**1. Write a program to compute the sume of the two given integer values. If the two values are the same, then return triple their sum.**

Sample Input

1, 2

3, 2

2, 2

Sample Output

3

5

12

Solution

#include <iostream>

using namespace std;

int test(int x, int y)

{

return x == y ? (x + y)\*3 : x + y;

}

int main()

{

cout << test(1, 2) << endl;

cout << test(3, 2) << endl;

cout << test(2, 2) << endl;

return 0;

}

**2. Write a program to get the absolute difference between n and 51. If n is greater than 51 return triple the absolute difference.**

Solution

#include <iostream>

using namespace std;

int test(int n)

{

const int x = 51;

if (n > x)

{

return (n - x)\*3;

}

return x - n;

}

int main()

{

cout << test(53) << endl;

cout << test(30) << endl;

cout << test(51) << endl;

return 0;

}

**3. Write a program to check two given integers, and return true if one of them is 30 or if their sum is 30.**

Sample Input:

30, 0

25, 5

20, 30

20, 25

Sample Output:

1

1

1

0

Solution

#include <iostream>

using namespace std;

bool test(int x, int y)

{

return x == 30 || y == 30 || (x + y == 30);

}

int main()

{

cout << test(30, 0) << endl;

cout << test(25, 5) << endl;

cout << test(20, 30) << endl;

cout << test(20, 25) << endl;

return 0;

}

**4. Write a program to check a given integer and return true if it is within 10 of 100 or 200.**

Sample Input:

103

90

89

Sample Output:

1

1

0

Solution

#include <iostream>

using namespace std;

bool test(int x)

{

if(abs(x - 100) <= 10 || abs(x - 200) <= 10)

return true;

return false;

}

int main()

{

cout << test(103) << endl;

cout << test(90) << endl;

cout << test(89) << endl;

return 0;

}

**5. Write a program to create a new string where ‘if’ is added to the front of a given string. If the string already begins with ‘if’, return the string unchanged.**

Sample Input:

“if else”

“else”

Sample Output:

if else

if else

Solution

#include <iostream>

using namespace std;

string test(string s)

{

if (s.length() > 2 && s.substr(0, 2)=="if")

{

return s;

}

return "if " + s;

}

int main()

{

cout << test("if else") << endl;

cout << test("else") << endl;

return 0;

}

**6. Write a program to remove the character in a given position of a given string. The given position will be in the range 0..string length -1 inclusive.**

Sample Input:

"Python", 1

"Python", o

"Python", 4

Sample Output:

Pthon

ython

Pythn

Solution

#include <iostream>

using namespace std;

string test(string str, int n)

{

return str.erase(n, 1);

}

int main()

{

cout << test("Python", 1) << endl;

cout << test("Python", 0) << endl;

cout << test("Python", 4) << endl;

return 0;

}

**7. Write a program to exchange the first and last characters in a given string and return the new string.**

Sample Input:

"abcd"

"a"

"xy"

Sample output:

dbca

a

yx

Solution

#include <iostream>

using namespace std;

string test(string str)

{

return str.length() > 1

? str.substr(str.length() - 1) + str.substr(1, str.length() - 2) + str.substr(0, 1) : str;

}

int main()

{

cout << test("abcd") << endl;

cout << test("a") << endl;

cout << test("xy") << endl;

return 0;

}

**8. Write a program to create a new string which is 4 copies of the 2 front characters of a given string. If the given string length is less than 2 return the original string.**

Sample Input:

"C Sharp"

"JS"

"a"

Sample Output:

C C C C

JSJSJSJS

A

Solution

#include <iostream>

using namespace std;

string test(string str)

{

return str.length() < 2 ? str : str.substr(0, 2) + str.substr(0, 2) + str.substr(0, 2) + str.substr(0, 2);

}

int main()

{

cout << test("C Sharp") << endl;

cout << test("JS") << endl;

cout << test("a") << endl;

return 0;

}

**9. Write a program to create a new string with the last char added at the front and back of a given string of length 1 or more.**

Sample Input:

"Red"

"Green"

"1"

Sample Output:

dRedd

nGreenn

111

Solution

#include <iostream>

using namespace std;

string test(string str)

{

string s = str.substr(str.length()-1);

return s + str + s;

}

int main()

{

cout << test("Red") << endl;

cout << test("Green") << endl;

cout << test("1") << endl;

return 0;

}

**10. Write a C++ program to count the string "aa" in a given string and assume "aaa" contains two "aa"**

Sample Input:

"bbaaccaag"

"jjkiaaasew"

"JSaaakoiaa"

Sample Output:

2

2

3

#include <iostream>

using namespace std;

int test(string s)

{

int ctr\_aa = 0;

for (int i = 0; i < s.length() - 1; i++)

{

if (s.substr(i, 2) == "aa")

{

ctr\_aa++;

}

}

return ctr\_aa;

}

int main()

{

cout << test("bbaaccaag") << endl;

cout << test("jjkiaaasew") << endl;

cout << test("JSaaakoiaa") << endl;

return 0;

}

**11. Write a program in C++ to find the sum of series 1 - X^2/2! + X^4/4!-.... upto nth term**

Sample Output:

Input the value of X: 3

Input the value for nth term: 4

term 1 value is: 1

term 2 value is: -4.5

term 3 value is: 3.375

term 4 value is: -1.0125

The sum of the above series is: -1.1375

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

float x, sum, term, fct, y, j, m;

int i, n;

y = 2;

cout << "\n\n Find the sum of the series 1 - X^2/2! + X^4/4!-....:\n";

cout << "---------------------------------------------------------\n";

cout << " Input the value of X: ";

cin >> x;

cout << " Input the value for nth term: ";

cin >> n;

sum = 1;

term = 1;

cout << " term 1 value is: " << term << endl;

for (i = 1; i < n; i++)

{

fct = 1;

for (j = 1; j <= y; j++)

{

fct = fct \* j;

}

term = term \* (-1);

m = pow(x, y) / fct;

m = m \* term;

cout << " term " << i + 1 << " value is: " << m << endl;

sum = sum + m;

y += 2;

}

cout << " The sum of the above series is: " << sum << endl;

}

**12. Write a C++ program to replace a given number until it become 1**

**If the given number(n) is even replace n with n/2 and if the given number(n) is odd replace n with either n+1 or n-1. Find the minimum number of replacements.**

Sample Input: n = 8

Number of replacements: 3

Sample Input: n = 10

Number of replacements: 4

#include <iostream>

#include <cmath>

using namespace std;

long long num\_replace(long long n)

{

if (n==1) return 0;

if (n%2==0)

return num\_replace(n/2)+1;

else

return min(num\_replace(n+1), num\_replace(n-1))+1;

}

int main()

{

long n = 8; // 8 -> 4 -> 2 -> 1

cout << "\nOriginal number: "<< n << " Number of replacements: " << num\_replace(n) << endl;

n = 10; // 10 -> 5 -> 4 -> 2 -> 1

cout << "\nOriginal number: "<< n << " Number of replacements: " << num\_replace(n) << endl;

n = 12; // 12 -> 6 -> 3 -> 2 -> 1

cout << "\nOriginal number: "<< n << " Number of replacements: " << num\_replace(n) << endl;

return 0;

}

**13. Write a program in C++ to calculate the series (1) + (1+2) + (1+2+3) + (1+2+3+4) + ... + (1+2+3+4+...+n).**

Sample Output:

Input the value for nth term: 5

1 = 1

1+2 = 3

1+2+3 = 6

1+2+3+4 = 10

1+2+3+4+5 = 15

The sum of the above series is: 35

#include <iostream>

using namespace std;

int main()

{

int i, j, n, sum = 0, tsum;

cout << "\n\n Find the sum of the series (1) + (1+2) + (1+2+3) + (1+2+3+4) + ... + (1+2+3+4+...+n):\n";

cout << "------------------------------------------------------------------------------------------\n";

cout << " Input the value for nth term: ";

cin >> n;

for (i = 1; i <= n; i++)

{

tsum = 0;

for (j = 1; j <= i; j++)

{

sum += j;

tsum += j;

cout << j;

if (j < i)

{

cout << "+";

}

}

cout << " = " << tsum << endl;

}

cout << " The sum of the above series is: " << sum << endl;

}

**14. Write a program in C++ to calculate the sum of the series 1.2+2.3+3.4+4.5+5.6+.......**

Sample Output:

Input the last integer between 1 to 98 without fraction you want to add: 10 1.2 + 2.3 + 3.4 + 4.5 + 5.6 + 6.7 + 7.8 + 8.9 + 9.1 + 10.11 The sum of the series =59.61

#include <iostream>

using namespace std;

int main()

{

int trm;

double num, sum, i, m;

cout << "\n\n calculate the sum of the series 1.2+2.3+3.4+4.5+5.6+......:\n";

cout << "----------------------------------------------------------------\n";

cout << " Input the last integer between 1 to 98 without fraction you want to add: ";

cin >> trm;

for (i = 1; i <= trm; i++)

{

if (i < 9)

{

m = .1;

}

else

{

m = .01;

}

num = i + ((i + 1) \* (m));

sum = sum + num;

cout << num;

if (i < trm)

{

cout << " + ";

}

}

cout << "\n The sum of the series =" << sum << endl;

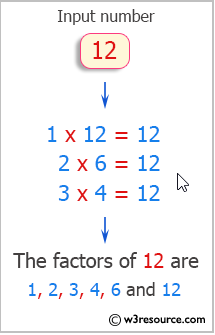
}

**15. Write a program in C++ to enter any number and print all factors of the number**

Sample Output:

Input a number: 63

The factors are: 1 3 7 9 21 63



#include <iostream>

using namespace std;

int main()

{

int num, i;

cout << "\n\n Print all factors of a number:\n";

cout << "-----------------------------------\n";

cout << " Input a number: ";

cin >> num;

cout << "The factors are: ";

for (i = 1; i <= num; i++)

{

if (num % i == 0)

{

cout << i << " ";

}

}

cout << endl;

}

**16. What is the output when the following code fragment is executed?**

int found = 0, count = 5;

if (!found !! –count == 0)

cout << “danger” << endl;

cout << “cout = “ << cout << endl;

**17. What is the output when the following code fragment is executed?**

char ch;

char title[] = “Titanic”;

ch = title[1];

Title = ch;

cout << title << endl;

cout << ch << endl;

**18. Given an array, find the total number of inversions of it. If (i < j) and (A[i] > A[j]), then pair (i, j) is called an inversion of an array A. We need to count all such pairs in the array.**

**Input:**  A[] = [1, 9, 6, 4, 5]

**Output:** The inversion count is 5

There are 5 inversions in the array: (9, 6), (9, 4), (9, 5), (6, 4), (6, 5)

#include <stdio.h>

// Function to find inversion count of a given array

int findInversionCount(int arr[], int n)

{

int inversionCount = 0;

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

{

for (int j = i + 1; j < n; j++)

{

if (arr[i] > arr[j]) {

inversionCount++;

}

}

}

return inversionCount;

}

int main()

{

int arr[] = { 1, 9, 6, 4, 5 };

int N = sizeof(arr)/sizeof(arr[0]);

printf("The inversion count is %d", findInversionCount(arr, N));

return 0;

}

**19. How would you optimally calculate p^k, where k is a non-negative integer? What is the complexity of the solution?**

**Answer:**

First, let’s mention that the trivial solution has the complexity of O(k). The problem can be solved by squaring and multiplying.

We know that p^k = p^x \* p^y if x+y=k. We also know that p^k = (p^a)^b if a\*b=k.

For an even value of k, choosing a = 2 and b = k/2, thus having p^k = (p^2)^(k/2), will reduce the number of required multiplications almost in half. For an odd value of k, choosing x = 1 and y=k-1 will result in y being even, and we can then simply repeat the same process as for the even case. This allows us to define a recursive function:

FUNCTION pow(base, exponent)

IF exponent == 0

RETURN 1

ELSE IF exponent is even

RETURN pow(base \* base, exponent / 2)

ELSE

RETURN base \* pow(base \* base, (exponent - 1) / 2)

END IF

This solution results in a complexity of O(log k).

20. A significantly large static set of string keys has been given, together with data for each of those keys. We need to create a data structure that allows us to update and access that data quickly, with constant time even in worst cases. How can we solve this problem?

**Answer:**

Let’s mark the number of keys as N. The problem presented is a problem of perfect hashing. We do a similar approach as a normal HashTable, but instead of storing collisions in a list, we store them in a secondary HashTable. We choose primary hashing functions until all buckets have a relatively small number of elements in them. After that, in buckets with K keys we place hash tables with K^2 buckets. Even though this seems to result in high memory complexity, expected memory complexity is O(N). By choosing this size of HashTables we have a 50% probability to have collisions. This makes it easy to choose hashing functions that will not result in collisions.

**21: What will i and j equal after the code below is executed? Explain your answer.**

**int i = 5;**

**int j = i++;**

After the above code executes, i will equal 6, but j will equal 5.

Understanding the reason for this is fundamental to understanding how the unary increment (++) and decrement (--) operators work in C++.

When these operators precede a variable, the value of the variable is modified first and then the modified value is used. For example, if we modified the above code snippet to instead say int j = ++i;, i would be incremented to 6 and then j would be set to that modified value, so both would end up being equal to 6.

However, when these operators follow a variable, the unmodified value of the variable is used and then it is incremented or decremented. That’s why, in the statement int j = i++; in the above code snippet, j is first set to the unmodified value of i (i.e., 5) and then i is incremented to 6.

**22: How many times will this loop execute? Explain your answer.**

**unsigned chart half\_limit = 150;**

**for (unsigned char i = 0 ; i < 2 \* half\_limit ; ++i) {**

**// do something**

**}**

If you said 300, you would have been correct if i had been declared as an int. However, since i was declared as an unsigned char, the corrct answer is that this code will result in an infinite loop.

Here’s why:

The expression 2 \* half\_limit will get promoted to an int (based on C++ conversion rules) and will have a value of 300. However, since i is an unsigned char, it is rerepsented by an 8-bit value which, after reaching 255, will overflow (so it will go back to 0) and the loop will therefore go on forever.

**23: Recursion is a common technique used in C++ programming. What are the advantages and disadvantages of using recursion?**

The simplest definition is that a recursive function is one that calls itself.

The main advantage is that it leads to simpler, more elegant code. This in turn is easier to develop and, at least theoretically, leads to fewer errors.

The main disadvantage is that recursion is often slower in practice compared to its iterative counterpart. This performance hit is not attributable to complexity, but rather, a recursive function uses more memory. Each recursive function call reserves additional space on the stack, causing it to grow. If the recursion is too “deep”, meaning that it goes for too long before returning, it can result in a “stack overflow” error.

**24: Maximum Sum Subarray Problem (Kadane’s Algorithm)**

Given an integer array, find a contiguous subarray within it that has the largest sum.

**Input:**  {-2, 1, -3, 4, -1, 2, 1, -5, 4}

**Output:** Subarray with the largest sum is {4, -1, 2, 1} with sum 6.

**#include <iostream>**

**#include <climits>**

**using namespace std;**

**// Function to find the maximum sum of a contiguous subarray**

**// in a given integer array (handles negative numbers as well)**

**int kadaneNeg(int arr[], int n)**

**{**

**// stores the maximum sum subarray found so far**

**int max\_so\_far = INT\_MIN;**

**// stores the maximum sum of subarray ending at the current position**

**int max\_ending\_here = 0;**

**// traverse the given array**

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

**{**

**// update the maximum sum of subarray "ending" at index `i` (by adding the**

**// current element to maximum sum ending at previous index `i-1`)**

**max\_ending\_here = max\_ending\_here + arr[i];**

**// maximum sum should be more than the current element**

**max\_ending\_here = max(max\_ending\_here, arr[i]);**

**// update the result if the current subarray sum is found to be greater**

**max\_so\_far = max(max\_so\_far, max\_ending\_here);**

**}**

**return max\_so\_far;**

**}**

**int main()**

**{**

**int arr[] = { -8, -3, -6, -2, -5, -4 };**

**int n = sizeof(arr)/sizeof(arr[0]);**

**cout << "The maximum sum of a contiguous subarray is " << kadaneNeg(arr, n);**

**return 0;**

**}**

**Output:**

**The maximum sum of a contiguous subarray is -2**

**25: Maximum Product Subset Problem**

**Given an integer array, find the maximum product of its elements among all its subsets.**

**For example,  
  
Input: nums[] = { -6, 4, -5, 8, -10, 0, 8 }  
Output: The maximum product of a subset is 15360  
The subset with the maximum product of its elements is { -6, 4, 8, -10, 8 }  
  
Input: nums[] = { 4, -8, 0, 8 }  
Output: The maximum product of a subset is 32  
The subset with the maximum product of its elements is { 4, 8 }**Brute-Force Solution  
**A naive solution is to** [**consider every subset**](https://www.techiedelight.com/generate-power-set-given-set/) **and find the product of their elements. Finally, return the maximum product found among all subsets.**

**#include <iostream>**

**#include <vector>**

**#include <climits>**

**using namespace std;**

**// Function to generate the product of all elements in a given set**

**// and update maximum product found so far**

**void findMaxProduct(vector<int> const &set, int &maximum)**

**{**

**int product = 1;**

**for (int j: set) {**

**product = product \* j;**

**}**

**// if the set is not empty, then update the product**

**if (set.size()) {**

**maximum = max (maximum, product);**

**}**

**}**

**// Function to generate a power set of a given set `S`**

**void findPowerSet(vector<int> const &S, vector<int> &set, int n, int &maximum)**

**{**

**// if we have considered all elements, we have generated a subset**

**if (n == 0)**

**{**

**// compute its product of elements and update the maximum product**

**// found so far**

**findMaxProduct(set, maximum);**

**return;**

**}**

**// consider the n'th element**

**set.push\_back(S[n - 1]);**

**findPowerSet(S, set, n - 1, maximum);**

**set.pop\_back(); // backtrack**

**// or don't consider the n'th element**

**findPowerSet(S, set, n - 1, maximum);**

**}**

**int main()**

**{**

**vector<int> S = { -6, 4, -5, 8, -10, 0, 8 };**

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

**vector<int> set;**

**int maximum = INT\_MIN;**

**findPowerSet(S, set, n, maximum);**

**printf("The maximum product of a subset is %d", maximum);**

**return 0;**

**}**

**The time complexity of the above solution is exponential and requires additional space for the recursion (call stack).**Linear-time Solution  
**We can solve this problem in linear time by finding a negative element with a minimum absolute value in the set. We also count the total number of negative elements present in the set. If the count of negative elements is odd, exclude that negative element (having minimum absolute value) from the subset; otherwise, consider it (since the multiplication of two negative numbers will give a positive number as output). We need to handle one more special case because 0 will never be part of the subset if at least one positive element or two negative elements are present in the subset. Rest all elements will form part of the subset.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <limits.h>**

**int min (int x, int y) {**

**return (x < y) ? x : y;**

**}**

**// Function to return the maximum product of a subset of a given array**

**int findMaxProduct(int nums[], int n)**

**{**

**// base case**

**if (n == 0) {**

**return 0;**

**}**

**// if the array contains only one element**

**if (n == 1) {**

**return nums[0];**

**}**

**int product = 1; // to store the maximum product subset**

**// stores the negative element having a minimum absolute value in the set**

**int abs\_min\_so\_far = INT\_MAX;**

**int negative = 0; // maintain the count of -ve elements in the set**

**int positive = 0; // maintain the count of +ve elements in the set**

**int contains\_zero = 0;**

**// traverse the given array**

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

**{**

**// if the current element is negative**

**if (nums[i] < 0)**

**{**

**negative++;**

**abs\_min\_so\_far = min(abs\_min\_so\_far, abs(nums[i]));**

**}**

**// if the current element is positive**

**else if (nums[i] > 0) {**

**positive++;**

**}**

**// if the current element is zero**

**if (nums[i] == 0) {**

**contains\_zero = 1;**

**}**

**else {**

**product = product \* nums[i];**

**}**

**}**

**// if an odd number of negative elements are present, exclude the negative**

**// element having a minimum absolute value from the subset**

**if (negative & 1) {**

**product = product / -abs\_min\_so\_far;**

**}**

**// special case – set contains one negative element and one or more 0's**

**if (negative == 1 && !positive && contains\_zero) {**

**product = 0;**

**}**

**// special case – set contains all 0's**

**if (!negative && !positive && contains\_zero) {**

**product = 0;**

**}**

**// return maximum product**

**return product;**

**}**

**int main()**

**{**

**int nums[] = { -6, 4, -5, 8, -10, 0, 8 };**

**int n = sizeof(nums) / sizeof(nums[0]);**

**printf("The maximum product of a subset is %d", findMaxProduct(nums, n));**

**return 0;**

**}**

**The time complexity of the above solution is O(n) and doesn’t require any extra space, where n is the size of the input.**

**26: Activity Selection Problem**

**Activity Selection Problem: Given a set of activities, along with the starting and finishing time of each activity, find the maximum number of activities performed by a single person assuming that a person can only work on a single activity at a time.  
For example,  
Input: Following set of activities  
(1, 4), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9), (6, 10), (8, 11), (8, 12), (2, 13), (12, 14)  
Output: (1, 4), (5, 7), (8, 11), (12, 14)**

**#include <iostream>**

**#include <vector>**

**#include <algorithm>**

**#include <unordered\_set>**

**using namespace std;**

**struct Pair**

**{**

**// stores start and finish time of the activities**

**int start, finish;**

**};**

**// Activity selection problem**

**void selectActivity(vector<Pair> activities) // no-ref, no-const**

**{**

**// `k` keeps track of the index of the last selected activity**

**int k = 0;**

**// set to store the selected activities index**

**unordered\_set<int> out;**

**// select 0 as the first activity**

**if (activities.size() > 0) {**

**out.insert(0);**

**}**

**// sort the activities according to their finishing time**

**sort(activities.begin(), activities.end(),**

**[](auto const &lhs, auto const &rhs) {**

**return lhs.finish < rhs.finish;**

**});**

**// start iterating from the second element of the**

**// vector up to its last element**

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

**{**

**// if the start time of the i'th activity is greater or equal**

**// to the finish time of the last selected activity, it**

**// can be included in the activities list**

**if (activities[i].start >= activities[k].finish)**

**{**

**out.insert(i);**

**k = i; // update `i` as the last selected activity**

**}**

**}**

**for (int i: out)**

**{**

**cout << "{" << activities[i].start << ", " << activities[i].finish**

**<< "}" << endl;**

**}**

**}**

**int main()**

**{**

**// List of pairs with each pair storing start time**

**// and finish time of the activities**

**vector<Pair> activities =**

**{**

**{1, 4}, {3, 5}, {0, 6}, {5, 7}, {3, 8}, {5, 9},**

**{6, 10}, {8, 11}, {8, 12}, {2, 13}, {12, 14}**

**};**

**selectActivity(activities);**

**return 0;**

**}**

The time complexity of the above solution is O(n.log(n)), where n is the total number of activities. The auxiliary space required by the program is constant.

**27: Insertion Sort Algorithm – Iterative & Recursive**

**Given an integer array, sort it using the insertion sort algorithm.**How Insertion Sort works?  
**The idea is to divide the array into two subsets – sorted subset and unsorted subset. Initially, a sorted subset consists of only one first element at index 0. Then for each iteration, insertion sort removes the next element from the unsorted subset, finds the location it belongs within the sorted subset and inserts it there. It repeats until no input elements remain. The following example explains it all:  
*i = 1* [ 3 8 5 4 1 9 -2 ]  
*i = 2* [ 3 8 5 4 1 9 -2 ]  
*i = 3* [ 3 5 8 4 1 9 -2 ]   
*i = 4* [ 3 4 5 8 1 9 -2 ]  
*i = 5* [ 1 3 4 5 8 9 -2 ]  
*i = 6* [ 1 3 4 5 8 9 -2 ]  
 [ -2 1 3 4 5 8 9 ]**

| **#include <stdio.h>**    **// Function to perform insertion sort on `arr[]`**  **void insertionSort(int arr[], int n)**  **{**  **// start from the second element (the element at index 0**  **// is already sorted)**  **for (int i = 1; i < n; i++)**  **{**  **int value = arr[i];**  **int j = i;**    **// find index `j` within the sorted subset `arr[0…i-1]`**  **// where element `arr[i]` belongs**  **while (j > 0 && arr[j - 1] > value)**  **{**  **arr[j] = arr[j - 1];**  **j--;**  **}**    **// note that the subarray `arr[j…i-1]` is shifted to**  **// the right by one position, i.e., `arr[j+1…i]`**    **arr[j] = value;**  **}**  **}**    **// Function to print `n` elements of array `arr`**  **void printArray(int arr[], int n)**  **{**  **for (int i = 0; i < n; i++) {**  **printf("%d ", arr[i]);**  **}**  **}**    **int main(void)**  **{**  **int arr[] = { 3, 8, 5, 4, 1, 9, -2 };**  **int n = sizeof(arr) / sizeof(arr[0]);**    **insertionSort(arr, n);**    **// print the sorted array**  **printArray(arr, n);**    **return 0;**  **}** |
| --- |

**We can implement the insertion sort algorithm recursively.**

**#include <stdio.h>**

**// Recursive function to perform insertion sort on subarray `arr[i…n]`**

**void insertionSort(int arr[], int i, int n)**

**{**

**int value = arr[i];**

**int j = i;**

**// find index `j` within the sorted subset `arr[0…i-1]`**

**// where element `arr[i]` belongs**

**while (j > 0 && arr[j - 1] > value)**

**{**

**arr[j] = arr[j - 1];**

**j--;**

**}**

**arr[j] = value;**

**// note that the subarray `arr[j…i-1]` is shifted to**

**// the right by one position, i.e., `arr[j+1…i]`**

**if (i + 1 <= n) {**

**insertionSort(arr, i + 1, n);**

**}**

**}**

**// Function to print `n` elements of array `arr`**

**void printArray(int arr[], int n)**

**{**

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

**printf("%d ", arr[i]);**

**}**

**}**

**int main(void)**

**{**

**int arr[] = { 3, 8, 5, 4, 1, 9, -2 };**

**int n = sizeof(arr) / sizeof(arr[0]);**

**// start from the second element (the element at index 0**

**// is already sorted)**

**insertionSort(arr, 1, n - 1);**

**// print the sorted array**

**printArray(arr, n);**

**return 0;**

**}**

**28: Selection Sort Algorithm – Iterative & Recursive**

**Given an integer array, sort it using the selection sort algorithm.**How Selection Sort works?  
**The idea is to divide the array into two subsets – sorted subset and unsorted subset. Initially, the sorted subset is empty, and the unsorted subset is the entire input list. The algorithm proceeds by finding the smallest *(or largest, depending on sorting order)* element in the unsorted subset, swapping it with the leftmost unsorted element *(putting it in sorted order)*, and moving the subset boundaries one element to the right. The following example explains it all:  
 3 5 8 4 1 9 -2  
*i = 0* -2 5 8 4 1 9 3  
*i = 1* -2 1 8 4 5 9 3  
*i = 2* -2 1 3 4 5 9 8  
*i = 3* -2 1 3 4 5 9 8  
*i = 4* -2 1 3 4 5 9 8  
*i = 5* -2 1 3 4 5 8 9**

**Following is an iterative implementation of the selection sort algorithm**

**#include <stdio.h>**

**// Utility function to swap values at two indices in an array**

**void swap(int arr[], int i, int j)**

**{**

**int temp = arr[i];**

**arr[i] = arr[j];**

**arr[j] = temp;**

**}**

**// Function to perform selection sort on `arr[]`**

**void selectionSort(int arr[], int n)**

**{**

**// run `n-1` times**

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

**{**

**// find the minimum element in the unsorted subarray `[i…n-1]`**

**// and swap it with `arr[i]`**

**int min = i;**

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

**{**

**// if `arr[j]` is less, then it is the new minimum**

**if (arr[j] < arr[min]) {**

**min = j; // update the index of minimum element**

**}**

**}**

**// swap the minimum element in subarray `arr[i…n-1]` with `arr[i]`**

**swap(arr, min, i);**

**}**

**}**

**// Function to print `n` elements of array `arr`**

**void printArray(int arr[], int n)**

**{**

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

**printf("%d ", arr[i]);**

**}**

**}**

**int main(void)**

**{**

**int arr[] = { 3, 5, 8, 4, 1, 9, -2 };**

**int n = sizeof(arr) / sizeof(arr[0]);**

**selectionSort(arr, n);**

**printArray(arr, n);**

**return 0;**

**}**

**The selection sort algorithm can be implemented recursively**

**#include <stdio.h>**

**// Utility function to swap values at two indices in an array**

**void swap(int arr[], int i, int j)**

**{**

**int temp = arr[i];**

**arr[i] = arr[j];**

**arr[j] = temp;**

**}**

**// Recursive function to perform selection sort on subarray `arr[i…n-1]`**

**void selectionSort(int arr[], int i, int n)**

**{**

**// find the minimum element in the unsorted subarray `[i…n-1]`**

**// and swap it with `arr[i]`**

**int min = i;**

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

**{**

**// if `arr[j]` is less, then it is the new minimum**

**if (arr[j] < arr[min]) {**

**min = j; // update the index of minimum element**

**}**

**}**

**// swap the minimum element in subarray `arr[i…n-1]` with `arr[i]`**

**swap(arr, min, i);**

**if (i + 1 < n) {**

**selectionSort(arr, i + 1, n);**

**}**

**}**

**// Function to print `n` elements of array `arr`**

**void printArray(int arr[], int n)**

**{**

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

**printf("%d ", arr[i]);**

**}**

**}**

**int main()**

**{**

**int arr[] = { 3, 5, 8, 4, 1, 9, -2 };**

**int n = sizeof(arr) / sizeof(arr[0]);**

**selectionSort(arr, 0, n);**

**printArray(arr, n);**

**return 0;**

**}**

**29: Bubble Sort Algorithm – Iterative & Recursive**

**Given an integer array, sort it using the bubble sort algorithm.**How Bubble Sort works?  
**Each pass of bubble sort steps through the list to be sorted compares each pair of adjacent items and swaps them if they are in the wrong order. *At the end of each pass, the next largest element will “Bubble” up to its correct position.* These passes through the list are repeated until no swaps are needed, which indicates that the list is sorted. In the worst-case, we might end up making an n-1 pass, where n is the input size.  
  3 5 8 4 1 9 -2   
*pass 1*  3 5 4 1 8 -2 9  
*pass 2*  3 4 1 5 -2 8 9  
*pass 3*  3 1 4 -2 5 8 9  
*pass 4*  1 3 -2 4 5 8 9  
*pass 5*  1 -2 3 4 5 8 9  
*pass 6* -2 1 3 4 5 8 9  
  
Following is an iterative implementation of the bubble sort algorithm**

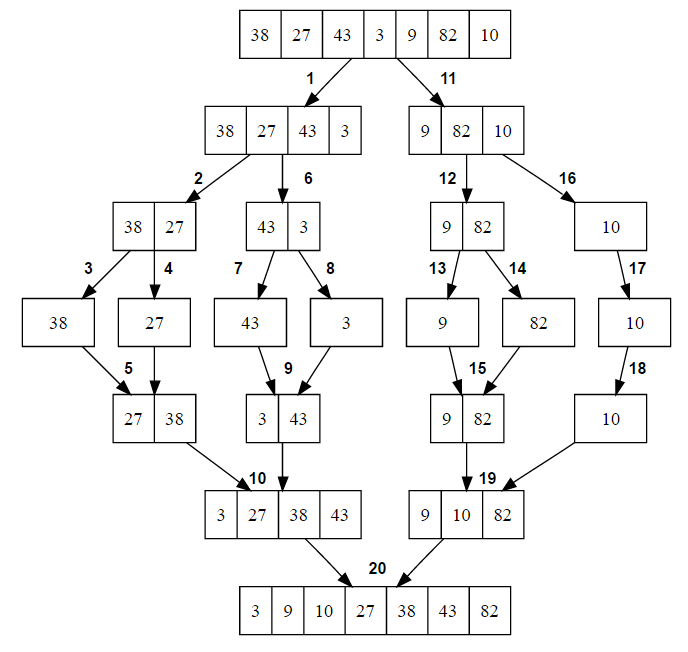
| **#include <stdio.h>**    **// Utility function to swap values at two indices in an array**  **void swap(int arr[], int i, int j)**  **{**  **int temp = arr[i];**  **arr[i] = arr[j];**  **arr[j] = temp;**  **}**    **// Function to print `n` elements of array `arr`**  **void printArray(int arr[], int n)**  **{**  **for (int i = 0; i < n; i++) {**  **printf("%d ", arr[i]);**  **}**  **}**    **// Function to perform bubble sort on a given array `arr[]`**  **void bubbleSort(int arr[], int n)**  **{**  **// `n-1` passes**  **for (int k = 0; k < n - 1; k++)**  **{**  **// last `k` items are already sorted, so the inner loop can**  **// avoid looking at the last `k` items**  **for (int i = 0; i < n - 1 - k; i++)**  **{**  **if (arr[i] > arr[i + 1]) {**  **swap(arr, i, i + 1);**  **}**  **}**    **// the algorithm can be terminated if the inner loop didn't do any swap**  **}**  **}**    **int main(void)**  **{**  **int arr[] = { 3, 5, 8, 4, 1, 9, -2 };**  **int n = sizeof(arr) / sizeof(arr[0]);**    **bubbleSort(arr, n);**  **printArray(arr, n);**    **return 0;**  **}** |
| --- |

**The bubble sort algorithm can be implemented recursively as well.**

| | **#include <stdio.h>**    **// Utility function to swap values at two indices in an array**  **void swap(int arr[], int i, int j)**  **{**  **int temp = arr[i];**  **arr[i] = arr[j];**  **arr[j] = temp;**  **}**    **// Function to print `n` elements of array `arr`**  **void printArray(int arr[], int n)**  **{**  **for (int i = 0; i < n; i++) {**  **printf("%d ", arr[i]);**  **}**  **}**    **// Recursive function to perform bubble sort on subarray `arr[i…n]`**  **void bubbleSort(int arr[], int n)**  **{**  **for (int i = 0; i < n - 1; i++)**  **{**  **if (arr[i] > arr[i + 1]) {**  **swap(arr, i, i + 1);**  **}**  **}**    **if (n - 1 > 1) {**  **bubbleSort(arr, n - 1);**  **}**  **}**    **int main(void)**  **{**  **int arr[] = { 3, 5, 8, 4, 1, 9, -2 };**  **int n = sizeof(arr) / sizeof(arr[0]);**    **bubbleSort(arr, n);**  **printArray(arr, n);**    **return 0;**  **}** | | --- | |
| --- | --- |

**30: Merge Sort Algorithm**

**Given an integer array, sort it using the merge sort algorithm.**How Merge sort works?  
**Merge sort is a** [**Divide and Conquer**](https://www.techiedelight.com/divide-and-conquer-interview-questions/) **algorithm. Like all divide-and-conquer algorithms, merge sort divides a large array into two smaller subarrays and then recursively sort the subarrays. Basically, two steps are involved in the whole process:  
  
 - Divide the unsorted array into n subarrays, each of size 1 (an array of size 1 is considered sorted).  
 - Repeatedly merge subarrays to produce new sorted subarrays until only 1 subarray is left, which would be our sorted array.  
  
The** [**following diagram**](https://en.wikipedia.org/wiki/File:Merge_sort_algorithm_diagram.svg) **represents a top-down view of the recursive merge sort algorithm used to sort an 7-element integer array:**

****

| **#include <stdio.h>**  **#include <stdlib.h>**  **#include <time.h>**    **#define N 15**    **// Merge two sorted subarrays `arr[low … mid]` and `arr[mid+1 … high]`**  **void Merge(int arr[], int aux[], int low, int mid, int high)**  **{**  **int k = low, i = low, j = mid + 1;**    **// While there are elements in the left and right runs**  **while (i <= mid && j <= high)**  **{**  **if (arr[i] <= arr[j]) {**  **aux[k++] = arr[i++];**  **}**  **else {**  **aux[k++] = arr[j++];**  **}**  **}**    **// Copy remaining elements**  **while (i <= mid) {  aux[k++] = arr[i++];**  **}**    **// No need to copy the second half (since the remaining items**  **// are already in their correct position in the auxiliary array)**    **// copy back to the original array to reflect sorted order**  **for (int i = low; i <= high; i++) {**  **arr[i] = aux[i];**  **}**  **}**    **// Sort array `arr[low…high]` using auxiliary array `aux`**  **void mergesort(int arr[], int aux[], int low, int high)**  **{**  **// Base case**  **if (high == low) { // if run size == 1**  **return;**  **}**    **// find midpoint**  **int mid = (low + ((high - low) >> 1));**    **// recursively split runs into two halves until run size == 1,**  **// then merge them and return up the call chain**    **mergesort(arr, aux, low, mid); // split/merge left half**  **mergesort(arr, aux, mid + 1, high); // split/merge right half**    **Merge(arr, aux, low, mid, high); // merge the two half runs.**  **}**    **// Function to check if arr is sorted in ascending order or not**  **int isSorted(int arr[])**  **{**  **int prev = arr[0];**  **for (int i = 1; i < N; i++)**  **{**  **if (prev > arr[i])**  **{**  **printf("MergeSort Fails!!");**  **return 0;**  **}**  **prev = arr[i];**  **}**    **return 1;**  **}**    **// Implementation of merge sort algorithm in C**  **int main(void)**  **{**  **int arr[N], aux[N];**  **srand(time(NULL));**    **// generate random input of integers**  **for (int i = 0; i < N; i++) {**  **aux[i] = arr[i] = (rand() % 100) - 50;**  **}**    **// sort array `arr` using auxiliary array `aux`**  **mergesort(arr, aux, 0, N - 1);**    **if (isSorted(arr))**  **{**  **for (int i = 0; i < N; i++) {**  **printf("%d ", arr[i]);**  **}**  **}**    **return 0;**  **}** |
| --- |

**31: Counting Sort Algorithm**

**Given a collection of n items, each of which has a non-negative integer key whose maximum value is at most k, effectively sort it using the counting sort algorithm.**Algorithm  
**The algorithm loops over the items, computing a histogram of each key’s number of times within the input collection. It then performs a prefix sum computation to determine, for each key, the starting position in the output array of the items having that key. Finally, it loops over the items again, moving each item into its sorted position in the output array.  
  
The algorithm can be implemented as below in C, Java, and Python. In the below code:  
  
 - After the first for-loop, freq[i] stores the total number of items with a key equal to i.  
 - After the second for-loop, it instead stores the total number of items with a key less than i, which is the same as the first index at which an item with key i should be stored in the output array.  
 - Throughout the third loop, freq[i] always stores the next position in the output array into which an item with key i should be stored, so each item is moved to its correct position in the output array**

| **#include <stdio.h>**  **#include <string.h>**    **/\***  **`arr` ——> the input integer array to be sorted**  **`n` ——> the length of the input array**  **`k` ——> a number such that all integers are in range `0…k-1`**  **\*/**  **void countsort(int arr[], int n, int k)**  **{**  **// create an integer array of size `n` to store the sorted array**  **int output[n];**    **// create an integer array of size `k + 1`, initialized by all zero**  **int freq[k + 1];**  **memset(freq, 0, sizeof(freq));**    **// 1. Using the value of each item in the input array as an index,**  **// store each integer's count in `freq[]`**  **for (int i = 0; i < n; i++) {**  **freq[arr[i]]++;**  **}**    **// 2. Calculate the starting index for each integer**  **int total = 0;**  **for (int i = 0; i < k + 1; i++)**  **{**  **int oldCount = freq[i];**  **freq[i] = total;**  **total += oldCount;**  **}**    **// 3. Copy to the output array, preserving the order of inputs with equal keys**  **for (int i = 0; i < n; i++)**  **{**  **output[freq[arr[i]]] = arr[i];**  **freq[arr[i]]++;**  **}**    **// copy the output array back to the input array**  **for (int i = 0; i < n; i++) {**  **arr[i] = output[i];**  **}**  **}**    **int main()**  **{**  **int arr[] = { 4, 2, 10, 10, 1, 4, 2, 1, 10 };**  **int n = sizeof(arr) / sizeof(arr[0]);**    **// range of array elements**  **int k = 10;**    **countsort(arr, n, k);**    **for (int i = 0; i < n; i++) {**  **printf("%d ", arr[i]);**  **}**    **return 0;**  **}** |
| --- |

**32: Binary Search Algorithm – Iterative and Recursive Implementation**

**Given a sorted array of n integers and a target value, determine if the target exists in the array in logarithmic time using the binary search algorithm. If target exists in the array, print the index of it.  
For example,**

**Input:  
nums[] = [2, 3, 5, 7, 9]  
target = 7  
Output: Element found at index 3  
  
Input:  
nums[] = [1, 4, 5, 8, 9]  
target = 2  
Output: Element not found**

| **#include <stdio.h>**    **// Iterative implementation of the binary search algorithm to return**  **// the position of `target` in array `nums` of size `n`**  **int binarySearch(int nums[], int n, int target)**  **{**  **// search space is nums[low…high]**  **int low = 0, high = n - 1;**    **// loop till the search space is exhausted**  **while (low <= high)**  **{**  **// find the mid-value in the search space and**  **// compares it with the target**    **int mid = (low + high)/2; // overflow can happen**  **// int mid = low + (high - low)/2;**  **// int mid = high - (high - low)/2;**    **// target value is found**  **if (target == nums[mid]) {**  **return mid;**  **}**    **// if the target is less than the middle element, discard all elements**  **// in the right search space, including the middle element**  **else if (target < nums[mid]) {**  **high = mid - 1;**  **}**    **// if the target is more than the middle element, discard all elements**  **// in the left search space, including the middle element**  **else {**  **low = mid + 1;**  **}**  **}**    **// target doesn't exist in the array**  **return -1;**  **}**    **int main(void)**  **{**  **int nums[] = { 2, 5, 6, 8, 9, 10 };**  **int target = 5;**    **int n = sizeof(nums)/sizeof(nums[0]);**  **int index = binarySearch(nums, n, target);**    **if (index != -1) {**  **printf("Element found at index %d", index);**  **}**  **else {**  **printf("Element not found in the array");**  **}**    **return 0;**  **}** |
| --- |

## Recursive Implementation

**We can easily convert the above iterative version of the binary search algorithm into a recursive one. The algorithm can be implemented recursively as follows**

| **#include <stdio.h>**    **// Recursive implementation of the binary search algorithm to return**  **// the position of `target` in subarray nums[low…high]**  **int binarySearch(int nums[], int low, int high, int target)**  **{**  **// Base condition (search space is exhausted)**  **if (low > high) {**  **return -1;**  **}**    **// find the mid-value in the search space and**  **// compares it with the target**    **int mid = (low + high)/2; // overflow can happen**  **// int mid = low + (high - low)/2;**    **// Base condition (target value is found)**  **if (target == nums[mid]) {**  **return mid;**  **}**    **// discard all elements in the right search space,**  **// including the middle element**  **else if (target < nums[mid]) {**  **return binarySearch(nums, low, mid - 1, target);**  **}**    **// discard all elements in the left search space,**  **// including the middle element**  **else {**  **return binarySearch(nums, mid + 1, high, target);**  **}**  **}**    **int main(void)**  **{**  **int nums[] = { 2, 5, 6, 8, 9, 10 };**  **int target = 5;**    **int n = sizeof(nums)/sizeof(nums[0]);**    **int low = 0, high = n - 1;**  **int index = binarySearch(nums, low, high, target);**    **if (index != -1) {**  **printf("Element found at index %d", index);**  **}**  **else {**  **printf("Element not found in the array");**  **}**    **return 0;**  **}** |
| --- |

**33: Exponential search**

**Given a sorted array of n integers and a target value, determine if the target exists in the array or not in logarithmic time. If the target exists in the array, return the index of it.  
  
For example,**

**Input: A[] = [2, 3, 5, 7, 9]  
target = 7  
Output: Element found at index 3  
  
Input: A[] = [1, 4, 5, 8, 9]  
target = 2  
Output: Element not found**

| **#include <stdio.h>**    **// Utility function to find a minimum of two numbers**  **int min(int x, int y) {**  **return (x < y) ? x : y;**  **}**    **// Binary search algorithm to return the position of key `x` in subarray A[low…high]**  **int binarySearch(int A[], int low, int high, int x)**  **{**  **// base condition (search space is exhausted)**  **if (low > high) {**  **return -1;**  **}**    **// find the mid-value in the search space and**  **// compares it with the key**    **int mid = (low + high)/2; // overflow can happen**  **// int mid = low + (high - low)/2;**    **// base condition (a key is found)**  **if (x == A[mid]) {**  **return mid;**  **}**    **// discard all elements in the right search space,**  **// including the middle element**  **else if (x < A[mid]) {**  **return binarySearch(A, low, mid - 1, x);**  **}**    **// discard all elements in the left search space,**  **// including the middle element**  **else {**  **return binarySearch(A, mid + 1, high, x);**  **}**  **}**    **// Returns the position of key `x` in a given array `A` of length `n`**  **int exponentialSearch(int A[], int n, int x)**  **{**  **// base case**  **if (n == 0) {**  **return -1;**  **}**    **int bound = 1;**    **// find the range in which key `x` would reside**  **while (bound < n && A[bound] < x) {**  **bound \*= 2; // calculate the next power of 2**  **}**    **// call binary search on A[bound/2 … min(bound, n-1)]**  **return binarySearch(A, bound/2, min(bound, n - 1), x);**  **}**    **// Exponential search algorithm**  **int main(void)**  **{**  **int A[] = {2, 5, 6, 8, 9, 10};**  **int target = 9;**    **int n = sizeof(A)/sizeof(A[0]);**  **int index = exponentialSearch(A, n, target);**    **if (index != -1) {**  **printf("Element found at index %d", index);**  **}**  **else {**  **printf("Element not found in the array");**  **}**    **return 0;**  **}** |
| --- |

**34: Interpolation search**

**Given a sorted integer array and a target, determine if the target exists in the array or not using an interpolation search algorithm. If the target exists in the array, return the index of it.  
  
For example,**

**Input:  
arr[] = [2, 3, 5, 7, 9]  
target = 7  
Output: Element found at index 3  
  
Input:  
arr[] = [1, 4, 5, 8, 9]  
target = 2  
Output: Element not found**

**#include <stdio.h>**

**// Function to determine if target exists in a sorted array `A` or not**

**// using an interpolation search algorithm**

**int interpolationSearch(int A[], int n, int target)**

**{**

**// base case**

**if (n == 0) {**

**return -1;**

**}**

**// search space is A[low…high]**

**int low = 0, high = n - 1, mid;**

**while (A[high] != A[low] && target >= A[low] && target <= A[high])**

**{**

**// estimate mid**

**mid = low + ((target - A[low]) \* (high - low) / (A[high] - A[low]));**

**// target value is found**

**if (target == A[mid]) {**

**return mid;**

**}**

**// discard all elements in the right search space, including the middle element**

**else if (target < A[mid]) {**

**high = mid - 1;**

**}**

**// discard all elements in the left search space, including the middle element**

**else {**

**low = mid + 1;**

**}**

**}**

**// if a target is found**

**if (target == A[low]) {**

**return low;**

**}**

**// target doesn't exist in the array**

**else {**

**return -1;**

**}**

**}**

**int main(void)**

**{**

**int A[] = {2, 5, 6, 8, 9, 10};**

**int target = 5;**

**int n = sizeof(A)/sizeof(A[0]);**

**int index = interpolationSearch(A, n, target);**

**if (index != -1) {**

**printf("Element found at index %d", index);**

**}**

**else {**

**printf("Element not found in the array");**

**}**

**return 0;**

**}**

**35: Longest Alternating Subarray Problem**

**Given an array containing positive and negative elements, find a subarray with alternating positive and negative elements, and in which the subarray is as long as possible.**

**#include <iostream>**

**#include <vector>**

**#include <algorithm>**

**using namespace std;**

**// Utility function to print subarray `nums[i…j]`**

**void printSubarray(vector<int> const &nums, int i, int j)**

**{**

**cout << "[";**

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

**cout << nums[k] << ", ";**

**}**

**cout << nums[j] << "]";**

**}**

**// Function to find the length of the longest subarray with alternating**

**// positive and negative elements**

**void findLongestSubarray(vector<int> const &nums)**

**{**

**// base case**

**if (nums.size() == 0) {**

**return;**

**}**

**// stores length of longest alternating subarray found so far**

**int max\_len = 1;**

**// stores ending index of longest alternating subarray found so far**

**int ending\_index = 0;**

**// stores length of alternating subarray ending at the current position**

**int curr\_len = 1;**

**// traverse the given array starting from the second index**

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

**{**

**// if the current element has an opposite sign than the previous element**

**if (nums[i] \* nums[i - 1] < 0)**

**{**

**// include the current element in the longest alternating subarray ending**

**// at the previous index**

**curr\_len++;**

**// update result if the current subarray length is found to be greater**

**if (curr\_len > max\_len)**

**{**

**max\_len = curr\_len;**

**ending\_index = i;**

**}**

**}**

**// reset length if the current element has the same sign as the previous**

**// element**

**else {**

**curr\_len = 1;**

**}**

**}**

**cout << "The longest alternating subarray is ";**

**printSubarray(nums, (ending\_index - max\_len + 1), ending\_index);**

**}**

**int main()**

**{**

**vector<int> nums = { 1, -2, 6, 4, -3, 2, -4, -3 };**

**findLongestSubarray(nums);**

**return 0;**

**}**

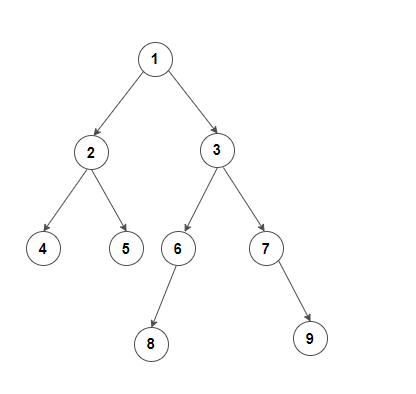
**36: Determine if a binary tree can be converted to another by doing any number of swaps of children**

**Given a binary tree, write an efficient algorithm to determine if it can be converted into another binary tree by doing any number of swaps of its right and left branches.**

| **#include <iostream>**  **using namespace std;**    **// Data structure to store a binary tree node**  **struct Node**  **{**  **int data;**  **Node \*left, \*right;**    **Node(int data)**  **{**  **this->data = data;**  **this->left = this->right = nullptr;**  **}**  **};**    **// Function to determine if two given binary trees can be transformed**  **// into each other by doing any number of left and right subtree swaps**  **bool equal(Node\* X, Node\* Y)**  **{**  **// base case: both trees are the same (handles the case when both trees are empty)**  **if (X == Y) {**  **return true;**  **}**    **return (X && Y) && (X->data == Y->data) &&**  **((equal(X->left, Y->left) && equal(X->right, Y->right)) ||**  **(equal(X->right, Y->left) && equal(X->left, Y->right)));**  **}**    **int main()**  **{**  **// construct the first tree**  **Node\* X = nullptr;**    **X = new Node(6);**  **X->left = new Node(3);**  **X->right = new Node(8);**  **X->left->left = new Node(1);**  **X->left->right = new Node(7);**  **X->right->left = new Node(4);**  **X->right->right = new Node(2);**  **X->right->left->left = new Node(1);**  **X->right->left->right = new Node(7);**  **X->right->right->right = new Node(3);**    **// construct the second tree**  **Node\* Y = nullptr;**    **Y = new Node(6);**  **Y->left = new Node(8);**  **Y->right = new Node(3);**  **Y->left->left = new Node(2);**  **Y->left->right = new Node(4);**  **Y->right->left = new Node(7);**  **Y->right->right = new Node(1);**  **Y->left->left->left = new Node(3);**  **Y->left->right->left = new Node(1);**  **Y->left->right->right = new Node(7);**    **if (equal(X, Y)) {**  **cout << "Binary tree can be converted";**  **}**  **else {**  **cout << "Binary tree cannot be converted";**  **}**    **return 0;**  **}** |
| --- |

**37:Print all paths from the root to leaf nodes of a binary tree**

**Given a binary tree, write an efficient algorithm to print all paths from the root node to every leaf node in it.**

**For example, consider the following binary tree:  
  
The binary tree has four root-to-leaf paths:  
1 —> 2 —> 4  
1 —> 2 —> 5  
1 —> 3 —> 6 —> 8  
1 —> 3 —> 7 —> 9**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**// Data structure to store a binary tree node**

**struct Node**

**{**

**int data;**

**Node \*left, \*right;**

**Node(int data)**

**{**

**this->data = data;**

**this->left = this->right = nullptr;**

**}**

**};**

**// Function to check if a given node is a leaf node or not**

**bool isLeaf(Node\* node) {**

**return (node->left == nullptr && node->right == nullptr);**

**}**

**// Recursive function to find paths from the root node to every leaf node**

**void printRootToLeafPaths(Node\* node, vector<int> &path)**

**{**

**// base case**

**if (node == nullptr) {**

**return;**

**}**

**// include the current node to the path**

**path.push\_back(node->data);**

**// if a leaf node is found, print the path**

**if (isLeaf(node))**

**{**

**for (int data: path) {**

**cout << data << " ";**

**}**

**cout << endl;**

**}**

**// recur for the left and right subtree**

**printRootToLeafPaths(node->left, path);**

**printRootToLeafPaths(node->right, path);**

**// backtrack: remove the current node after the left, and right subtree are done**

**path.pop\_back();**

**}**

**// The main function to print paths from the root node to every leaf node**

**void printRootToLeafPaths(Node\* node)**

**{**

**// vector to store root-to-leaf path**

**vector<int> path;**

**printRootToLeafPaths(node, path);**

**}**

**int main()**

**{**

**/\* Construct the following tree**

**1**

**/ \**

**/ \**

**2 3**

**/ \ / \**

**4 5 6 7**

**/ \**

**8 9**

**\*/**

**Node\* root = new Node(1);**

**root->left = new Node(2);**

**root->right = new Node(3);**

**root->left->left = new Node(4);**

**root->left->right = new Node(5);**

**root->right->left = new Node(6);**

**root->right->right = new Node(7);**

**root->right->left->left = new Node(8);**

**root->right->right->right = new Node(9);**

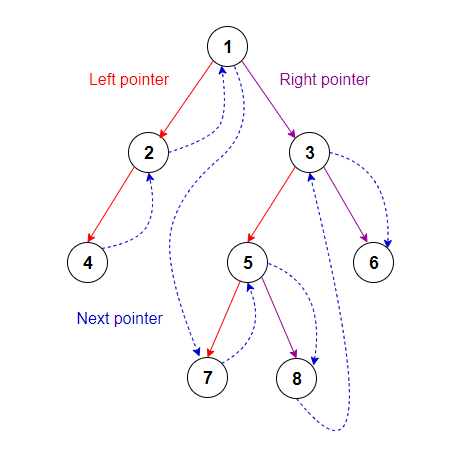
**// print all root-to-leaf paths**

**printRootToLeafPaths(root);**

**return 0;**

**}**

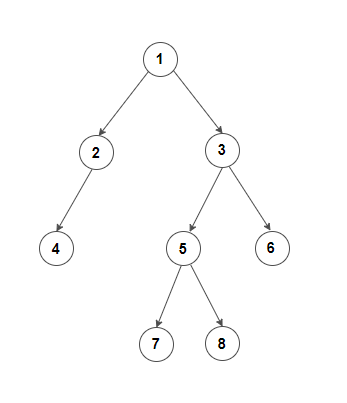
**38: Set next pointer to the inorder successor of all nodes in a binary tree**

**Given a binary tree where each node has one extra pointer next, set it to the inorder successor for all binary tree nodes.  
For example, consider the following tree. Here, the blue dotted line represents the next pointer for each node.  
**

**The inorder successor of node 4 is 2  
The inorder successor of node 2 is 1  
The inorder successor of node 1 is 7  
The inorder successor of node 7 is 5  
The inorder successor of node 5 is 8  
The inorder successor of node 8 is 3  
The inorder successor of node 3 is 6  
The inorder successor of node 6 is null**

| **#include <iostream>**  **using namespace std;**    **// Data structure to store a binary tree node**  **struct Node**  **{**  **int data;**  **Node \*left, \*right, \*next;**    **Node(int data)**  **{**  **this->data = data;**  **this->left = this->right = this->next = nullptr;**  **}**  **};**    **// Function to set the next pointer of all nodes in a binary tree.**  **// curr —> current node**  **// prev —> previously processed node (passed by reference)**  **void setNextNode(Node\* curr, Node\* &prev)**  **{**  **// return if the tree is empty**  **if (curr == nullptr) {**  **return;**  **}**    **// recur for the left subtree**  **setNextNode(curr->left, prev);**    **// set the previous node's next pointer to the current node**  **if (prev != nullptr) {**  **prev->next = curr;**  **}**    **// update the previous node to the current node**  **prev = curr;**    **// recur for the right subtree**  **setNextNode(curr->right, prev);**  **}**    **// Function to print inorder successor of all nodes of**  **// binary tree using the next pointer**  **void inorderSuccessor(Node\* root)**  **{**  **Node\* prev = nullptr;**  **Node\* curr = root;**    **// set next pointer of all nodes**  **setNextNode(curr, prev);**    **// go to the leftmost node**  **curr = root;**  **while (curr->left != nullptr) {**  **curr = curr->left;**  **}**    **// print inorder successor of all nodes**  **while (curr->next)**  **{**  **cout << "The inorder successor of " << curr->data << " is "**  **<< curr->next->data << endl;**  **curr = curr->next;**  **}**  **}**    **int main()**  **{**  **/\* Construct the following tree**  **1**  **/ \**  **/ \**  **2 3**  **/ / \**  **/ / \**  **4 5 6**  **/ \**  **/ \**  **7 8**  **\*/**    **Node\* root = new Node(1);**  **root->left = new Node(2);**  **root->right = new Node(3);**  **root->left->left = new Node(4);**  **root->right->left = new Node(5);**  **root->right->right = new Node(6);**  **root->right->left->left = new Node(7);**  **root->right->left->right = new Node(8);**    **inorderSuccessor(root);**    **return 0;**  **}** |
| --- |

**39: Find difference between sum of all nodes present at odd and even levels in a binary tree**

**Given a binary tree, calculate the difference between the sum of all nodes present at odd levels and the sum of all nodes present at even level.  
For example, consider the following binary tree. The required difference is:  
(1 + 4 + 5 + 6) - (2 + 3 + 7 + 8) = -4  
**

| **#include <iostream>**  **using namespace std;**  **// Data structure to store a binary tree node**  **struct Node {  int data;**  **Node \*left, \*right;**  **Node(int data)**  **{**  **this->data = data;**  **this->left = this->right = nullptr;**  **}**  **};**    **// Helper function**  **void findDiff(Node\* root, int &diff, int level)**  **{**  **// base case**  **if (root == nullptr) {**  **return;**  **}**    **// if the current level is odd**  **if (level & 1) {**  **diff += root->data;**  **}**  **// if the current level is even**  **else {**  **diff -= root->data;**  **}**    **// recur for the left and right subtree**  **findDiff(root->left, diff, level + 1);**  **findDiff(root->right, diff, level + 1);**  **}**    **// Function to calculate the difference between the sum of all nodes present**  **// at odd levels and the sum of all nodes present at even level**  **int findDiff(Node\* root)**  **{**  **int diff = 0;**  **findDiff(root, diff, 1);**    **return diff;**  **}**    **int main()**  **{**  **/\* Construct the following tree**  **1**  **/ \**  **/ \**  **2 3**  **/ / \**  **/ / \**  **4 5 6**  **/ \**  **/ \**  **7 8**  **\*/**    **Node\* root = new Node(1);**  **root->left = new Node(2);**  **root->right = new Node(3);**  **root->left->left = new Node(4);**  **root->right->left = new Node(5);**  **root->right->right = new Node(6);**  **root->right->left->left = new Node(7);**  **root->right->left->right = new Node(8);**    **cout << findDiff(root);**    **return 0;**  **}** |
| --- |

**40: Clone a Binary Tree**

**Given a binary tree, efficiently create copy of it.**

**#include <iostream>**

**using namespace std;**

**// A Binary Tree Node**

**class Node**

**{**

**public:**

**int data;**

**Node\* left, \*right;**

**Node(int data)**

**{**

**this->data = data;**

**this->right = this->left = nullptr;**

**}**

**};**

**// Function to print the inorder traversal on a given binary tree**

**void inorder(Node\* root)**

**{**

**if (root == nullptr) {**

**return;**

**}**

**// recur for the left subtree**

**inorder(root->left);**

**// print the current node's data**

**cout << root->data << " ";**

**// recur for the right subtree**

**inorder(root->right);**

**}**

**// Recursive function to clone a binary tree**

**Node\* cloneBinaryTree(Node\* root)**

**{**

**// base case**

**if (root == nullptr) {**

**return nullptr;**

**}**

**// create a new node with the same data as the root node**

**Node\* root\_copy = new Node(root->data);**

**// clone the left and right subtree**

**root\_copy->left = cloneBinaryTree(root->left);**

**root\_copy->right = cloneBinaryTree(root->right);**

**// return cloned root node**

**return root\_copy;**

**}**

**int main()**

**{**

**Node\* root = new Node(1);**

**root->left = new Node(2);**

**root->right = new Node(3);**

**root->left->left = new Node(4);**

**root->left->right = new Node(5);**

**root->right->left = new Node(6);**

**root->right->right = new Node(7);**

**Node\* clone = cloneBinaryTree(root);**

**cout << "Inorder traversal of the cloned tree: ";**

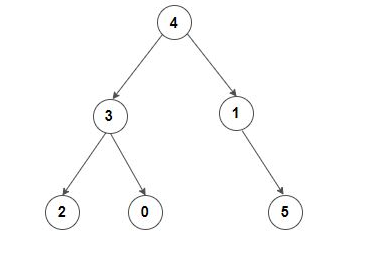
**inorder(clone);**

**return 0;**

**}**

**41: Construct an ancestor matrix from a binary tree**

**Given a binary tree whose nodes are labeled from 0 to N-1, construct an N × N ancestor matrix. An ancestor matrix is a boolean matrix, whose cell (i, j) is true if i is an ancestor of j in the binary tree.  
For example, consider the following binary tree:**

**  
The output should be the following ancestor matrix  
 0 0 0 0 0 0  
 0 0 0 0 0 1  
 0 0 0 0 0 0  
 1 0 1 0 0 0  
 1 1 1 1 0 1  
 0 0 0 0 0 0**

| **#include <iostream>**  **#include <vector>**  **#include <unordered\_set>**  **using namespace std;**    **// Data structure to store a binary tree node**  **struct Node**  **{**  **int data;**  **Node\* left, \*right;**    **// Constructor**  **Node(int data)**  **{**  **this->data = data;**  **this->left = this->right = nullptr;**  **}**  **};**    **// Recursive function to calculate the size of the binary tree**  **int size(Node\* root)**  **{**  **// base case**  **if (root == nullptr) {**  **return 0;**  **}**    **return size(root->left) + 1 + size(root->right);**  **}**    **// Traverse the tree in a preorder fashion and update the ancestors of**  **// all nodes in the boolean ancestor matrix**  **void constructAncestorMatrix(Node\* root, unordered\_set<Node\*> &ancestors,**  **vector<vector<bool>> &ancestorMatrix)**  **{**  **// base case**  **if (root == nullptr) {**  **return;**  **}**    **// update all ancestors of the current node**  **for (Node\* node: ancestors) {**  **ancestorMatrix[node->data][root->data] = true;**  **}**    **// add the current node to the set of ancestors**  **ancestors.insert(root);**    **// recur for the left and right subtree**  **constructAncestorMatrix(root->left, ancestors, ancestorMatrix);**  **constructAncestorMatrix(root->right, ancestors, ancestorMatrix);**    **// remove the current node from the set of ancestors since all**  **// descendants of the current node are already processed**  **ancestors.erase(root);**  **}**    **// Function to construct an ancestor matrix from a given binary tree**  **vector<vector<bool>> constructAncestorMatrix(Node\* root)**  **{**  **// calculate the size of the binary tree**  **int n = size(root);**    **// create an ancestor matrix of size `n × n`, initialized by false**  **vector<vector<bool>> ancestorMatrix(n, vector<bool>(n));**    **// stores ancestors of a node**  **unordered\_set<Node\*> ancestors;**    **// construct the ancestor matrix**  **constructAncestorMatrix(root, ancestors, ancestorMatrix);**    **return ancestorMatrix;**  **}**    **int main()**  **{**  **/\* Construct the following tree**  **4**  **/ \**  **3 1**  **/ \ \**  **2 0 5**  **\*/**    **Node\* root = new Node(4);**  **root->left = new Node(3);**  **root->right = new Node(1);**  **root->left->left = new Node(2);**  **root->left->right = new Node(0);**  **root->right->right = new Node(5);**    **// construct the ancestor matrix**  **vector<vector<bool>> ancestorMatrix = constructAncestorMatrix(root);**    **// print the ancestor matrix**  **for (auto const &row: ancestorMatrix)**  **{**  **for (auto val: row) {**  **cout << val << " ";**  **}**  **cout << endl;**  **}**    **return 0;**  **}** |
| --- |

**42: Check if binary representation of a number is palindrome or not**

**Check if the binary representation of a positive number is palindrome or not.  
For example, 101, 11, 11011, 1001001 are palindromes. 100, 110, 1011, etc., are not palindromes.**

**#include <iostream>**

**using namespace std;**

**// Returns true if the binary representation of `n` is a palindrome**

**bool isPalindrome(unsigned n)**

**{**

**// `rev` stores reverse of a binary representation of `n`**

**unsigned rev = 0;**

**// do till all bits of `n` are processed**

**unsigned k = n;**

**while (k)**

**{**

**// add the rightmost bit to `rev`**

**rev = (rev << 1) | (k & 1);**

**k = k >> 1; // drop last bit**

**}**

**// Returns true if `reverse(n)` is the same as `n`**

**return n == rev;**

**}**

**int main()**

**{**

**int n = 9; // 1001**

**if (isPalindrome(n)) {**

**cout << n << " is a palindrome";**

**}**

**else {**

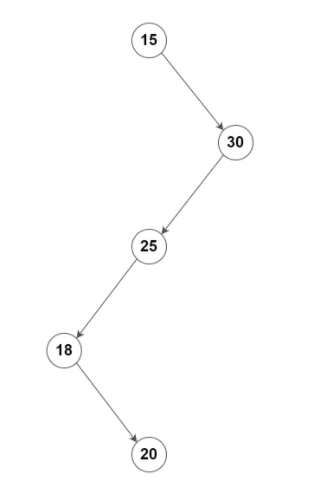
**cout << n << " is not a palindrome";**

**}**

**return 0;**

**}**

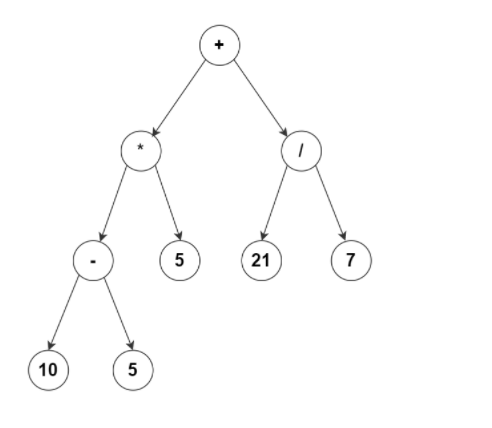
**43: Check if each node of a binary tree has exactly one child**

**Given a binary tree, check if each node has exactly one child or not. In other words, check whether the binary tree is skewed or not.  
For example, the following binary tree is also skewed:  
**

| **#include <iostream>**  **using namespace std;**    **// Data structure to store a binary tree node**  **struct Node**  **{**  **int key;**  **Node \*left, \*right;**    **Node(int key)**  **{**  **this->key = key;**  **this->left = this->right = nullptr;**  **}**  **};**    **// Recursive function to calculate the size of a binary tree**  **int getSize(Node\* root)**  **{**  **// Base case: empty tree has size 0**  **if (root == nullptr) {**  **return 0;**  **}**    **// recur for the left and right subtree**  **return 1 + getSize(root->left) + getSize(root->right);**  **}**    **// Recursive function to calculate the height of a binary tree**  **int findHeight(Node\* root)**  **{**  **// Base case: an empty tree has a height of 0**  **if (root == nullptr) {**  **return 0;**  **}**    **// recur for the left and right subtree and consider the maximum depth**  **return 1 + max(findHeight(root->left), findHeight(root->right));**  **}**    **// Function to check if each node of a binary tree has exactly one child**  **bool isSkewedTree(Node\* root) {**  **return getSize(root) == findHeight(root);**  **}**    **int main()**  **{**  **Node\* root = new Node(15);**  **root->right = new Node(30);**  **root->right->left = new Node(25);**  **root->right->left->left = new Node(18);**  **root->right->left->left->right = new Node(20);**    **bool isSkewed = isSkewedTree(root);**  **if (isSkewed) {**  **cout << "The binary tree is skewed";**  **}**  **else {**  **cout << "The binary tree is not skewed";**  **}**    **return 0;**  **}** |
| --- |

**44:Evaluate a Binary Expression Tree**

**Evaluate a given binary expression tree representing algebraic expressions.  
A binary expression tree is a** [**binary tree**](https://www.techiedelight.com/binary-tree-interview-questions/)**, where the operators are stored in the tree’s internal nodes, and the leaves contain constants.  
Assume that each node of the binary expression tree has zero or two children. The supported operators are +(addition), −(subtraction), \*(multiplication), ÷(division) and ^(exponentiation).  
For example, the value of the following expression tree is 28:**

****

**#include <iostream>**

**#include <string>**

**using namespace std;**

**// Data structure to store a binary tree node**

**struct Node**

**{**

**string val;**

**Node \*left, \*right;**

**Node(string val)**

**{**

**this->val = val;**

**this->left = this->right = nullptr;**

**}**

**};**

**// Utility function to check if a given node is a leaf node**

**bool isLeaf(Node\* node) {**

**return node->left == nullptr && node->right == nullptr;**

**}**

**// Function to apply an operator 'op' to values 'x' and 'y' and return the result**

**double process(string op, double x, double y)**

**{**

**if (op == "+") { return x + y; }**

**if (op == "-") { return x - y; }**

**if (op == "\*") { return x \* y; }**

**if (op == "/") { return x / y; }**

**return 0;**

**}**

**// Recursive function to evaluate a binary expression tree**

**double evaluate(Node\* root)**

**{**

**// base case: invalid input**

**if (root == nullptr) {**

**return 0;**

**}**

**// the leaves of a binary expression tree are operands**

**if (isLeaf(root)) {**

**return stod(root->val);**

**}**

**// recursively evaluate the left and right subtree**

**double x = evaluate(root->left);**

**double y = evaluate(root->right);**

**// apply the operator at the root to the values obtained in the previous step**

**return process(root->val, x, y);**

**}**

**int main()**

**{**

**Node\* root = new Node("+");**

**root->left = new Node("\*");**

**root->right = new Node("/");**

**root->left->left = new Node("-");**

**root->left->right = new Node("5");**

**root->right->left = new Node("21");**

**root->right->right = new Node("7");**

**root->left->left->left = new Node("10");**

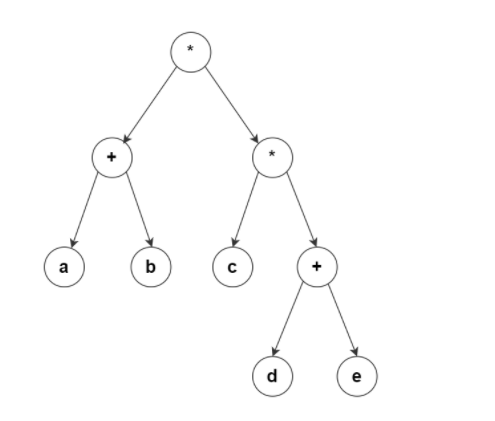
**root->left->left->right = new Node("5");**

**cout << "The value of the expression tree is " << evaluate(root) << endl;**

**return 0;**

**}**

**45:Construction of an expression tree**

**Construct an expression tree from a given postfix notation and print the infix notation.  
The** [**binary expression tree**](https://en.wikipedia.org/wiki/Binary_expression_tree) **is a** [**binary tree**](https://www.techiedelight.com/binary-tree-interview-questions/) **whose leaves are operands, such as constants or variable names, and the other nodes contain operators.  
For example, the postfix notation a b + c d e + \* \* results in the following expression tree. The corresponding infix notation is (a+b)\*(c\*(d+e)) which can be produced by traversing the expression tree in an** [**inorder fashion**](https://www.techiedelight.com/inorder-tree-traversal-iterative-recursive/)**. However, an opening and closing parenthesis must be added at the beginning and end of each expression (every subtree represents a subexpression).  
**

**#include <iostream>**

**#include <stack>**

**#include <string>**

**using namespace std;**

**// Data structure to store a binary tree node**

**struct Node**

**{**

**char data;**

**Node \*left, \*right;**

**Node(char data)**

**{**

**this->data = data;**

**this->left = this->right = nullptr;**

**};**

**Node(char data, Node \*left, Node \*right)**

**{**

**this->data = data;**

**this->left = left;**

**this->right = right;**

**};**

**};**

**// Function to check if a given token is an operator**

**bool isOperator(char c) {**

**return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');**

**}**

**// Print the postfix expression for an expression tree**

**void postorder(Node\* root)**

**{**

**if (root == nullptr) {**

**return;**

**}**

**postorder(root->left);**

**postorder(root->right);**

**cout << root->data;**

**}**

**// Print the infix expression for an expression tree**

**void inorder(Node\* root)**

**{**

**if (root == nullptr) {**

**return;**

**}**

**// if the current token is an operator, print open parenthesis**

**if (isOperator(root->data)) {**

**cout << "(";**

**}**

**inorder(root->left);**

**cout << root->data;**

**inorder(root->right);**

**// if the current token is an operator, print close parenthesis**

**if (isOperator(root->data)) {**

**cout << ")";**

**}**

**}**

**// Function to construct an expression tree from the given postfix expression**

**Node\* construct(string postfix)**

**{**

**// base case**

**if (postfix.length() == 0) {**

**return nullptr;**

**}**

**// create an empty stack to store tree pointers**

**stack<Node\*> s;**

**// traverse the postfix expression**

**for (char c: postfix)**

**{**

**// if the current token is an operator**

**if (isOperator(c))**

**{**

**// pop two nodes `x` and `y` from the stack**

**Node\* x = s.top();**

**s.pop();**

**Node\* y = s.top();**

**s.pop();**

**// construct a new binary tree whose root is the operator and whose**

**// left and right children point to `y` and `x`, respectively**

**Node\* node = new Node(c, y, x);**

**// push the current node into the stack**

**s.push(node);**

**}**

**// if the current token is an operand, create a new binary tree node**

**// whose root is the operand and push it into the stack**

**else {**

**s.push(new Node(c));**

**}**

**}**

**// a pointer to the root of the expression tree remains on the stack**

**return s.top();**

**}**

**int main()**

**{**

**string postfix = "ab+cde+\*\*";**

**Node\* root = construct(postfix);**

**cout << "Postfix Expression: ";**

**postorder(root);**

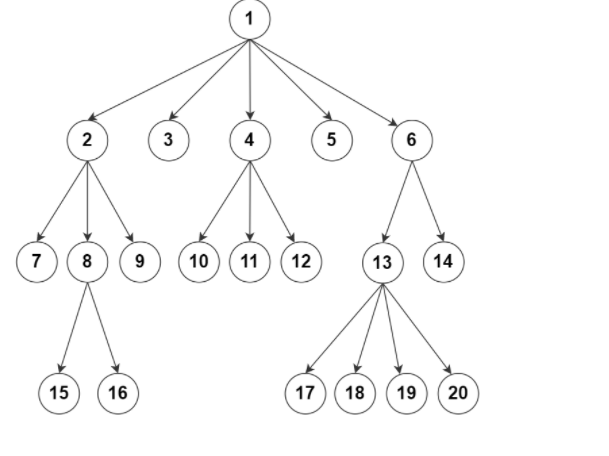
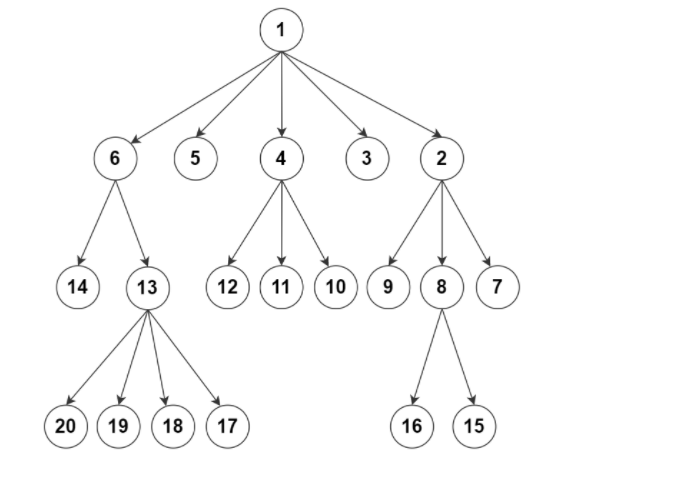
**cout << "\nInfix Expression: ";**

**inorder(root);**

**return 0;**

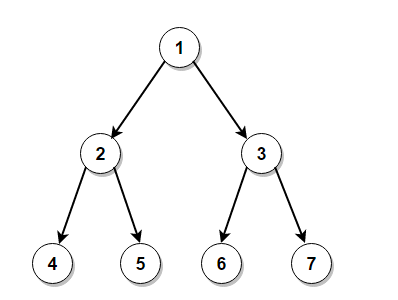
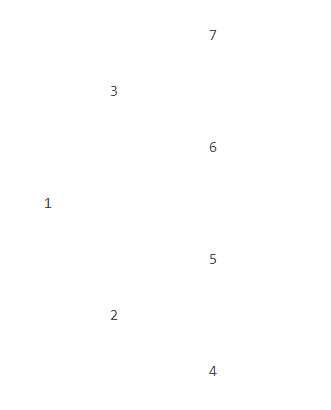
**}**

**46: Create a mirror of an m–ary tree**

**Given an m–ary tree, write an efficient algorithm to convert the tree into its mirror.  
An** [**m-ary tree**](https://en.wikipedia.org/wiki/M-ary_tree) **(aka k–ary tree) is a tree in which each node has no more than m children. Each node of an m–ary tree has an array for storing pointers to each of its children.  
The binary tree and the ternary tree are special cases of the m–ary tree where m = 2 and m = 3. Consider the following diagram, which displays an m–ary tree with m = 5:  
  
The following m–ary tree is a mirror image of the above tree:  
**

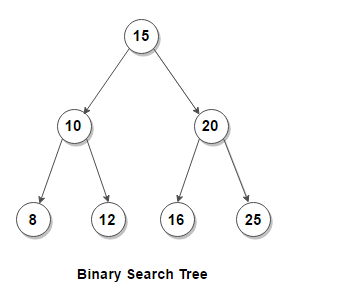
| **#include <iostream>**  **#include <vector>**  **#include <algorithm>**  **using namespace std;**    **// Data structure to store an m–ary tree node**  **struct Node**  **{**  **int data;**  **vector<Node\*> child;**    **Node(int data) {**  **this->data = data;**  **}**  **};**    **// Traverse and print an m–ary tree using preorder traversal**  **void printTree(Node\* root)**  **{**  **// base case**  **if (root == nullptr) {**  **return;**  **}**    **// print the current node**  **cout << root->data << ' ';**    **// recur for all children nodes from left to right**  **for (Node\* child: root->child) {**  **printTree(child);**  **}**  **}**    **// Recursive function to convert an m–ary tree into its mirror image**  **void convertToMirror(Node\* root)**  **{**  **// base case**  **if (root == nullptr) {**  **return;**  **}**    **// recur for each child**  **for (Node\* child: root->child) {**  **convertToMirror(child);**  **}**    **// Reverse the order of the elements in the children**  **reverse(root->child.begin(), root->child.end());**    **// std::reverse is equivalent to the following code**    **/\***  **for (int i = 0, j = (root->child).size() - 1; i < j; i++, j--) {**  **swap(root->child[i], root->child[j]);**  **}**  **\*/**  **}**    **int main()**  **{**  **// construct an m–ary tree**  **Node\* root = new Node(1);**    **(root->child).push\_back(new Node(2));**  **(root->child).push\_back(new Node(3));**  **(root->child).push\_back(new Node(4));**  **(root->child).push\_back(new Node(5));**  **(root->child).push\_back(new Node(6));**    **(root->child[0]->child).push\_back(new Node(7));**  **(root->child[0]->child).push\_back(new Node(8));**  **(root->child[0]->child).push\_back(new Node(9));**    **(root->child[2]->child).push\_back(new Node(10));**  **(root->child[2]->child).push\_back(new Node(11));**  **(root->child[2]->child).push\_back(new Node(12));**    **(root->child[4]->child).push\_back(new Node(13));**  **(root->child[4]->child).push\_back(new Node(14));**    **(root->child[0]->child[1]->child).push\_back(new Node(15));**  **(root->child[0]->child[1]->child).push\_back(new Node(16));**    **(root->child[4]->child[0]->child).push\_back(new Node(17));**  **(root->child[4]->child[0]->child).push\_back(new Node(18));**  **(root->child[4]->child[0]->child).push\_back(new Node(19));**  **(root->child[4]->child[0]->child).push\_back(new Node(20));**    **convertToMirror(root);**  **printTree(root);**    **return 0;**  **}** |
| --- |

**47: Print a two-dimensional view of a binary tree**

**Write an efficient algorithm to print the two-dimensional view of a binary tree.  
  
For example, the above binary tree can be displayed as:  
One can notice the following things in the above binary tree illustration:  
 - The rightmost node is printed in the first line.  
 - The leftmost node is printed in the last line.  
 - A fixed amount of space is increased at each level.**

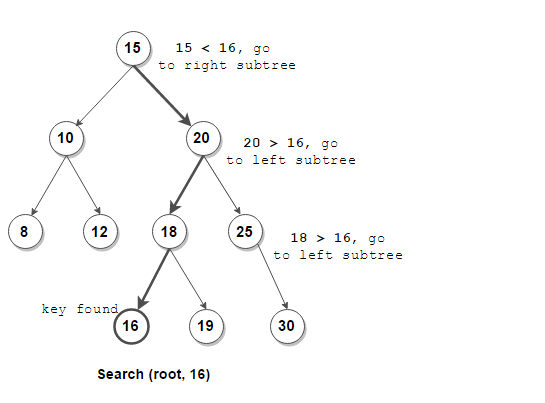
| **#include <iostream>**  **#include <utility>**  **using namespace std;**    **// Data structure to store a binary tree node**  **struct Node**  **{**  **int data;**  **Node \*left, \*right;**    **Node(int data)**  **{**  **this->data = data;**  **this->left = this->right = nullptr;**  **}**  **};**    **// Utility function to print the two-dimensional view of a binary tree using**  **// reverse inorder traversal**  **void printBinaryTree(Node\* root, int space = 0, int height = 10)**  **{**  **// Base case**  **if (root == nullptr) {**  **return;**  **}**    **// increase distance between levels**  **space += height;**    **// print right child first**  **printBinaryTree(root->right, space);**  **cout << endl;**    **// print the current node after padding with spaces**  **for (int i = height; i < space; i++) {**  **cout << ' ';**  **}**  **cout << root->data;**    **// print left child**  **cout << endl;**  **printBinaryTree(root->left, space);**  **}**    **int main()**  **{**  **/\* Construct the following tree**  **1**  **/ \**  **/ \**  **2 3**  **/ \ / \**  **/ \ / \**  **4 5 6 7**  **\*/**    **Node\* root = new Node(1);**  **root->left = new Node(2);**  **root->right = new Node(3);**  **root->left->left = new Node(4);**  **root->left->right = new Node(5);**  **root->right->left = new Node(6);**  **root->right->right = new Node(7);**    **// print binary tree**  **printBinaryTree(root);**    **return 0;**  **}** |
| --- |

**48:Insertion in a BST – Iterative and Recursive Solution**

**A Binary Search Tree (BST) is a rooted binary tree, whose nodes each store a key (and optionally, an associated value), and each has two distinguished subtrees, commonly denoted left and right. The tree should satisfy the BST property, which states that each node’s key must be greater than all keys stored in the left subtree and not greater than all keys in the right subtree. Ideally, unique values should be present in the tree.  
  
Binary search trees are a fundamental data structure used to construct more abstract data structures such as sets, multisets, and associative arrays (maps, multimaps, etc.).**

| **#include <iostream>**  **#include <vector>**  **using namespace std;**    **// Data structure to store a BST node**  **struct Node**  **{**  **int data;**  **Node\* left = nullptr, \*right = nullptr;**    **Node() {}**  **Node(int data): data(data) {}**  **};**    **// Function to perform inorder traversal on the tree**  **void inorder(Node\* root)**  **{**  **if (root == nullptr) {**  **return;**  **}**    **inorder(root->left);**  **cout << root->data << " ";**  **inorder(root->right);**  **}**    **// Recursive function to insert a key into a BST**  **Node\* insert(Node\* root, int key)**  **{**  **// if the root is null, create a new node and return it**  **if (root == nullptr) {**  **return new Node(key);**  **}**    **// if the given key is less than the root node, recur for the left subtree**  **if (key < root->data) {**  **root->left = insert(root->left, key);**  **}**  **// if the given key is more than the root node, recur for the right subtree**  **else {**  **root->right = insert(root->right, key);**  **}**    **return root;**  **}**    **// Function to construct a BST from given keys**  **Node\* constructBST(vector<int> const &keys)**  **{**  **Node\* root = nullptr;**  **for (int key: keys) {**  **root = insert(root, key);**  **}**  **return root;**  **}**    **int main()**  **{**  **vector<int> keys = { 15, 10, 20, 8, 12, 16, 25 };**    **Node\* root = constructBST(keys);**  **inorder(root);**    **return 0;**  **}** |
| --- |

**49:Search a given key in BST – Iterative and Recursive Solution**

**Given a BST, write an efficient function to search a given key in it. The algorithm should return the parent node of the key and print if the key is the left or right node of the parent node. If the key is not present in the BST, the algorithm should be able to determine that.  
Searching a binary search tree for a specific key can be programmed recursively or iteratively.**

**#include <iostream>**

**using namespace std;**

**// Data structure to store a BST node**

**struct Node**

**{**

**int data;**

**Node\* left = nullptr, \*right = nullptr;**

**Node() {}**

**Node(int data): data(data) {}**

**};**

**// Recursive function to insert a key into a BST**

**Node\* insert(Node\* root, int key)**

**{**

**// if the root is null, create a new node and return it**

**if (root == nullptr) {**

**return new Node(key);**

**}**

**// if the given key is less than the root node, recur for the left subtree**

**if (key < root->data) {**

**root->left = insert(root->left, key);**

**}**

**// if the given key is more than the root node, recur for the right subtree**

**else {**

**root->right = insert(root->right, key);**

**}**

**return root;**

**}**

**// Recursive function to search in a given BST**

**void search(Node\* root, int key, Node\* parent)**

**{**

**// if the key is not present in the key**

**if (root == nullptr)**

**{**

**cout << "Key not found";**

**return;**

**}**

**// if the key is found**

**if (root->data == key)**

**{**

**if (parent == nullptr) {**

**cout << "The node with key " << key << " is root node";**

**}**

**else if (key < parent->data)**

**{**

**cout << "The given key is the left node of the node with key "**

**<< parent->data;**

**}**

**else {**

**cout << "The given key is the right node of the node with key "**

**<< parent->data;**

**}**

**return;**

**}**

**// if the given key is less than the root node, recur for the left subtree;**

**// otherwise, recur for the right subtree**

**if (key < root->data) {**

**search(root->left, key, root);**

**} else {**

**search(root->right, key, root);**

**}**

**}**

**int main()**

**{**

**int keys[] = { 15, 10, 20, 8, 12, 16, 25 };**

**Node\* root = nullptr;**

**for (int key: keys) {**

**root = insert(root, key);**

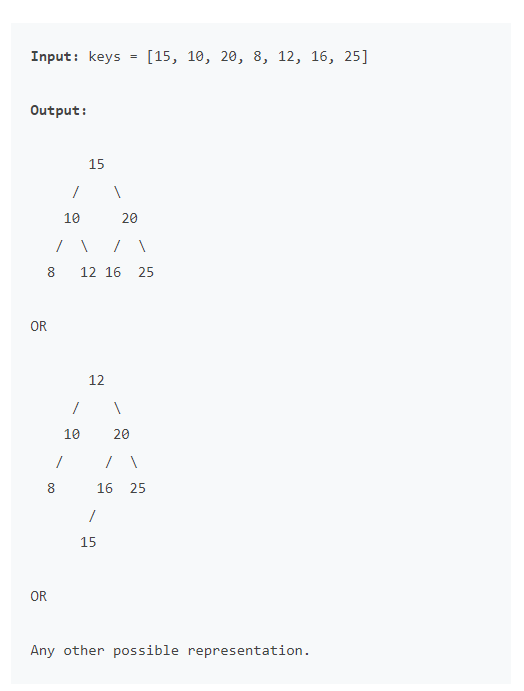
**}**

**search(root, 25, nullptr);**

**return 0;**

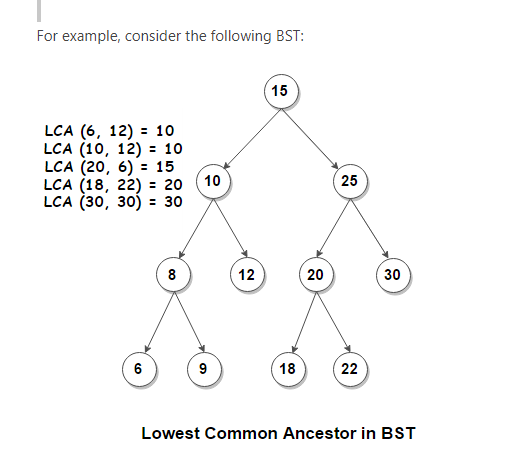
**}**

**50: Construct a balanced BST from the given keys**

**Given an unsorted integer array that represents binary search tree (BST) keys, construct a height-balanced BST from it. For each node of a height-balanced tree, the difference between its left and right subtree height is at most 1.**

| **#include <iostream>**  **#include <vector>**  **#include <algorithm>**  **using namespace std;**    **// Data structure to store a BST node**  **struct Node**  **{**  **int data;**  **Node\* left = nullptr, \*right = nullptr;**    **Node() {}**  **Node(int data): data(data) {}**  **};**    **// Function to perform inorder traversal on the tree**  **void inorder(Node\* root)**  **{**  **if (root == nullptr) {**  **return;**  **}**    **inorder(root->left);**  **cout << root->data << " ";**  **inorder(root->right);**  **}**    **// Recursive function to insert a key into a BST**  **Node\* insert(Node\* root, int key)**  **{**  **// if the root is null, create a new node and return it**  **if (root == nullptr) {**  **return new Node(key);**  **}**    **// if the given key is less than the root node, recur for the left subtree**  **if (key < root->data) {**  **root->left = insert(root->left, key);**  **}**    **// if the given key is more than the root node, recur for the right subtree**  **else {**  **root->right = insert(root->right, key);**  **}**    **return root;**  **}**    **// Function to construct balanced BST from the given sorted array.**  **// Note that the root of the tree is passed by reference here**  **void convert(vector<int> const &keys, int low, int high, Node\* &root)**  **{**  **// base case**  **if (low > high) {**  **return;**  **}**    **// find the middle element of the current range**  **int mid = (low + high) / 2;**    **// construct a new node from the middle element and assign it to the root**  **root = new Node(keys[mid]);**    **// left subtree of the root will be formed by keys less than middle element**  **convert(keys, low, mid - 1, root->left);**    **// right subtree of the root will be formed by keys more than the middle element**  **convert(keys, mid + 1, high, root->right);**  **}**    **// Function to construct balanced BST from the given unsorted array**  **Node\* convert(vector<int> keys)**  **{**  **// sort the keys first**  **sort(keys.begin(), keys.end());**    **// construct a balanced BST**  **Node\* root = nullptr;**  **convert(keys, 0, keys.size() - 1, root);**    **// return root node of the tree**  **return root;**  **}**    **int main()**  **{**  **// input keys**  **vector<int> keys = { 15, 10, 20, 8, 12, 16, 25 };**    **// construct a balanced binary search tree**  **Node\* root = convert(keys);**    **// print the keys in an inorder fashion**  **inorder(root);**    **return 0;**  **}** |
| --- |

**51: Find the Lowest Common Ancestor (LCA) of two nodes in a BST**

**Given a BST and two nodes x and y in it, find the lowest common ancestor (LCA) of x and y. The solution should return null if either x or y is not the actual node in the tree.  
The lowest common ancestor (LCA) of two nodes x and y in the BST is the lowest (i.e., deepest) node that has both x and y as descendants, where each node can be a descendant of itself (so if x is reachable from w, w is the LCA). In other words, the LCA of x and y is the shared ancestor of x and y that is located farthest from the root.  
**

**#include <iostream>**

**using namespace std;**

**// Data structure to store a BST node**

**struct Node**

**{**

**int data;**

**Node\* left = nullptr, \*right = nullptr;**

**Node() {}**

**Node(int data): data(data) {}**

**};**

**// Function to perform inorder traversal on the tree**

**void inorder(Node\* root)**

**{**

**if (root == nullptr) {**

**return;**

**}**

**inorder(root->left);**

**cout << root->data << " ";**

**inorder(root->right);**

**}**

**// Recursive function to insert a key into a BST**

**Node\* insert(Node\* root, int key)**

**{**

**// if the root is null, create a new node and return it**

**if (root == nullptr) {**

**return new Node(key);**

**}**

**// if the given key is less than the root node, recur for the left subtree**

**if (key < root->data) {**

**root->left = insert(root->left, key);**

**}**

**// if the given key is more than the root node, recur for the right subtree**

**else {**

**root->right = insert(root->right, key);**

**}**

**return root;**

**}**

**// Iterative function to search a given node in a BST**

**bool search(Node\* root, Node\* key)**

**{**

**// traverse the tree and search for the key**

**while (root)**

**{**

**// if the given key is less than the current node, go to the left**

**// subtree; otherwise, go to the right subtree**

**if (key->data < root->data) {**

**root = root->left;**

**}**

**else if (key->data > root->data) {**

**root = root->right;**

**}**

**// if the key is found, return true**

**else if (key == root) {**

**return true;**

**}**

**else {**

**return false;**

**}**

**}**

**// we reach here if the key is not present in the BST**

**return false;**

**}**

**// Recursive function to find the lowest common ancestor of given nodes**

**// `x` and `y`, where both `x` and `y` are present in a BST**

**Node\* LCARecursive(Node\* root, Node\* x, Node\* y)**

**{**

**// base case: empty tree**

**if (root == nullptr) {**

**return nullptr;**

**}**

**// if both `x` and `y` is smaller than the root, LCA exists in the left subtree**

**if (root->data > max(x->data, y->data)) {**

**return LCARecursive(root->left, x, y);**

**}**

**// if both `x` and `y` are greater than the root, LCA exists in the right subtree**

**else if (root->data < min(x->data, y->data)) {**

**return LCARecursive(root->right, x, y);**

**}**

**// if one key is greater (or equal) than the root and one key is smaller**

**// (or equal) than the root, then the current node is LCA**

**return root;**

**}**

**// Print lowest common ancestor of two nodes in a BST**

**void LCA(Node\* root, Node\* x, Node\* y)**

**{**

**// return if the tree is empty, or `x` or `y` is not present in the tree**

**if (root == nullptr || !search(root, x) || !search(root, y)) {**

**return;**

**}**

**// `lca` stores the lowest common ancestor of `x` and `y`**

**Node\* lca = LCARecursive(root, x, y);**

**// if the lowest common ancestor exists, print it**

**if (lca != nullptr) {**

**cout << "LCA is " << lca->data;**

**}**

**else {**

**cout << "LCA does not exist";**

**}**

**}**

**int main()**

**{**

**int keys[] = { 15, 10, 20, 8, 12, 16, 25 };**

**/\* Construct the following tree**

**15**

**/ \**

**/ \**

**10 20**

**/ \ / \**

**/ \ / \**

**8 12 16 25**

**\*/**

**Node\* root = nullptr;**

**for (int key: keys) {**

**root = insert(root, key);**

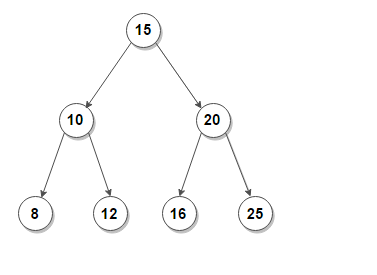
**}**

**LCA(root, root->left->left, root->left->right);**

**return 0;**

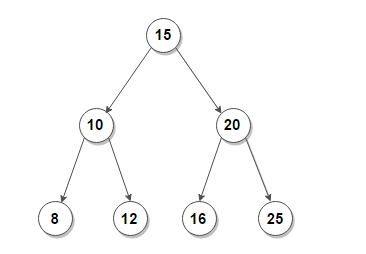
**}**

**52: Find k’th smallest node in a BST**

**Given a BST and a positive number k, find the k'th smallest node in it.  
For example, the 4th smallest node in the following BST is 15, and the 6th smallest is 20. The 8th smallest node does not exist.**

| **#include <stdio.h>**  **#include <stdlib.h>**    **// Data structure to store a binary tree node**  **struct Node**  **{**  **int data;**  **struct Node \*left, \*right;**  **};**    **// Function to create a new binary tree node having a given key**  **struct Node\* newNode(int key)**  **{**  **struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));**  **node->data = key;**  **node->left = node->right = NULL;**    **return node;**  **}**    **// Recursive function to find the k'th smallest node**  **// in the BST (using inorder traversal)**  **struct Node\* kthSmallest(struct Node\* root, int\* counter, int k)**  **{**  **// base case**  **if (root == NULL) {**  **return NULL;**  **}**    **// recur for the left subtree**  **struct Node\* left = kthSmallest(root->left, counter, k);**    **// if k'th smallest node is found**  **if (left != NULL) {**  **return left;**  **}**    **// if the root is k'th smallest node**  **if (++(\*counter) == k) {**  **return root;**  **}**    **// recur for the right subtree only if k'th smallest node is not found**  **// in the right subtree**  **return kthSmallest(root->right, counter, k);**  **}**    **// Function to find the k'th smallest node in a BST**  **struct Node\* findKthSmallest(struct Node\* root, int k)**  **{**  **// counter to keep track of the total number of the visited nodes**  **int counter = 0;**    **// recursively find the k'th smallest node**  **return kthSmallest(root, &counter, k);**  **}**    **int main(void)**  **{**  **/\* Construct the following BST**  **15**  **/ \**  **/ \**  **10 20**  **/ \ / \**  **/ \ / \**  **8 12 16 25**  **\*/**    **struct Node\* root = newNode(15);**  **root->left = newNode(10);**  **root->right = newNode(20);**  **root->left->left = newNode(8);**  **root->left->right = newNode(12);**  **root->right->left = newNode(16);**  **root->right->right = newNode(25);**    **int k = 4;**    **// find the k'th smallest node**  **struct Node\* result = findKthSmallest(root, k);**    **if (result) {**  **printf("%d'th smallest node is %d", k, result->data);**  **}**  **else {**  **printf("%d'th smallest node does not exist.", k);**  **}**    **return 0;**  **}** |
| --- |

**53: Find k’th largest node in a BST**

**Given a BST and a positive number k, find the k'th largest node in the BST.  
For example, consider the following binary search tree. If k = 2, the k'th largest node is 20.**

**#include <iostream>**

**#include <climits>**

**using namespace std;**

**// Data structure to store a BST node**

**struct Node**

**{**

**int data;**

**Node\* left = nullptr, \*right = nullptr;**

**Node() {}**

**Node(int data): data(data) {}**

**};**

**// Function to perform inorder traversal on the tree**

**void inorder(Node\* root)**

**{**

**if (root == nullptr) {**

**return;**

**}**

**inorder(root->left);**

**cout << root->data << " ";**

**inorder(root->right);**

**}**

**// Recursive function to insert a key into a BST**

**Node\* insert(Node\* root, int key)**

**{**

**// if the root is null, create a new node and return it**

**if (root == nullptr) {**

**return new Node(key);**

**}**

**// if the given key is less than the root node, recur for the left subtree**

**if (key < root->data) {**

**root->left = insert(root->left, key);**

**}**

**// if the given key is more than the root node, recur for the right subtree**

**else {**

**root->right = insert(root->right, key);**

**}**

**return root;**

**}**

**// Function to find the k'th largest node in a BST.**

**// Here, `i` denotes the total number of nodes processed so far**

**Node\* kthLargest(Node\* root, int \*i, int k)**

**{**

**// base case**

**if (root == nullptr) {**

**return nullptr;**

**}**

**// search in the right subtree**

**Node\* left = kthLargest(root->right, i, k);**

**// if k'th largest is found in the left subtree, return it**

**if (left) {**

**return left;**

**}**

**// if the current node is k'th largest, return its value**

**if (++\*i == k) {**

**return root;**

**}**

**// otherwise, search in the left subtree**

**return kthLargest(root->left, i, k);**

**}**

**// Function to find the k'th largest node in a BST**

**Node\* findKthLargest(Node\* root, int k)**

**{**

**// maintain index to count the total number of nodes processed so far**

**int i = 0;**

**// traverse the tree in an inorder fashion and return k'th node**

**return kthLargest(root, &i, k);**

**}**

**int main()**

**{**

**int keys[] = { 15, 10, 20, 8, 12, 16, 25 };**

**Node\* root = nullptr;**

**for (int key: keys) {**

**root = insert(root, key);**

**}**

**int k = 2;**

**Node\* node = findKthLargest(root, k);**

**if (node != nullptr) {**

**cout << node->data;**

**}**

**else {**

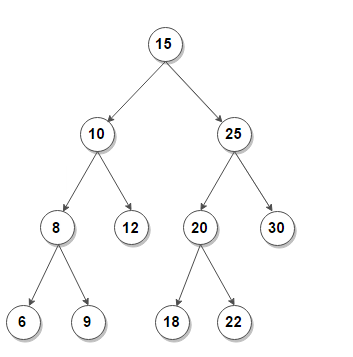
**cout << "Invalid Input";**

**}**

**return 0;**

**}**

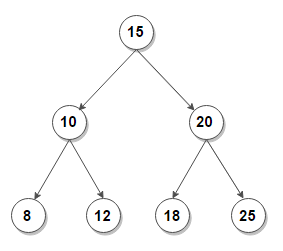
**54: Find a pair with the given sum in a BST**

**Given a binary search tree, find a pair with a given sum present in it.  
For example, consider the following BST. If the given sum is 14, the pair is (8, 6).  
**

| **#include <iostream>**  **#include <unordered\_set>**  **using namespace std;**    **// Data structure to store a BST node**  **struct Node**  **{**  **int data;**  **Node\* left = nullptr, \*right = nullptr;**    **Node() {}**  **Node(int data): data(data) {}**  **};**    **// Recursive function to insert a key into a BST**  **Node\* insert(Node\* root, int key)**  **{**  **// if the root is null, create a new node and return it**  **if (root == nullptr) {**  **return new Node(key);**  **}**    **// if the given key is less than the root node, recur for the left subtree**  **if (key < root->data) {**  **root->left = insert(root->left, key);**  **}**    **// if the given key is more than the root node, recur for the right subtree**  **else {**  **root->right = insert(root->right, key);**  **}**    **return root;**  **}**    **// Function to find a pair with a given sum in a BST**  **bool findPair(Node\* root, int target, auto &set)**  **{**  **// base case**  **if (root == nullptr) {**  **return false;**  **}**    **// return true if pair is found in the left subtree; otherwise, continue**  **// processing the node**  **if (findPair(root->left, target, set)) {**  **return true;**  **}**    **// if a pair is formed with the current node, print the pair and return true**  **if (set.find(target - root->data) != set.end())**  **{**  **cout << "Pair found (" << target - root->data << ", " << root->data << ")";**  **return true;**  **}**    **// insert the current node's value into the set**  **else {**  **set.insert(root->data);**  **}**    **// search in the right subtree**  **return findPair(root->right, target, set);**  **}**    **int main()**  **{**  **int keys[] = { 15, 10, 20, 8, 12, 16, 25 };**    **Node\* root = nullptr;**  **for (int key: keys) {**  **root = insert(root, key);**  **}**    **// find pair with the given sum**  **int target = 28;**    **// create an empty set**  **unordered\_set<int> set;**    **if (!findPair(root, target, set)) {**  **cout << "Pair does not exist";**  **}**    **return 0;**  **}** |
| --- |

**55: Print complete Binary Search Tree (BST) in increasing order**

**Given a** [**level order**](https://www.techiedelight.com/level-order-traversal-binary-tree/) **representation of a complete binary search tree, print its elements in increasing order.  
For example, the level order representation of the complete BST below is [15, 10, 20, 8, 12, 18, 25]. The solution should print [8, 10, 12, 15, 18, 20, 25].**

****

**#include <stdio.h>**

**// Recursive function to print a complete binary search tree in increasing order**

**void printIncreasingOrder(int keys[], int low, int high)**

**{**

**if (low > high) {**

**return;**

**}**

**// recur for the left subtree**

**printIncreasingOrder(keys, low\*2 + 1, high);**

**// print the root node**

**printf("%d ", keys[low]);**

**// recur for the right subtree**

**printIncreasingOrder(keys, low\*2 + 2, high);**

**}**

**int main(void)**

**{**

**int keys[] = { 15, 10, 20, 8, 12, 18, 25 };**

**int n = sizeof(keys) / sizeof(int);**

**printIncreasingOrder(keys, 0, n - 1);**

**return 0;**

**}**

**56: Find the number of rotations in a circularly sorted array**

**Given a circularly sorted integer array, find the total number of times the array is rotated. Assume there are no duplicates in the array, and the rotation is in the anti-clockwise direction.  
For example,**

**Input: nums = [8, 9, 10, 2, 5, 6]=  
Output: The array is rotated 3 times  
  
Input: nums = [2, 5, 6, 8, 9, 10]  
Output: The array is rotated 0 times**

**#include <stdio.h>**

**// Function to find the total number of times the array is rotated**

**int findRotationCount(int nums[], int n)**

**{**

**// search space is nums[low…high]**

**int low = 0, high = n - 1;**

**// loop till the search space is exhausted**

**while (low <= high)**

**{**

**// if the search space is already sorted, we have**

**// found the minimum element (at index `low`)**

**if (nums[low] <= nums[high]) {**

**return low;**

**}**

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

**// find the next and previous element of the `mid` element**

**// (in a circular manner)**

**int next = (mid + 1) % n;**

**int prev = (mid - 1 + n) % n;**

**// if the `mid` element is less than both its next and previous**

**// neighbor, it is the array's minimum element**

**if (nums[mid] <= nums[next] && nums[mid] <= nums[prev]) {**

**return mid;**

**}**

**// if nums[mid…high] is sorted, and `mid` is not the minimum element,**

**// then the pivot element cannot be present in nums[mid…high],**

**// discard nums[mid…high] and search in the left half**

**else if (nums[mid] <= nums[high]) {**

**high = mid - 1;**

**}**

**// if nums[low…mid] is sorted, then the pivot element cannot be present in it;**

**// discard nums[low…mid] and search in the right half**

**else if (nums[mid] >= nums[low]) {**

**low = mid + 1;**

**}**

**}**

**// invalid input**

**return -1;**

**}**

**int main(void)**

**{**

**int nums[] = { 8, 9, 10, 2, 5, 6 };**

**int n = sizeof(nums) / sizeof(nums[0]);**

**int count = findRotationCount(nums, n);**

**printf("Array is rotated %d times", count);**

**return 0;**

**}**

**57: Find the first or last occurrence of a given number in a sorted array**

**Given a sorted integer array, find the index of a given number’s first or last occurrence. If the element is not present in the array, report that as well.**

**Input:**

**nums = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9]**

**target = 5**

**Output:**

**The first occurrence of element 5 is located at index 1**

**The last occurrence of element 5 is located at index 3**

**Input:**

**nums = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9]**

**target = 4**

**Output:Element not found in the array**

**#include <stdio.h>**

**// Function to find the first occurrence of a given number in a sorted integer array**

**int findFirstOccurrence(int nums[], int n, int target)**

**{**

**// search space is nums[low…high]**

**int low = 0, high = n - 1;**

**// initialize the result by -1**

**int result = -1;**

**// loop till the search space is exhausted**

**while (low <= high)**

**{**

**// find the mid-value in the search space and compares it with the target**

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

**// if the target is located, update the result and**

**// search towards the left (lower indices)**

**if (target == nums[mid])**

**{**

**result = mid;**

**high = mid - 1;**

**}**

**// if the target is less than the middle element, discard the right half**

**else if (target < nums[mid]) {**

**high = mid - 1;**

**}**

**// if the target is more than the middle element, discard the left half**

**else {**

**low = mid + 1;**

**}**

**}**

**// return the leftmost index, or -1 if the element is not found**

**return result;**

**}**

**int main(void)**

**{**

**int nums[] = {2, 5, 5, 5, 6, 6, 8, 9, 9, 9};**

**int n = sizeof(nums)/sizeof(nums[0]);**

**int target = 5;**

**int index = findFirstOccurrence(nums, n, target);**

**if (index != -1)**

**{**

**printf("The first occurrence of element %d is located at index %d",**

**target, index);**

**}**

**else {**

**printf("Element not found in the array");**

**}**

**return 0;**

**}**

**58: Find floor and ceil of a number in a sorted integer array**

**Given a sorted integer array, find the floor and ceil of a given number in it. The floor and ceil map the given number to the largest previous or the smallest following integer in the array.  
More precisely, for a number x, floor(x) is the largest integer in the array less than or equal to x, and ceil(x) is the smallest integer in the array greater than or equal to x. If the floor or ceil doesn’t exist, consider it to be -1. For example,  
Input:**

**nums[] = [1, 4, 6, 8, 9]  
Number: 0 to 10  
  
Output:  
Number 0 —> ceil is 1, floor is -1  
Number 1 —> ceil is 1, floor is 1  
Number 2 —> ceil is 4, floor is 1  
Number 3 —> ceil is 4, floor is 1  
Number 4 —> ceil is 4, floor is 4  
Number 5 —> ceil is 6, floor is 4  
Number 6 —> ceil is 6, floor is 6  
Number 7 —> ceil is 8, floor is 6  
Number 8 —> ceil is 8, floor is 8  
Number 9 —> ceil is 9, floor is 9  
Number 10 —> ceil is -1, floor is 9**

| **#include <stdio.h>**    **// Function to find the ceil of `x` in a sorted array nums[0…n-1]**  **// i.e., the smallest integer greater than or equal to `x`**  **int getCeil(int nums[], int n, int x)**  **{**  **// search space is nums[low…high]**  **int low = 0, high = n - 1, mid;**    **// initialize ceil to -1**  **int ceil = -1;**    **// loop till the search space is exhausted**  **while (low <= high)**  **{**  **// find the mid-value in the search space**  **mid = (low + high) / 2;**    **// if `x` is equal to the middle element, it is the ceil**  **if (nums[mid] == x) {**  **return nums[mid];**  **}**    **// if `x` is less than the middle element, the ceil exists in the**  **// subarray nums[low…mid]; update ceil to the middle element**  **// and reduce our search space to the left subarray nums[low…mid-1]**  **else if (x < nums[mid])**  **{**  **ceil = nums[mid];**  **high = mid - 1;**  **}**    **// if `x` is more than the middle element, the ceil exists in the**  **// right subarray nums[mid+1…high]**  **else {**  **low = mid + 1;**  **}**  **}**    **return ceil;**  **}**    **// Function to find the floor of `x` in a sorted array nums[0…n-1],**  **// i.e., the largest integer less than or equal to `x`**  **int getFloor(int nums[], int n, int x)**  **{**  **int low = 0, high = n - 1, mid;**    **// initialize floor to -1**  **int floor = -1;**    **// loop till the search space is exhausted**  **while (low <= high)**  **{**  **// find the mid-value in the search space**  **mid = (low + high) / 2;**    **// if `x` is equal to the middle element, it is the floor**  **if (nums[mid] == x) {**  **return nums[mid];**  **}**    **// if `x` is less than the middle element, the floor exists in the left**  **// subarray nums[low…mid-1]**  **else if (x < nums[mid]) {**  **high = mid - 1;**  **}**    **// if `x` is more than the middle element, the floor exists in the**  **// subarray nums[mid…high]; update floor to the middle element**  **// and reduce our search space to the right subarray nums[mid+1…high]**  **else {**  **floor = nums[mid];**  **low = mid + 1;**  **}**  **}**    **return floor;**  **}**    **int main(void)**  **{**  **int nums[] = { 1, 4, 6, 8, 9 };**  **int n = sizeof(nums) / sizeof(nums[0]);**    **for (int i = 0; i <= 10; i++)**  **{**  **printf("Number %d —> ", i);**  **printf("ceil is %d, ", getCeil(nums, n, i));**  **printf("floor is %d\n", getFloor(nums, n, i));**  **}**    **return 0;**  **}** |
| --- |

**59:Find the number of 1’s in a sorted binary array**

**Given a sorted binary array, efficiently count the total number of 1’s in it.  
Input: nums[] = [0, 0, 0, 0, 1, 1, 1]  
Output: The total number of 1’s present is 3  
Input: nums[] = [0, 0, 1, 1, 1, 1, 1]  
Output: The total number of 1’s present is 5**

**#include <stdio.h>**

**// Function to find the total number of 1's in a sorted binary array**

**int count(int nums[], int n)**

**{**

**// if the last array element is 0, no 1's can**

**// be present since it is sorted**

**if (nums[n - 1] == 0) {**

**return 0;**

**}**

**// if the first array element is 1, all its elements**

**// are ones only since it is sorted**

**if (nums[0]) {**

**return n;**

**}**

**// divide the array into left and right subarray and recur**

**return count(nums, n/2) + count(nums + n/2, n - n/2);**

**}**

**int main(void)**

**{**

**int nums[] = { 0, 0, 0, 0, 1, 1, 1 };**

**int n = sizeof(nums) / sizeof(nums[0]);**

**printf("The total number of 1's present is %d", count(nums, n));**

**return 0;**

**}**

**60: Print all numbers between 1 to N without using any loop | 4 methods**

**Write a program to print all numbers between 1 and N without using a loop.**

**#include <iostream>**

**using namespace std;**

**#define N 100**

**int main()**

**{**

**static int i = 1;**

**if (i <= N && cout << i++ << " ") {**

**main();**

**}**

**return 0;**

**}  
  
61: Activity Selection Problem**

**Activity Selection Problem: Given a set of activities, along with the starting and finishing time of each activity, find the maximum number of activities performed by a single person assuming that a person can only work on a single activity at a time.**

**#include <stdio.h>**

**// Naive iterative solution to calculate `pow(x, n)`**

**long power(int x, unsigned n)**

**{**

**// initialize result by 1**

**long pow = 1L;**

**// multiply `x` exactly `n` times**

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

**pow = pow \* x;**

**}**

**return pow;**

**}**

**int main(void)**

**{**

**int x = -2;**

**unsigned n = 10;**

**printf("pow(%d, %d) = %d", x, n, power(x, n));**

**return 0;**

**}**

**62:Find floor and ceil of a number in a sorted array (Recursive solution)**

**Given a sorted integer array, find the floor and ceiling of a given number in it. The floor and ceiling map the given number to the largest previous or the smallest following integer in the array.  
Input:  
nums = [1, 4, 6, 8, 9]  
Number: 0 to 10  
Output:  
Number 0 —> ceiling is 1, floor is -1  
Number 1 —> ceiling is 1, floor is 1  
Number 2 —> ceiling is 4, floor is 1  
Number 3 —> ceiling is 4, floor is 1  
Number 4 —> ceiling is 4, floor is 4  
Number 5 —> ceiling is 6, floor is 4  
Number 6 —> ceiling is 6, floor is 6  
Number 7 —> ceiling is 8, floor is 6  
Number 8 —> ceiling is 8, floor is 8  
Number 9 —> ceiling is 9, floor is 9  
Number 10 —> ceiling is -1, floor is 9**

**#include <stdio.h>**

**// Function to find the ceiling of `x` in a sorted array nums[low…high].**

**// i.e., the smallest integer greater than or equal to `x`**

**int findCeiling(int nums[], int low, int high, int x)**

**{**

**// search space is nums[low…high]**

**// base case**

**if (high < low) {**

**return -1;**

**}**

**// if `x` is less than equal to the lowest element in search**

**// space, the lowest element is the ceiling**

**if (x <= nums[low]) {**

**return nums[low];**

**}**

**// if `x` is more than the highest element in the search space,**

**// its ceiling doesn't exist**

**if (x > nums[high]) {**

**return -1;**

**}**

**// find the middle index**

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

**// if `x` is equal to the middle element, it is the ceiling**

**if (nums[mid] == x) {**

**return nums[mid];**

**}**

**// if `x` is more than the middle element, the ceiling exists in the right**

**// subarray nums[mid+1…high]**

**else if (nums[mid] < x) {**

**return findCeiling(nums, mid + 1, high, x);**

**}**

**// if `x` is less than the middle element, the ceiling exists in the left**

**// subarray nums[low…mid] (Note – middle element can also be ceiling)**

**else {**

**return findCeiling(nums, low, mid, x);**

**}**

**}**

**// Function to find the floor of `x` in a sorted array nums[low…high].**

**// i.e., the largest integer less than or equal to `x`**

**int findFloor(int nums[], int low, int high, int x)**

**{**

**// search space is nums[low…high]**

**// base case**

**if (high < low) {**

**return -1;**

**}**

**// if `x` is less than the lowest element in search**

**// space, its floor doesn't exist**

**if (x < nums[low]) {**

**return -1;**

**}**

**// if `x` is more than equal to the highest element in**

**// the search space, it is the floor**

**if (x >= nums[high]) {**

**return nums[high];**

**}**

**// find the middle index**

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

**// this check is placed to handle infinite loop for**

**// a call to `findFloor(nums, mid, right, x)`**

**if (mid == low) {**

**return nums[low];**

**}**

**// if `x` is equal to the middle element, it is the floor**

**if (nums[mid] == x) {**

**return nums[mid];**

**}**

**// if `x` is more than the middle element, the floor exists in the right**

**// subarray nums[mid…high] (Note – middle element can also be the floor)**

**else if (nums[mid] < x) {**

**return findFloor(nums, mid, high, x);**

**}**

**// if `x` is less than the middle element, the floor exists in the left**

**// subarray nums[low…mid-1]**

**else {**

**return findFloor(nums, low, mid - 1, x);**

**}**

**}**

**int main(void)**

**{**

**int nums[] = { 1, 4, 6, 8, 9 };**

**int n = sizeof(nums) / sizeof(nums[0]);**

**for (int i = 0; i <= 10; i++)**

**{**

**printf("Number %2d —> ceiling is %2d, floor is %2d\n", i,**

**findCeiling(nums, 0, n - 1, i),**

**findFloor(nums, 0, n - 1, i));**

**}**

**return 0;**

**}**

**63: Find the frequency of each element in a sorted array containing duplicates**

**Given a sorted array containing duplicates, efficiently find each element’s frequency without traversing the whole array.  
Input: [2, 2, 2, 4, 4, 4, 5, 5, 6, 8, 8, 9]**

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

**Explanation:**

**2 and 4 occurs thrice**

**5 and 8 occurs twice**

**6 and 9 occurs once**

**#include <iostream>**

**#include <unordered\_map>**

**using namespace std;**

**// Function to find the frequency of each element in a sorted array**

**void findFrequency(int nums[], int n, unordered\_map<int, int> &freq)**

**{**

**// if every element in subarray nums[0…n-1] is equal,**

**// then increment the element's count by `n`**

**if (nums[0] == nums[n - 1])**

**{**

**freq[nums[0]] += n;**

**return;**

**}**

**// divide the array into left and right subarray and recur**

**findFrequency(nums, n/2, freq);**

**findFrequency(nums + n/2, n - n/2, freq);**

**}**

**int main()**

**{**

**int nums[] = { 2, 2, 2, 4, 4, 4, 5, 5, 6, 8, 8, 9 };**

**int n = sizeof(nums) / sizeof(int);**

**// find the frequency of each array element and store it in a map**

**unordered\_map<int, int> freq;**

**findFrequency(nums, n, freq);**

**// print the frequency**

**for (auto pair: freq) {**

**cout << pair.first << " occurs " << pair.second << " times\n";**

**}**

**return 0;**

**}**

**64: Remove duplicates from a sorted linked list**

**Given a linked list sorted in increasing order, write a function that removes duplicate nodes from it by traversing the list only once.**

**For example, the list {1, 2, 2, 2, 3, 4, 4, 5} should be converted into the list {1, 2, 3, 4, 5}.**

**#include <stdio.h>**

**#include <stdlib.h>**

**// A Linked List Node**

**struct Node**

**{**

**int data;**

**struct Node\* next;**

**};**

**// Helper function to print a given linked list**

**void printList(struct Node\* head)**

**{**

**struct Node\* ptr = head;**

**while (ptr)**

**{**

**printf("%d —> ", ptr->data);**

**ptr = ptr->next;**

**}**

**printf("NULL");**

**}**

**// Helper function to insert a new node at the beginning of the linked list**

**void push(struct Node\*\* head, int data)**

**{**

**struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));**

**newNode->data = data;**

**newNode->next = \*head;**

**\*head = newNode;**

**}**

**// Remove duplicates from a sorted list**

**void removeDuplicates(struct Node\* head)**

**{**

**// do nothing if the list is empty**

**if (head == NULL) {**

**return;**

**}**

**struct Node\* current = head;**

**// compare the current node with the next node**

**while (current->next != NULL)**

**{**

**if (current->data == current->next->data)**

**{**

**struct Node\* nextNext = current->next->next;**

**free(current->next);**

**current->next = nextNext;**

**}**

**else {**

**current = current->next; // only advance if no deletion**

**}**

**}**

**}**

**int main(void)**

**{**

**// input keys**

**int keys[] = {1, 2, 2, 2, 3, 4, 4, 5};**

**int n = sizeof(keys)/sizeof(keys[0]);**

**// points to the head node of the linked list**

**struct Node\* head = NULL;**

**// construct a linked list**

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

**push(&head, keys[i]);**

**}**

**removeDuplicates(head);**

**// print linked list**

**printList(head);**

**return 0;**

**}**

**65: Find the square root of a number using a binary search**

**Given a positive number, return the square root of it. If the number is not a perfect square, return the floor of its square root.  
Input: x = 12**

**Output: 3**

**Input: x = 16**

**Output: 4**

**#include <stdio.h>**

**// Function to find the floor of the square root of `x`**

**int findSqrt(int x)**

**{**

**// find the first positive number `i` such that `i×i` is greater than `x`**

**int i = 1;**

**while (i\*i <= x) {**

**i++;**

**}**

**return i - 1;**

**}**

**int main(void)**

**{**

**for (int i = 0; i <= 16; i++) {**

**printf("sqrt(%d) = %d\n", i, findSqrt(i));**

**}**

**return 0;**

**}**

**66: Longest Common Prefix (LCP) Problem**

**Write an efficient algorithm to find the longest common prefix (LCP) between a given set of strings.  
Input: technique, technician, technology, technical**

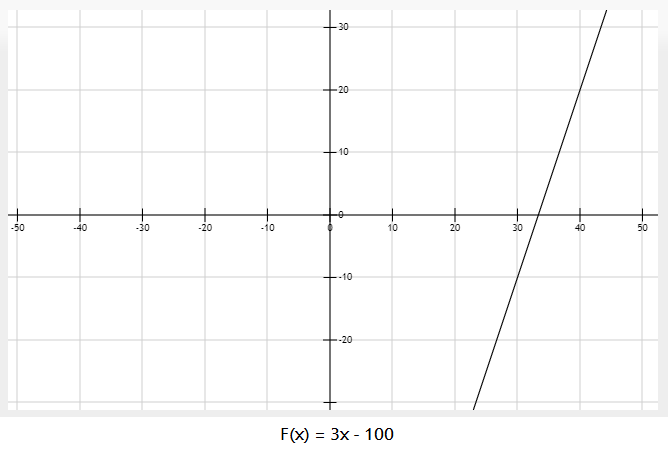
**Output: The longest common prefix is techn**

**Input: techie delight, tech, techie, technology, technical**

**Output: The longest common prefix is tech**

| **#include <iostream>**  **#include <vector>**  **#include <string>**  **using namespace std;**    **// Function to find the longest common prefix between two strings**  **string LCP(string X, string Y)**  **{**  **int i = 0, j = 0;**  **while (i < X.length() && j < Y.length())**  **{**  **if (X[i] != Y[j]) {**  **break;**  **}**    **i++, j++;**  **}**    **return X.substr(0, i);**  **}**    **// Function to find the longest common prefix (LCP) between a given set of strings**  **string findLCP(vector<string> const &words)**  **{**  **string prefix = words[0];**  **for (string s: words) {**  **prefix = LCP(prefix, s);**  **}**    **return prefix;**  **}**    **int main()**  **{**  **vector<string> words { "techie delight", "tech", "techie",**  **"technology", "technical" };**    **cout << "The longest common prefix is " << findLCP(words);**    **return 0;**  **}** |
| --- |

**67:Unbounded Binary Search**

**Given a monotonically increasing function f(x) on positive numbers, find the value of x when f(x) becomes positive for the first time. In other words, find a positive number x such that f(x-1), f(x-2), … are negative and f(x+1), f(x+2), … are positive.  
A function is called monotonically increasing, if f(x) <= f(y) is true for all x and y such that x <= y. For example, f(x) = 3x - 100 is a monotonically increasing function. It becomes positive for the first time when x = 34, as shown below:  
**

**#include <stdio.h>**

**// A monotonically increasing function `f(x) = 3x - 100`**

**int f(int x) {**

**return 3\*x - 100;**

**}**

**// Find the value of `x` in the search space [low, high] using binary search**

**// where f(x) becomes positive for the first time**

**int binarySearch(int low, int high)**

**{**

**// base condition (search space is exhausted)**

**if (high < low) {**

**return -1;**

**}**

**// find the mid-value in the search space**

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

**// if `f(mid)` is positive**

**if (f(mid) > 0)**

**{**

**// return `mid` if it is the first element of the search space or**

**// when f(mid-1) is not positive**

**if (mid == low || f(mid - 1) <= 0) {**

**return mid;**

**}**

**// otherwise, discard all elements in the right search space**

**return binarySearch(low, mid - 1);**

**}**

**// if f(mid) is zero or negative,**

**// discard all elements in the left search space**

**return binarySearch(mid + 1, high);**

**}**

**// Returns the positive value `x`, where f(x) becomes positive for the first time**

**int exponentialSearch()**

**{**

**// find the range in which the result would reside**

**int i = 1;**

**while (f(i) <= 0)**

**{**

**// calculate the next power of 2**

**i \*= 2;**

**}**

**// call binary search on [i/2, i]**

**return binarySearch(i/2, i);**

**}**

**int main(void)**

**{**

**int x = exponentialSearch();**

**printf("f(x) becomes positive for the first time when x = %d", x);**

**return 0;**

**}**

**68: Find all n-digit binary numbers without any consecutive 1’s**

**Given a positive integer n, count all n–digit binary numbers without any consecutive 1's.  
For example, for n = 5, the binary numbers that satisfy the given constraints are: [00000, 00001, 00010, 00100, 00101, 01000, 01001, 01010, 10000, 10001, 10010, 10100, 10101].**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**// Function to count all n–digit binary numbers without any consecutive 1's**

**int countStrings(int n, int last\_digit)**

**{**

**// base case**

**if (n == 0) {**

**return 0;**

**}**

**// if only one digit is left**

**if (n == 1)**

**{**

**if (last\_digit) { // if the last digit is 1**

**return 1;**

**}**

**else { // otherwise, if the last digit is 0**

**return 2;**

**}**

**}**

**// if the last digit is 0, we can have both 0 and 1 at the current position**

**if (last\_digit == 0) {**

**return countStrings(n - 1, 0) + countStrings(n - 1, 1);**

**}**

**// if the last digit is 1, we can have only 0 at the current position**

**else {**

**return countStrings(n - 1, 0);**

**}**

**}**

**int main()**

**{**

**// total number of digits**

**int n = 5;**

**cout << "The total number of " << n << "–digit binary numbers without any "**

**"consecutive 1's are " << countStrings(n, 0);**

**return 0;**

**}**

**69: Count all paths in a matrix from the first cell to the last cell**

**Given an M × N rectangular grid, efficiently count all paths starting from the first cell (0, 0) to the last cell (M-1, N-1). We can either move down or move towards right from a cell.  
Input: 3 × 3 matrix**

**Output: Total number of paths are 6**

**(0, 0) —> (0, 1) —> (0, 2) —> (1, 2) —> (2, 2)**

**(0, 0) —> (0, 1) —> (1, 1) —> (1, 2) —> (2, 2)**

**(0, 0) —> (0, 1) —> (1, 1) —> (2, 1) —> (2, 2)**

**(0, 0) —> (1, 0) —> (2, 0) —> (2, 1) —> (2, 2)**

**(0, 0) —> (1, 0) —> (1, 1) —> (1, 2) —> (2, 2)**

**(0, 0) —> (1, 0) —> (1, 1) —> (2, 1) —> (2, 2)**

**#include <stdio.h>**

**// Top-down recursive function to count all paths from cell (m, n)**

**// to the last cell (M-1, N-1) in a given `M × N` rectangular grid**

**int countPaths(int m, int n, int M, int N)**

**{**

**// there is only one way to reach the last cell**

**// when we are at the last row or the last column**

**if (m == M - 1 || n == N - 1) {**

**return 1;**

**}**

**return countPaths(m + 1, n, M, N) // move down**

**+ countPaths(m, n + 1, M, N); // move right**

**}**

**int main(void)**

**{**

**// `M × N` matrix**

**int M = 3;**

**int N = 3;**

**int k = countPaths(0, 0, M, N);**

**printf("The total number of paths is %d", k);**

**return 0;**

**}**

**70: Program to find n’th Fibonacci number**

**Write a program to calculate the nth Fibonacci number where n is a given positive number.  
Fibonacci’s sequence is characterized by the fact that every number after the first two is the sum of the two preceding ones. For example, consider the following series:  
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, … and so on.  
As we can see above, each subsequent number is the sum of the previous two numbers. The starting point of the sequence is sometimes considered 1, resulting in the first two numbers in the Fibonacci sequence as 1 and 1.**

**#include <stdio.h>**

**// Function to find the nth Fibonacci number**

**int fib(int n)**

**{**

**if (n <= 1) {**

**return n;**

**}**

**return fib(n - 1) + fib(n - 2);**

**}**

**int main()**

**{**

**int n = 8;**

**printf("F(n) = %d", fib(n));**

**return 0;**

**}**

**71. How to get the matching characters in a string?**

**Answer: To get the matching characters in a string, the below steps are followed:**

1. **Hash Map data structure is taken which works with the key-value pair.**
2. **Loop the strings, character by character, and verify if that character of the string exists in the hash map or not.**
3. **If the result is true, the counter for the character in the hash map is increased or else then put a count as 1.**
4. **Once the loop ends, then the Hash map is traversed and print the characters with more than 1 count.**

**Code snippet:**

| **HashMap<Character, Integer> mp = new HashMap<> ();**  **for (int j = 0; j<text.length (); j++) {**  **char ch = text.charAt(j);**  **if(mp.containsKey(ch)){**  **int cnt = mp.get(ch);**  **mp.put(ch, ++cnt);**  **}else{**  **mp.put(ch, 1);**  **}**  **}**  **Set<Character> charct = map.keySet();**    **for (Character ch: charct){**  **int c= mp.get(ch);**  **if(c>1){**  **System.out.println(ch+ " - " + c);**  **}**  **}** |
| --- |

**72. How to calculate the number of vowels and consonants in a string?**

**Answer: To calculate the number of vowels and consonants in a string, the below steps are followed:**

1. Get the string on which count has to be performed.
2. Run a loop from 0 to the length of the string.
3. Take a single character at a time and verify if they are a part of the group of vowels.
4. If the result is true, increase the count of vowels or else increment the count of consonants.

**Code snippet:**

| for (int k = 0; k < text.length(); k++) {  char c = text.charAt(k);  if (c == 'a' || c == 'e' || c == 'i' ||  c == 'o' || c == 'u')  owls += vowls  else  consonts += consonts  }  System.out.println("Vowel count is " + vowls);  System.out.println("Consonant count is: " + consonts); |
| --- |

**Q #11) How do you prove that the two strings are anagrams?**

**Answer:** Two strings are called anagrams if they accommodate a similar group of characters in a varied sequence.

**To check if two strings are anagrams, the below steps are followed:**

1. Initialize two strings in two variables.
2. Check if the length of the two strings is similar, if not then the strings are not an anagram.
3. If the result is true, take the two strings and store them in a character array.
4. Sort the two character arrays, then check if the two sorted arrays are alike.
5. If the result is true, the two strings are anagram else, not anagram.

**Code snippet:**

| if (str1.length() != str2.length()) {  System.out.println(str1 + " and " +str2 + " not anagrams string");  }else{  char[] anagram1 = str1.toCharArray();  char[] anagram2 = str2.toCharArray();  Arrays.sort(anagram1);  Arrays.sort(anagram2);  anagrmstat = Arrays.equals(anagram1, anagram2);  }  if (anagrmstat == true) {  System.out.println(str1 + " and " +str2 + " anagrams string");  }else{  System.out.println(str1 + " and " +str2 + " not anagrams string");  }  } |
| --- |

**73. Find the count for the occurrence of a particular character in a string.**

**Answer: To count the occurrence of a particular character in a string, the below steps are followed:**

1. Start with a string and a particular character whose occurrence shall be counted.
2. Start a loop from 0 to the length of the string.
3. Compare if a particular character of the string equals to the character that is being searched.
4. If the result is true, then increment the value of the counter.

**Code snippet:**

| for (int l=0; l<strng.length(); l++)  {  if ( strng.charAt(l)== searchedcharacter)  rslt ++;    }  System.out.println(rslt); |
| --- |

**74. How to verify if two strings are a rotation mutually?**

**Answer: To verify if two strings are a rotation mutually, the below steps are followed:**

1. Initialize the two strings in two variables.
2. Check if the length of two strings is similar, if not return false.
3. Join the string to itself.
4. Verify if the string which is rotated is present in the joined string.
5. If the result is true, the second string is a rotation of the first string.

**Code snippet:**

| String concat = org\_string + org\_string;  if (concat.indexOf (rotat) ! = -1) {  return true;  } |
| --- |

**75. How to calculate the number of numerical digits in a string?**

**Answer: To calculate the number of digits in a string, the below steps are followed:**

1. Get the string on which count has to be performed
2. Use the replaceAll function, which replaces all the numerical digits with “”.
3. Get the length of the string without digits.

**Code snippet:**

| package introduction;    public class GG {    public static void main(String[] args) {  // TODO Auto-generated method stub  String str = "TESTu45";  str=str.replaceAll("\\d", "");  int l = str.length();  System.out.println("The length of the string without digit is:" + l);    }    } |
| --- |

In this solution, a regular expression is utilized.

**76. How to compute the first character of a string that is not repeated?**

**Answer: To compute the first character of a string which is not repeated, the below steps are followed:**

1. A Set data structure for the repeated characters and a list for the non-repeated character is taken.
2. After segregating the repeated and non-repeated, at the end of the iteration, the first element of the list is printed in the console.

**Code snippet:**

| Set<Character> repeated = new HashSet<>();  List<Character> nonRepeated = new ArrayList<>();  for (int m = 0; m < wrd.length(); m++) {  char l = wrd.charAt(m);  if (repeated.contains(l)) {  continue;  }  if (nonRepeated.contains(l)) {  nonRepeated.remove((Character) l);  repeated.add(l);  } else {  nonRepeated.add(l);  }  }  return nonRepeated.get(0);  } |
| --- |

**77. How to search a missing number in an array that contains integers from 1 to 100?**

**Answer: To search a missing number in an array which contains integers from 1 to 100, the below steps are followed:**

1. Take an integer array with the numbers from 1 to 100.
2. Compute the sum of the numbers, the summation shall be= l\*(l+1)/2, where l is the number of integers.
3. Perform subtraction of the missing element from the total addition of numbers.

**78. How to get the matching elements in an integer array?**

**Answer: To get the matching elements in an integer array, the below steps are followed:**

1. Build two loops.
2. In the first loop, collect elements one at a time and add up the number of instances of the selected element.

**Code snippet:**

| for (m = 0; m < size; m++)  {  for (n = m + 1; n < size; n++)  {  if (arry[m] == arry[n])  System.out.print(arr[m]);  }  } |
| --- |

**79. How to delete the repeated elements in an integer array?**

**Answer: To delete the repeated elements in an integer array, the below steps are followed:**

1. Build a hashmap that will pick all the elements that are present before.
2. Loop through the array and verify if the element already exists in the hash map
3. If the result is true, array traversal is continued, otherwise, the element is printed out in the console.

**Code snippet:**

| HashMap<Integer,Boolean> m = new HashMap<>();  for (int j = 0; j < a.length); j++)  {  if (m.get(a[j]) == null)  System.out.print(a[j] + " ");  mp.put(a[j], true);  }  } |
| --- |

**80. Determine the largest and the smallest element of an array which is not sorted.**

**Answer: To determine the largest and the smallest element of an array the below steps need to be followed:**

1. Traverse the array, and monitor the maximum element found so far, till we are at the border of the array, the largest element is achieved.
2. Traverse the array, and monitor the minimum element found so far, till we are at the border of the array, the smallest element is achieved.

**81. Explain the bubble sort algorithm.**

**Answer: The bubble sort algorithm includes the following steps:**

1. Begin from the first element, then perform a comparison with the following element in the array
2. If the present element is larger than the following element of the array, then swap their positions.
3. If the present element is lesser than the following element of the array, shift to the next element, and again repeat step 1.

**Code snippet:**

| for(k = 0; k < arry.length; k++)  {  for(l = 0; l < arry.length-l-1; l++)  {  if( arry[l] > arr[l+1])  {  t = arry[l];  arry[l] = arry[l+1];  arry[l+1] = t;  }  } |
| --- |

**Q #21) Implement the insertion sort algorithm.**

**Answer:** Implementation of insertion sort.

**Code snippet:**

| for (m = 1; m < arry.length; m++)  {  n = m;  while (n> 0 && arry[n - 1] > arry[n])  {  k = arry[n];  arry[n] = arry[n - 1];  arry[n - 1] = k;  n--;  }  } |
| --- |

**82. Determine the second largest element of an array.**

**Answer: The second largest element of an array can be computed by the following steps:**

1. State the largest element as the first element of the array and the second largest element as the second element of the array.
2. Iterate through the loop for traversing the array.
3. IF arry[i] is greater than the largest element THEN  
   Second element ? largest element  
   Largest element ?arry[i]  
   IF second element is less than arry[i] THEN  
   Second element?arry[i]

**Code snippet:**

| if(arry[0] > arry[1]) {  l = arry[0];  s = arry[1];  } else {  l = arry[1];  s = arry[0];  }    for(i = 2; i < arry.length; i++) {  if( l < a[i] ) {  s = l;  l = arry[i];  } else if( s < arry[i] ) {  s = arry[i];  }  } |
| --- |

**83. Explain the reversal of an array.**

**Answer: Array reversal is performed in the following ways:**

1. Take an array with elements.
2. Now exchange the position of the first element with the final element, and similarly the second element with the penultimate element.
3. This will continue until the entire array is reversed.

**Code snippet:**

| for (t = 0; t < arr.length / 2; t++) {  tmp = arr[t];  arr[t] = arr[arr.length - t - 1];  arr[arr.length - t- 1] = tmp;  } |
| --- |

**Q #24) How to remove special characters in a string that is in lowercase?**

**Answer:** Special characters in a string can be removed by using the replaceAll function in Java.

**Code snippet:**

| string str = “Testing@”  str.replaceAll(“[^a-z0-9]”,””) |
| --- |

In this solution, a regular expression is utilized.

**84. How to perform swapping two strings by not using a third variable?**

**Answer: Two strings are swapped without the help of the third variable by the following steps:**

**(i)** Take two strings i, j, and append them then store in the first string.

**(ii)** Using the substring method extract the string:

j = substring(0,i.length()-j.length())

**(iii)** Store the string j in string i

i= subsrtirng(j.length)

**Code snippet:**

| string i = “abc”, j =”def”;  i = i+j;  j = i. substring(0,i.length()-j.length());  i = i.substring(j.length())  System.out.println(i +””+j); |
| --- |

**85. How to traverse to the middle of a linked list?**

**Answer: To traverse to the middle of a linked list the below steps are followed:**

1. Declare two pointers first and second which are initialized to the linked list head.
2. Increment the first linked list by two nodes and second by one node in each loop.
3. While the first node reaches the end of the list, the second node will point to the middle.

**Code snippet:**

| first = second = head;  while(first !=null) {  first = first.next;  if(first != null && first.next != null) {  second = second.next;  first = first.next;  }  }  return second;    } |
| --- |

**86. Implement the process of reversing a linked list.**

**Answer: A linked list can be reversed by the below steps:**

1. Declare three nodes preceding, present, and following.
2. While in the present node, the preceding will be null.
3. Let the present.next be preceding to reverse the list.
4. In each looping, present and preceding are incremented by 1.

**Code snippet:**

| Node preceding=null;  Node following  while(present!=null)  {  following=present.next;  present.next=preceding;  preceding=present;  present=following;  }  return preceding;  } |
| --- |

**87. What is the process of deleting matched elements from a linked list which is not sorted.**

**Answer: To delete matched elements from a linked list which is not sorted, the below steps are followed:**

1. Travel from the head to the tail of the linked list.
2. For every value in the linked list, verify if it’s already present in the hash table.
3. If the result is true, the element is not added to the hash table.

**Code snippet:**

| HashSet<Integer> h = new HashSet<>();    node present = head;  node preceding = null;  while (present != null)  {  int presentval = present.value;    if (h.contains(presentval)) {  preceding.next = present.next;  } else {  h.add(presentval);  preceding = present;  }  present = present.next;  }    } |
| --- |

**Q #29) How to get the length of a linked list?**

**Answer:** **To get the length of a linked list, the below steps are followed:**

1. Start a counter with a value of 0 and present node as head.
2. Till the present node is not null , perform these :
   * present = present -> next
   * counter = counter + 1
3. Counter value is returned.

**Code snippet:**

| {  Node present = head;  int c = 0;  while (present != null)  {  c = c + 1;  present = present.next;  }  return c;  } |
| --- |

**88. How to search a specific value in a linked list?**

**Answer: To search a specific value in a linked list, the below steps are followed:**

1. Declare present node as head.
2. Till the present node is not null, perform these :
   * present -> value is equal to the value being looked for return true.
   * present = present -> next.
3. If not found, false is returned.

**Code snippet:**

| Node present = head;  while (present != null)  {  if (present.value == value)  return true;  present = present.next;  }  return false;  } |
| --- |

**Q #31) How to verify if a number is prime or not?**

**Answer:** **To verify if a number is prime or not, the below steps are followed:**

1. Start a loop from value 2(k) up to (number / 2)
2. If the number is perfectly divisible by k, then the number is non – prime.
3. If the number is not perfectly divisible except for 1 and by itself, then the number is prime.

**Code snippet:**

| for(k = 2; k <= number / 2; k++)  {  if(number % k == 0)  {  stat = false;  break;  }  }  if (stat)  System.out.println("Prime";    else  System.out.println("Not prime"); |
| --- |

**89. How to get the third node of a linked list?**

**Answer: To get to the third node of the linked list the below steps are followed:**

1. Start a counter with a value 0.
2. Iterate through the linked list and perform these steps:
   * If the value of the counter is 3, then the present node is returned.
   * Counter is increased by 1.
   * Modify the present such that it implies to the next of the present.

**Code snippet:**

| Node present = head;  int c = 0;  while (c != null)  {  if (c == 3)  return present.val;  c = c+1;  present = present.next;  } |
| --- |

**Q #33) Compute the first five Fibonacci numbers.**

**Answer:** 0 and 1 are the first two Fibonacci numbers and all the numbers after 0 and 1 are the addition of the two previous numbers.

**Code snippet:**

| int num1=0, num2=1,t;  for ( int k = 0; k<5,k++)  {  System.out.println(num1);  t = num1 + num2;  num1 = num2; num2 = t;  } |
| --- |

**90. How to reverse a number?**

**Answer: Reversal of a number is achieved in the following steps:**

1. Take out the rightmost digit of the number.
2. Sum up the digit with the new reversed number.
3. Perform multiplication by 10.
4. Divide the number by 10.

**100. Determine the factors of a number.**

**Answer: The factors of a number is expressed by the following code snippet:**

| int no = 75;    for(int j = 1; j <= no; j++) {  if (no % j == 0) {  System.out.print(j);  } |
| --- |

#### 