1. Algorithms with time complexities such as n and 100n are called linear algorithms.
2. Algorithms with time complexities such as n2 are called quadratic-time algorithms (True or False). True
3. Any quadratic-time algorithm is eventually more efficient than any linear-time algorithm (True or False). False
4. Functions such as 5n2 and 5n2 +100 are called quadratic functions.

|  |  |
| --- | --- |
| T(N) | Growth function |
| n2 | 6 |
| 480 | 1 |
| 2n | 7 |
| logN | 2 |
| 24 | 5 |
| 380N | 4 |
| 1/2N | 3 |

|  |  |
| --- | --- |
| T(N) | Growth function |
| N logN | 4 |
| N4 | 8 |
| 2n | 9 |
| log8N | 3 |
| nlog4N | 5 |
| log2N | 2 |
| nlog6N | 6 |
| 300 | 1 |
| 6N3 | 7 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **T(n)** | **Constant** | **Linear** | **Polynomial** | **Exponential** |
| **1** | yes |  |  |  |
| **2n3** |  |  | yes |  |
| **(4/3)n** |  | yes |  |  |
| **2n** |  |  |  | yes |
| **4n2** |  |  | yes |  |
| **5600** | yes |  |  |  |
| **2493n** |  | yes |  |  |
| **3/2n** |  |  |  | yes |

1. f( n ) = 5n + 12 - O(n)
2. f( n ) = 109 - O(1)
3. f( n ) = n2+ 3n + 112 - O(n2)
4. f( n ) = n3 + 1999n + 1337 - O(n3)

What is the complexity of the functions below?

|  |
| --- |
| int sum = 0; for (int i = 1; i < N; i \*= 2)    for (int j = 0; j < N; j++)       sum++; |

O(N2)

|  |
| --- |
| function isEven(value){ if (value % 2 == 0){ return true; } else return false; } |

O(1)

|  |
| --- |
| arrayMax(A, n) { currentMax = A[0] For (i=1; i< A.length; i++){ If (A[i] > currentMax) then currentMax = A[i] } |

O(N)

|  |
| --- |
| function areYouHere(arr1, arr2) {     for (let i=0; i<arr1.length; i++) {        const el1 = arr1[i];            for (let j=0; j<arr2.length; j++) {            const el2 = arr2[j];                 if (el1 === el2) return true;        }       }    return false; } |

O(N2)

|  |
| --- |
| function isPrime(n) {     if (n < 2 || n % 1 != 0) {        return false;    }     for (let i = 2; i < n; ++i) {         if (n % i == 0) return false;    }    return true; } |

O(N)

|  |
| --- |
| function findRandomElement(arr) {    return arr[Math.floor(Math.random() \* arr.length)];  } |

O(1)

|  |
| --- |
| function createPairs(arr) {     for (let i = 0; i < arr.length; i++) {        for(let j = i+1; j < arr.length; j++) {            console.log(arr[i] + ", " +  arr[j] );        }     } } |

O(N2)

|  |
| --- |
| public static int binarySearch(int[] a, int key)   {      int lo = 0, hi = a.length-1;      while (lo <= hi)      {          int mid = lo + (hi - lo) / 2;          if      (key < a[mid]) hi = mid - 1;          else if (key > a[mid]) lo = mid + 1;          else return mid;     }     return -1;  } |

O(N)

|  |
| --- |
| sum = 0; for(inti=0;i<n;++i){ for(int j = 0; j < n\*n; ++j) { sum++; } } |

O(N3)

f(n) = 8*n* + 5

f(n)≤cg(n) ∀ n≥n0

f(n) = 8*n* + 5 <= c\*g(n)

C=9

8n+5 <= 9n

5<=9n-8n

5<=n

f(n)≤9g(n) ∀ n≥5