DSA

<u>Problems with their solutions</u>

1. Problem: Number of Bids

You are given a list of project costs. Find the number of distinct pairs (a, b) where |a - b| = target.
Each pair should be unique and order does not matter.

Optimized Approach: Using Hashing (unordered_set)

Instead of using **nested loops ($O(N^2)$)**, we can use a **hash set (unordered_set)** to achieve **O(N)** time complexity.

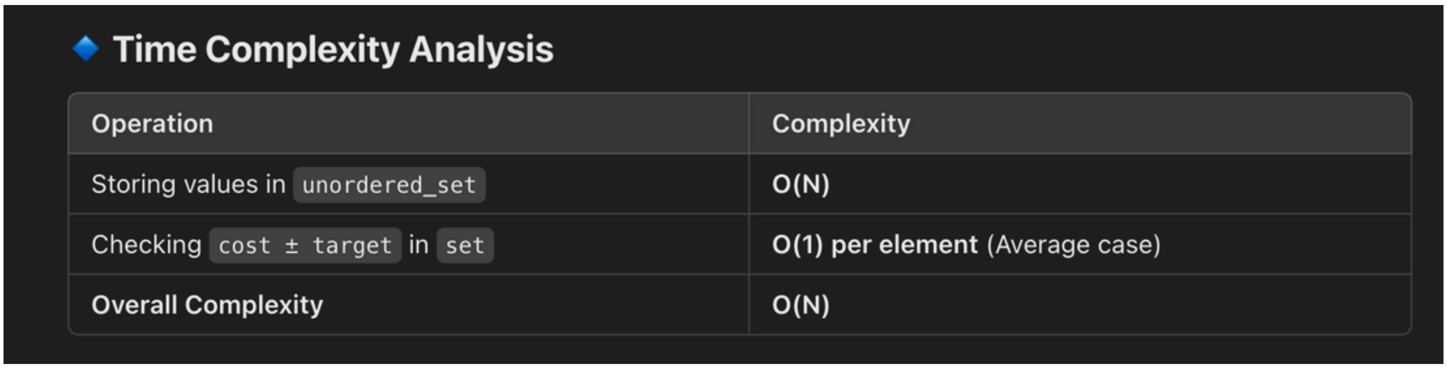
Steps to Solve

1. **Use a hash set (set<int>)** to store project costs.
2. **Iterate through the costs** and check if cost - target exists in the set.
3. **Count valid pairs** using the condition |cost - target| = target.

Code (Optimized - O(N))

```
#include <iostream>

#include <unordered_set>

#include <vector>

using namespace std;

int countPairs(vector<int>& projectCost, int target) {

   unordered_set<int> seen;

   int count = 0;

   for (int cost : projectCost) { //RANGE BASED LOOP

      if (seen.count(cost - target)) count++; //checking the difference

      if (seen.count(cost + target)) count++;

      seen.insert(cost);   } //inserting the values of the array projectcost
```

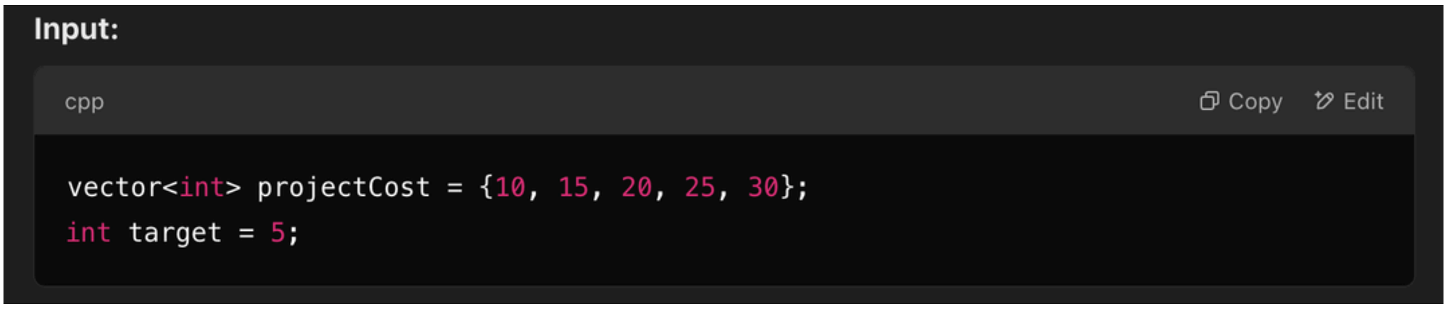
```
    return count;

}

int main() {

    vector<int> projectCost = {1, 3, 5}; //EXAMPLE INPUT

    int target = 2; //EXAMPLE INPUT

    cout << countPairs(projectCost, target) << endl;

    return 0;

}
```



🔹 **Time Complexity Analysis**

Operation	Complexity
Storing values in `unordered_set`	**O(N)**
Checking `cost ± target` in `set`	**O(1) per element** (Average case)
Overall Complexity	**O(N)**

TRY A NEW INPUT:



Input:

```cpp
vector<int> projectCost = {10, 15, 20, 25, 30};
int target = 5;
```

References:

Unordered Sets in C++ Standard Template Library - GeeksforGeeks

2. Problem: Longest Common Subsequence (LCS) of Two Strings

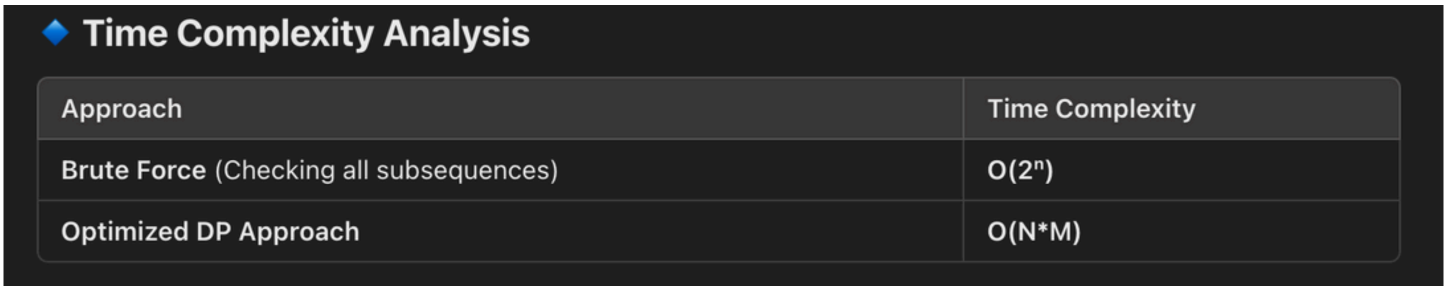
Find the length of the longest common subsequence between two strings using **Dynamic Programming (DP).**
Instead of checking all subsequences (**O(2^n)**), **we use DP to reduce time complexity to O(N*M).**

Steps to Solve

1. **Create a DP table dp[i][j]**
   - dp[i][j] stores the LCS length of s1[0...i-1] and s2[0...j-1].
2. **Fill the DP table:**
   - If s1[i-1] == s2[j-1], we include this character:
     dp[i][j] = dp[i-1][j-1] + 1
   - Else, take the max of the previous results:
     dp[i][j] = max(dp[i-1][j], dp[i][j-1])
3. **Return dp[n][m]**, the answer for the full strings.

Code (Optimized - O(N*M))

```
#include <iostream>

#include <vector>

using namespace std;

int LCS(string s1, string s2) {

   int n = s1.length(), m = s2.length();

   vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

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

      for (int j = 1; j <= m; j++) {

         if (s1[i-1] == s2[j-1]) //IF EQUAL THEN INCREMENT THE DIAGONAL VALUE

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

         else

            dp[i][j] = max(dp[i-1][j], dp[i][j-1]);      }}

      //OR ELSE TAKE THE MAX OF THE ABOVE AND LEFT VALUE

   return dp[n][m];
}

int main() {

   string s1 = "abcde", s2 = "ace"; //EXAMPLE INPUT

   cout << LCS(s1, s2) << endl;

   return 0;

}
```



🔷 **Time Complexity Analysis**

Approach	Time Complexity
Brute Force (Checking all subsequences)	**$O(2^n)$**
Optimized DP Approach	**O(N*M)**

TRY A NEW INPUT:

Consider the strings:

s1 = "abcde"

s2 = "ace"

s3 = "aec"

References:

🌐 Longest Common Subsequence—Root of *almost* all substring questions.

🌐 Dynamic Programming or DP - GeeksforGeeks

3. Problem: Space Separated Strings

Find and return repeated words in **lexicographical order** as a space-separated string.

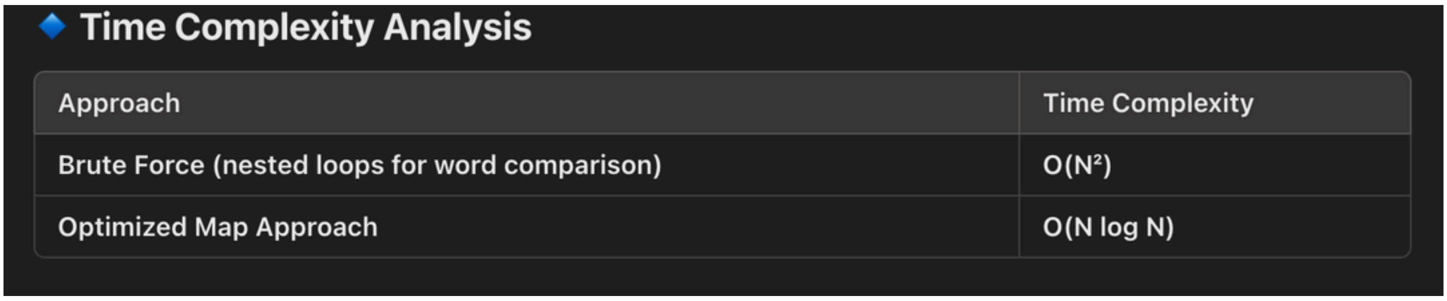
Steps to Solve

1. **Use stringstream to split words** from the input.

2. **Use map<string, int> to store word counts**.
3. **Iterate through the map** and collect words that appear **more than once**.
4. **Return space-separated result** or "NA" if no words are repeated.

Code (Optimized - O(N log N)):

```
#include <iostream>

#include <sstream>

#include <map>

using namespace std;

string findRepeatedWords(string text) {

  stringstream ss(text); //SPLIT THE STRING

  string word, result = "";

  map<string, int> wordCount;

  while (ss >> word) wordCount[word]++; //FREQUENCY IS MARKED IN THE MAP

  for (auto &entry : wordCount)

    if (entry.second > 1) result += entry.first + " ";

//if occurred more than once then add it to the result

  return result.empty() ? "NA" : result.substr(0, result.size() - 1);

}

int main() {

  string text = "cat batman latt matter cat matter cat"; //EXAMPLE INPUT

  cout << findRepeatedWords(text) << endl;

  return 0;

}
```



🔷 Time Complexity Analysis

Approach	Time Complexity
Brute Force (nested loops for word comparison)	$O(N^2)$
Optimized Map Approach	O(N log N)

TRY A NEW INPUT:

Input:

how hat has when has hat one

References:

🌐 Map in C++ Standard Template Library (STL) - GeeksforGeeks

🌐 stringstream in C++ and its Applications - GeeksforGeeks

4. Problem: Count of Specific Character

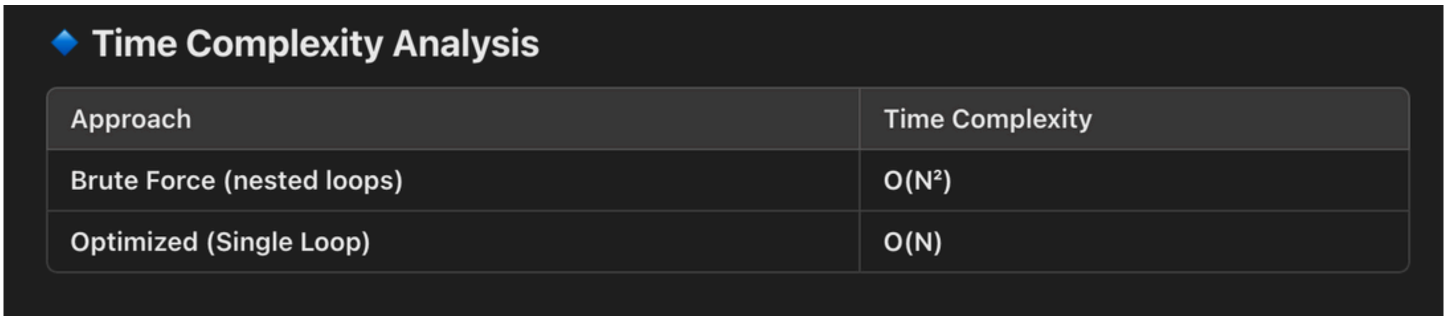
Find the count of a specific character in a string.

Steps to Solve

1. **Iterate through the string** and count occurrences of the given character.
2. **Return the count**.

Code (Optimized - O(N))

```
#include <iostream>

using namespace std;

int countCharacter(string data, char target) {

  int count = 0;

  for (char c : data) //range based loop is used to maintain efficiency and simplicity

    if (c == target) count++;

  return count;
}

int main() {

    string data = "hello world!"; //EXAMPLE INPUT

    char target = 'o';  //EXAMPLE INPUT

    cout << countCharacter(data, target) << endl;

    return 0;

}
```



🔷 Time Complexity Analysis

Approach	Time Complexity
Brute Force (nested loops)	$O(N^2)$
Optimized (Single Loop)	O(N)

TRY A NEW INPUT:
Input:

haveagoodday

References:

Range-Based for Loop in C++ - GeeksforGeeks

5. Problem: Unsold Products

Move all 0s (unsold products) to the end while maintaining order.

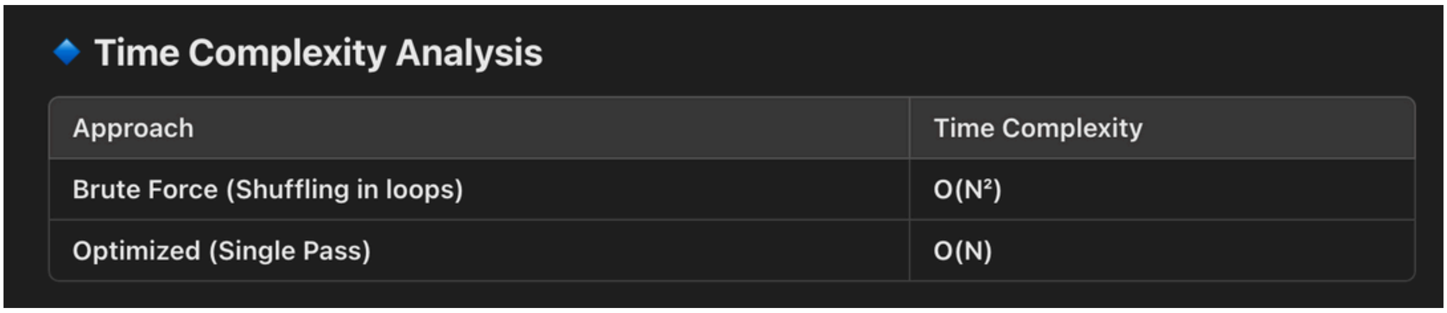
Steps to Solve

1. **Iterate through array, keeping only non-zero values.**
2. **Fill remaining places with zeroes.**

Code (Optimized - O(N))

```
#include <iostream>
```

```
#include <vector>

using namespace std;

vector<int> moveZerosToEnd(vector<int>& arr) {

   int index = 0;

   for (int num : arr) if (num != 0) arr[index++] = num;

//range based loop is used to maintain efficiency and simplicity

   while (index < arr.size()) arr[index++] = 0;

   return arr;

}

int main() {

   vector<int> arr = {5, 2, 0, 8, 0, 2, 1}; //EXAMPLE INPUT

   vector<int> result = moveZerosToEnd(arr);

   for (int num : result) cout << num << " ";

   return 0;

}
```



🔷 **Time Complexity Analysis**

Approach	Time Complexity
Brute Force (Shuffling in loops)	$O(N^2)$
Optimized (Single Pass)	$O(N)$

TRY A NEW INPUT:

Input1 : {3,0,0,9,0,1,1}

6. Problem: Alice and String

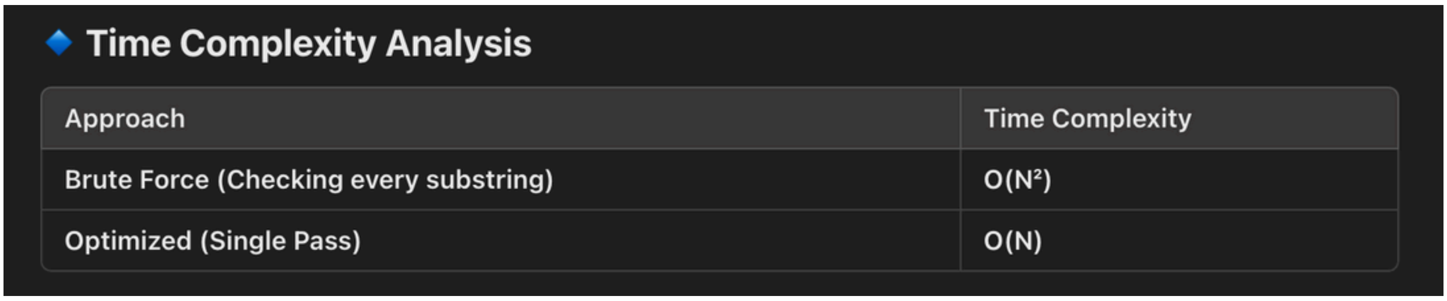
Find the longest substring **without '.' interruptions**.

Steps to Solve

1. **Use a counter** to track the longest streak.
2. **Reset when encountering ..**

Code (Optimized - O(N))

```cpp
#include <iostream>

using namespace std;

int longestUninterruptedSubstring(string s) {

   int maxLen = 0, currLen = 0;

   for (char c : s) {

      if (c == '.') currLen = 0;

      else maxLen = max(maxLen, ++currLen);

   }

   return maxLen;

}

int main() {

   string s = "abc.b.cc"; //EXAMPLE INPUT

   cout << longestUninterruptedSubstring(s) << endl;

   return 0;

}
```



Time Complexity Analysis

Approach	Time Complexity
Brute Force (Checking every substring)	$O(N^2)$
Optimized (Single Pass)	O(N)

TRY A NEW INPUT:

Input1 : ....

References:

🌐 std::max in C++ - GeeksforGeeks

7. Problem: Even Odd

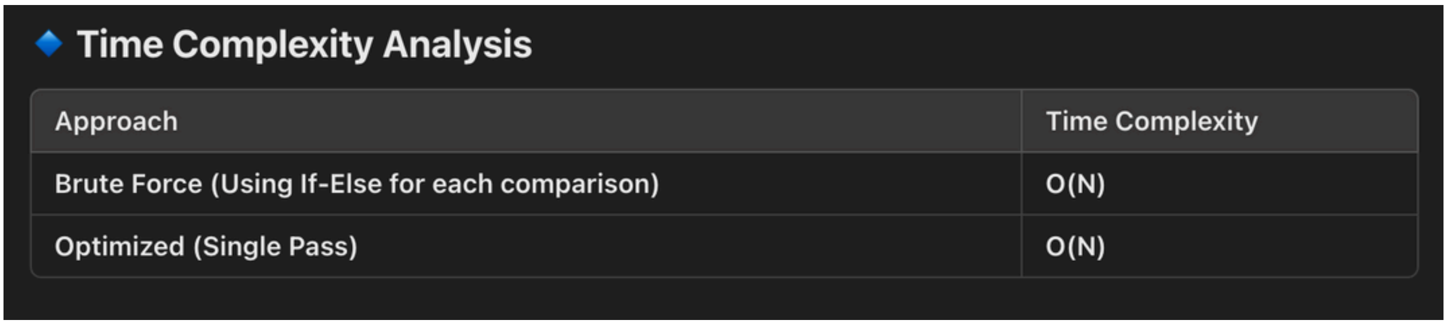
Convert an array into a string representation of "EvenOdd".

Steps to Solve

1. **Iterate through the array** and check if numbers are even or odd.
2. **Build the output string** accordingly.

Code (Optimized - O(N))

```
#include <iostream>

#include <vector>

using namespace std;

string evenOddLabels(vector<int>& arr) {

    string result = "";

    for (int num : arr)

        result += (num % 2 == 0 ? "Even" : "Odd");

    return result;

}

int main() {

    vector<int> arr = {1, 2, 3, 4, 5, 6}; //EXAMPLE INPUT

    cout << evenOddLabels(arr) << endl;

    return 0;

}
```



🔹 Time Complexity Analysis

Approach	Time Complexity
Brute Force (Using If-Else for each comparison)	O(N)
Optimized (Single Pass)	O(N)

TRY A NEW INPUT:

Input1 : [2,2,3,4,4,-1]

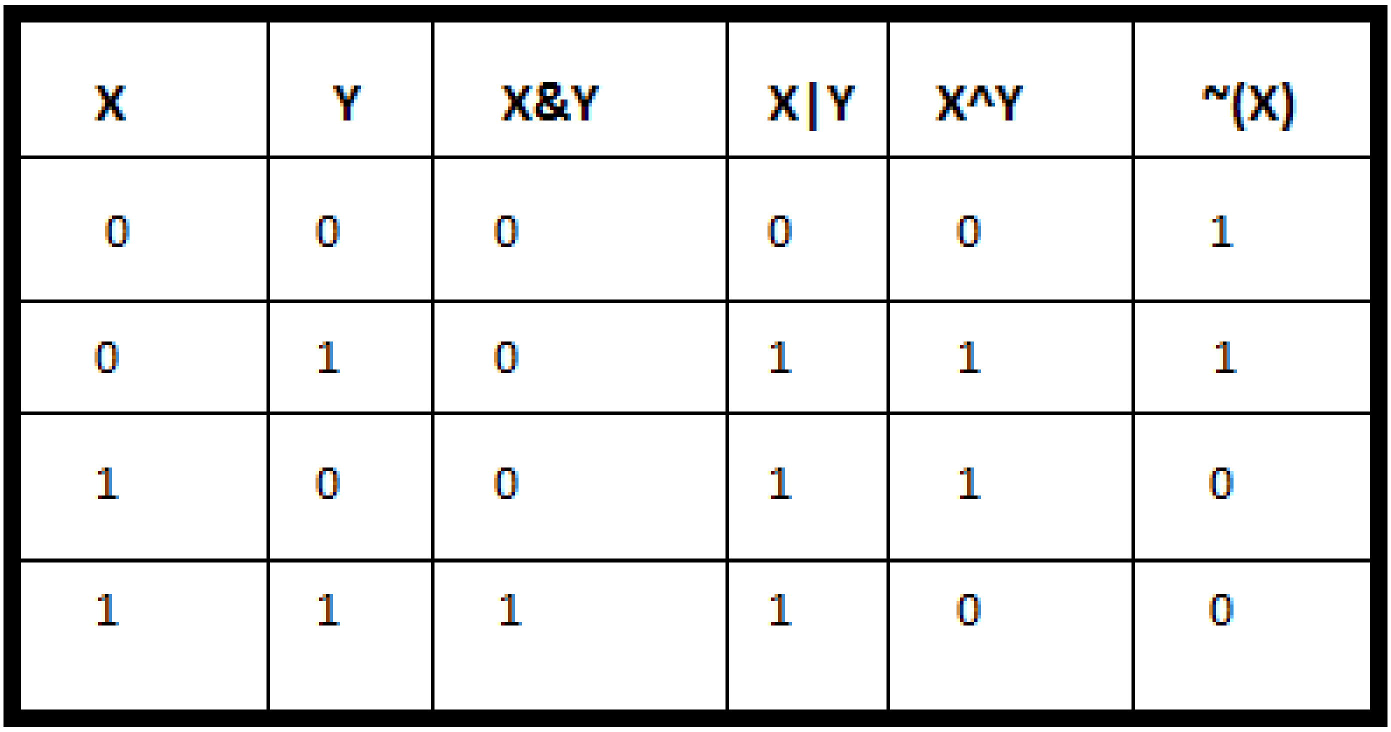
References:

🌐 Strings in C++ - GeeksforGeeks

Bit manipulation

Bitwise Operators:

NOT (~), AND (&), OR (|), XOR (^)



X	Y	X&Y	X|Y	X^Y	~(X)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Left Shift (<<): Left shift operator is a binary operator which shift the some number of bits, in the given bit pattern, to the left and append 0 at the end. Left shift is equivalent to multiplying the bit pattern with 2^k (if we are shifting k bits).

Right Shift (>>): Right shift operator is a binary operator which shift the some number of bits, in the given bit pattern, to the right and append 1 at the end. Right shift is equivalent to dividing the bit pattern with 2^k (if we are shifting k bits).

Problems

---

1) How to check if a given number is a power of 2 ?

Brute force approach:

```
bool isPowerOfTwo(int x)
    {
        if(x == 0)
            return false;
        else
        {
            while(x % 2 == 0) x /= 2;
            return (x == 1);
            }
    }
```

Above function will return true if x is a power of 2, otherwise false.

Time complexity of the above code is **O(logN).**

Optimized approach:

```
bool isPowerOfTwo(int x)
  {
      // x will check if x == 0 and !(x & (x - 1)) will check if x is a power of 2 or not
      return (x && !(x & (x - 1)));
  }
```

2) Count the number of ones in the binary representation of the given number.

Optimized approach:

```
int count_one (int n)
    {
        while( n )
        {
        n = n&(n-1);
           count++;
        }
        return count;
}
```

Brute force approach:

The basic approach to evaluate the binary form of a number is to traverse on it and count the number of ones. But this approach takes **log2N of time in every case.**

Why log2N ?

As to get a number in its binary form, we have to divide it by 2, until it gets 0, which will take log2N of time.

Optimized approach:

With bitwise operations, we can use an algorithm whose running time depends on the number of ones present in the binary form of the given number. This algorithm is much better, as it will reach to **logN, only in its worst case.**

3) Check if the ith bit is set in the binary form of the given number.

```
bool check (int N)
    {
        if( N & (1 << i) )
            return true;
        else
            return false;
    }
```

If i is a constant, the function runs in **O(1)** time.

If i is calculated dynamically, the function may take **O(log N)** time, depending on how i is derived.

Explanation and other topics given below:

Basics of Bit Manipulation Tutorials & Notes | Basic Programming | HackerEarth

---

References:

DSA Cheat sheets:

takeuforward - Best Coding Tutorials for Free

DSA Sheet by Love Babbar - GeeksforGeeks

Competitive programming for placements:

Top Interview 150 - Study Plan - LeetCode