Solucion

Code 1

```
For ( int i = 0; i < n; i++){      O(n)      = O(n)
}
```

Code 2

```
For ( int i = 0; i < n; i++){          O(n)
    For ( int j = 0; j <m; j++){       O(m)        O(n * m)
    }
}
```

Code 3

```
For ( int i = 0; i < n; i++){          O(n)
    For ( int j = i ; j < n; j++){     O(n)        O(n^2)
    }
}
```

Code 4

```
int   index = -1;                 O(1)
For ( int i = 0; i < n; i++){     O(n)
    if (array[i] == target){      O(1)        O(n)
          index = i;              O(1)
          break;
    }
}
```

Code 5

```
int left = 0, right = n-1, index = -1;    O(1) → todas
while (left <= right){    O(1)
    int mid = left + (right - left) / 2;    O(log n)
    if(array[mid] == target){    O(1)
        index = mid;    O(1)
        break;    O(1)
    } else if(array[mid] < target){    O(1)
        left = mid + 1;    O(1)
    } else {    O(1)
        right = mid - 1;    O(1)
    }
}
```

= O(log n)

Code 6

```
int row = 0, col = matrix[0].length - 1, indexRow = -1, indexCol = -1;    O(1)
while(row < matrix.length && col >= 0){    O(1)
    if(matrix[row][col] == target){    O(log n)
        indexRow = row;    O(1)
        indexCol = col;    O(1)
        break;    O(1)
    } else if(matrix[row][col] < target){    O(1)
        row++;    O(1)
    } else {    O(1)
        col--; }    O(1)
}
```

O(Log n)

Code 7

```
void bubbleSort(int[] array){                    O(1)
    int n = array.length;                        O(1)
    for (int i = 0; i < n-1; i++){               O(n)
        for (int j = 0; j < n-i-1; j++){         O(n)
            if(array[j] > array[j+1]){           O(1)
                int temp = array[j];             O(1)
                array[j] = array[j+1];           O(1)
                array[j+1] = temp;               O(1)
            }
        }
    }
}
```

$O(n^2)$

Code 8

```
void selectionSort(int[] array){                 O(1)
  int n = array.length;                          O(1)
  for (int i = 0; i < n-1; i++){                 O(n)
      int minIndex = i;                          O(1)
      for (int j = 0; j < n-i-1; j++){           O(n)
          if (array[j] < array[minIndex]){       O(1)
              minIndex = j; }                    O(1)
      }
      int temp = array[i];                       O(1)
      array[i] = array[minIndex];                O(1)
      array[minIndex] = temp;}                   O(1)
}
```

$O(n^2)$

Code 9

```
void insertionSort(int[] array) {                    O(1)
    int n = array.length;                            O(1)
    for (int i = 1; i < n; i++) {                    O(n)
        int key = array[n];                          O(1)        O(n^2)
        int j = i - 1;                               O(1)
        while (j >= 0 && array[j] > key) {           O(n) O(n)
            array[j + 1] = array[j];                 O(log n)
            j--;                                     O(1)
        }
    array[j + 1] = key;                              O(1)
    }
}
```

Code 10

```
void mergeSort(int[] array, int left, int right) {   O(1)
    if (left < right) {                              O(1)
        int mid = left + (right - left) / 2;         O(1)
        mergeSort(array, left, mid);                 O(n log n)
        mergeSort(array, mid + 1, right);            O(n log n)
        merge(array, left, mid, right);              O(n log n)
    }                                                O(n log n)
}
```

Code 11

```
void quickSort(int[] array, int low, int high){ O(nlogn)
    if(low < high){ O(1)
        int pivotIndex = partition(array, low, high); O(1)
        quickSort(array, low, pivotIndex-1); O(nlogn)
        quickSort(array, pivotIndex+1, high); O(nlogn)
    }
                                                O(nlogn)
}
```

Code 12

```
int fibonacci(int n){ O(1)
    if(n <= 1){ O(1)
        return n; O(1)
    }
    int[] dp = new int[n+1]; O(1)          O(n)
    dp[0] = 0; O(1)
    dp[1] = 1; O(1)
    for(int i=2; i<=n; i++){ O(n)
        dp[i]= dp[i-1] + dp[i-2]; O(1)
    }
    return dp[n]; O(1)
}
```

Code 13

```
void linearSearch(int[] array, int target){   O(1)
    for(int i = 0; i < array.length; i++){   O(n)
        if(array[i] == target){   O(1)
                                                    O(n)
            // Encontrado
            return;   O(1)
        }
    }
    // No Encontrado
}
```

Code 15

```
int factorial(int n){   O(1)
    if(n == 0 || n == 1){   O(1)
        return 1;   O(1)                    O(n)
    }
    return n * factorial(n-1);   O(n)
}
```

Code 14

```
int binarySearch(int[] sortedArray, int target){   O(1)
    int left = 0, right = sortedArray.length-1;    O(1)
    while(left <= right){    O(1)
      int mid = left + (right - left) / 2;    O(log n)
      if (sortedArray[mid] == target){    O(1)
          return mid; // indice elemento encontrado O(1)
      } else if(sortedArray[mid] < target){    O(1)
          left = mid + 1;    O(1)
      }else{    O(1)
          right = mid -1;    O(1)
      }
                                          O(log n)
  }
  return -1;    O(1)
}
```