+ New chat

Not seeing what you expected here? Don't worry, your conversation data is preserved! Check back soon.

Upgrade to Plus NEW

Dark mode

Updates & FAQ

Log out

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;
    }
}
```

The time complexity of the given code can be calculated by counting the number of times the inner loop is executed.

The outer loop runs N times. For each iteration of the outer loop, the inner loop runs N - i times. Therefore, the total number of times the inner loop is executed is:

(N-0) + (N-1) + (N-2) + ... + 1

This is an arithmetic series with a first term of N and a common difference of -1. The sum of an arithmetic series can be computed using the formula:

sum = (n/2) * (first term + last term)

where n is the number of terms in Regenerate response = N and the last term is 1.

The time complexity of the code is O(N^2) since the total number of iterations of the inner loop is proportional to N^2.

The best case occur in binary search algorithm when

a.
Item is the middle element of the array or is not there at all

b.
Item is the first element in the array

c.
Item is not in the array at all

d.
Item is the middle element of the array

Regenerate response

middle position of the array. Therefore, the correct option is d.

Let f(n) = 7n + 8 and g(n) = n, find c (a suitable constant) such that O(g(n)) = f(n) for n ≥ n0

a.
7

b.
8

c.
There is no such constant

To find a constant c such that O(g(Regenerate response eed to find a positive constant c

Substituting the given functions, we get:

c * n ≥ 7n + 8 for n ≥ n0.

Rearranging the inequality, we get:

n ≥ (7/c) * n + 8/c for n ≥ n0.

For this inequality to hold for all n ≥ n0, we must have (7/c) ≤ 1, since the coefficient of n on the right-hand side is less than 1. Therefore, we can choose any constant c such that 7 ≤ c.

In other words, there are infinitely many constants c that satisfy the condition, and the smallest possible value is 7. Therefore, the correct option is a. 7.

The worst case complexity of Bubble sort algorithm is

a.
O(n2)

Regenerate response

c.
O(n)

d.
O(n log(n))

The worst-case time complexity of the Bubble sort algorithm is O(n^2), where n is the number of elements in the array to be sorted.

In the worst case, the algorithm needs to make n passes over the array, each pass requiring n-1 comparisons and swaps. Therefore, the total number of comparisons and swaps required is:

(n-1) + (n-2) + ... + 1 = (n * (n-1))/2

This expression is O(n^2) in terms Regenerate response its, and hence the worst-case time complexity of the Bubble sort

ChatGPT Feb 13 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

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;
    }
}
```

a.
O(N*Sqrt(N))

b.
O(N*log(N))

c.
O(N)

Regenerate response

ChatGPT Feb 13 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

O(N*N)

The time complexity of the given code can be calculated by counting the number of times the inner loop is executed.

The outer loop runs N/2 times. For each iteration of the outer loop, the inner loop runs log(N) times (base 2). Therefore, the total number of times the inner loop is executed is:

log(2) + log(4) + ... + log(N)

This can be simplified as:

log(24...*N) = log(2^(log(N/2)) * (N/2)!) = (log(N/2) + 1) * (N/2)

The time complexity of the code is therefore O(N log(N)) since the total number of iterations of the inner loop is proportional to N log(N).

Therefore, the correct option is b. (

No. of elements
in the array (N) Execution time
(micro seconds)
1024 51
2048 202
4096 805
8192 3227
16384 12900
32768 51592
What can be the possible average case time complexity of this sorting algorithm?
Select one:

a.
O(4N)

b.
O(Nlg(N))

d.
O(N)

We can use the empirical method to estimate the time complexity of the sorting algorithm based on the given execution time measurements. One common method is to plot the execution time against the input size on a log-log scale, and check if the plot looks approximately linear.

Taking the logarithm of both sides of the time complexity equation, we get:

log(T(N)) = k + c * log(N)

where T(N) is the execution time for an array of size N, k is a constant, and c is the time complexity exponent.

Plotting the log(T(N)) values agains[...]he given data, we obtain a straight line on the log-log scale, i[...]omplexity of the sorting

algorithm is O(N^c) = O(N^1.99), which is closer to O(N^2) than O(N log(N)).

Therefore, the correct option is c. O(N^2).

Express the function in terms of Θ-notation.

a.
Θ(n2)

b.
Θ(n3)

c.
Θ(n−−√)

d.

Θ(lg(n))

Regenerate response

Express the function (n^3)/1000 - 100n^2 - 100n + 3 in terms of Θ-notation.

a.
Θ(n2)

b.
Θ(n3)

c.
Θ(n−−√)

d.
Θ(lg(n))

To express the given function in te Regenerate response ed to find two functions g(n) and

(n^3)/1000 - 100n^2 - 100n + 3 = (1/1000)n^3 - 100n^2 - 100n + 3

For large values of n, the dominant term in the function is (1/1000)n^3. Therefore, we can choose g(n) = (1/1000)n^3 and h(n) = n^3.

For n > 1, we have:

g(n) = (1/1000)n^3 ≤ (1/100)n^3 = (1/100)h(n)

hence, for all n > 1:

(1/1000)n^3 ≤ f(n) ≤ n^3

Therefore, we can conclude that the given function is in Θ(n^3).

Therefore, the correct option is b. Θ(n^3).

Which of the following case does r Regenerate response eory

b.
Best Case

c.
Worst Case

d.
Null Case

In complexity theory, the concept of a "null case" does not exist. Therefore, option d. Null Case is the correct answer.

In complexity theory, we typically analyze the performance of algorithms in terms of their worst-case, best-case, or average-case running time, as well as their space complexity, among other factors. The worst-case running time represents the maximum amount of time an algorithm takes to run on any input of a given size. The best-case running time represents the minimum amount of time an algorithm takes to run on any input of a given size. The average-case running time represents the expected amount of time an algorithm takes to run on random inputs of a given size, according to some probability distribution.