Recall that a function g(·) is monotonically increasing if g(x) <= g(y) for all values of x and y such that x <= y. For example, g(x) = 2x + 3 is a monotonically increasing function. Now given a monotonically increasing function f(x) defined on x > 0, we aim at finding the minimum integer value of x for which f(x) >= 0. Of course considering all possible values of x starting from 0 and checking until the value of x for which f(x) gets positive, is a solution yielding a complexity of O(x). Your task is to solve this problem in logarithmic time, i.e. in log(x) time. Hint: Think of applying binary search to solve the problem. However this is a case where the search space is not bounded since the upper bound is not known in advance. You can think of finding an initial upper bound by using exponential, and then use binary search to find the exact value. Note that exponential search complexity will be logarithmic. Example: (Input) f(x) = 3x - 100 (you can hard code the function) (Output) The value of x = 34 yields f(x) to be positive

SOLUTION CODE

#include<stdio.h>

#include<limits.h>

# define int\_max 1e9

int f(int x);

int getup(int i);

//defining the function f(x)

int f(int x)

{

return(3\*x-100);

}

//reducing the search space exponentially by binary search

int getup(int i)

{

for(int i=1;i<10000;i\*=2)

{

if(f(i)>=0) return i;

}

}

//finding the lowest poosible value f(x)for which f(x)>=0

int find (int low,int high)

{

if(low==high) return low;

int mid=(low+high)/2;

if(low<high)

{

if(f(mid)>=0&&f(mid-1)>=0) return(find(low,mid));

else if(f(mid)>=0&&f(mid-1)<0) return mid;

else if(f(mid)<0&&f(mid+1)<0) return(find(mid,high));

else if(f(mid)<0&&f(mid+1)>0) return(mid+1);

}

}

int main()

{

int p,q;

q=getup(1);

p=find(0,q);

printf("f(x) = 3x - 100\n");

printf("The value of x = %d yields f(x) to be positive",p);

return 0;

}