source	913
160 The Google Foo Bar Challenge 914 161 My Experience with the Google Foo Bar Challenge 915	
162 The Challenge 918 Source	920
163 The painter's partition problem Set 2 921 Source	930

164 Tiling with Dominoes 931	Source	937
165 Tips and Tricks for Competitive Programmers Set 1 (For Beginners) 938	Source	940
166 Tips and Tricks for Competitive Programmers Set 2 (Language to be used for Competitive Programming) 941	Source	944
167 Top 10 Algorithms and Data Structures for Competitive Programming 945	Source	949
168 Traversal of tree with k jumps allowed between nodes of same height 950	Source	955
169 Trick for modular division ($(x_1 * x_2 \dots x_n) / b \bmod m$) 956	Source	960
170 Triplet with no element divisible by 3 and sum N 961	Source	968
171 Two Dimensional Segment Tree Sub-Matrix Sum 969	Source	976
172 Using Chinese Remainder Theorem to Combine Modular equations 977	Source	980
173 Vantieghems Theorem for Primality Test 981	Source	982
174 Variation in Nim Game 983	Source	990
175 Water Connection Problem 991	Source	997
176 Water drop problem 998	Source	1000
177 What coding habits improve timing in coding contest? 1001	Source	1002
178 What to do at the time of Wrong Answer (WA)? 1003	Source	1004
179 Why is programming important for first year or school students? 1005	Source	1007

180 Why is python best suited for Competitive Coding? 1008	Source
1012
181 Writing C/C++ code efficiently in Competitive programming 1013	Source
1018
182 Writing code faster in C++ STL 1019	Source
1020
183 getchar_unlocked() – faster input in C/C++ for Competitive Pro gramming 1021	
	Source1022

Chapter 1

A Better Way To Approach Competitive Programming

A Better Way To Approach Competitive Programming - GeeksforGeeks

[This](#) article helps to all those who want to begin with Competitive Programming. The only prerequisite one need is the knowledge of a programming language.

Now, Let us find a better approach to Competitive Programming. Please note:

1. One should read the proper Input and Output format because most of the beginners make mistakes of having extra print statements in the output. So please be careful to the output format. **Example** – “Please Enter Next Number :” and “Output is : .

2. Analyze the problem constraints before writing code because most of the time you have written the code that is brute force and will not run in the given time limit constraint.

Following are the some of the issues that you might face if you are a beginner.:

1. **Time Limit** : It is given in seconds. It gives you the insight about the order of the correct solution. Consider following situation. Time Limit = 1 Sec
Let us Assume 1 sec approximately can perform 10^8 operations.
If you write a program that is of order $O(N^2)$ and the problem has T test cases.
Then total order of your program becomes $O(T*N^2)$.
If $T \leq 1000$ and $N \leq 1000$, then (ignoring hidden constants in asymptotic notations) your code may not be accepted in worst case as $1000*1000*1000$ is 10^9 operations which means 10 seconds.
To avoid TLE, always think of the worst test cases that are possible for the problem and analyze your code in that situation.
2. **Run Time Error** : It is the one of the most encountered problem by the beginners.
The main reason could be :

13

Chapter 1. A Better Way To Approach Competitive Programming

- (a) Segmentation Fault : It is the illegal accessing of memory address. **e.g** `int[] array = new int[100]; System.out.println(array[101]);`
 - (b) Declaration of an array of more than 10^8 .
 - (c) Divide & Taking Modulus with 0 .
 - (d) You should know how to use GDB that will help you to correct your run time error.
3. **Compilation Error** : It is one of the errors that is frequently faced when programming in C/C++.
 4. **Wrong Answer** : Whenever you encounter WA, write a brute force code & make sure that it is perfect. Now generate test cases using random function in C++. Run your code on these test cases and match the output. Now think of the corner cases that will help you to find the problem in your algorithm.
 5. **IntOverflow** : Many times unknowingly you will exceed the maximum value that can be stored in primitive type int. e.g. Constraints : $0 < \text{num1}, \text{num2} \leq 10^9$

```
#include<iostream.h>
int main()
{
    int num1, num2;
    cin >> num1 >> num2;
    int answer = num1 + num2;

    // error if num1 and num2 are large
    // numbers
    cout<< answer;
```

```

    return 0;
}

```

The above program won't run correct always because ($2 \cdot 10^9$) it may exceed the maximum value that can be stored in INTS. i.e 10^9 . So you will have to use long longint or unsigned int primitive data type for answer in such a case.

6. Comparing Doubles :

```

int main()
{
    float a ;
    cin << a;
    if (a == 10)
        cout << "YES";
}

```

float and double data types don't have infinite precision . **Beware** (6/15 digit precision for them respectively). So always use a margin of (~ 0.0000001) in comparing. Example –

14

Chapter 1. A Better Way To Approach Competitive Programming

```

if (abs(a - 10) < (0.0000001))
{
    cout << "YES";
}

```

Other Useful Points :

Sometimes when you are stuck. Check running time of other accepted code and analyze about the order of solution is required and the amount of memory that is allowed.

1. 4 MB ~ integer array of size 10^6 (assuming int takes 4 bytes) or 2-d array of size $10^3 \cdot 10^3$

Standard Memory limits in most of the problem are of order of 256MB.

If you have to allocate large array, then it is NOT a good idea to do allocation inside a function as memory is allocated and released for every test case, and memory is allocated on function call stack (stack size is limited at many places). Thus if you have to make an array of large size, make it global.

I hope this article was of help to you and hope that you will now be able to attempt questions being better prepared and thereby perform better, quicker. ff

Source

Chapter 2

AKS Primality Test

AKS Primality Test - GeeksforGeeks

There are several primality test available to check whether the number is prime or not like **Fermat's Theorem**, **Miller-Rabin** Primality test and alot more. But problem with all of them is they all are probabilistic in nature. So, here comes one another method i.e **AKS primality test (Agrawal–Kayal–Saxena primality test)** and it is **deterministically correct** for any general number.

Features of AKS primality test :

1. The AKS algorithm can be used to verify the primality of any general number given.
2. The maximum running time of the algorithm can be expressed as a polynomial over the number of digits in the target number.
3. The algorithm is guaranteed to distinguish deterministically whether the target number is prime or composite.
4. The correctness of AKS is not conditional on any subsidiary unproven hypothesis.

The AKS primality test is based upon the following theorem: An integer n greater than 2 is prime if and only if the polynomial congruence relation

$$(x + a)^n \equiv (x^n + a) \pmod{n}$$

holds for some **a coprime to n**. Here x is just a formal symbol .

The AKS test evaluates the equality by making complexity dependent on the size of r .

This is expressed as

$$(x + a)^n \equiv (x^n + a) \pmod{(x^n - 1) \cdot (x^n + 1) \cdots (x^n + a_{r-1})}$$

which can be expressed in simpler term as

$$(x + a)^n \equiv (x^n + a) \pmod{(x^r - 1) \cdot f(x) \cdot g(x)}$$

for some polynomials f and g .

This congruence can be checked in polynomial time when r is polynomial to the digits of n. The AKS algorithm evaluates this congruence for a large set of a values, whose size is polynomial to the digits of n. The proof of validity of the AKS algorithm shows that one can find r and a set of a values with the above properties such that if the congruences hold

then n is a power of a prime. The brute force approach would require the expansion of the $(x - a)^n$ polynomial and a reduction (mod n) of the resulting n + 1 coefficients .

As a should be co-prime to n. So, to implement this algorithm we can check by taking a = 1, but for large values of n we should take large values of a.

The algorithm is based on the condition that if n is any number, then it is prime if, $(x - 1)^n - (x^n - 1)$ is divisible by n.

Checking for n = 3 :

$$\begin{aligned} (x-1)^3 &= (x^3 - 1) \\ &= (x^3 - 3x^2 + 3x - 1) - (x^3 - 1) \\ &= -3x^2 + 3x \end{aligned}$$

As all the coefficients are divisible by n i.e. 3, so 3 (n) is prime. As the number increases, size increases.

The code here is based on this condition and can check primes till 64

. Below is the implementation of above approach:

C++

```
// C++ code to check if number is prime. This
// program demonstrates concept behind AKS
// algorithm and doesn't implement the actual
// algorithm (This works only till n = 64)
#include <bits/stdc++.h>
using namespace std;

// array used to store coefficients .
long long c[100];

// function to calculate the coefficients
```



```

// of  $(x - 1)^n - (x^n - 1)$  with the help
// of Pascal's triangle .
void coef(int n)
{
    c[0] = 1;
    for (int i = 0; i < n; c[0] = -c[0], i++) {
        c[1 + i] = 1;

        for (int j = i; j > 0; j--)
            c[j] = c[j - 1] - c[j];
    }
}

// function to check whether
// the number is prime or not
bool isPrime(int n)

```

```

{
    // Calculating all the coefficients by
    // the function coef and storing all
    // the coefficients in c array .
    coef(n);

    // subtracting c[n] and adding c[0] by 1
    // as  $(x - 1)^n - (x^n - 1)$ , here we
    // are subtracting c[n] by 1 and adding
    // 1 in expression.
    c[0]++, c[n]--;

    // checking all the coefficients whether
    // they are divisible by n or not.
    // if n is not prime, then loop breaks
    // and (i > 0).
    int i = n;
    while (i-- && c[i] % n == 0)
        ;

    // Return true if all coefficients are
    // divisible by n.
    return i < 0;
}

// driver program
int main()
{
    int n = 37;
    if (isPrime(n))
        cout << "Prime" << endl;
}

```

```

else
    cout << "Not Prime" << endl;
return 0;
}

```

Java

// Java code to check if number is prime. This
 // program demonstrates concept behind AKS
 // algorithm and doesn't implement the actual
 // algorithm (This works only till $n = 64$)

```

class GFG {
    // array used to store coefficients .
    static long c[] = new long[100];

    // function to calculate the coefficients

```

```

    // of  $(x - 1)^n - (x^n - 1)$  with the help
    // of Pascal's triangle .
    static void coef(int n)
    {
        c[0] = 1;
        for (int i = 0; i < n; c[0] = -c[0], i++) {
            c[1 + i] = 1;

            for (int j = i; j > 0; j--)
                c[j] = c[j - 1] - c[j];
        }
    }
}

```

```

    // function to check whether
    // the number is prime or not
    static boolean isPrime(int n)
    {
        // Calculating all the coefficients by
        // the function coef and storing all
        // the coefficients in c array .
        coef(n);

        // subtracting c[n] and adding c[0] by 1
        // as  $(x - 1)^n - (x^n - 1)$ , here we
        // are subtracting c[n] by 1 and adding
        // 1 in expression.
        c[0]++;
        c[n]--;

        // checking all the coefficients whether

```

```

// they are divisible by n or not.
// if n is not prime, then loop breaks
// and (i > 0).
int i = n;
while ((i-- > 0) && c[i] % n == 0)
    ;

// Return true if all coefficients are
// divisible by n.
return i < 0;
}
// Driver code
public static void main(String[] args)
{
    int n = 37;
    if (isPrime(n))
        System.out.println("Prime");
    else

```

```

        System.out.println("Not Prime");
    }
}

```

// This code is contributed by Anant Agarwal.

Python3

```

# Python3 code to check if
# number is prime. This
# program demonstrates concept
# behind AKS algorithm and
# doesn't implement the actual
# algorithm (This works only
# till n = 64)

```

```

# array used to
# store coefficients .
c = [0] * 100;

```

```

# function to calculate the
# coefficients of (x - 1)^n -
# (x^n - 1) with the help
# of Pascal's triangle .
def coef(n):
    c[0] = 1;
    for i in range(n):
        c[1 + i] = 1;
        for j in range(i, 0, -1):

```

```

    c[j] = c[j - 1] - c[j];
    c[0] = -c[0];

```

```

# function to check whether
# the number is prime or not
def isPrime(n):

```

```

    # Calculating all the coefficients
    # by the function coef and storing
    # all the coefficients in c array .
    coef(n);

```

```

    # subtracting c[n] and adding
    # c[0] by 1 as ( x - 1 )^n -
    # ( x^n - 1), here we are
    # subtracting c[n] by 1 and
    # adding 1 in expression.
    c[0] = c[0] + 1;
    c[n] = c[n] - 1;

```

```

    # checking all the coefficients
    # whether they are divisible by
    # n or not. if n is not prime,
    # then loop breaks and (i > 0).
    i = n;
    while (i > -1 and c[i] % n == 0):
        i = i - 1;

```

```

    # Return true if all coefficients
    # are divisible by n.
    return True if i < 0 else False;

```

```

# Driver Code
n = 37;
if (isPrime(n)):
    print("Prime");
else:
    print("Not Prime");

```

```

# This code is contributed by mits

```

C#

```

// C# code to check if number is prime. This
// program demonstrates concept behind AKS
// algorithm and doesn't implement the actual
// algorithm (This works only till n = 64)

```

using System;

class GFG {

// array used to store coefficients .
static long []c = new long[100];

// function to calculate the coefficients
// of $(x - 1)^n - (x^n - 1)$ with the help
// of Pascal's triangle .
static void coef(int n)

{
 c[0] = 1;

 for (int i = 0; i < n; c[0] = -c[0], i++)
 {
 c[1 + i] = 1;

 for (int j = i; j > 0; j--)
 c[j] = c[j - 1] - c[j];
 }

 }
}

// function to check whether
// the number is prime or not
static bool isPrime(int n)
{

 // Calculating all the coefficients by
 // the function coef and storing all
 // the coefficients in c array .
 coef(n);

 // subtracting c[n] and adding c[0] by 1
 // as $(x - 1)^n - (x^n - 1)$, here we
 // are subtracting c[n] by 1 and adding
 // 1 in expression.
 c[0]++;
 c[n]--;

 // checking all the coefficients whether
 // they are divisible by n or not.
 // if n is not prime, then loop breaks
 // and (i > 0).
 int i = n;
 while ((i-- > 0 && c[i] % n == 0)
 ;

```

        // Return true if all coefficients are
        // divisible by n.
        return i < 0;
    }

    // Driver code
    public static void Main()
    {
        int n = 37;
        if (isPrime(n))
            Console.WriteLine("Prime");
        else
            Console.WriteLine("Not Prime");
    }
}

// This code is contributed by anuj_67.

```

PHP

```

<?php
// PHP code to check if number
// is prime. This program
// demonstrates concept behind
// AKS algorithm and doesn't
// implement the actual
// algorithm (This works only
// till n = 64)

// array used to
// store coefficients .
global $c;

// function to calculate
// the coefficients
// of  $(x - 1)^n$  -
//  $(x^n - 1)$  with the help
// of Pascal's triangle .
function coef($n)
{
    $c[0] = 1;
    for ($i = 0; $i < $n; $c[0] = -$c[0], $i++)
    {
        $c[1 + $i] = 1;

        for ($j = $i; $j > 0; $j--)

```

```

        $c[$j] = $c[$j - 1] - $c[$j];
    }
}

// function to check whether
// the number is prime or not
function isPrime($n)
{
    global $c;

    // Calculating all the
    // coefficients by the
    // function coef and
    // storing all the
    // coefficients in c array .
    coef($n);

    // subtracting c[n] and
    // adding c[0] by 1 as
    //  $(x - 1)^n - (x^n - 1)$ ,
    // here we are subtracting c[n]
    // by 1 and adding 1 in expression.

```

```

// $c[0]++; $c[$n]--;

// checking all the coefficients whether
// they are divisible by n or not.
// if n is not prime, then loop breaks
// and (i > 0).
$i = $n;
while ($i-- && $c[$i] % $n == 0)

// Return true if all
// coefficients are
// divisible by n.
return $i < 0;
}

// Driver Code
$n = 37;
if (isPrime($n))
    echo "Not Prime", "\n";
else
    echo "Prime", "\n";

// This code is contributed by aj_36
?>

```

Output:

Prime

References:

https://en.wikipedia.org/wiki/AKS_primality_test
https://rosettacode.org/wiki/AKS_test_for_primes#C
<https://www.youtube.com/watch?v=HvMSRWTE2ml>

Improved By : jit_t, vt_m, Mithun Kumar

Source

<https://www.geeksforgeeks.org/aks-primality-test/>

24

Chapter 3

Algorithm Library | C++ Magicians STL Algorithm

Algorithm Library | C++ Magicians STL Algorithm - GeeksforGeeks

For all those who aspire to excel in competitive programming, only having a knowledge about containers of STL is of less use till one is not aware what all STL has to offer. STL has an ocean of algorithms, for all < algorithm > library functions : Refer [here](#).

Some of the most used algorithms on vectors and most useful one's in Competitive Program ming are mentioned as follows :

Non-Manipulating Algorithms

1. [sort\(first_iterator, last_iterator\)](#) – To sort the given vector.
2. [reverse\(first_iterator, last_iterator\)](#) – To reverse a vector.

3. ***max_element (first_iterator, last_iterator)** – To find the maximum element of a vector.
4. ***min_element (first_iterator, last_iterator)** – To find the minimum element of a vector.
5. **accumulate(first_iterator, last_iterator, initial value of sum)** – Does the summation of vector elements

```
// A C++ program to demonstrate working of sort(),
// reverse()
#include <algorithm>
#include <iostream>
#include <vector>
#include <numeric> //For accumulate operation
using namespace std;
```

```
int main()
{
    // Initializing vector with array values
    int arr[] = {10, 20, 5, 23, 42, 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Vector is: ";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    // Sorting the Vector in Ascending order
    sort(vect.begin(), vect.end());

    cout << "\nVector after sorting is: ";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    // Reversing the Vector
    reverse(vect.begin(), vect.end());

    cout << "\nVector after reversing is: ";
    for (int i=0; i<6; i++)
        cout << vect[i] << " ";

    cout << "\nMaximum element of vector is: ";
    cout << *max_element(vect.begin(), vect.end());

    cout << "\nMinimum element of vector is: ";
    cout << *min_element(vect.begin(), vect.end());
```

```

// Starting the summation from 0
cout << "\nThe summation of vector elements is: ";
cout << accumulate(vect.begin(), vect.end(), 0);

return 0;
}

```

Output:

```

Vector before sorting is: 10 20 5 23 42 15
Vector after sorting is: 5 10 15 20 23 42
Vector before reversing is: 5 10 15 20 23 42
Vector after reversing is: 42 23 20 15 10 5
Maximum element of vector is: 42
Minimum element of vector is: 5
The summation of vector elements is: 115

```

6. **count(first_iterator, last_iterator,x)** – To count the occurrences of x in vector. 26

Chapter 3. Algorithm Library | C++ Magicians STL Algorithm

7. **find(first_iterator, last_iterator, x)** – Points to last address of vector ((name_of_vector).end()) if element is not present in vector.

```

// C++ program to demonstrate working of count()
// and find()
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {10, 20, 5, 23 ,42, 20, 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Occurrences of 20 in vector : ";

    // Counts the occurrences of 20 from 1st to
    // last element
    cout << count(vect.begin(), vect.end(), 20);

    // find() returns iterator to last address if
    // element not present
    find(vect.begin(), vect.end(),5) != vect.end()?
        cout << "\nElement found":
        cout << "\nElement not found";

    return 0;
}

```

```
}
```

Output:

Occurrences of 20 in vector: 2
Element found

8. **binary_search**(first_iterator, last_iterator, x) – Tests whether x exists in sorted vector or not.
9. **lower_bound**(first_iterator, last_iterator, x) – returns an iterator pointing to the first element in the range [first,last) which has a value not less than 'x'.
10. **upper_bound**(first_iterator, last_iterator, x) – returns an iterator pointing to the first element in the range [first,last) which has a value greater than 'x'.

```
// C++ program to demonstrate working of lower_bound()
// and upper_bound().
```

27

Chapter 3. Algorithm Library | C++ Magicians STL Algorithm

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    // Sort the array to make sure that lower_bound()
    // and upper_bound() work.
    sort(vect.begin(), vect.end());

    // Returns the first occurrence of 20
    auto q = lower_bound(vect.begin(), vect.end(), 20);

    // Returns the last occurrence of 20
    auto p = upper_bound(vect.begin(), vect.end(), 20);

    cout << "The lower bound is at position: ";
    cout << q-vect.begin() << endl;

    cout << "The upper bound is at position: ";
    cout << p-vect.begin() << endl;

    return 0;
}
```

```
}
```

Output:

The lower bound is at position: 3

The upper bound is at position: 5

Some Manipulating Algorithms

11. **arr.erase(position to be deleted)** – This erases selected element in vector and shifts and resizes the vector elements accordingly.

12. **arr.erase(unique(arr.begin(),arr.end()),arr.end())** – This erases the duplicate occurrences in sorted vector in a single line.

```
// C++ program to demonstrate working of erase()
#include <algorithm>
#include <iostream>
#include <vector>
```

28

Chapter 3. Algorithm Library | C++ Magicians STL Algorithm

```
using namespace std;
```

```
int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Vector is :";
    for (int i=0; i<6; i++)
        cout << vect[i]<<" ";

    // Delete second element of vector
    vect.erase(vect.begin()+1);

    cout << "\nVector after erasing the element: ";
    for (int i=0; i<5; i++)
        cout << vect[i] << " ";

    // sorting to enable use of unique()
    sort(vect.begin(), vect.end());

    cout << "\nVector before removing duplicate "
         << " occurrences: ";
    for (int i=0; i<5; i++)
        cout << vect[i] << " ";
```

```

// Deletes the duplicate occurrences
vect.erase(unique(vect.begin(), vect.end()), vect.end());
cout << "\nVector after deleting duplicates: ";
for (int i=0; i< vect.size(); i++)
    cout << vect[i] << " ";

return 0;
}

```

Output:

Vector before erasing the element: 5 20 5 23 20 20
 Vector after erasing the element: 5 5 23 20 20
 Vector before removing duplicate occurrences: 5 5 20 20 23 Vector
 after deleting duplicates: 5 20 23

13. **next_permutation(first_iterator, last_iterator)** – This modified the vector to its next permutation.

29

Chapter 3. Algorithm Library | C++ Magicians STL Algorithm

14. **prev_permutation(first_iterator, last_iterator)** – This modified the vector to its previous permutation.

```

// C++ program to demonstrate working of next_permutation() // and
prev_permutation()
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Given Vector is:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    // modifies vector to its next permutation order
    next_permutation(vect.begin(), vect.end());
    cout << "\nVector after performing next permutation:\n";    for (int
i=0; i<n; i++)
        cout << vect[i] << " ";

    prev_permutation(vect.begin(), vect.end());
}

```

```

        cout << "\nVector after performing prev permutation:\n";    for (int
i=0; i<n; i++)
        cout << vect[i] << " ";

    return 0;
}

```

Output:

Given Vector is:
5 10 15 20 20 23 42 45
Vector after performing next permutation:
5 10 15 20 20 23 45 42
Vector after performing prev permutation:
5 10 15 20 20 23 42 45

14. **distance(first_iterator,desired_position)** – It returns the distance of desired position from the first iterator. This function is very useful while finding the index.

30

Chapter 3. Algorithm Library | C++ Magicians STL Algorithm

```

// C++ program to demonstrate working of distance()
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    // Return distance of first to maximum element
    cout << "Distance between first to max element: ";
    cout << distance(vect.begin(),
        max_element(vect.begin(), vect.end()));    return 0;
}

```

Output:

Distance between first to max element: 7

More – [STL Articles](#)

Source

Chapter 4

Array algorithms in C++ STL (all_of, any_of, none_of, copy_n and iota)

Array algorithms in C++ STL (all_of, any_of, none_of, copy_n and iota) - GeeksforGeeks

From C++11 onwards, some new and interesting algorithms are added in STL of C++. These algorithms operate on an array and are useful in saving time during coding and hence useful in competitive programming as well.

all_of()

This function operates on whole range of array elements and can save time to run a loop to check each elements one by one. It checks for a given property on every element and returns true when each element in range satisfies specified property, else returns false.

```
// C++ code to demonstrate working of all_of()
#include<iostream>
#include<algorithm> // for all_of()
```

```

using namespace std;
int main()
{
    // Initializing array
    int ar[6] = {1, 2, 3, 4, 5, -6};

    // Checking if all elements are positive
    all_of(ar, ar+6, [](int x) { return x>0; })?
        cout << "All are positive elements" :
        cout << "All are not positive elements";

    return 0;
}

```

32

Chapter 4. Array algorithms in C++ STL (all_of, any_of, none_of, copy_n and iota)

```

}

```

Output:

All are not positive elements

In the above code, -6 being a negative element negates the condition and returns false.

any_of()

This function checks for a given range if there's even one element satisfying a given property mentioned in function. Returns true if at least one element satisfies the property else returns false.

```

// C++ code to demonstrate working of any_of()
#include<iostream>
#include<algorithm> // for any_of()
using namespace std;
int main()
{
    // Initializing array
    int ar[6] = {1, 2, 3, 4, 5, -6};

    // Checking if any element is negative
    any_of(ar, ar+6, [](int x){ return x<0; })?
        cout << "There exists a negative element" :
        cout << "All are positive elements";

    return 0;
}

```


Output:

There exists a negative element

In above code, -6 makes the condition positive.

none_of()

This function returns true if none of elements satisfies the given condition else returns false.

```
// C++ code to demonstrate working of none_of()
#include<iostream>
```

33

Chapter 4. Array algorithms in C++ STL (all_of, any_of, none_of, copy_n and iota)

```
#include<algorithm> // for none_of()
using namespace std;
int main()
{
    // Initializing array
    int ar[6] = {1, 2, 3, 4, 5, 6};

    // Checking if no element is negative
    none_of(ar, ar+6, [](int x){ return x<0; })?
        cout << "No negative elements" :
        cout << "There are negative elements";

    return 0;
}
```

Output:

No negative elements

Since all elements are positive, the function returns true.

copy_n()

copy_n() copies one array elements to new array. This type of copy creates a deep copy of array. This function takes 3 arguments, source array name, size of array and the target array name.

```
// C++ code to demonstrate working of copy_n()
#include<iostream>
#include<algorithm> // for copy_n()
using namespace std;
int main()
```

```

{
    // Initializing array
    int ar[6] = {1, 2, 3, 4, 5, 6};

    // Declaring second array
    int ar1[6];

    // Using copy_n() to copy contents
    copy_n(ar, 6, ar1);

    // Displaying the copied array
    cout << "The new array after copying is : ";
    for (int i=0; i<6 ; i++)
        cout << ar1[i] << " ";
}

```

34

Chapter 4. Array algorithms in C++ STL (all_of, any_of, none_of, copy_n and iota)

```

return 0;

}

```

Output:

The new array after copying is : 1 2 3 4 5 6

In the above code, the elements of ar are copied in ar1 using copy_n()

iota()

This function is used to assign continuous values to array. This function accepts 3 arguments, the array name, size, and the starting number.

```

// C++ code to demonstrate working of iota()
#include<iostream>
#include<numeric> // for iota()
using namespace std;
int main()
{
    // Initializing array with 0 values
    int ar[6] = {0};

    // Using iota() to assign values
    iota(ar, ar+6, 20);

    // Displaying the new array
    cout << "The new array after assigning values is : ";    for (int
i=0; i<6 ; i++)
        cout << ar[i] << " ";
}

```

```
    return 0;  
}
```

Output:

The new array after assigning values is : 20 21 22 23 24 25

In the above code, continuous values are assigned to array using `iota()`. **Improved By** : [mohitw16](#)

Source

<https://www.geeksforgeeks.org/useful-array-algorithms-in-c-stl/>

35

Chapter 5

Array range queries over range queries

Array range queries over range queries - GeeksforGeeks

Given an array of size n and a give set of commands of size m . The commands are enumerated from 1 to m . These commands can be of the following two types of commands:

1. **Type 1** [$l\ r\ (1 \leq l \leq r \leq n)$] : Increase all elements of the array by one, whose indices belongs to the range $[l, r]$. In these queries of the index is inclusive in the range.
2. **Type 2** [$l\ r\ (1 \leq l \leq r \leq m)$] : Execute all the commands whose indices are in the range $[l, r]$. In these queries of the index is inclusive in the range. It's guaranteed that r is strictly less than the enumeration/number of the current command.

Note : The array indexing is from 1 as per the problem statement.

Example 1

Input : 5 5
 1 1 2
 1 4 5
 2 1 2

2 1 3
 2 3 4
 Output : 7 7 0 7 7

Explanation of Example 1 :

Our array initially is of size 5 whose each element has been initialized to 0. So now the question states that we have 5 queries for the above example.

36

Chapter 5. Array range queries over range queries

1. Query 1 is of type 1 : As stated above we will simply increment the array indices by 1 the given indices are 1 and 2 so after the execution of the first our array turns down to be 1 1 0 0 0 .
2. Query 2 is of type 1 : As stated above we will simply increment the array indices by 1 the given indices are 4 and 5 so after the execution of the first our array turns down to be 1 1 0 1 1 .
3. Query 3 is of type 2 : As stated in the definition of this type of query we will execute the queries stated in the range i.e. we will operate the queries instead of the array. The range given is 1 and 2 so we will execute queries 1 and 2 again i.e. we will use repetitive approach for the type 2 queries so we will execute query 1 again and our array will be 2 2 0 1 1. Now when we execute the query we will execute query 2 and our resultant array will be 2 2 0 2 2 .
4. Query 4 is of type 2 : As stated in the definition of this type of query we will execute the queries stated in the range i.e. we will operate the queries instead of the array. The range given is 1 and 3 so we will execute queries 1, 2 and 3 again i.e. using repetitive approach queries 1, 2 and 3 will be executed. After the execution of the query 1 again the array will be 3 3 0 2 2 . After the execution of the query 2 again the array will be 3 3 0 3 3 . Now due to query 3 inclusive in the range we will execute query 3 the resultant array will be 4 4 0 4 4 . As explained above.
5. Query 5 is of type 2 : The last query will execute the 3rd and 4th query which has been explained above. After the execution of the 3rd query our array will be 5 5 0 5 5 . And after the execution of the 4th query i.e. execution of query 1, 2 and 3 our array will be 7 7 0 7 7 The above is the desired result

Example 2

Input : 1 2
 1 1 1
 1 1 1
 Output : 2

Explanation of the example 2:

Our array initially is of size 1 whose each element has been initialized to 0. So now the question states that we have 2 queries for the above example.

1. Query 1 is of type 1 : As stated above we will simply increment the array indices by 1 the given indices are 1 and 1 so after the execution of the first our array turns down to be 1 .
2. Query 2 is of type 1 : As stated above we will simply increment the array indices by 1 the given indices are 1 and 1 so after the execution of the first our array turns down to be 2 . This gives us the desired result

Method 1 :

This method is the brute force method where by simple recursion is applied on the type 2 queries and for type 1 queries simple increment in the array index is performed.

37

Chapter 5. Array range queries over range queries

```
// CPP program to perform range queries over range
// queries.
```

```
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to execute type 1 query
void type1(int arr[], int start, int limit)
{
    // incrementing the array by 1 for type
    // 1 queries
    for (int i = start; i <= limit; i++)
        arr[i]++;
}
```

```
// Function to execute type 2 query
void type2(int arr[], int query[][3], int start, int limit) {
    for (int i = start; i <= limit; i++) {

        // If the query is of type 1 function
        // call to type 1 query
        if (query[i][0] == 1)
            type1(arr, query[i][1], query[i][2]);

        // If the query is of type 2 recursive call
        // to type 2 query
        else if (query[i][0] == 2)
            type2(arr, query, query[i][1], query[i][2]);
    }
}
```

```
// Driver code
int main()
{
    // Input size of array amd number of queries
    int n = 5, m = 5;
    int arr[n + 1];
```

```

for (int i = 1; i <= n; i++)
    arr[i] = 0;

// Build query matrix
int temp[15] = { 1, 1, 2, 1, 4, 5, 2,
                1, 2, 2, 1, 3, 2, 3, 4 };
int query[5][3];
int j = 0;
for (int i = 1; i <= m; i++) {
    query[i][0] = temp[j++];
    query[i][1] = temp[j++];

```

38

Chapter 5. Array range queries over range queries

```

    query[i][2] = temp[j++];
}

// Perform queries
for (int i = 1; i <= m; i++)
    if (query[i][0] == 1)
        type1(arr, query[i][1], query[i][2]);
    else if (query[i][0] == 2)
        type2(arr, query, query[i][1], query[i][2]);
// printing the result
for (int i = 1; i <= n; i++)
    cout << arr[i] << " ";

return 0;
}

```

Output:

7 7 0 7 7

The Time complexity of the above code is $O(2^m)$

Method 2 :

In this method we use an extra array for creating the record array to find the number of time a particular query is being executed and after creating the record array we simply execute the queries of type 1 and the contains of the record array is simply added to the main array the and this would give us the resultant array.

```

// CPP program to perform range queries over range
// queries.
#include <bits/stdc++.h>
using namespace std;

// Function to create the record array
void record_sum(int record[], int l, int r,
               int n, int adder)

```

```

{
    for (int i = l; i <= r; i++)
        record[i] += adder;
}

```

```

// Driver Code
int main()
{
    int n = 5, m = 5;
    int arr[n];

```

39

Chapter 5. Array range queries over range queries

```

// Build query matrix
memset(arr, 0, sizeof arr);
int query[5][3] = { { 1, 1, 2 }, { 1, 4, 5 },
                    { 2, 1, 2 }, { 2, 1, 3 },
                    { 2, 3, 4 } };
int record[m];
memset(record, 0, sizeof record);

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

    // If query is of type 2 then function
    // call to record_sum
    if (query[i][0] == 2)
        record_sum(record, query[i][1] - 1,
                    query[i][2] - 1, m, record[i] + 1);

    // If query is of type 1 then simply add
    // 1 to the record array
    else
        record_sum(record, i, i, m, 1);

}

// for type 1 queries adding the contains of
// record array to the main array record array
for (int i = 0; i < m; i++) {
    if (query[i][0] == 1)
        record_sum(arr, query[i][1] - 1,
                    query[i][2] - 1, n, record[i]);
}

// printing the array
for (int i = 0; i < n; i++)
    cout << arr[i] << ' ';

return 0;

```

```
}
```

Output :

7 7 0 7 7

The Time complexity of the above code is $O(n^2)$

Method 3 :

This method has been made more efficient by applying [square root decomposition](#) to the record array.

40

Chapter 5. Array range queries over range queries

```
// CPP program to perform range queries over range  
// queries.
```

```
#include <bits/stdc++.h>
```

```
#define max 10000
```

```
using namespace std;
```

```
// For prefix sum array
```

```
void update(int arr[], int l)
```

```
{
```

```
    arr[l] += arr[l - 1];
```

```
}
```

```
// This function is used to apply square root
```

```
// decomposition in the record array
```

```
void record_func(int block_size, int block[],
```

```
    int record[], int l, int r, int value)
```

```
{
```

```
    // traversing first block in range
```

```
    while (l < r && l % block_size != 0 && l != 0) {
```

```
        record[l] += value;
```

```
        l++;
```

```
    }
```

```
    // traversing completely overlapped blocks in range
```

```
    while (l + block_size <= r + 1) {
```

```
        block[l / block_size] += value;
```

```
        l += block_size;
```

```
    }
```

```
    // traversing last block in range
```

```
    while (l <= r) {
```

```
        record[l] += value;
```

```
        l++;
```

```
    }
```

```
}
```

```
// Function to print the resultant array
```

```
void print(int arr[], int n)
```



```

{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver code
int main()
{
    int n = 5, m = 5;
    int arr[n], record[m];
    int block_size = sqrt(m);
    int block[max];
    int command[5][3] = { { 1, 1, 2 }, { 1, 4, 5 },

```

41

Chapter 5. Array range queries over range queries

```

        { 2, 1, 2 }, { 2, 1, 3 },
        { 2, 3, 4 } };
memset(arr, 0, sizeof arr);
memset(record, 0, sizeof record);
memset(block, 0, sizeof block);

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

    // If query is of type 2 then function
    // call to record_func
    if (command[i][0] == 2) {
        int x = i / (block_size);
        record_func(block_size, block, record,
                    command[i][1] - 1, command[i][2] - 1,
                    (block[x] + record[i] + 1));
    }
    // If query is of type 1 then simply add
    // 1 to the record array
    else
        record[i]++;
}

// Merging the value of the block in the record array    for (int i =
0; i < m; i++) {
    int check = (i / block_size);
    record[i] += block[check];
}

for (int i = 0; i < m; i++) {
    // If query is of type 1 then the array
    // elements are over-written by the record
    // array
    if (command[i][0] == 1) {
        arr[command[i][1] - 1] += record[i];

```

```

        if ((command[i][2] - 1) < n - 1)
            arr[(command[i][2])] -= record[i];
    }

    // The prefix sum of the array
    for (int i = 1; i < n; i++)
        update(arr, i);

    // Printing the resultant array
    print(arr, n);
    return 0;
}

```

42

Chapter 5. Array range queries over range queries

Output :
7 7 0 7 7

Method 4 :

This method has been made more efficient by applying [Binary Indexed Tree or Fenwick Tree](#) by creating two binary indexed tree for query 1 and query 2 respectively.

```

// CPP program to perform range queries over range
// queries.
#include <bits/stdc++.h>
using namespace std;

// Updates a node in Binary Index Tree (BITree) at given index // in
// BITree. The given value 'val' is added to BITree[i] and // all of its
// ancestors in tree.
void updateBIT(int BITree[], int n, int index, int val)
{
    // index in BITree[] is 1 more than the index in arr[]    index =
    index + 1;

    // Traverse all ancestors and add 'val'
    while (index <= n) {

        // Add 'val' to current node of BI Tree
        BITree[index] = (val + BITree[index]);

        // Update index to that of parent in update View
        index = (index + (index & (-index)));
    }
    return;
}

// Constructs and returns a Binary Indexed Tree for given // array of

```

```

size n.
int* constructBITree(int n)
{
    // Create and initialize BITree[] as 0
    int* BITree = new int[n + 1];
    for (int i = 1; i <= n; i++)
        BITree[i] = 0;

    return BITree;
}

// Returns sum of arr[0..index]. This function assumes
// that the array is preprocessed and partial sums of
// array elements are stored in BITree[]

```

43

Chapter 5. Array range queries over range queries

```

int getSum(int BITree[], int index)
{
    int sum = 0;
    // index in BITree[] is 1 more than the index in arr[]    index =
    index + 1;

    // Traverse ancestors of BITree[index]
    while (index > 0) {

        // Add element of BITree to sum
        sum = (sum + BITree[index]);

        // Move index to parent node in getSum View
        index -= index & (-index);
    }
    return sum;
}

// Function to update the BITree
void update(int BITree[], int l, int r, int n, int val)
{
    updateBIT(BITree, n, l, val);
    updateBIT(BITree, n, r + 1, -val);
    return;
}

// Driver code
int main()
{
    int n = 5, m = 5;
    int temp[15] = { 1, 1, 2, 1, 4, 5, 2, 1, 2,
                    2, 1, 3, 2, 3, 4 };
    int q[5][3];
}

```

```

int j = 0;
for (int i = 1; i <= m; i++) {
    q[i][0] = temp[j++];
    q[i][1] = temp[j++];
    q[i][2] = temp[j++];
}

// BITree for query of type 2
int* BITree = constructBITree(m);

// BITree for query of type 1
int* BITree2 = constructBITree(n);

// Input the queries in a 2D matrix
for (int i = 1; i <= m; i++)

```

44

Chapter 5. Array range queries over range queries

```

    cin >> q[i][0] >> q[i][1] >> q[i][2];

    // If query is of type 2 then function call
    // to update with BITree
    for (int i = m; i >= 1; i--)
        if (q[i][0] == 2)
            update(BITree, q[i][1] - 1, q[i][2] - 1, m, 1);
    for (int i = m; i >= 1; i--) {
        if (q[i][0] == 2) {
            long int val = getSum(BITree, i - 1);
            update(BITree, q[i][1] - 1, q[i][2] - 1, m, val);
        }
    }

    // If query is of type 1 then function call
    // to update with BITree2
    for (int i = m; i >= 1; i--) {
        if (q[i][0] == 1) {
            long int val = getSum(BITree, i - 1);
            update(BITree2, q[i][1] - 1, q[i][2] - 1,
                n, (val + 1));
        }
    }

    for (int i = 1; i <= n; i++)
        cout << (getSum(BITree2, i - 1)) << " ";

    return 0;
}

```

Output :

7 7 0 7 7

The Time complexity of Method 3 and Method 4 is $O(\log n)$.

Source

<https://www.geeksforgeeks.org/array-range-queries-range-queries/>

45

Chapter 6

Array with GCD of any of its subset belongs to the given array

Array with GCD of any of its subset belongs to the given array - GeeksforGeeks Given a set of N elements such that N ≤ 1000 , task is to generate an array such that the GCD of any subset of the generated array lies in the given set of elements. The generated array should **not be more than thrice the length of the set of the GCD**.

Prerequisite : [GCD of an Array](#) | [Subset of Array](#)

Examples :

Input : 3

1 2 7

Output : 1 1 2 1 7

Input : 4

2 4 6 12

Output : 4 6 12

Input : 5

2 5 6 7 11

Output : No array can be build

Explanation :

Calculate the [GCD of an array](#) or in this case a set. Now, first sort the given set of GCD. If the GCD of this set is equal to the minimum number of the given set, then just by putting this GCD between each number. But, if this GCD is not the minimum element of the given

46

Chapter 6. Array with GCD of any of its subset belongs to the given array

set, then unfortunately “no array can be build”.

C++

```
// C++ implementation to generate the
// required array
#include <bits/stdc++.h>
using namespace std;

// Function to return gcd of a and b
int gcd(int a, int b)
{
    if (a == 0)
        return b;
    return gcd(b % a, a);
}

// Function to find gcd of
// array of numbers
int findGCD(vector<int> arr, int n)
{
    int result = arr[0];
    for (int i = 1; i < n; i++)
        result = gcd(arr[i], result);
    return result;
}

// Function to generate the array
// with required constraints.
void compute(vector<int> arr, int n)
{
    vector<int> answer;

    // computing GCD of the given set
    int GCD_of_array = findGCD(arr, n);

    // Solution exists if GCD of array is equal
    // to the minimum element of the array
    if(GCD_of_array == arr[0])
    {
```

```

    answer.push_back(arr[0]);
    for(int i = 1; i < n; i++)
    {
        answer.push_back(arr[0]);
        answer.push_back(arr[i]);
    }

```

```

// Printing the built array

```

47

Chapter 6. Array with GCD of any of its subset belongs to the given array

```

        for (int i = 0; i < answer.size(); i++)
            cout << answer[i] << " ";
    }
    else
        cout << "No array can be build";
}

// Driver function
int main()
{
    // Taking in the input and initializing
    // the set STL set in cpp has a property
    // that it maintains the elements in
    // sorted order, thus we do not need
    // to sort them externally
    int n = 3;
    int input[] = {2, 5, 6, 7, 11};
    set<int> GCD(input, input + n);
    vector<int> arr;
    set<int>::iterator it;

    for(it = GCD.begin(); it!= GCD.end(); ++it)
        arr.push_back(*it);

    // Calling the computing function.
    compute(arr,n);

    return 0;
}

```

Java

```

// Java implementation
// to generate the
// required array
import java.io.*;
import java.util.*;

```

```

class GFG
{
// Function to return
// gcd of a and b
static int gcd(int a,
               int b)
{
    if (a == 0)
        return b;

```

48

Chapter 6. Array with GCD of any of its subset belongs to the given array

```

        return gcd(b % a, a);
    }

// Function to find gcd
// of array of numbers
public static int findGCD(ArrayList<Integer>
                        arr, int n)
{
    int result = arr.get(0);
    for (int i = 1; i < n; i++)
        result = gcd(arr.get(i),
                    result);
    return result;
}

// Function to generate
// the array with required
// constraints.
public static void compute(ArrayList<Integer>
                        arr, int n)
{
    ArrayList<Integer> answer =
        new ArrayList<Integer>();

    // computing GCD of
    // the given set
    int GCD_of_array = findGCD(arr, n);

    // Solution exists if GCD
    // of array is equal to the
    // minimum element of the array
    if(GCD_of_array == arr.get(0))
    {
        answer.add(arr.get(0));
        for(int i = 1; i < n; i++)
        {
            answer.add(arr.get(0));
            answer.add(arr.get(i));

```



```

    }

    // Printing the
    // built array
    for (int i = 0;
        i < answer.size(); i++)
        System.out.print(answer.get(i) + " ");
    }
    else
        System.out.print("No array " +

```

49

Chapter 6. Array with GCD of any of its subset belongs to the given array

```

        "can be build");
    }

    // Driver Code
    public static void main(String args[])
    {

        // Taking in the input and
        // initializing the set STL
        // set in cpp has a property
        // that it maintains the
        // elements in sorted order,
        // thus we do not need to
        // sort them externally
        int n = 3;
        Integer input[] = {2, 5, 6, 7, 11};
        HashSet<Integer> GCD = new HashSet<Integer>
            (Arrays.asList(input));
        ArrayList<Integer> arr =
            new ArrayList<Integer>();

        for (int v : GCD)
            arr.add(v);

        // Calling the
        // computing function.
        compute(arr, n);
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)

```

C#

```

// C# implementation
// to generate the

```

```
// required array
using System;
using System.Collections.Generic;
```

```
class GFG
{
    // Function to return
    // gcd of a and b
    static int gcd(int a, int b)
    {
        if (a == 0)
```

50

Chapter 6. Array with GCD of any of its subset belongs to the given array

```
        return b;
        return gcd(b % a, a);
    }

    // Function to find gcd
    // of array of numbers
    static int findGCD(List<int> arr,
                        int n)
    {
        int result = arr[0];
        for (int i = 1; i < n; i++)
            result = gcd(arr[i],
                        result);
        return result;
    }

    // Function to generate
    // the array with required
    // constraints.
    static void compute(List<int> arr,
                        int n)
    {
        List<int> answer = new List<int>();

        // computing GCD of
        // the given set
        int GCD_of_array = findGCD(arr, n);

        // Solution exists if GCD
        // of array is equal to the
        // minimum element of the array
        if(GCD_of_array == arr[0])
        {
            answer.Add(arr[0]);
            for(int i = 1; i < n; i++)
            {
```

```

        answer.Add(arr[0]);
        answer.Add(arr[i]);
    }

    // Printing the
    // built array
    for (int i = 0; i < answer.Count; i++)
        Console.Write(answer[i] + " ");
}
else
    Console.WriteLine("No array " +
        "can be build");

```

51

Chapter 6. Array with GCD of any of its subset belongs to the given array

```

}

// Driver Code
static void Main()
{
    // Taking in the input and
    // initializing the set STL
    // set in cpp has a property
    // that it maintains the
    // elements in sorted order,
    // thus we do not need to
    // sort them externally
    int n = 3;
    int []input= new int[]{2, 5, 6, 7, 11};
    HashSet<int> GCD = new HashSet<int>(input);
    List<int> arr = new List<int>();

    foreach (int b in GCD)
        arr.Add(b);

    // Calling the
    // computing function.
    compute(arr, n);
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)

```

Output:

No array can be build

Time Complexity : $O(n \log(n))$, where n is the size of array given.

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/array-gcd-subset-belongs-given-array/> 52

Chapter 7

BFS using STL for competitive coding

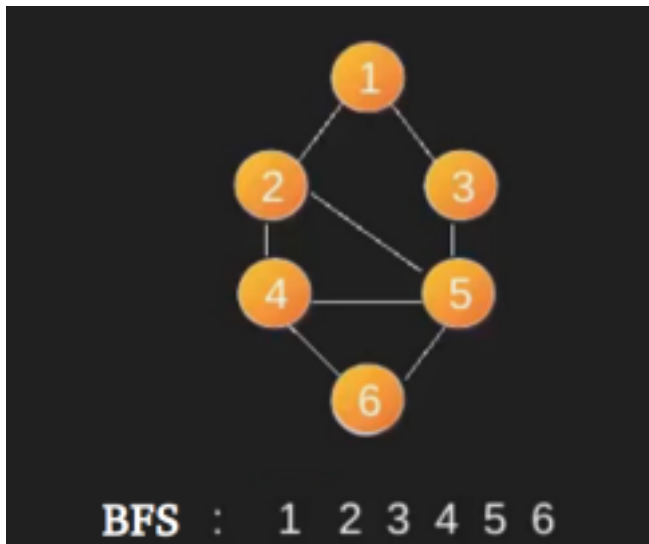
BFS using STL for competitive coding - GeeksforGeeks

A STL based simple implementation of BFS using [queue](#) and [vector](#) in STL. The adjacency list is represented using vectors of vector.

In BFS, we start with a node.

- 1) Create a queue and enqueue source into it.
Mark source as visited.
- 2) While queue is not empty, do following
 - a) Dequeue a vertex from queue. Let this be f .
 - b) Print f
 - c) Enqueue all not yet visited adjacent of f and mark them visited.

Below is an example BFS starting from source vertex 1. Note that there can be multiple BFSs possible for a graph (even from a particular vertex).



For more details of BFS, refer this post .

The code here is simplified such that it could be used in competitive coding.

```
// A Quick implementation of BFS using
// vectors and queue
#include <bits/stdc++.h>
#define pb push_back

using namespace std;

vector<bool> v;
vector<vector<int> > g;

void edge(int a, int b)
{
    g[a].pb(b);

    // for undirected graph add this line
```

```

    // g[b].pb(a);
}

void bfs(int u)
{
    queue<int> q;

    q.push(u);
    v[u] = true;

    while (!q.empty()) {

```

54

Chapter 7. BFS using STL for competitive coding

```

        int f = q.front();
        q.pop();

        cout << f << " ";

        // Enqueue all adjacent of f and mark them visited
        for (auto i = g[f].begin(); i != g[f].end(); i++) {
            if (!v[*i]) {
                q.push(*i);
                v[*i] = true;
            }
        }
    }
}

// Driver code
int main()
{
    int n, e;
    cin >> n >> e;

    v.assign(n, false);
    g.assign(n, vector<int>());

    int a, b;
    for (int i = 0; i < e; i++) {
        cin >> a >> b;
        edge(a, b);
    }

    for (int i = 0; i < n; i++) {
        if (!v[i])
            bfs(i);
    }

    return 0;

```

```
}
```

Input:

8 10

0 1

0 2

0 3

0 4

1 5

2 5

3 6

55

Chapter 7. BFS using STL for competitive coding

4 6

5 7

6 7

Output:

0 1 2 3 4 5 6 7

Source

<https://www.geeksforgeeks.org/bfs-using-stl-competitive-coding/>

Chapter 8

Bit Tricks for Competitive Programming

Bit Tricks for Competitive Programming - GeeksforGeeks

In competitive programming or in general some problems seems difficult but can be solved very easily with little bit magic. We have discussed some tricks in below previous post.

Bitwise Hacks for Competitive Programming

We have considered below facts in this article –

- 0 based indexing of bits from left to right.
- Setting i-th bit means, turning i-th bit to 1
- Clearing i-th bit means, turning i-th bit to 0

1) Clear all bits from LSB to ith bit

```
mask = ~((1 << i+1) - 1);  
x &= mask;
```

Logic: To clear all bits from LSB to i-th bit, we have to AND x with mask having LSB to i-th bit 0. To obtain such mask, first left shift 1 i times. Now if we minus 1 from that, all the bits from 0 to i-1 become 1 and remaining bits become 0. Now we can simply take complement of mask to get all first i bits to 0 and remaining to 1.

Example

```
x = 29 (00011101) and we want to clear LSB to 3rd bit, total 4 bits  
mask -> 1 << 4 -> 16(00010000)  
mask -> 16 - 1 -> 15(00001111)  
mask -> ~mask -> 11110000  
x & mask -> 16 (00010000)
```

2) Clearing all bits from MSB to i-th bit

57

Chapter 8. Bit Tricks for Competitive Programming

```
mask = (1 << i) - 1;  
x &= mask;
```

Logic: To clear all bits from MSB to i-th bit, we have to AND x with mask having MSB to i-th bit 0. To obtain such mask, first left shift 1 i times. Now if we minus 1 from that, all the bits from 0 to i-1 become 1 and remaining bits become 0.

Example

```
x = 215 (11010111) and we want to clear MSB to 4th bit, total 4 bits  
mask -> 1 << 4 -> 16(00010000)  
mask -> 16 - 1 -> 15(00001111)  
x & mask -> 7(00000111)
```

3) Divide by 2

```
x >>= 1;
```

Logic: When we do arithmetic right shift, every bit is shifted to right and blank position is substituted with sign bit of number, 0 in case of positive and 1 in case of negative number. Since every bit is a power of 2, with each shift we are reducing the value of each bit by factor of 2 which is equivalent to division of x by 2.

Example

```
x = 18(00010010)
```

```
x >> 1 = 9 (00001001)
```

4) Multiplying by 2

```
x <<= 1;
```

Logic: When we do arithmetic left shift, every bit is shifted to left and blank position is substituted with 0. Since every bit is a power of 2, with each shift we are increasing the value of each bit by a factor of 2 which is equivalent to multiplication of x by 2. Example

```
x = 18(00010010)
```

```
x << 1 = 36 (00100100)
```

5) Upper case English alphabet to lower case

```
ch |= ' ';
```

Logic: The bit representation of upper case and lower case English alphabets are –

```
A -> 01000001 a -> 01100001
```

```
B -> 01000010 b -> 01100010
```

```
C -> 01000011 c -> 01100011
```

```
...
```

```
...
```

```
Z -> 01011010 z -> 01111010
```

As we can see if we set 5th bit of upper case characters, it will be converted into lower case character. We have to prepare a mask having 5th bit 1 and other 0 (00100000). This mask is bit representation of space character (' '). The character 'ch' then ORed with mask.

Example

```
ch = 'A' (01000001)
mask = '_' (00100000)
ch | mask = 'a' (01100001)
```

Please refer [Case conversion \(Lower to Upper and Vice Versa\)](#) for details. **6) Lower case English alphabet to upper case**

```
ch &= '_';
```

Logic: The bit representation of upper case and lower case English alphabets are –

```
A -> 01000001 a -> 01100001
B -> 01000010 b -> 01100010
C -> 01000011 c -> 01100011
..
..
Z -> 01011010 z -> 01111010
```

As we can see if we clear 5th bit of lower case characters, it will be converted into upper case character. We have to prepare a mask having 5th bit 0 and other 1 (10111111). This mask is bit representation of underscore character ('_'). The character 'ch' then AND with mask.

Example

```
ch = 'a' (01100001)
mask = '_' (10111111)
ch | mask = 'A' (01000001)
```

Please refer [Case conversion \(Lower to Upper and Vice Versa\)](#) for details. **7) Count set bits in integer**

```
int countSetBits(int x)
{
    int count = 0;
    while (x)
    {
```

```
        x &= (x-1);
        count++;
    }
    return count;
}
```

Logic: This is [Brian Kernighan's algorithm](#).

8) Find log base 2 of 32 bit integer

```

int log2(int x)
{
    int res = 0;
    while (x >= 1)
        res++;
    return res;
}

```

Logic: We right shift x repeatedly until it becomes 0, meanwhile we keep count on the shift operation. This count value is the $\log_2(x)$.

9) Checking if given 32 bit integer is power of 2

```

int isPowerof2(int x)
{
    return (x && !(x & x-1));
}

```

Logic: All the power of 2 have only single bit set e.g. 16 (00010000). If we minus 1 from this, all the bits from LSB to set bit get toggled, i.e., $16-1 = 15$ (00001111). Now if we AND x with (x-1) and the result is 0 then we can say that x is power of 2 otherwise not. We have to take extra care when $x = 0$.

Example

$x = 16$ (00010000)

$x - 1 = 15$ (00001111)

$x \& (x-1) = 0$

so 16 is power of 2

Please refer [this](https://www.geeksforgeeks.org/bit-tricks-competitive-programming/) article for more bit hacks.

Source

<https://www.geeksforgeeks.org/bit-tricks-competitive-programming/> 60

Chapter 9

Bits manipulation (Important tactics)

Bits manipulation (Important tactics) - GeeksforGeeks

Prerequisites : [Bitwise operators in C](#), [Bitwise Hacks for Competitive Programming](#), [Bit Tricks for Competitive Programming](#)

1. Compute XOR from 1 to n (direct method) :

```
// Direct XOR of all numbers from 1 to n
int computeXOR(int n)
{
    if (n % 4 == 0)
        return n;
    if (n % 4 == 1)
        return 1;
    if (n % 4 == 2)
        return n + 1;
    else
        return 0;
}
```

Input: 6

Output: 7

Refer [Compute XOR from 1 to n](#) for details.

2. We can quickly calculate the total number of combinations with numbers smaller than or equal to with a number whose sum and XOR are equal. Instead of using looping (Brute force method), we can directly find it by a mathematical trick i.e.

61

Chapter 9. Bits manipulation (Important tactics)

// Refer Equal Sum and XOR for details.

Answer = pow(2, count of zero bits)

3. How to know if a number is a power of 2?

```
// Function to check if x is power of 2
bool isPowerOfTwo(int x)
{
    // First x in the below expression is
    // for the case when x is 0
    return x && !(x & (x - 1));
}
```

Refer [check if a number is power of two](#) for details.

4. Find XOR of all subsets of a set. We can do it in $O(1)$ time. The answer is always 0 if given set has more than one elements. For set with single element, the answer is value of single element. Refer [XOR of the XOR's of all subsets](#) for details.
5. We can quickly find number of leading, trailing zeroes and number of 1's in a binary code of an integer in C++ using GCC. It can be done by using inbuilt function i.e.

Number of leading zeroes: `builtin_clz(x)`

Number of trailing zeroes : `builtin_ctz(x)`

Number of 1-bits: `__builtin_popcount(x)`

Refer [GCC inbuilt functions](#) for details.

6. Convert binary code directly into an integer in C++.

```
// Conversion into Binary code//
#include <iostream>
using namespace std;

int main()
{
    auto number = 0b011;
    cout << number;
    return 0;
}
```

Output: 3

7. The Quickest way to swap two numbers:

```
a ^= b;
b ^= a;
a ^= b;
```

Refer [swap two numbers](#) for details.

8. Simple approach to flip the bits of a number: It can be done by a simple way, just simply subtract the number from the value obtained when all the bits are equal to 1 . For example:

Number : Given Number

Value : A number with all bits set in given number.
Flipped number = Value – Number.

Example :

Number = 23,

Binary form: 10111;

After flipping digits number will be: 01000;

Value: 11111 = 31;

9. We can find the most significant set bit in $O(1)$ time for a fixed size integer. For example below code is for 32 bit integer.

```
int setBitNumber(int n)
{
    // Below steps set bits after
    // MSB (including MSB)

    // Suppose n is 273 (binary
    // is 100010001). It does following
    // 100010001 | 010001000 = 110011001
    n |= n>>1;

    // This makes sure 4 bits
    // (From MSB and including MSB)
    // are set. It does following
    // 110011001 | 001100110 = 111111111
    n |= n>>2;

    n |= n>>4;
    n |= n>>8;
    n |= n>>16;

    // Increment n by 1 so that
    // there is only one set bit
    // which is just before original
    // MSB. n now becomes 1000000000
```

Chapter 9. Bits manipulation (Important tactics)

```
n = n + 1;

// Return original MSB after shifting.
// n now becomes 100000000
return (n >> 1);
}
```

Refer [Find most significant set bit of a number](#) for details.

10. We can quickly check if bits in a number are in alternate pattern (like 101010). We compute $n \wedge (n \gg 1)$. If n has an alternate pattern, then $n \wedge (n \gg 1)$ operation will produce a number having set bits only. ‘^’

is a bitwise XOR operation. Refer [check if a number has bits in alternate pattern](#) for details.

Source

<https://www.geeksforgeeks.org/bits-manipulation-important-tactics/> 64

Chapter 10

Bitwise Hacks for Competitive Programming

It is recommended to refer [Interesting facts about Bitwise Operators](#) as a prerequisite. **1. How to set a bit in the number 'num' :**

If we want to set a bit at nth position in number 'num' ,it can be done using 'OR' operator(|).

- First we left shift '1' to n position via $(1 \ll n)$
- Then, use 'OR' operator to set bit at that position.'OR' operator is used because it will set the bit even if the bit is unset previously in binary representation of number 'num'.

```
#include<iostream>
using namespace std;
// num is the number and pos is the position
// at which we want to set the bit.
void set(int & num,int pos)
{
    // First step is shift '1', second
    // step is bitwise OR
    num |= (1 << pos);
}
int main()
{
    int num = 4, pos = 1;
    set(num, pos);
    cout << (int)(num) << endl;
    return 0;
}
```

Output:

6

We have passed the parameter by 'call by reference' to make permanent changes in the number.

2. How to unset/clear a bit at n'th position in the number 'num' :

Suppose we want to unset a bit at nth position in number 'num' then we have to do this with the help of 'AND' (&) operator.

- First we left shift '1' to n position via $(1 \ll n)$ then we use bitwise NOT operator '~' to unset this shifted '1'.
- Now after clearing this left shifted '1' i.e making it to '0' we will 'AND'(&) with the number 'num' that will unset bit at nth position position.

```
#include <iostream>
```

```

using namespace std;
// First step is to get a number that has all 1's except the given position. void unset(int
&num,int pos)
{
    //Second step is to bitwise and this number with given number    num &=
(~(1 << pos));
}
int main()
{
    int num = 7;
    int pos = 1;
    unset(num, pos);
    cout << num << endl;
    return 0;
}

```

Output:

5

3. Toggling a bit at nth position :

Toggling means to turn bit 'on'(1) if it was 'off'(0) and to turn 'off'(0) if it was 'on'(1) previously. We will be using 'XOR' operator here which is this '^'. The reason behind 'XOR' operator is because of its properties.

- Properties of 'XOR' operator.

- $1 \wedge 1 = 0$
- $0 \wedge 0 = 0$

66

Chapter 10. Bitwise Hacks for Competitive Programming

- $1 \wedge 0 = 1$
- $0 \wedge 1 = 1$

- If two bits are different then 'XOR' operator returns a set bit(1) else it returns an unset bit(0).

```

#include <iostream>
using namespace std;
// First step is to shift 1, Second step is to XOR with given number void
toggle(int &num,int pos)
{
    num ^= (1 << pos);
}
int main()
{
    int num = 4;
    int pos = 1;
    toggle(num, pos);
}

```

```

    cout << num << endl;
    return 0;
}

```

Output:

6

4. Checking if bit at nth position is set or unset:

It is quite easily doable using 'AND' operator.

- Left shift '1' to given position and then 'AND'('&').

```

#include <iostream>
using namespace std;

bool at_position(int num,int pos)
{
    bool bit = num & (1<<pos);
    return bit;
}

int main()
{
    int num = 5;
    int pos = 0;
    bool bit = at_position(num, pos);
    cout << bit << endl;
    return 0;
}

```

67

Chapter 10. Bitwise Hacks for Competitive Programming

Output:

1

Observe that we have first left shifted '1' and then used 'AND' operator to get bit at that position. So if there is '1' at position 'pos' in 'num', then after 'AND' our variable 'bit' will store '1' else if there is '0' at position 'pos' in the number 'num' then after 'AND' our variable bit will store '0'.

Some more quick hacks:

- **Inverting every bit of a number/1's complement:**

If we want to invert every bit of a number i.e change bit '0' to '1' and bit '1' to '0'. We can do this with the help of '~' operator. For example : if number is num=00101100 (binary representation) so '~num' will be '11010011'.

This is also the '1s complement of number'.

```
#include <iostream>
using namespace std;
int main()
{
    int num = 4;

    // Inverting every bit of number num
    cout << (~num);
    return 0;
}
```

Output:
-5

- **Two's complement of the number:** 2's complement of a number is 1's complement + 1.

So formally we can have 2's complement by finding 1s complement and adding 1 to the result i.e ($\sim\text{num}+1$) or what else we can do is using '-' operator.

```
#include <iostream>
using namespace std;
int main()
{
    int num = 4;
    int twos_complement = -num;
    cout << "This is two's complement " << twos_complement << endl;    cout
    << "This is also two's complement " << (~num+1) << endl;    return 0;
}
```

Output:

This is two's complement -4
This is also two's complement -4

- **Stripping off the lowest set bit :**

In many situations we want to strip off the lowest set bit for example in Binary Indexed tree data structure, counting number of set bit in a number.

We do something like this:

$X = X \& (X-1)$

But how does it even work ?

Let us see this by taking an example, let $X = 1100$.

$(X-1)$ inverts all the bits till it encounter lowest set '1' and it also invert that lowest set '1'.

$X-1$ becomes 1011. After 'ANDing' X with $X-1$ we get lowest set bit stripped.

```
#include <iostream>
using namespace std;
void strip_last_set_bit(int &num)
{
    num = num & (num-1);
}
int main()
{
    int num = 7;
    strip_last_set_bit(num);
    cout << num << endl;
    return 0;
}
```

Output:

6

- **Getting lowest set bit of a number:**

This is done by using expression ' $X \& (-X)$ '. Let us see this by taking an example: Let $X = 00101100$. So $\sim X$ (1's complement) will be '11010011' and 2's complement will be $(\sim X + 1$ or $-X)$ i.e '11010100'. So if we 'AND' original number ' X ' with its two's complement which is ' $-X$ ', we get lowest set bit.

```
00101100
& 11010100
-----
00000100
```

```
#include <iostream>
using namespace std;
int lowest_set_bit(int num)
{
    int ret = num & (-num);
    return ret;
}
```

```
int main()
{
    int num = 10;
    int ans = lowest_set_bit(num);
    cout << ans << endl;
    return 0;
}
```

Output:

2

[Bit Tricks for Competitive Programming](#)

Refer [BitWise Operators Articles](#) for more articles on Bit Hacks.

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/bitwise-hacks-for-competitive-programming/> 70

Chapter 11

Burst Balloon to maximize coins

Burst Balloon to maximize coins - GeeksforGeeks

We have been given N balloons, each with a number of coins associated with it. On bursting a balloon i, the number of coins gained is equal to $A[i-1] * A[i] * A[i+1]$. Also, balloons i-1 and i+1 now become adjacent. Find the maximum possible profit earned after bursting all the balloons. Assume an extra 1 at each boundary.

Examples:

Input : 5, 10

Output : 60

Explanation - First Burst 5, Coins = $1 \times 5 \times 10$

Then burst 10, Coins += $1 \times 10 \times 1$

Total = 60

Input : 1, 2, 3, 4, 5

Output : 110

A recursive solution is discussed [here](#). We can solve this problem using dynamic programming.

First, consider a sub-array from indices Left to Right(inclusive).

If we assume the balloon at index Last to be the last balloon to be burst in this sub-array, we would say the coins gained to be $A[\text{left}-1] \times A[\text{last}] \times A[\text{right}+1]$.

Also, the total Coin Gained would be this value, plus $\text{dp}[\text{left}][\text{last} - 1] + \text{dp}[\text{last} + 1][\text{right}]$, where $\text{dp}[i][j]$ means maximum coin gained for sub-array with indices i, j. Therefore, for each value of Left and Right, we need find and choose a value of Last with maximum coin gained, and update the dp array.

Our Answer is the value at $\text{dp}[1][N]$.

Python3

Python program burst balloon problem.

71

Chapter 11. Burst Balloon to maximize coins

```
def getMax(A):
    N = len(A)
    A = [1] + A + [1] # Add Bordering Balloons
    dp = [[0 for x in range(N + 2)] for y in range(N + 2)] # Declare DP Array
    for length in range(1, N + 1):
        for left in range(1, N-length + 2):
            right = left + length - 1

            # For a sub-array from indices left, right
            # This innermost loop finds the last balloon burst
            for last in range(left, right + 1):
                dp[left][right] = max(dp[left][right], \
                    dp[left][last-1] + \
                    A[left-1]*A[last]*A[right + 1] + \
                    dp[last + 1][right])
            return(dp[1][N])

# Driver code
A = [1, 2, 3, 4, 5]
print(getMax(A))
```

Output:

110

Source

<https://www.geeksforgeeks.org/burst-balloon-to-maximize-coins/>

72

Chapter 12

C qsort() vs C++ sort()

C qsort() vs C++ sort() - GeeksforGeeks

Standard C library provides qsort function that can be used for sorting an array. Following is the prototype of qsort() function.

```
// Sort an array of any type. The parameters are, base  
// address of array, size of array and pointer to  
// comparator function  
void qsort (void* base, size_t num, size_t size,  
            int (*comparator)(const void*, const void*));
```

It requires a pointer to the array, the number of elements in the array, the size of each element and a comparator function. We have discussed qsort comparator in detail [here](#).

C++ Standard Library provides a similar function `sort()` that originated in the STL. We have discussed C++ sort [here](#). Following are prototypes of C++ `sort()` function.

```
// To sort in default or ascending order.
template
void sort(T first, T last);

// To sort according to the order specified
// by comp.
template
void sort(T first, T last, Compare comp);
```

The order of equal elements is not guaranteed to be preserved. C++ provides `std::stable_sort` that can be used to preserve order.

Comparison to `qsort` and `sort()`

1. Implementation details:

As the name suggests, `qsort` function uses QuickSort algorithm to sort the given array, although the C standard does not require it to implement quicksort.

C++ `sort` function uses introsort which is a hybrid algorithm. Different implementations use different algorithms. The GNU Standard C++ library, for example, uses a 3-part hybrid sorting algorithm: introsort is performed first (introsort itself being a hybrid of quicksort and heap sort) followed by an insertion sort on the result.

2. Complexity :

The C standard doesn't talk about its complexity of `qsort`. The new C++11 standard requires that the complexity of `sort` to be $O(N \log(N))$ in the worst case. Previous versions of C++ such as C++03 allow possible worst case scenario of $O(N^2)$. Only average complexity was required to be $O(N \log N)$.

3. Running time:

STL's `sort` ran faster than C's `qsort`, because C++'s templates generate optimized code for a particular data type and a particular comparison function.

STL's `sort` runs 20% to 50% faster than the hand-coded quicksort and 250% to 1000% faster than the C `qsort` library function. C might be the fastest language but `qsort` is very slow.

When we tried to sort one million integers on C++14, Time taken by C `qsort()` was 0.247883 sec and time taken by C++ `sort()` was only 0.086125 sec

```
// C++ program to demonstrate performance of
// C qsort and C++ sort() algorithm
#include <bits/stdc++.h>
using namespace std;
```

```
// Number of elements to be sorted
```

```

#define N 1000000

// A comparator function used by qsort
int compare(const void * a, const void * b)
{
    return ( *(int*)a - *(int*)b );
}

// Driver program to test above functions
int main()
{
    int arr[N], dupArr[N];

    // seed for random input
    srand(time(NULL));

    // to measure time taken by qsort and sort
    clock_t begin, end;
    double time_spent;

```

74

Chapter 12. C qsort() vs C++ sort()

```

// generate random input
for (int i = 0; i < N; i++)
    dupArr[i] = arr[i] = rand()%100000;

begin = clock();
qsort(arr, N, sizeof(int), compare);
end = clock();

// calculate time taken by C qsort function
time_spent = (double)(end - begin) / CLOCKS_PER_SEC;

cout << "Time taken by C qsort() - "
    << time_spent << endl;

time_spent = 0.0;

begin = clock();
sort(dupArr, dupArr + N);
end = clock();

// calculate time taken by C++ sort
time_spent = (double)(end - begin) / CLOCKS_PER_SEC;

cout << "Time taken by C++ sort() - "
    << time_spent << endl;

return 0;

```

```
}
```

Output :

Time taken by C qsort() - 0.247883

Time taken by C++ sort() - 0.086125

C++ sort() is blazingly faster than qsort() on equivalent data due to inlining. sort() on a container of integers will be compiled to use std::less::operator() by default, which will be inlined and sort() will be comparing the integers directly. On the other hand, qsort() will be making an indirect call through a function pointer for every comparison which compilers fails to optimize.

4. Flexibility:

STL's sort works for all data types and for different data containers like C arrays, C++ vectors, C++ deque, etc and other containers that can be written by the user. This kind of flexibility is rather difficult to achieve in C.

5. Safety:

Compared to qsort, the templated sort is more type-safe since it does not require access to data items through unsafe void pointers, as qsort does.

References:

<http://theory.stanford.edu/~amitp/rants/c++-vs-c>

[https://en.wikipedia.org/wiki/Sort_\(C%2B%2B\)](https://en.wikipedia.org/wiki/Sort_(C%2B%2B))

Source

<https://www.geeksforgeeks.org/c-qsort-vs-c-sort/>

Chapter 13

C++ tricks for competitive programming (for C++ 11)

C++ tricks for competitive programming (for C++ 11) - GeeksforGeeks We have discussed some tricks in below post. In this post, some more tricks are discussed. [Writing C/C++ code efficiently in Competitive programming](#)

Although practice is the only way that ensures increased performance in programming contests but having some tricks up your sleeve ensures an upper edge and fast debugging.

1) Checking if the number is even or odd without using the % operator: Although this trick is not much better than using % operator but is sometimes efficient (with large numbers). Use & operator:

```
if (num & 1)
    cout << "ODD";
else
    cout << "EVEN";
```

Example:

num = 5

Binary: "101 & 1" will be 001, so true

num = 4

Binary: "100 & 1" will be 000, so false.

2) Fast Multiplication or Division by 2

Multiplying by 2 means shifting all the bits to left and dividing by 2 means shifting to the right.

Example : 2 (Binary 10): shifting left 4 (Binary 100) and right 1 (Binary 1)

```
n = n << 1; // Multiply n with 2
n = n >> 1; // Divide n by 2
```

3) Swapping of 2 numbers using XOR:

This method is fast and doesn't require the use of 3rd variable.

```
// A quick way to swap a and b
a ^= b;
b ^= a;
a ^= b;
```

4) Avoiding use of strlen():

```
// Use of strlen() can be avoided by:
for (i=0; s[i]; i++)
```

```
{
}
// loop breaks when the character array ends.
```

5) Use of `emplace_back()` (Discussed [here](#), [here](#) and [here](#))

Instead of `push_back()` in STL `emplace_back` can be used because it is much faster and instead of allocating memory somewhere else and then appending it directly allocates memory in the container.

6) Inbuilt GCD function: C++ has inbuilt GCD function and there is no need to explicitly code it. Syntax: `__gcd(x, y);`

7) Using Inline functions: We can create inline functions and use them without having to code them up during the contest. Examples: (a) function for sieve, (b) function for palindrome.

8) Maximum size of the array: We must be knowing that the maximum size of array declared inside the main function is of the order of 10^6 but if you declare array globally then you can declare its size upto 10^7 .

9) Calculating the most significant digit: To calculate the most significant digit of any number `log` can be directly used to calculate it.

Suppose the number is `N` then
 Let `double K = log10(N);`
 now `K = K - floor(K);`
`int X = pow(10, K);`
`X` will be the most significant digit.

10) Calculating the number of digits directly: To calculate number of digits in a number, instead of looping you can efficiently use `log` :

Number of digits in `N` = `floor(log10(N)) + 1;`

11) Efficient trick to know if a number is a power of 2 sing the normal technique of division the complexity comes out to be $O(\log N)$, but it can be solved using $O(v)$ where `v` are the number of digits of number in binary form.

```
/* Function to check if x is power of 2*/
bool isPowerOfTwo (int x)
{
  /* First x in the below expression is
   for the case when x is 0 */
  return x && !(x&(x-1));
}
```

12) C++11 has in built algorithms for following:

```
// are all of the elements positive?
all_of(first, first+n, ispositive());

// is there at least one positive element?
any_of(first, first+n, ispositive());

// are none of the elements positive?
none_of(first, first+n, ispositive());
```

Please refer [Array algorithms in C++ STL \(all_of, any_of, none_of, copy_n and itoa\)](#) for details.

13) Copy Algorithm: used to copy elements from one container to another.

```
int source[5] = {0, 12, 34, 50, 80};
int target[5];
// copy 5 elements from source to target
copy_n(source, 5, target);
```

Please refer [Array algorithms in C++ STL \(all_of, any_of, none_of, copy_n and itoa\)](#) for details.

14) The Iota Algorithm The algorithm `iota()` creates a range of sequentially increasing values, as if by assigning an initial value to `*first`, then incrementing that value using prefix `++`. In the following listing, `iota()` assigns the consecutive values {10, 11, 12, 13, 14} to the array `arr`, and {'a', 'b', 'c'} to the char array `c[]`.

```
int a[5] = {0};
char c[3] = {0};

// changes a to {10, 11, 12, 13, 14}
iota(a, a+5, 10);
iota(c, c+3, 'a'); // {'a', 'b', 'c'}
```

79

Chapter 13. C++ tricks for competitive programming (for C++ 11)

Please refer [Array algorithms in C++ STL \(all_of, any_of, none_of, copy_n and itoa\)](#) for details.

15) Initialization in Binary form: In C++ 11 assignments can also be made in binary form.

```
// C++ code to demonstrate working of
// "binary" numbers
#include<iostream>
using namespace std;
```

```
int main()
{
    auto number = 0b011;
    cout << number;
    return 0;
}
```

Output :

3

16) Use of “and” Though not a very productive one, this tip helps you to just use conditional operator and instead of typing &.

```
// C++ code to demonstrate working of "and"
#include<iostream>
using namespace std;
int main()
{
    int a = 10;
    if (a < 20 and a > 5)
        cout << "Yes";
    return 0;
}
```

Output :

Yes

Improved By : [bugsanderrors](#)

Source

<https://www.geeksforgeeks.org/c-tricks-competitive-programming-c-11/> 80