Subject: 279: Design & Analysis: Algorithms (28009)
Assignment: 1

Name: Mansi Jainendra Tandel
SCU id: W1606463

1)

Mathematical induction:

For any natural number, $n^3 - n$ is divisible by 3.

Step 1: Proves that the statement is true for the initial value.

So, here for $n=1$, we have to check for $n^3 - n$. By putting $n=1$,

$n^3 - n = 1^3 - 1 = 0$, which is divisibly by 3.

Step 2: Assume that the statement is true for any value of $n=k$.

∴ Assume that $k^3 - k$ is divisible by 3.

Then prove that the statement is true for $n=k+1$.

∴ for $(k+1)^3 - (k+1)$, we have to prove that $(k+1)^3 - (k+1)$ is divisible by 3.

$$(k+1)^3 - (k+1) = (k^3 + 3k^2 + 3k + 1) - (k+1)$$
$$= k^3 + 3k^2 + 3k + 1 - k - 1$$
$$= (k^3 - k) + (3k^2 + 3k)$$

Let $k^3 - k = P$.

here, $(k+1)^3 - (k+1) =$

$k^3 - k$ is divisibly by 3, which means that $k^3 - k = 3P$ where P is an integer.

∴ For k, $(k+1)^3 - (k+1) = 3P + 3k^2 + 3k$
$= 3(P + k^2 + k)$

Here, $3(p+k^2+k)$ is divisible by 3.

Hence, it is proved that for any natural number n, n^3-n is divisible by 3.

2] (a) Write the algorithm for factorial using loop.

```
factorial = 1
for (i=1 ; i≤n ; i++)
{
    factorial = factorial * i
}
```

(b) Loop invariant:

→ A loop invariant is a statement about program variables that is true before and after each iteration of a loop.

→ Loop invariants are useful to make us understand why the algorithm is correct.

→ Three properties of the loop invariant are following.

(i) Initialization: The loop invariant must be true before the first iteration of the loop.

(ii) Maintenance: If it is true before an iteration of the loop, it remains true Before the next iteration.

(iii) Termination: When the loop terminates, the invariant gives us a useful property that help show the algorithm is correct.

→ Loop Invariants are useful to prove the properties of loops.

→ For the factorial algorithm given in (a), the Loop invariants are i=1 and factorial = 1.

→ Here, the initialization property states that the factorial = 1 and i=1. Here, i is initially I At the beginning.

→ The maintenance property checks for every number which is $i \leq n$. and performs the condition of factorial = factorial * i. Here, for every $i \leq n$, we get the factorial number.

→ The termination property is satisfied by the algorithm when, the for loop is giving the correct number for the calculation of factorial.

→ For example, if we want to find the factorial of 5. then,

we take n = 5.
factorial = 1 and i is initalized to 1 at the beginning of the loop.

→ For the 1st iteration,
i=1; i<=5; i++
gives factorial = 1 * 1 = 1

→ For i=2;
we have factorial = 1 * 2 = 2

→ For i=3;
we have factorial = 2 * 3 = 6

→ For i=4;
we have factorial = 6 * 4 = 24

→ For i=5;
we have factorial = 24 * 5 = 120.

→ As per above, iterations, the loop invariants hold all the three properties.
→ So, we can say that the algorithm is correct.
→ Here, loop invariants are tows before and after each iteration of a loop.

(c) Running time

```
factorial = 1
for (i = 1; i ≤ n; i++) {
    factorial = factorial * i
}
```

Running time = $C_1 + C_2(n+1) + C_3(n)$

$= C_1 + C_2(n+1) + C_3(n)$

$\therefore$ C_1 is the constant for factorial = 1 calculation. $C_2(n+1)$ is for the for loop. $C_3(n)$ is for the operation of factorial = factorial * i.

$\therefore$ The time complexity will be in $O(n)$.

(d) Recursive algorithm

I/P : An integer numbe n, wher $n \geq 0$.

O/P : factorial of n.

```
factorial(n) {
    if (n == 0)
        return 1
    else
        return n * factorial(n-1)
}
```

3) (b) Calculate the running time of your algorithm as a function of your input size.

→ Assume that the input size is n.

```
int maxAscendingSum(vector<int>& nums){                    n
    int n = nums.size();
    int dp[n]; ← c1
    for(int i=0; i<n; i++){ ← n+1
        dp[i] = nums[i] ← n
    }
    for(int i=1; i<n; i++){ ← n+1
        if(nums[i-1]<nums[i]){ ← n
            dp[i]=max(dp[i], dp[i-1]+nums[i]); ← n
        }
    }
    int ans=0; ← c2
    for(int i=0; i<n; i++){ ← n+1
        ans=max(ans, dp[i]); ← n
    }
    return ans; ← c3
}
```

∴ The running time, will be $c_1 + 8(n) + 8(n+1)$

$+ 8(n+1) + n + n + n + c_3 +$

$c_2 + n + 1 + n = c_1 + c_2 + c_3 + 3 + 8n$

∴ The running time will be in $O(n)$.

Success Details >

Runtime: 4 ms, faster than 54.28% of C++ online submissions for Maximum Ascending Subarray Sum.

Memory Usage: 8.5 MB, less than 32.84% of C++ online submissions for Maximum Ascending Subarray Sum.

Next challenges:

3Sum Closest | Longest Repeating Character Replacement | Fruit Into Baskets

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
04/10/2021 00:29	Accepted	4 ms	8.5 MB	cpp