Advance Algorithmic Problem solving

Course Code : R1UC601B

LAB FILE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

GALGOTIAS UNIVERSITY, GREATER NOIDA

UTTAR PRADESH

Student Name:Shivam Pandey
Admission No: 21SCSE1010177
Submitted to : Mr. Gopal Jana
Section – 09
Semester : VI

PROBLEM NO : 01

AIM : Calculate xor from 1 to n .

Program :

```
#include <iostream>

int xor_1_to_n(int n) {
    if (n % 4 == 0)
        return n;
    else if (n % 4 == 1)
        return 1;
    else if (n % 4 == 2)
        return n + 1;
    else
        return 0;
}

int main() {
    int n;
    std::cout << "Enter a number (n): ";
    std::cin >> n;

    int result = xor_1_to_n(n);
    std::cout << "XOR of all numbers from 1 to " << n
<< " is " << result << std::endl;

    return 0;
}
```

OUTPUT : Enter a number (n): 10
XOR of all numbers from 1 to 10 is 11

PROBLEM NO : 02

AIM: Enter a number (n): 10
XOR of all numbers from 1 to 10 is 11

```
#include <iostream>

// Function to calculate XOR of all numbers from 1 to n
int calculateXOR(int n) {
    // Using the properties of XOR
    switch (n & 3) {
        case 0: return n;     // If n is multiple of 4
        case 1: return 1;     // If n % 4 gives remainder 1
        case 2: return n + 1; // If n % 4 gives remainder 2
        case 3: return 0;     // If n % 4 gives remainder 3
    }
    return 0; // to suppress warning; this line is actually
unreachable
}

int main() {
    int n;
    std::cout << "Enter a number (n): ";
    std::cin >> n;

    int result = calculateXOR(n);
    std::cout << "XOR of all numbers from 1 to " << n << " is " <<
result << std::endl;

    return 0;
}
```

OUTPUT: Enter a number (n): 10
XOR of all numbers from 1 to 10 is 11

PROBLEM NO : 03

AIM: · How to know if a number is a power of 2?

```
#include <iostream>

bool is_power_of_2(int n) {
   return n != 0 && (n & (n - 1)) == 0;
}

int main() {
   int numbers[] = {1, 2, 4, 8, 16, 32, 64, 128, 256,
512, 1024, 2048};
   int size = sizeof(numbers) / sizeof(numbers[0]);

   for (int i = 0; i < size; ++i) {
      std::cout << numbers[i] << " is a power of 2: "
<< std::boolalpha << is_power_of_2(numbers[i]) <<
std::endl;
   }

   return 0;
}
```

OUTPUT : 1 is a power of 2: true
2 is a power of 2: true

```
4 is a power of 2: true
8 is a power of 2: true
16 is a power of 2: true
32 is a power of 2: true
64 is a power of 2: true
128 is a power of 2: true
256 is a power of 2: true
512 is a power of 2: true
1024 is a power of 2: true
2048 is a power of 2: true
```

PROBLEM NO : 04

AIM: Find xor of all subsets of a set

```
#include <iostream>
#include <vector>

int xor_of_all_subsets(std::vector<int>& nums) {
    int result = 0;
    int n = nums.size();

    for (int i = 0; i < (1 << n); ++i) {
        int xor_subset = 0;
        for (int j = 0; j < n; ++j) {
            if (i & (1 << j)) {
                xor_subset ^= nums[j];
            }
        }
        result ^= xor_subset;
    }
    return result;
```

```
}

int main() {
    std::vector<int> nums = {1, 2, 3};
    int xor_result = xor_of_all_subsets(nums);
    std::cout << "XOR of all subsets: " << xor_result
<< std::endl;
    return 0;
}
```

OUTPUT : XOR of all subsets: 0

PROBLEM NO : 05

AIM:find the number of leading trailling zeores and number of 1

```
#include <iostream>

// Function to count the number of leading zeros
int count_leading_zeros(unsigned int n) {
    int count = 0;
    while ((n & (1 << 31)) == 0) {
        ++count;
        n <<= 1;
    }
    return count;
}

// Function to count the number of trailing zeros
```

```
int count_trailing_zeros(unsigned int n) {
    int count = 0;
    while ((n & 1) == 0) {
        ++count;
        n >>= 1;
    }
    return count;
}

// Function to count the number of 1s
int count_ones(unsigned int n) {
    int count = 0;
    while (n != 0) {
        count += n & 1;
        n >>= 1;
    }
    return count;
}

int main() {
    unsigned int num = 48; // Example number
    std::cout << "Number of leading zeros: " <<
count_leading_zeros(num) << std::endl;
    std::cout << "Number of trailing zeros: " <<
count_trailing_zeros(num) << std::endl;
    std::cout << "Number of ones: " << count_ones(num)
<< std::endl;
    return 0;
}
```

OUTPUT : Number of leading zeros: 26
Number of trailing zeros: 4
Number of ones: 4

PROBLEM NO : 06

AIM: convert binary directly into c ++

```
#include <iostream>

int main() {
    // Example binary number: 1010 (decimal 10)
    int binary_number = 0b1010;

    std::cout << "Binary number in decimal: " <<
binary_number << std::endl;
    return 0;
}
```

OUTPUT : Binary number in decimal: 10

PROBLEM NO : 07

AIM: The quickest way to swap two numbers

```
#include <iostream>

void swap(int& a, int& b) {
   if (&a != &b) { // Ensure a and b are distinct
variables
       a ^= b;
       b ^= a;
       a ^= b;
   }
}

int main() {
   int x = 5, y = 10;

   std::cout << "Before swapping: x = " << x << ", y
= " << y << std::endl;

   swap(x, y);

   std::cout << "After swapping: x = " << x << ", y =
" << y << std::endl;

   return 0;
}
```

OUTPUT : Before swapping: x = 5, y = 10

After swapping: x = 10, y = 5

PROBLEM NO : 08

AIM: simple approach to flip the bits of a number

```
#include <iostream>

int main() {
   unsigned int num = 10; // Example number
   unsigned int flipped_num = ~num;

   std::cout << "Original number: " << num <<
std::endl;
   std::cout << "Flipped number: " << flipped_num
<< std::endl;

   return 0;
}
```

OUTPUT : Original number: 10
Flipped number: 4294967285

PROBLEM NO : 09

AIM: finding the most significant set bit(MSB)

```
#include <iostream>
```

```
int find_msb(int n) {
    if (n == 0)
        return -1; // No set bit found
    int msb = 0;
    while (n != 0) {
        n = n >> 1;
        msb++;
    }
    return msb - 1; // Adjust for zero-based index
}

int main() {
    int num = 48; // Example number
    int msb_position = find_msb(num);

    std::cout << "Most significant set bit position:
" << msb_position << std::endl;

    return 0;
}
```

OUTPUT : Most significant set bit position: 5

PROBLEM NO : 10

AIM:check if a number has bits in and alternative pattern

```
#include <iostream>

bool has_alternating_bits(int n) {
    int prev_bit = n & 1;
    n >>= 1;
    while (n != 0) {
        int curr_bit = n & 1;
        if (curr_bit == prev_bit) {
            return false; // Current bit is same as previous bit, not alternating
        }
        prev_bit = curr_bit;
        n >>= 1;
    }
    return true;
}

int main() {
    int num = 10; // Example number
```

```
    bool is_alternating =
has_alternating_bits(num);

    if (is_alternating) {
        std::cout << num << " has bits in an
alternating pattern." << std::endl;
    } else {
        std::cout << num << " does not have bits in
an alternating pattern." << std::endl;
    }

    return 0;
}
```

OUTPUT:

10 does not have bits in an alternating pattern.
5 has bits in an alternating pattern.