Assignment 1

Piyush Kumar Uttam - EE18BTECH11036

Download all latex-tikz codes from

```
https://github.com/piyushSTK/C−DS/blob/main/
    Assignment1/assignment1.tex
```

1 Problem

(Q 26) Consider the following C function.

```
void convert(int n){
    if(n<0) printf("%d", n);
    else{
        convert(n/2);
        printf("%d", n%2);
    }
}
```

Which of the following will happen when the function **convert** is called with any positive integer n as argument?

1) It will print the binary representation of **n** and terminate.
2) It will print the binary representation of **n** in the reverse order and terminate.
3) It will print the binary representation of **n** and will not terminate.
4) It will not print anything and will not terminate.

2 Solution

Answer : D) It will not print anything and will not terminate.

Explanation

This is a problem involving recursion, as $n > 0$ the portion

```
    else{
        convert(n/2);
        printf("%d", n%2);
    }
```

will be evaluated. In this portion, convert is called again with value of **n/2** and subsequently **n** goes from $n- > 0$.

However as **n** = 0; the condition

```
    if(n<0) printf("%d", n);
```

is still not satisfied and $0/2 = 0$. Hence it will not terminate and will not print anything as print statement is after **convert(n/2)** call.

3 Problem

Obtain a new code to get option 1.

Solution

The first option can be found using the following piece of code.

```
void convert(int n){
    if(n<=0) return;
    else{
        convert(n/2);
        printf("%d", n%2);
    }
}
```

The flowchart for the given problem is given in the Figures section.

4 Problem

Obtain a new code using only for loops.

Solution

Getting binary representation using only for loop can be found in following piece of code. The given code uses only 2 int variables to remove the need for additional stack memory.

```
void convert(int n){
    if(n == 0){
        printf("%d", 0);
    }
    int p = floor(log2(n));
    int div = pow(2,p);
    for(int i = p; i>=0; i--){
        if(n - div>=0){
            printf("%d", 1);
            n = n-div;
            div = div/2;
        }
        else{
            printf("%d", 0);
            div = div/2;
        }
    }
    return;
}
```

5 Problem

Explain which method is better out of recursion and iteration.

Solution

Using iterative method is better than using recursion for the following reasons:

1) Recursion utilizes call stack where all the local variables and return addresses are stored in a stack frame above the previous recursive call's stack frame.
2) It generally uses more memory than optimised iterative program.
3) Stack overflow is a common error which recursive functions encounter. This happens when the number of recursive calls exceed the allocated stack memory limit.
4) So since time complexity is same for both solutions, hence iterative solution with lesser space usage is better approach.

The visualisation of recursive solution stack memory diagram and iterative solution stack memory diagram is given in the Figures section.

6 Problem

Derive the time complexity of the original solution.

Solution

Let's assume the time taken to process the given input N to be $T(n)$.
So time taken to process N/2 will be $T(n/2)$
Lets assume print() function takes some constant C time to process.
Therefore from the program we can see that:

$$T(n) = T(n/2) + C \tag{6.0.1}$$

and

$$T(n/2) = T(n/4) + C \tag{6.0.2}$$

Substituting equation (6.0.2) in (6.0.1) we get

$$T(n) = T(n/4) + C + C \tag{6.0.3}$$

On repeated substitution till n $- >$ 0 we get:

$$T(n) = T(0) + C + C + C + C.....log_2(n)times \tag{6.0.4}$$

$$\implies T(n) = C * log_2(n) + T(0) \tag{6.0.5}$$

$$\implies T(n) \leqslant C * log_2(n) \tag{6.0.6}$$

$$\implies T(n) \in O(log(n)) \tag{6.0.7}$$

Hence the program has logarithmic time complexity.

7 Verification

The running time of the program was recorded to confirm the time complexity and time vs n was plotted using the collected data.
Code for verification of time complexity can be found at

https://github.com/piyushSTK/C-DS/blob/main/Assignment1/codes/verify.py

The graph of T(n) vs n can be found in the figures section.

8 Figures

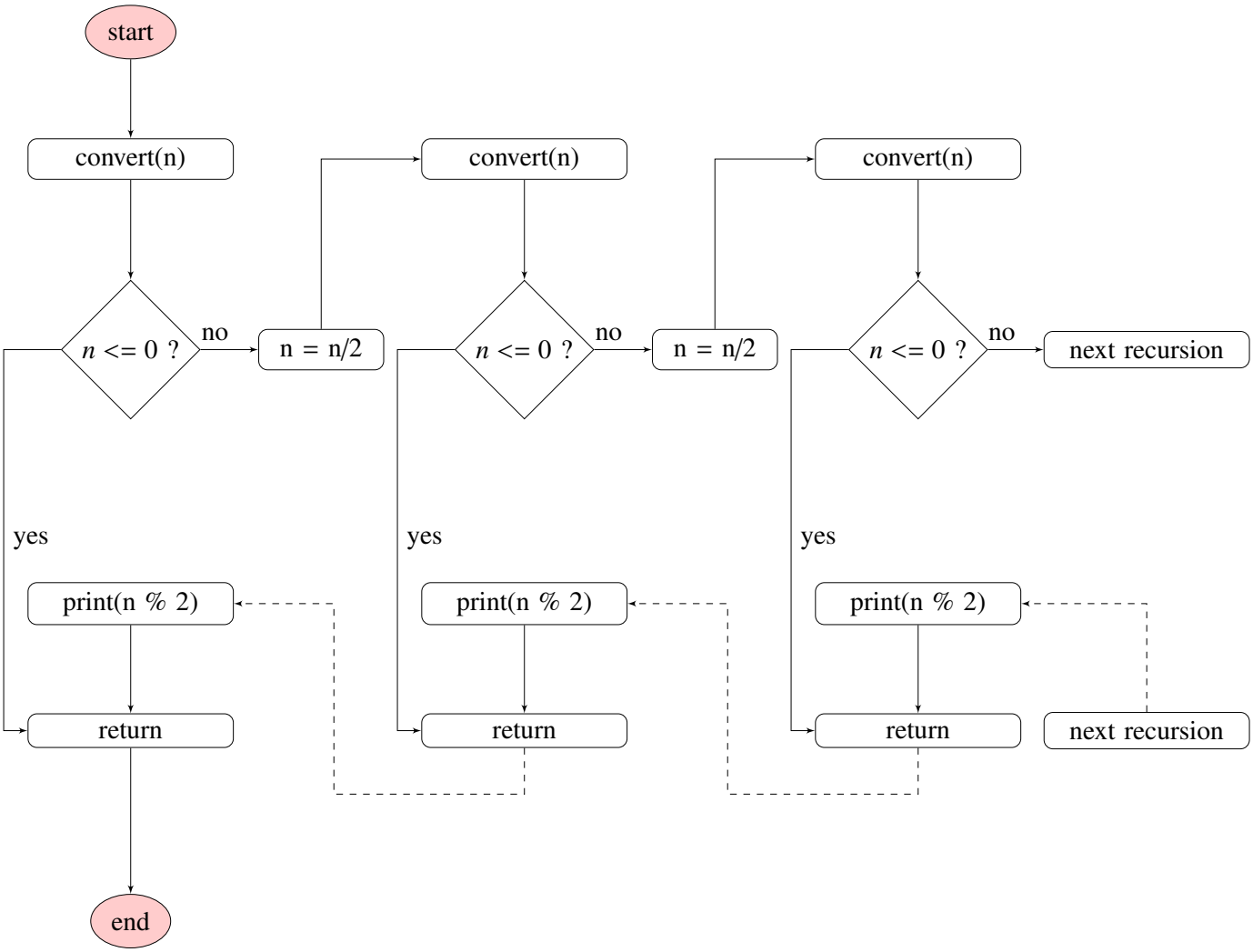

Fig. 4: Flowchart of recursive calls of convert(n)

main()
Local Variables int n int p int div

Fig. 4: Stack memory diagram for iterative solution

...
...
...
...

convert(n)
Local Variables int n

convert(n)
Local Variables int n

convert(n)
Local Variables int n

convert(n)
Local Variables int n

main()
Local Variables int n

Fig. 4: Stack Memory diagram for recursive solution

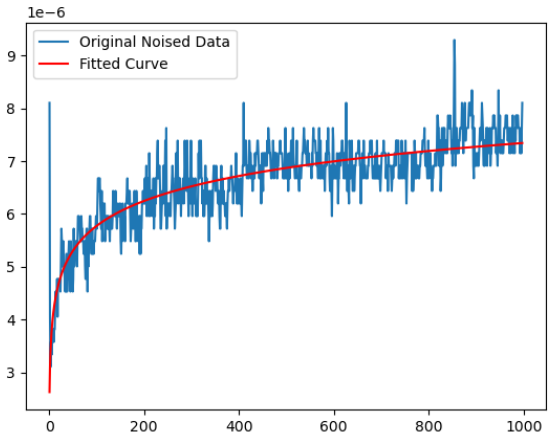

Fig. 4: T(n) vs n