1    maxValue = numArr[0]            *# 2 ops       (assign and access)*

2    i = 1                           *# 1 op        (assign)*

3    while i < n - 1:                *# 2 ops \* n   (compare and arithmetic repeated n times)*

4        if maxValue < numArr[i]:    *# 2 ops \* n-1 (compare and access repeated for n-1 times)*

6            maxValue = numArr[i]    *# 2 ops \* n-1 (assign and access repeated for n-1 times)*

7        i = i + 1                   *# 2 ops \* n-1 (assign and arithmetic repeated n-1 times)*

8    return maxValue                 *# 1 op        (return)*

*# Best case: t(n) = 2 + 1 + 2n + 4(n-1) + 1 = 6n*

*# Worst case: t(n) = 2 + 1 + 2n + 6(n-1) + 1 = 8n-2*

1    int sum = 0;                    *// 1 op = O(1)*

2    for (int i = 0; i < n; i++)     *// (1 op) + (1 op \* n+1 loops) + (1 op \* n loops) = O(1 + (n+1) + n) = O(2n + 2)*

3    {

4        sum += arr[i];              *// 3 op \* n loop = O(3n)*

5    }

*// Time complexity: (1) + (2n + 2) + (3n) = O(5n + 3) = O(n)*

1   for (int i = 0; i < n; i++)      *// (1 op) + (1 op \* n+1 loops) + (1 op \* n loops) = O(1 + (n+1) + n) = O(2n + 2)*

2   {

3        for (int j = 0; j < n; j++) *// ((1 op) + (1 op \* n+1 loops) + (1 op \* n loops)) \* n loops = O(2n2 + 2n)*

4        {

5            arr[i][j] = i \* j;      *// (3 ops \* n loops) \* n loops = 3n = O(3n2)*

6        }

7    } *// Time complexity: (2n + 2) + (2n2 + 2n) + 3n2 = 5n2 + 4n + 2 = O(n2)*

1   for (int i = 0; i < n; i++)      *// O(n)*

2   {

3        for (int j = 0; j < n; j++) *// O(n) \* n loop 1 iterations = O(n2)*

4        {

5            arr[i][j] = i \* j;      *// O(1) \* n loop 1 iterations \* n loop 2 iterations = O(n2)*

6        }

7    } *// Time complexity: O(n2)*

1    for (int i = 1; i <= n; i++)        *// O(n)*

2    {

3        for (int j = 1; j <= i; j++)    *// Executes the sum of 0 + 1 + 2 + … n-1 times = n(n+1)/2*

4        {

5            ...                         *// O(1) \* n(n+1)/2 loops = O(n2)*

6        }

7    }                                   *// Time complexity: O(n2)*

1    for (int i = 0; i < n; i++)         *// O(n)*

2    {

3        for (int j = n; j > 1; j/=2)    *// O(log n) \* n loops = O(n log n)*

4        {

5            ...

6        }

7    }                                   *// Time complexity: O(n log n)*

1   int factorial(int n)

2   {

3       if (n <= 1) { return 1; }

4       else { return n \* factorial(n-1); }

5   } *// Time complexity: O(1) \* n recursive method calls = O(n)*

1   int fib(int n)

2   {

3       if (n == 0 || n == 1) { return n; }

4       else { return fib(n-1) + fib(n-2); }

5   }