rohit

Basic Problems on Bit Manipulation

Below are some problems which can be solved very easily using some of the basic concepts of Bit Manipulation. Let's look at each of these problems and the Bitwise approach of solving them:

**Problem 1**: Given a number N, the task is to check whether the given number is a power of 2 or not.

**Example**:

**Input** : N = 4

**Output** : Yes

22 = 4

**Input** : N = 7

**Output** : No

**Input** : N = 32

**Output** : Yes

25 = 32

* **Bitwise Solution**: If we subtract a number which is a power of 2 with 1 then all of it's unset bits after the only set bit become set; and the set bit become unset. For example, consider **4**( Binary representation: 100) and **16**(Binary representation: 10000), we get following after subtracting 1 from them:

3 –> 011

15 –> 01111

* You can clearly see that **bitwise-AND(&)** of 4 and 3 gives zero, similarly 16 and 15 also gives zero. So, if a number N is a power of 2 then **bitwise-AND(&)** of **N**and**N-1** will be zero. We can say that N is a power of 2 or not based on the value of N&(N-1).

**Problem 2**: Given a number N, find the most significant set bit in the given number.

**Examples**:

**Input** : N = 10

**Output** : 8

Binary representation of 10 is 1010

The most significant bit corresponds

to decimal number 8.

**Input** : N = 18

**Output** : 16

* **Bitwise Solution**: The most-significant bit in binary representation of a number is the highest ordered bit, that is it is the bit-position with highest value.   
  One of the solution is first find the bit-position corresponding to the **MSB** in the given number, this can be done by calculating logarithm base 2 of the given number, i.e., **log2(N)** gives the position of the MSB in N.   
  Once, we know the position of the **MSB**, calculate the value corresponding to it by raising 2 by the power of calculated position.   
  That is, **value = 2log2(N)**.

**Problem 3**: Given a number N, the task is to find the XOR of all numbers from 1 to N.

**Examples** :

**Input** : N = 6

**Output** : 7

// 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 = 7

**Input** : N = 7

**Output** : 0

// 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7 = 0

* **Solution**:
  1. Find the remainder of N by moduling it with 4.
  2. If rem = 0, then xor will be same as N.
  3. If rem = 1, then xor will be 1.
  4. If rem = 2, then xor will be N+1.
  5. If rem = 3 ,then xor will be 0.

heck if Kth bit is set or not

Given a number **n**, check if the Kth bit of n is set or not.

**Examples:**

***Input****: n = 5, k = 1*  
***Output:****SET*  
***Explanation:****5 is represented as 101 in binary and has its first bit set.*

***Input:****n = 2, k = 3*  
***Output:****NOT SET*  
***Explanation:****2 is represented as 10 in binary, all higher i.e. beyond MSB, bits are NOT SET.*

**Check whether the K-th bit is set or not Using Left Shift Operator:**

To solve the problem follow the below idea:

* *Left shift given number 1 by k-1 to create a number that has only set bit as k-th bit.*  
  *temp = 1 << (k-1)*
* *If****bitwise AND of n and temp****is non-zero, then result is SET else result is NOT SET.*

Dry run of the above approach:

***Input:****n = 75 and k = 4*  
*temp = 1 << (k-1) = 1 << 3 = 8*  
*Binary Representation of temp = 0..00001000*  
*Binary Representation of n = 0..01001011*  
*Since bitwise AND of n and temp is non-zero, result is SET.*

Below is the implementation of the above approach:

C++Java

// CPP program to check if k-th bit

// of a given number is set or not

#include <bits/stdc++.h>

using namespace std;

void isKthBitSet(int n, int k)

{

if (n & (1 << (k - 1)))

cout << "SET";

else

cout << "NOT SET";

}

// Driver code

int main()

{

int n = 5, k = 1;

// Function call

isKthBitSet(n, k);

return 0;

}

**Output**

SET

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

**Check whether the K-th bit is set or not Using Right Shift Operator:**

To solve the problem follow the below idea:

*If we right shift n by k-1, we get the last bit as 1 if the Kth bit is set else 0*

Below is the implementation of the above approach:

C++Java

// CPP program to check if k-th bit

// of a given number is set or not using

// right shift operator.

#include <iostream>

using namespace std;

void isKthBitSet(int n, int k)

{

if ((n >> (k - 1)) & 1)

cout << "SET";

else

cout << "NOT SET";

}

// Driver code

int main()

{

int n = 5, k = 1;

// Function call

isKthBitSet(n, k);

return 0;

}

**Output**

SET

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

Power of Two

Given a positive integer **n**, write a function to find if it is a power of 2 or not

**Examples:**

***Input :****n = 4*  
***Output :****Yes*  
***Explanation:****22 = 4*

***Input :****n = 32*  
***Output :****Yes*  
***Explanation:****25 = 32*

To solve the problem follow the below idea:

*A simple method for this is to simply take the log of the number on base 2 and if you get an integer then the number is the power of 2*

Below is the implementation of the above approach:

C++Java

// C++ Program to find whether a

// no is a power of two

#include <bits/stdc++.h>

using namespace std;

// Function to check if x is power of 2

bool isPowerOfTwo(int n)

{

if (n == 0)

return false;

return (ceil(log2(n)) == floor(log2(n)));

}

// Driver code

int main()

{

// Function call

isPowerOfTwo(31) ? cout << "Yes" << endl

: cout << "No" << endl;

isPowerOfTwo(64) ? cout << "Yes" << endl

: cout << "No" << endl;

return 0;

}

**Output**

No

Yes

**Find whether a given number is a power of 2 using Brian Kernighan’s algorithm:**

To solve the problem follow the below idea:

*As we know that the number which will be the power of two have only one set bit , therefore when we do bitwise AND with the number which is just less than the number which can be represented as the power of (2) then the result will be 0 .*

***Example :****4 can be represented as (2^2 ) ,*  
*(4 & 3)=0  or in binary (100 & 011=0)*

Below is the implementation of the above approach:

C++Java

// C++ program of the above approach

#include <bits/stdc++.h>

using namespace std;

/\* Function to check if x is power of 2\*/

bool isPowerofTwo(long long n)

{

return (n != 0) && ((n & (n - 1)) == 0);

}

// Driver code

int main()

{

// Function call

isPowerofTwo(30) ? cout << "Yes\n" : cout << "No\n";

isPowerofTwo(128) ? cout << "Yes\n" : cout << "No\n";

return 0;

}

**Output**

No

Yes

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

One Odd Occurring

Given an array of positive integers. All numbers occur an even number of times except one number which occurs an odd number of times. Find the number in O(n) time & constant space.

**Examples :**

***Input :****arr = {1, 2, 3, 2, 3, 1, 3}*  
***Output :****3*

***Input :****arr = {5, 7, 2, 7, 5, 2, 5}*  
***Output :****5*

A **Simple Solution** is to run two nested loops. The outer loop picks all elements one by one and the inner loop counts the number of occurrences of the element picked by the outer loop. The time complexity of this solution is O(n2).

Below is the implementation of the brute force approach :

C++Java

// C++ program to find the element

// occurring odd number of times

#include <bits/stdc++.h>

using namespace std;

// Function to find the element

// occurring odd number of times

int getOddOccurrence(int arr[], int arr\_size)

{

for (int i = 0; i < arr\_size; i++) {

int count = 0;

for (int j = 0; j < arr\_size; j++) {

if (arr[i] == arr[j])

count++;

}

if (count % 2 != 0)

return arr[i];

}

return -1;

}

// driver code

int main()

{

int arr[] = { 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };

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

// Function calling

cout << getOddOccurrence(arr, n);

return 0;

}

**Output :**

5

***Time Complexity:****O(n^2)*  
***Auxiliary Space:****O(1)*

The **Best Solution** is to do bitwise XOR of all the elements. XOR of all elements gives us odd occurring elements.

Here **^** is the **XOR** operators;

**Note :**

x^0 = x

x^y=y^x (**Commutative property holds**)

(x^y)^z = x^(y^z) (**Distributive property holds**)

x^x=0

Below is the implementation of the above approach.

C++Java

// C++ program to find the element

// occurring odd number of times

#include <bits/stdc++.h>

using namespace std;

// Function to find element occurring

// odd number of times

int getOddOccurrence(int ar[], int ar\_size)

{

int res = 0;

for (int i = 0; i < ar\_size; i++)

res = res ^ ar[i];

return res;

}

/\* Driver function to test above function \*/

int main()

{

int ar[] = {2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};

int n = sizeof(ar)/sizeof(ar[0]);

// Function calling

cout << getOddOccurrence(ar, n);

return 0;

}

**Output :**

5

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

One Odd Occurring

Given an array of positive integers. All numbers occur an even number of times except one number which occurs an odd number of times. Find the number in O(n) time & constant space.

**Examples :**

***Input :****arr = {1, 2, 3, 2, 3, 1, 3}*  
***Output :****3*

***Input :****arr = {5, 7, 2, 7, 5, 2, 5}*  
***Output :****5*

A **Simple Solution** is to run two nested loops. The outer loop picks all elements one by one and the inner loop counts the number of occurrences of the element picked by the outer loop. The time complexity of this solution is O(n2).

Below is the implementation of the brute force approach :

C++Java

// C++ program to find the element

// occurring odd number of times

#include <bits/stdc++.h>

using namespace std;

// Function to find the element

// occurring odd number of times

int getOddOccurrence(int arr[], int arr\_size)

{

for (int i = 0; i < arr\_size; i++) {

int count = 0;

for (int j = 0; j < arr\_size; j++) {

if (arr[i] == arr[j])

count++;

}

if (count % 2 != 0)

return arr[i];

}

return -1;

}

// driver code

int main()

{

int arr[] = { 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };

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

// Function calling

cout << getOddOccurrence(arr, n);

return 0;

}

**Output :**

5

***Time Complexity:****O(n^2)*  
***Auxiliary Space:****O(1)*

The **Best Solution** is to do bitwise XOR of all the elements. XOR of all elements gives us odd occurring elements.

Here **^** is the **XOR** operators;

**Note :**

x^0 = x

x^y=y^x (**Commutative property holds**)

(x^y)^z = x^(y^z) (**Distributive property holds**)

x^x=0

Below is the implementation of the above approach.

C++Java

// C++ program to find the element

// occurring odd number of times

#include <bits/stdc++.h>

using namespace std;

// Function to find element occurring

// odd number of times

int getOddOccurrence(int ar[], int ar\_size)

{

int res = 0;

for (int i = 0; i < ar\_size; i++)

res = res ^ ar[i];

return res;

}

/\* Driver function to test above function \*/

int main()

{

int ar[] = {2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};

int n = sizeof(ar)/sizeof(ar[0]);

// Function calling

cout << getOddOccurrence(ar, n);

return 0;

}

**Output :**

5

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

Two Odd Occurring

Given an unsorted array that contains even number of occurrences for all numbers except two numbers. Find the two numbers which have odd occurrences in O(n) time complexity and O(1) extra space.

Examples:

Input: {12, 23, 34, 12, 12, 23, 12, 45}

Output: 34 and 45

Input: {4, 4, 100, 5000, 4, 4, 4, 4, 100, 100}

Output: 100 and 5000

Input: {10, 20}

Output: 10 and 20

A **naive method**to solve this problem is to run two nested loops. The outer loop picks an element and the inner loop counts the number of occurrences of the picked element. If the count of occurrences is odd then print the number. The time complexity of this method is O(n^2).

We can **use sorting** to get the odd occurring numbers in O(nLogn) time. First sort the numbers using an O(nLogn) sorting algorithm like Merge Sort, Heap Sort.. etc. Once the array is sorted, all we need to do is a linear scan of the array and print the odd occurring number.

We can also **use hashing**. Create an empty hash table which will have elements and their counts. Pick all elements of input array one by one. Look for the picked element in hash table. If the element is found in hash table, increment its count in table. If the element is not found, then enter it in hash table with count as 1. After all elements are entered in hash table, scan the hash table and print elements with odd count. This approach may take O(n) time on average, but it requires O(n) extra space.

**A O(n) time and O(1) extra space solution using XOR:**   
Let the two odd occurring numbers be x and y. We **use bitwise XOR** to get x and y. We try to make 2 groups such that x and y go to different groups. E.g. [a, a, b, b, x], . Then the problem will become “Find ‘one’ number with odd occurrence in an unsorted array”, which becomes a simple problem and will be solved using XOR. Below are steps to group x and y differently.

**1.**    The first step is to do XOR of all elements present in array. XOR of all elements gives us XOR of x and y because of the following properties of XOR operation.   
        1) XOR of any number n with itself gives us 0, i.e., n ^ n = 0   
        2) XOR of any number n with 0 gives us n, i.e., n ^ 0 = n   
        3) XOR is cumulative and associative.

        So we have XOR of x and y after the first step, in decimal form. E.g. 5 ^ 6 returns 3, which is computed in bit form as 101 ^ 110 = 011. Let the ‘value’ of XOR be xor2. Every Set bit\*\* in xor2 indicates that ‘the corresponding bits in x and y have values different from each other’ (XOR property- ‘1 when bits are different’).

\*\* ( Set-bits are 1’s in binary form. E.g. 101 has 2 set bits(1’s), at 0th index and at 2nd index. )

For example, if x = 6 (0110) and y = 15 (1111), then xor2 will be (1001), the two set bits in xor2 indicate that the corresponding bits in x and y are different, at 0th index and at 3rd index both.

**2.**    In the second step, we pick a set bit of xor2. Idea is to use the fact that xor2 is ‘1’ in indexes where bits of x and y are different. So we separate x and y to different groups, along with rest of the numbers of list, based on whether the number has same set-bit or not.

     We choose the rightmost set bit of xor2 as it is easy to get rightmost set bit of a number (bit magic). If we **bitwise AND** a number with its negative counterpart, we get rightmost set bit. (just an observation based property, do remember). So, (xor2) & (-xor2) will give us right set bit. Find (-number) by 2’s complement, that is ((1’s complement) +1 ). It can also be written as (~number)+1.

   a)    Example of 2’s complement :

 7 is 00111(any no. of preceding zeroes). 1’s complement is obtained by flipping bits , 11000. Then add 1, so 2’s complement of 7 is 11001. Since first bit is 1, its a negative no.

 (-1)\*16 + 1\*8 +1\*1  =  -7

   b)    Example of (number) & (-number) = right set bit :

 Continuing example of 7,  7 is 00111 and -7 is 11001 , 7 & -7 is  00001. So, rightmost set bit of 7 is 1.

 Another example with 12 & -12:

 12 is 01100 \*\* and -12 is calculated by flipping digits and adding 1. So, 10011 and adding 1 gives 10100.  12 & -12,  01100 & 10100  gives  00100 as set bit, that is returned as 4 in decimal system, also referred as **Set-bit Number** here.

 \*\* (since number is 32 bit, there are 28 0’s left of ‘left set-bit’, but taking only a few is okay. Positive numbers have leftmost bit 0 and negative have 1 )

**3.**    In third step, we separate x and y in different groups : We now know that for selected set bit index, x and y have different corresponding bits. If we AND all numbers in list with set bit, some will give 0 and others will give 1. We will put all numbers giving zeroes in one group and ones in another. x and y will fall in different groups.

Explained with example:-

E.g. arr = [4, 2, 4, 10, 2, 3, 3, 12] ,

Step 1) XOR of all in arr will cancel all repeating nos. 10 ^12 will be ans. 1010 ^ 1100 will be 0110 that is xor=6.

Step 2) Set bit is 10 from 0110 from visualization. (number) & (-number) is also a quick way to find right set bit.

            xor & (-xor) can be coded directly. 6 is 0110 and finding -6 by flipping digits and adding 1, 1001 +1 = 1010.

            So 6 AND -6 is essentially 0110 & 1010, that is 0010 i.e. 2 – Set-bit Number.

Step 3) AND of all in list with 2 (Set bit no.) will give us numbers that give either 1 or 0, and we make groups.

            [4, 4, 12] and [2, 10, 2, 3, 3], giving 0 and 1 respectively on AND with Set-bit number.

Step 4) XOR of 1st group will give us x=12, x ^ y is known from 1st step i.e. 6. x ^(x ^y) will give us y. 12 ^6 is 10.

           x=12, y=10

    This step works because of the same properties of XOR. All the occurrences of a number will go in same set. XOR of all occurrences of a number which occur even number of times will result in 0 in its set. And the xor of a set will be one of the odd occurring elements.

C++Java

// C++ Program to find the two odd occurring elements

#include <bits/stdc++.h>

using namespace std;

/\* Prints two numbers that occur odd number of times. The

function assumes that the array size is at least 2 and

there are exactly two numbers occurring odd number of times. \*/

void printTwoOdd(int arr[], int size)

{

int xor2 = arr[0]; /\* Will hold XOR of two odd occurring elements \*/

int set\_bit\_no; /\* Will have only single set bit of xor2 \*/

int i;

int n = size - 2;

int x = 0, y = 0;

/\* Get the xor of all elements in arr[]. The xor will basically

be xor of two odd occurring elements \*/

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

xor2 = xor2 ^ arr[i];

/\* Get one set bit in the xor2. We get rightmost set bit

in the following line as it is easy to get \*/

set\_bit\_no = xor2 & ~(xor2-1);

/\* Now divide elements in two sets:

1) The elements having the corresponding bit as 1.

2) The elements having the corresponding bit as 0. \*/

for(i = 0; i < size; i++)

{

/\* XOR of first set is finally going to hold one odd

occurring number x \*/

if(arr[i] & set\_bit\_no)

x = x ^ arr[i];

/\* XOR of second set is finally going to hold the other

odd occurring number y \*/

else

y = y ^ arr[i];

}

cout << "The two ODD elements are " << x << " & " << y;

}

/\* Driver code \*/

int main()

{

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

int arr\_size = sizeof(arr)/sizeof(arr[0]);

printTwoOdd(arr, arr\_size);

return 0;

}

**Output**

The two ODD elements are 5 & 1

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

**Naive Solution:**

C++Java

#include <iostream>

using namespace std;

void oddAppearing(int arr[], int n)

{

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

{

int count = 0;

for(int j = 0; j < n; j++)

{

if(arr[i] == arr[j])

count++;

}

if(count % 2 != 0)

cout<<arr[i]<<" ";

}

}

int main() {

int arr[]= {3, 4, 3, 4, 5, 4, 4, 6, 7, 7}, n = 10;

oddAppearing(arr, n);

}

**Output:**

5 6

**Time Complexity: O(n^2)**  
**Auxiliary Space: O(1)**

Power Set using Bitwise

**Power Set:**Power set **P(S)** of a set **S** is the set of all subsets of **S**. For example S = {a, b, c} then P(s) = {{}, {a}, {b}, {c}, {a,b}, {a, c}, {b, c}, {a, b, c}}.  
If **S** has **n** elements in it then **P(s)** will have **2n** elements

**Example:**

*Set  = [a,b,c]*  
*power\_set\_size = pow(2, 3) = 8*  
*Run for binary counter = 000 to 111*

*Value of Counter            Subset*  
*000                    -> Empty set*  
*001                    -> a*  
*010                    -> b*  
*011                    -> ab*  
*100                    -> c*  
*101                    -> ac*  
*110                    -> bc*  
*111                    -> abc*

**Algorithm:**

Input: Set[], set\_size  
1. Get the size of power set  
      powet\_set\_size = pow(2, set\_size)  
2  Loop for counter from 0 to pow\_set\_size  
    (a) Loop for i = 0 to set\_size  
         (i) If ith bit in counter is set  
                Print ith element from set for this subset  
   (b) Print separator for subsets i.e., newline

**Method 1:**  
For a given set[] S, the power set can be found by generating all binary numbers between **0** and **2n-1**, where **n** is the size of the set.   
For example, for the set **S {*x*, *y*, *z*}**, generate all binary numbers from**0** to **23-1** and for each generated number, the corresponding set can be found by considering set bits in the number.

Below is the implementation of the above approach.

C++Java

// C++ program for the above approach

#include <bits/stdc++.h>

using namespace std;

// Function to print all the power set

void printPowerSet(char\* set, int set\_size)

{

// Set\_size of power set of a set with set\_size

// n is (2^n-1)

unsigned int pow\_set\_size = pow(2, set\_size);

int counter, j;

// Run from counter 000..0 to 111..1

for (counter = 0; counter < pow\_set\_size; counter++) {

for (j = 0; j < set\_size; j++) {

// Check if jth bit in the counter is set

// If set then print jth element from set

if (counter & (1 << j))

cout << set[j];

}

cout << endl;

}

}

/\*Driver code\*/

int main()

{

char set[] = { 'a', 'b', 'c' };

printPowerSet(set, 3);

return 0;

}

**Output**

a

b

ab

c

ac

bc

abc

**Time Complexity:**O(n2n)  
**Auxiliary Space:**O(1)

Maximum AND Value | Explanation

Given an array arr[] of N positive elements. The task is to find the Maximum AND Value generated by any pair of the element from the array.  
  
**Note**: AND is bitwise '&' operator.  
  
**Examples**: 

**Input**: a[] = {4, 8, 12, 16}   
**Output**: 8   
The pairs 8 and 12 gives us the '&' value as 8.   
  
**Input**: a[] = {4, 8, 16, 2}   
**Output**: 0

A **naive** approach is to iterate for all the pairs using two for loops and check for the maximum '&' value of any pair.   
  
**Note**: This approach will not fit in the given time limit since the complexity of the above method is O(N^2). 

C++

int findMaxium(int a[], int n)

{

int maxi = 0;

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

{

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

maxi = max(maxi, a[i] & a[j]);

}

return maxi;

}

**Efficient Approach:** An efficient approach will be to look at this problem bitwise. Since we need to find the maximum '&' value. The first thing that strikes our mind is that the answer should have its MSB as far as possible. So, if two elements are considered as a pair, then their MSB should be set to as much left as possible. Let's take an example to understand this. Consider three elements {10, 8, 2}, so to get a maximum '&' value we need to take those elements whose MSB is as far as possible. In the given example, we can clearly see that 10(1010) and 8(1000) have their 4th-bit from the left set and hence will maximize the answer. Taking 2 and 10 will give our 2nd bit to be set which won't maximize our answer.   
  
So since the constraints permit till 10^4, hence the '&' value will also be less than that. 10^4 will range in 2^0 to 2^14, which means we need to start our checking from the 15th bit. Initially we loop from 15 to 0 and check for the count of numbers whose that particular bit is set. Once we get the count more than 2, the answer will have that bit set, and for the next bit from the left to be set we need to check for both the previous all bits and the current i-th bit. The previous bits can be added to the current bit using a '|' operator. In this way, we get all the positions of the bit which are set, which can be easily represented as a number.   
  
**Note**: We have started checking from bits 31 so that if the constraints were high, it can easily fit in. 

C++

// CPP Program to find maximum AND value of a pair

#include <bits/stdc++.h>

using namespace std;

// Utility function to check number of elements

// having set msb as of pattern

int checkBit(int pattern, int arr[], int n)

{

int count = 0;

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

if ((pattern & arr[i]) == pattern)

count++;

return count;

}

// Function for finding maximum and value pair

int maxAND(int arr[], int n)

{

int res = 0, count;

// iterate over total of 32bits from msb to lsb

for (int bit = 31; bit >= 0; bit--) {

// find the count of element having same pattern as

// obtained by adding bits on every iteration.

count = checkBit(res | (1 << bit), arr, n);

// if count >= 2 set particular bit in result

if (count >= 2)

res = res | (1 << bit); // this is the pattern

we continued

}

return res;

}

// Driver function

int main()

{

int arr[] = { 4, 8, 6, 2 };

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

cout << "Maximum AND Value = " << maxAND(arr, n);

return 0;

}

Maximum AND value of a pair in an array

Given an **array** of N positive elements, the task is to find the maximum AND value generated by any pair of elements from the array.

**Examples:**

**Input:** arr1[] = {16, 12, 8, 4}  
**Output:** 8  
**Explanation:** **8** **AND12** will give us the maximum value**8**

**Input:** arr1[] = {4, 8, 16, 2}  
**Output:** 0

**Maximum AND value of a pair in an array using Nested loop**

The idea is to iterate over all the possible pairs and calculate the AND value of those all. and pick the largest value among them.

Follow the steps mentioned below to implement the approach:

* Declare a variable**res**to store the final answer.
* Create a nested loop and traverse all pairs, storing the maximum value of **AND**of a pair on**res.**
* Return res.

Below is the implementation of the above approach:

C++JavaPython3C#PHPJavascript

// CPP Program to find maximum XOR value of a pair

#include <bits/stdc++.h>

using namespace std;

// Function for finding maximum and value pair

int maxAND(int arr[], int n)

{

int res = 0;

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

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

res = max(res, arr[i] & arr[j]);

return res;

}

// Driver function

int main()

{

int arr[] = { 4, 8, 6, 2 };

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

cout << "Maximum AND Value = " << maxAND(arr, n);

return 0;

}

**Output**

Maximum AND Value = 4

**Time Complexity:**O(N2)  
**Auxiliary Space:**O(1)

**Maximum AND value of a pair in an array using Bit Manipulation**

The**idea** is based on the property of AND operator. AND operation of any two bits results in 1 if both bits are 1. We start from the MSB and check whether we have a minimum of two elements of array having set value. If yes then that MSB will be part of our solution and be added to result otherwise we will discard that bit. Similarly, iterating from MSB to LSB (32 to 1) for bit position we can easily check which bit will be part of our solution and will keep adding all such bits to our solution.

**Illustration:**  
Below is the illustration of example arr[] = { 4, 8, 12, 16}

* **step 1**: Write Bit-representation of each element :   
  4 = 100, 8 = 1000, 12 = 1100, 16 = 10000
* **step 2**: Check for 1st MSB , pattern = 0 + 16 = 16 means we have 10000 as our pattern in binary equivalent. Now 5th bit in 16 is set but no other element has 5-bit as set bit so this will not add up to our RES, still RES = 0 and pattern = 0
* **step 3:**Check 4th bit, pattern = 0 + 8 = 8 means we have 1000 as our pattern in binary equivalent. Now 8 and 12 both have set bit on 4th bit position so that will add up in our solution , RES = 8 and pattern = 8  i.e. RES = 1000 and pattern = 1000.
* **step 4:**Check 3rd bit, pattern = 8 + 4 = 12.  i.e. after adding 3rd bit our pattern becomes 1100 .when we check how many no. in the array have this pattern we find now only 12 satisfies it less two numbers .  so we will discard 3rd bit, RES = 8 (1000)and pattern = 8(1000)
* **step 5:** Check 2nd bit, pattern = 8 + 2 = 10 , after adding 2nd bit our pattern becomes 1010 .when we check how many no. in the array have this pattern we find no element have this pattern inside it ie. No element has set bit same as pattern so we will discard 2nd bit, RES = 8 (1000) and pattern = 8(1000).
* **step 4:** Check 1st bit, pattern = 8 + 1 = 9 ie.  i.e. after adding 3rd bit our pattern becomes 1001. No element has set bit same as pattern so we will discard 1st bit, RES = 8(1000) and pattern = 8(1000).

Follow the steps mentioned below to implement the approach:

* Create a variable **res** to store the final answer.
* Traverse a loop from 31 to 0 to see if there are more than two elements in the array whose AND with res equals res if we add the current bit to res and keep updating res.
* Return **res** as the final answer

Below is the implementation of the above approach:

C++JavaPython3C#PHPJavascript

// CPP Program to find maximum XOR value of a pair

#include <bits/stdc++.h>

using namespace std;

// Utility function to check number of elements

// having set msb as of pattern

int checkBit(int pattern, int arr[], int n)

{

int count = 0;

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

if ((pattern & arr[i]) == pattern)

count++;

return count;

}

// Function for finding maximum and value pair

int maxAND(int arr[], int n)

{

int res = 0, count;

// iterate over total of 32bits from msb to lsb

for (int bit = 31; bit >= 0; bit--) {

// find the count of element having same pattern as

// obtained by adding bits on every iteration.

count = checkBit(res | (1 << bit), arr, n);

// if count >= 2 set particular bit in result

if (count >= 2)

res = res | (1 << bit); // this is the pattern

// we continued

}

return res;

}

// Driver function

int main()

{

int arr[] = { 4, 8, 6, 2 };

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

cout << "Maximum AND Value = " << maxAND(arr, n);

return 0;

}

**Output**

Maximum AND Value = 4

**Time Complexity:**O(N\*log(M))where **M**is the maximum element from the array and**N** is the size of the array.  
**Auxiliary Space:**O(1)