**1. What is the time, space complexity of the following code:**

int a = 0, b = 0;

for (i = 0; i < N; i++) {

a = a + rand();

}

for (j = 0; j < M; j++) {

b = b + rand();

}

Options:

O(N \* M) time, O(1) space

O(N + M) time, O(N + M) space

O(N + M) time, O(1) space

O(N \* M) time, O(N + M) space

Output:

O(N + M) time, O(1) space

Explanation: The first loop is O(N) and the second loop is O(M). Since we don’t know which is bigger, we say this is O(N + M). This can also be written as O(max(N, M)).

Since there is no additional space being utilized, the space complexity is constant / O(1)

**2. What is the time complexity of the following code:**

int a = 0;

for (i = 0; i < N; i++) {

for (j = N; j > i; j--) {

a = a + i + j;

}

}

Options:

O(N)

O(N\*log(N))

O(N \* Sqrt(N))

O(N\*N)

Output:

O(N\*N)

Explanation:

The above code runs total no of times

= N + (N – 1) + (N – 2) + … 1 + 0

= N \* (N + 1) / 2

= 1/2 \* N^2 + 1/2 \* N

O(N^2) times.

**3. What is the time complexity of the following code:**

int i, j, k = 0;

for (i = n / 2; i <= n; i++) {

for (j = 2; j <= n; j = j \* 2) {

k = k + n / 2;

}

}

Options:

O(n)

O(nLogn)

O(n^2)

O(n^2Logn)

Output:

O(nLogn)

Explanation: If you notice, j keeps doubling till it is less than or equal to n. Several times, we can double a number till it is less than n would be log(n).

Let’s take the examples here.

for n = 16, j = 2, 4, 8, 16

for n = 32, j = 2, 4, 8, 16, 32

So, j would run for O(log n) steps.

i runs for n/2 steps.

So, total steps = O(n/ 2 \* log (n)) = O(n\*logn)

**4. What does it mean when we say that an algorithm X is asymptotically more efficient than Y?**

Options:

X will always be a better choice for small inputs

X will always be a better choice for large inputs

Y will always be a better choice for small inputs

X will always be a better choice for all inputs

Answer:

X will always be a better choice for large inputs

Explanation: In asymptotic analysis, we consider the growth of the algorithm in terms of input size. An algorithm X is said to be asymptotically better than Y if X takes smaller time than y for all input sizes n larger than a value n0 where n0 > 0.

**5. What is the time complexity of the following code:**

int a = 0, i = N;

while (i > 0) {

a += i;

i /= 2;

}

Options:

O(N)

O(Sqrt(N))

O(N / 2)

O(log N)

Output: O(log N)

Explanation: We have to find the smallest x such that N / 2^x N

x = log(N)

**6. Which of the following best describes the useful criterion for comparing the efficiency of algorithms?**

Time

Memory

Both of the above

None of the above

Answer:

Both of the above

Explanation: Comparing the efficiency of an algorithm obviously depends on time and memory taken by an algorithm.

**7. How is time complexity measured?**

By counting the number of algorithms in an algorithm.

By counting the number of primitive operations performed by the algorithm on given input size.

By counting the size of data input to the algorithm.

None of the above

Answer:

By counting the number of primitive operations performed by the algorithm on given input size.

**8. What will be the time complexity of the following code?**

for(var i=0;i<n;i++)

i\*=k

O(n)

O(k)

O(logkn)

O(lognk)

Output:

O(logkn)

Explanation: because loops for the kn-1 times, so after taking log it becomes logkn.

**9. What will be the time complexity of the following code?**

var value = 0;

for(var i=0;i<n;i++)

for(var j=0;j<i;j++)

value += 1;

n

(n+1)

n(n-1)/2

n(n+1)/2

Output: n(n-1)/2

Explanation: First for loop will run for (n) times and another for loop will be run for (n-1) times so overall time will be n(n-1)/2.

**10. Algorithm A and B have a worst-case running time of O(n) and O(logn), respectively. Therefore, algorithm B always runs faster than algorithm A.**

True

False

Answer:

False

Explanation: The Big-O notation provides an asymptotic comparison in the running time of algorithms. For n < n0​​, algorithm A might run faster than algorithm B, for instance.