

1. Solve the following recurrence relations a) $x(n)=x(n-1) + 5$ for $n > 1$ $x(1)=0$ b) $x(n)=3x(n-1)$ for $n > 1$ $x(1)=4$ c) $x(n)=x(n/2)+n$ for $n > 1$ $x(1)=1$ (solve for $n=2k$) d) $x(n)=x(n/3)+1$ for $n > 1$ $x(1)=1$ (solve for $n=3k$)

Program:

```
def recurrence_a(n):
    if n == 1:
        return 0
    return recurrence_a(n-1) + 5
print("a)  $x(n) = x(n-1) + 5$ ,  $x(1) = 0$ ")
for i in range(1, 6):
    print(f"x({i}) = {recurrence_a(i)}")
def recurrence_b(n):
    if n == 1:
        return 4
    return 3 * recurrence_b(n-1)
print("\nb)  $x(n) = 3*x(n-1)$ ,  $x(1) = 4$ ")
for i in range(1, 6):
    print(f"x({i}) = {recurrence_b(i)}")
def recurrence_c(n):
    if n == 1:
        return 1
    return recurrence_c(n//2) + n
print("\nc)  $x(n) = x(n/2) + n$ ,  $x(1) = 1$  (solve for  $n=2k$ )")
for i in range(1, 11):
    if i % 2 == 0:
        print(f"x({i}) = {recurrence_c(i)}")
def recurrence_d(n):
    if n == 1:
        return 1
    return recurrence_d(n//3) + 1
print("\nd)  $x(n) = x(n/3) + 1$ ,  $x(1) = 1$  (solve for  $n=3k$ )")
for i in range(1, 16):
    if i % 3 == 0:
        print(f"x({i}) = {recurrence_d(i)}")
```

output:

```
a)  $x(n) = x(n-1) + 5, x(1) = 0$   
 $x(1) = 0$   
 $x(2) = 5$   
 $x(3) = 10$   
 $x(4) = 15$   
 $x(5) = 20$   
  
b)  $x(n) = 3*x(n-1), x(1) = 4$   
 $x(1) = 4$   
 $x(2) = 12$   
 $x(3) = 36$   
 $x(4) = 108$   
 $x(5) = 324$   
  
c)  $x(n) = x(n/2) + n, x(1) = 1$  (solve for  $n=2^k$ )  
 $x(2) = 3$   
 $x(4) = 7$   
 $x(6) = 10$   
 $x(8) = 15$   
 $x(10) = 18$   
  
d)  $x(n) = x(n/3) + 1, x(1) = 1$  (solve for  $n=3^k$ )  
 $x(3) = 2$ 
```

Result: program has been successfully executed.

2. Evaluate the following recurrences completely i) $T(n) = T(n/2) + 1$, where $n=2^k$ for all $k \geq 0$ ii) $T(n) = T(n/3) + T(2n/3) + cn$, where 'c' is a constant and 'n' is the input size

Program:

def recurrence_i(n):

```

    if n == 1:
        return 0
    return recurrence_i(n // 2) + 1

# Recurrence ii
def recurrence_ii(n, c):
    if n == 0:
        return 0
    return recurrence_ii(n // 3, c) + recurrence_ii(2 * n // 3, c) + c

# Test the recurrences
n = 8
c = 2

print("Recurrence i for n =", n, ":", recurrence_i(n))
print("Recurrence ii for n =", n, "and c =", c, ":", recurrence_ii(n, c))
output:

```

```

Recurrence i for n = 8 : 3
Recurrence ii for n = 8 and c = 2 : 18

=== Code Execution Successful ===

```

Result: program successfully executed.

3. Consider the following recursion algorithm $\text{Min1}(A[0 \text{ ----- } n-1])$ If $n=1$ return $A[0]$ Else $\text{temp} = \text{Min1}(A[0 \text{ } n-2])$ If $\text{temp} \leq A[n-1]$ return temp Else Return $A[n-1]$ a) What does this algorithm compute? b) Setup a recurrence relation for the algorithms basic operation count and solve it

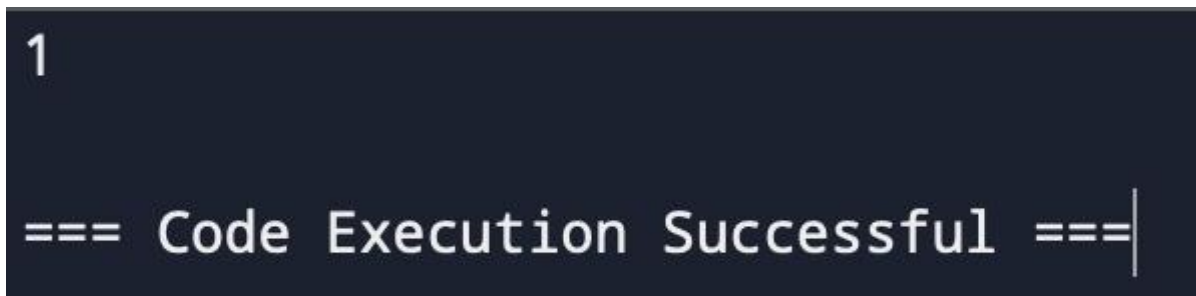
Program:

```
def Min1(A):
```

```

n = len(A)
if n == 1:
    return A[0]
else:
    temp = Min1(A[:n-1])
    if temp <= A[n-1]:
        return temp
    else:
        return A[n-1]
# Test the algorithm
A = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
result = Min1(A)
print(result)
output:

```



Result: program successfully executed.

4. Analyze the order of growth. (i). $F(n) = 2n^2 + 5$ and $g(n) = 7n$. Use the $\Omega(g(n))$ notation.

Program:

```

def F(n):
    return 2 * n**2 + 5

```

```
def g(n):
```

```
    return 7 * n
```

```
# Checking if F(n) is  $\Omega(g(n))$ 
```

```
n = 10
```

```
if F(n) >= 7 * g(n):
```

```
    print("F(n) is  $\Omega(g(n))$  for n =", n)
```

```
else:
```

```
    print("F(n) is not  $\Omega(g(n))$  for n =", n)
```

```
output:
```

```
F(n) is not  $\Omega(g(n))$  for n = 10
```

```
=== Code Execution Successful ===
```

Result: program successfully executed