**Chapter 2 - Big O**

1. Find the complexity of the function used to find the kth smallest integer in an unordered array of integers:

int selectkth(int a[], int k, int n)

{

int i, j, mini, tmp;

for (i = 0; i < k; i++)

{

mini = i;

for (j = i + 1; j < n; j++)

if (a[j] < a[mini])

mini = j;

tmp = a[i];

a[i] = a[mini];

a[mini] = tmp;

}

return a[k - 1];

}

Answer:

The complexity of the function to find smallest kth integer is in the two for loops.

The outer loop: for (i = 0; i < k; i++)

* K = # of iterations
* Runs k times (k-1)

The inner loop: for (j = i + 1; j < n; j++)

* Iterates n-i times for every outer loop

Complexity:

= (n-1) +(n-2)+(n-3)+…+(n-k+1)

= (k-1)n-1-2-3-…-(k-1)

complexity = O(kn)

2. Determine the complexity of the following implementations of the algorithms for adding, multiplying, and transposing n 3 n matrices:

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

for (j = 0; j < n; j++)

a[i][j] = b[i][j] + c[i][j];

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

for (j = 0; j < n; j++)

for (k = a[i][j] = 0; k < n; k++)

a[i][j] += b[i][k] \* c[k][j];

for (i = 0; i < n - 1; i++)

for (j = i + 1; j < n; j++) {

tmp = a[i][j];

a[i][j] = a[j][i];

a[j][i] = tmp;

}

Answer

Adding of matrix:

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

for (j = 0; j < n; j++)

a[i][j] = b[i][j] + c[i][j];

In the outer for loop runs for n times, while the inner for loop runs for n iterations. n\*n

If i = 0 (inner loop) goes from j = 0 to j = n-1 - - - n times

Therefore,

The complexity is n\*n, O(n^2)

Multiplication of a matrix:

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

for (j = 0; j < n; j++)

for (k = a[i][j] = 0; k < n; k++)

a[i][j] += b[i][k] \* c[k][j];

For multiplying, there are 2 nested for loops and 1 outer for loop. The process is like that of adding of a matrix except that we have another inner for loop. The 2 inner for loops can be read as 0(n^2) that then multiplies outer loop \* n

Therefore,

The complexity is O(n^3)

Transpose of matrix:

for (i = 0; i < n - 1; i++)

for (j = i + 1; j < n; j++)

{

tmp = a[i][j];

a[i][j] = a[j][i];

a[j][i] = tmp;

}

Answer

Here, we have the outer loop run for n times, for each time the inner loops run for n times.

Therefore, the complexity for transpose of a matrix is O(n^2)

3) If something is O(n^3), is it also O(n^2) ?

Answer:

No, O(n^3) is not also O(n^2). In fact, the opposite is true. If something is O(n^2) then it is also O(n^3). O notation can also be read as “<, less than”.

This is because if x < 4. Then it must also mean x is less than any integer that is high than 4.

In this problem, O(n^3) is cannot also be O(n^2), because O(n^3) is slower.

4) If an algorithm had an average case of O(n), a best case of O(log n), and a worst case of O(n^2), what is its overall Big O?

Answer:

We can find the algorithm’s overall Big O by finding the function with the highest value when substituting for n.

We can test by substituting n = 2^2048 (infinity)

Average case:

O(n) = O(2^2048) = 2^2048

Best Case:

O(log n) = O(log 2^2048) = 2048

Worst Case:

O(n^2) = O((2^2048)^2)) = 2^2048 \* 2^2048

Therefore:

O(log n) < O(n) < O(n^2)

Thus, O(n^2) is overall Big O of this algorithm because it has the biggest potential for value.

5) What is #4's Omega?

Answer:

Problem #4’s Omega is O(logn)