Assignment 1

NAME: RITU RAJ ANAND

REG NO: 1041012910

CSE R 2ND YEAR

1. Write a java program to count the number of bits that are set 1 in an integer. Also prove that time complexity is O(n) where n is the number of bits.

Ans:

**import** java.util.Scanner;

**public** **class** NumberOfSetBits {

**public** **static** **void** main(String[] args)

{

**int** n;

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enter the number:");

n=input.nextInt();

**int** x=n;

**int** count=0;

**while**(n>0)

{

**if**((n&1)==1)

count++;

n>>=1;

}

System.***out***.println("The number of set bits in "+Integer.*toBinaryString*(x)+" is "+count);

}

}

1. Write a program to find the parity bit of a number in O(n) time, where nis the word size.

Ans:

**import** java.util.Scanner;

**public** **class** ParityGenerator {

**public** **static** **void** main(String[] args)

{

**int** n;

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enter the number:");

n=input.nextInt();

**int** x=n;

**int** count=0;

**while**(n>0)

{

**if**((n&1)==1)

count++;

n>>=1;

}

System.***out***.println("Parity bit is:"+(count&1));

}

}

1. Write a program to find the parity bit of a number in O(k) time, where kis the number of set bits.

Ans:

**import** java.util.Scanner;

**public** **class** ParityGenerator {

**public** **static** **void** main(String[] args)

{

**int** n;

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enter the number:");

n=input.nextInt();

**int** x=n;

**int** count=0;

**while**(n>0)

{

count^=1;

n&=(n-1);

}

System.***out***.println("Parity bit is:"+count);

}

}

1. Write a program to find the parity bit of a number in O(k) time, where kis the number of set bits.

Ans:

**import** java.util.Scanner;

**public** **class** ParityGenerator {

**public** **static** **void** main(String[] args)

{

**int** n;

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enter the number:");

n=input.nextInt();

**int** x=n;

**int** count=0;

**while**(n>0)

{

count^=1;

n&=(n-1);

}

System.***out***.println("Parity bit is:"+count);

}

}

1. Define a function to create a lookup table of size 216 whose value is the parity bits of the index.

Ans:

**static** **void** lookupcreator(**int** parity[])

{

**for**(**int** i=0;i<65536;i++)

{

**int** x=i;

x^=x>>8;

x^=x>>4;

x^=x>>2;

x^=x>>1;

parity[i]=(x&1);

}

}

1. Write a program to calculate the parity bit of a 64 bit word using look up table in O(n/L) time, where n is the word size and L is the group size. Note: Consider group size is 16 bit for the problem.

Ans:

**import** java.util.Scanner;

**public** **class** ParityBitLookUpTable

{

**static** **void** lookupcreator(**int** parity[])

{

**for**(**int** i=0;i<65536;i++)

{

**int** x=i;

x^=x>>8;

x^=x>>4;

x^=x>>2;

x^=x>>1;

parity[i]=(x&1);

}

}

**public** **static** **void** main(String[] args)

{

Scanner input=**new** Scanner(System.***in***);

**int** parity[]=**new** **int**[65536];

*lookupcreator*(parity);

System.***out***.println("Enter the binary word whose parity is to be calculated:");

**long** x=input.nextLong();

**int** m=0xFFFF;

**int** prt=parity[(**int**)(x>>48)&m]^parity[(**int**)(x>>32)&m]^parity[(**int**)(x>>16)&m]^parity[(**int**)x&m];

System.***out***.println("Parity of the entered number is:"+prt);

}

}

1. Write a program to calculate parity bit of a 64 bit word using only xor and right shift operator.

Ans:

**import** java.util.Scanner;

**public** **class** ParitUsingOnlyXorAndRightOperator {

**public** **static** **void** main(String[] args)

{

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enter the number whose parity is to be calculated:");

**long** x=input.nextLong();

x^=x>>32;

x^=x>>16;

x^=x>>8;

x^=x>>4;

x^=x>>2;

x^=x>>1;

System.***out***.println("Parity is:"+(x&1));

}

}

1. Write a program to swap the i th bit with j th bit of a number.

Ans:

**import** java.util.Scanner;

**public** **class** SwappingBits {

**public** **static** **void** main(String[] args)

{

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enrter the number:");

**int** n=input.nextInt();

System.***out***.println(Integer.*toBinaryString*(n));

System.***out***.println("Enter the positions to swap:");

**int** i=input.nextInt();

**int** j=input.nextInt();

**if**((1&(n>>i))!=(1&(n>>j)))

{

**int** bitMask=(1<<i)|(1<<j);

n^=bitMask;

}

System.***out***.println(Integer.*toBinaryString*(n));

}

}

1. Design a function to create a lookup table A such that for every 16 bit number y, A[y] holds the bit-reversal of y.

Ans:

**static** **void** reversallookup(**int** lookup[])

{

**for**(**int** i=0;i<65536;i++)

{

**int** n=i;

**int** r=0;

**while**(n>0)

{

r<<=1;

**if**((n&1)==1)

r^=1;

n>>=1;

}

lookup[i]=r;

}

}

1. Write a program to find the bit reversal of a number using the lookup table created in Q9.

Ans:

**import** java.util.Scanner;

**public** **class** BitReversalUsingLookUpTable

{

**static** **void** reversallookup(**int** lookup[])

{

**for**(**int** i=0;i<65536;i++)

{

**int** n=i;

**int** r=0;

**while**(n>0)

{

r<<=1;

**if**((n&1)==1)

r^=1;

n>>=1;

}

lookup[i]=r;

}

}

**public** **static** **void** main(String[] args)

{

Scanner input=**new** Scanner(System.***in***);

**int** lookup[]=**new** **int**[65536];

*reversallookup*(lookup);

System.***out***.println("Enter the number whose bit reversal is to be calculated:");

**int** n=input.nextInt();

**if**(n>=65536)

System.***out***.println("The entered number is outside the range of the lookup table");

**else**

System.***out***.println(lookup[n]);

}

}

1. Write a program to find the closest integer with the same weight.

Ans:

**import** java.util.Scanner;

**public** **class** ClosestWeight

{

**public** **static** **void** main(String[] args)

{

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enter the number:");

**int** n=input.nextInt();

System.***out***.println(Integer.*toBinaryString*(n));

**if**((n&1)==0)

{

**int** m=n&(n-1);

**int** x=m^n;

x>>=1;

n&=n-1;

n|=x;

}

**else**

{

**int** m=(~n)&(~((~n)-1));

n^=1;

n|=m;

}

System.***out***.println(Integer.*toBinaryString*(n));

System.***out***.println(n);

}

}

1. Write a program to compute XXY using bit wise operator.

Ans:

**import** java.util.Scanner;

**public** **class** MultiplyTwoNumbers

{

**public** **static** **void** main(String[] args)

{

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enter two number:");

**int** a=input.nextInt();

**int** b=input.nextInt();

**int** res=0;

**while**(b!=1)

{

**if**((b&1)==1)

res+=a;

a<<=1; b>>=1;

}

res+=a;

System.***out***.println("Their product is:"+res);

}

}

1. Write a program to compute X/Y using bit wise operator.

Ans:

**import** java.util.Scanner;

**public** **class** DivideTwoNumbers {

**public** **static** **void** main(String[] args)

{

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enter the numbers:");

**int** a=input.nextInt();

**int** b=input.nextInt();

**int** t=0, q=0;

**for**(**int** i=0;a>=b;i++)

{

**if**((b<<i)>a)

{

q|=(1<<(i-1));

a-=(b<<(i-1));

i=-1;

}

}

System.***out***.println(q);

}

}

1. . Write a program to compute XY using bit wise operator.

Ans:

**import** java.util.Scanner;

**public** **class** Power {

**public** **static** **void** main(String[] args)

{

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enter the numbers:");

**int** x=input.nextInt();

**int** y=input.nextInt();

**int** res=1;

**while**(y!=0)

{

**if**((y&1)==1)

res\*=x;

x\*=x;

y>>=1;

}

System.***out***.println(res);

}

}

1. Write a program to check if a decimal number is a palindrome.

Ans:

**import** java.util.Scanner;

**public** **class** PalindromeNumber {

**public** **static** **void** main(String[] args)

{

Scanner input=**new** Scanner(System.***in***);

System.***out***.println("Enter the number:");

**int** n=input.nextInt();

**int** nod=(**int**) Math.*log10*(n)+1;

System.***out***.println(nod);

**int** msd=(**int** ) Math.*pow*(10, nod-1);

System.***out***.println(msd);

**while**(n!=0)

{

System.***out***.println((n/msd)+" "+(n%10));

**if**((n/msd)!=(n%10))

{

System.***out***.println("Entered Numer is not palindrome.");

System.*exit*(-1);

}

n%=msd;

n/=10;

msd/=100;

System.***out***.println(n);

}

System.***out***.println("Entered number is plaindrome.");

}

}

1. Write a program which test if two rectangle have a nonempty intersection. If the intersection is nonempty, return the rectangle formed by their intersection.

Ans:

**class** Rectangle{

**int** x,y,H,W;

Rectangle(**int** x,**int** y,**int** W,**int** H){

**this**.x = x;

**this**.y = y;

**this**.H = H;

**this**.W = W;

}

}

**class** RectangleIntersection {

**public** **static** **void** main(String[] args) {

Rectangle r1 = **new** Rectangle(1,1,2,2);

Rectangle r2 = **new** Rectangle(1,4,2,2);

System.***out***.println(*area*(r1,r2));

}

// public static boolean isIntersecting(Rectangle r1,Rectangle r2){

// return r1.x <= r2.x + r2.W && r1.x + r1.W >= r2.x && r1.y <= r2.y + r2.H && r1.y + r1.H >= r2.y;

// }

**public** **static** **double** area(Rectangle r1,Rectangle r2){

**int** x\_intersection = Math.*abs*(Math.*max*(r1.x,r2.x) - Math.*min*(r1.x+r1.W,r2.x+r2.W));

**int** y\_intersection = Math.*abs*(Math.*max*(r1.y,r2.y) - Math.*min*(r1.y+r1.H,r2.y+r2.H));

**if**(x\_intersection == 0 || y\_intersection == 0)

**return** 0; // not intersecting

**return** x\_intersection \* y\_intersection;

}

}