**Answer to the Question No. - 1**

**(a)** For the given while loop, the loop invariant is –

At the beginning of the while loop, k = 2c.

**(b)** Before the first iteration of the while loop, c is assigned to 0 and k is assigned to 1.

So, the base case is, k1 = 20 = 1, which is equal to k.

Now assuming the loop invariant is true for first p iterations, so kp = 2p.

We multiply k with 2 during the iteration. It can be written that,

K2 = 21 \* 2

K3 = 22 \* 2

K4 = 23 \* 2 and so on.

Here, k = 2p+1 and it is the same as the beginning of p+1-th iterations.

At the end: c = n, so k = 2n.

**(c)** The runtime of the algorithm:

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Cost** | **Times** |
| **function** pow(*n*) |  |  |
| *k* = 1 | C1 | 1 |
| *c* = 0 | C2 | 1 |
| **while** *c < n* **do** | C3 | n+1 |
| *k* = *k ·* 2 | C4 | n |
| *c* = *c* + 1 | C5 | n |
| **end while** |  |  |
| **return** *k* | C6 | 1 |
| **end function** |  |  |

Here C is a constant and independent of the input.

We assume that each statement will take constant amount of time which is 1.

So, the runtime will be = C1 + C2 + (n+1)C3 + n (C4 + C5) + C6

= C1 + C2 + C3 + C6 + n(C3 + C4 + C5)

= a + bn = O(n), Where a and b are constants.

The loop invariant holds before the first iteration and if the invariant holds before an iteration then it also holds before the next one and the invariant holds at the end of the iteration. So, the algorithm is correct.

**Answer to the Question No. – 2**

**(a)** For the outer loop,

|  |  |
| --- | --- |
| **Iterations** | **Value of i** |
| 1 | i = 3n |
| 2 | i = 3n – 4 |
| 3 | i = 3n – (2 \* 4) |
| 4 | i = 3n – (3 \* 4) |
| …. | … |
| k | i = 3n – (k - 1)\*4 |

From the given statement, 3n – (k - 1)\*4 = 1. So, k = (3n - 1)/4 + 1 = ¾ \* n + ¾ .

For outer loop we get O(n) after discarding the constant terms.

For the inner loop,

|  |  |
| --- | --- |
| **Iterations** | **Value of j** |
| 1 | j = 20n/30 |
| 2 | j = 20n/31 |
| 3 | j = 20n/32 |
| 4 | j = 20n/33 |
| …. | … |
| K | j = 20n/3(k-1) |

From the given statement, 20n/3(k-1) = 1

⇒ n = 3(k-1)/ 20

Taking log on both side we get, = (k-1) -

⇒ k = + / + 1

For inner loop we get O(), after discarding the constant terms.

So, we get the complexity is, n \* that is, θ(n) and n0 = 1. We know that we can replace big-O with big-θ.

**(b)** For the outer loop,

|  |  |
| --- | --- |
| Iterations | Value of i |
| 1 | i = 3n2 - 0 |
| 2 | i = 3n2 - 1 |
| 3 | i = 3n2 - 2 |
| 4 | i = 3n2 - 3 |
| …. | … |
| K | i = 3n2 – (k-1) |

From the given statement, 3n2 – (k-1) = 1. So, k = 3n2

For outer loop we get O(n2).

For the inner loop,

|  |  |
| --- | --- |
| Iterations | Value of j |
| 1 | j = 3n2/20 |
| 2 | j = 3n2/21 |
| 3 | j = 3n2/22 |
| 4 | j = 3n2/23 |
| …. | … |
| K | j = 3n2/2(k-1) |

From the given statement, 3n2/2(k-1) = 1

⇒ n2 = 2(k-1)/3

Taking log on both side we get, 2 = (k-1) -

⇒ k =(2/ \* + + 1

So, we get the complexity is, n2 \* that is, θ(n2) and n0 = 1. We know that big-O can be replaced with big-θ.

**Answer to the Question No. – 3**

**(a)** Let g and f be the function from the set of natural numbers to itself. The function f is said to be θ(g), if there are constants c1, c2 > 0 and a natural number n0 such that c1\* g(n) ≤ f(n) ≤ c2 \* g(n) for all n ≥ n0.

From the given expression we get, f(n) = 4n5 – 50n2 + 10n and g(n) = n5.

Now, 4n5 – 50n2 + 10n ≤ n5, (when n > 3)

≤ 4n5 + 10n

≤ 4n5 + 10n5

≤ 14n5 for c = 14 and n0 = 3

That is, 4n5 – 50n2 + 10n = O(n5)

Again, n5 ≤ n5 – 50n2 + 10n, (when n > 1)

≤ 4n5 – 50n2 + 10n for c = 1 and n0 = 1

That is, 4n5 – 50n2 + 10n = Ω(n5)

So we have, 4n2 – 50n2 + 10n θ(n5) is true.

**(b)** o(g(n)) = {f(n): for any positive constant c, there exists positive constant n0 such that 0 ≤ f(n) < cg(n) for all n ≥ n0}. So, the set of functions f(n) are strictly smaller than c\*g(n), meaning that small-o notation is a stronger upper bound than big-O notation.

Intuitively, this means that as the n approaches infinity, f(n) becomes insignificant compared to g(n). In mathematical terms:

Here, f(n) = 5n2/3 + 8 and g(n) = n. So,

This proves that 5n2/3 + 8 o(n) is true.

**(c)** Let g and f be the function from the set of natural numbers to itself. The function f is said to be Ω(g), if there is a constant c > 0 and a natural number n0 such that c\*g(n) ≤ f(n) for all n ≥ n0.

From the given expression we get, f(n) = n5 + 4n2 + 15 and g(n) = n3.

According to the definition we can write, n5 + 4n2 + 15 ≥ 1 \* n5; (When n>1)

⇒ 1 \* n3 (As n3< n5); For c1 = 1 and n0 = 1.

So, n5 – 4n2 + 15 Ω(n3) is true.

**Answer to the Question No. – 4**

Let’s assume, f1 = , f2 = , f3 = n1/3, f4 = n5, f5 = 10n and f6 = nn.

Comparing Asymptotic growth of function using logarithms we get,

|  |  |  |  |
| --- | --- | --- | --- |
| = | ≤ | = ; | f1 O(f2) |
| = | ≤ | = \* ; | f2 O(f3) |
| = \* | ≤ | = 5 \* ; | f3 O(f4) |
| = 5 \* | ≤ | = n \* ; | f4 O(f5) |
| = n \* | ≤ | = n \* ; | f5 O(f6) |

So we can write, ≤ ≤ n1/3 ≤ n5 ≤ 10n ≤ nn.