Braian Felipe Ramirez Ortiz 506222015

Analisis Big O

1. `for (int i = 0; i < n; i++)` O(n)

O(n)

2. `for (int i = 0; i < n; i++) {` O(n)

`for (int j = 0; j < m; j++) {` O(m)

`}`

`}`

O(n) x O(m)

O(n x m)

3. `for (int i = 0; i < n; i++) {` O(n)

`for (int j = 0; j < n; j++) {` O(n)

`}}`

O(n) x O(n) = O(n²)

4. `int index = -1;` O(1)

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

`if (array[i] == target) {` O(1)

`index = i;` O(1)

`Break;`

`}}`

~~O(1)~~ + O(n) + ~~O(1)~~ + ~~O(1)~~

O(n)

5.

```
int left = 0, right = n-1, index = -1;          O(1)
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(n) + O(1) + O(log n) + O(1) + O(1) + O(1) + O(1) + O(1)

O(Log n)

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(n+m)
    indexRow = Row;                              O(n)
    indexCol = col;                              O(m)
    Break;                                       O(1)
  } else if (matrix[row][col] < target) {        O(n+m)
    row++;                                       O(1)
  } else {                                       O(1)
    Col--;                                       O(1)
  }
}
```

O(1) + O(m+n) + O(n) + O(m) + O(1) + O(n+m) + O(1)

O(m+n)

TEMA / MATERIA DIA MES AÑO

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-1-i; 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(1)+O(1)+O(n×n)+O(1)+O(1)+O(1)+O(1)
$O(n^2)$

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 = i+1; j < n; 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(1)+O(1)+O(n×n)+O(1)+O(1)
$O(n^2)$

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

10. Void mergeSort(int[] array, int left, int rigth) { O(1)
if(left < Rigth) { O(1)
int mid = left + (right - left)/2; O(log n)
mergeSort(array, left, rigth); O(1)
mergeSort(array, mid+1, rigth); O(1)
merge(array, left, mid, rigth); O(1)
}}

O(1) + O(1) + O(log n) + O(1) + O(1) + O(1)

O(log n)

11. Void quickSort (int[] array, int low, int high) { O(1)
if (low < high) { O(1)
int pivotIndex = partition (array, low, high); O(1)
quickSort (array, low, pivotIndex - 1); O(n) - O(nlogn)
quickSort (array, pivotIndex + 1, high); O(n) - O(nlogn)
}} O(1) + O(1) + O(1) + O(nlogn) + O(nlogn)
$O(n^2)$

12. int Fibonacci (int n) { O(1)
if (n <= 1) { O(1)
return n; O(1)
}
int[] dp = new int[n+1]; O(1)
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)
} O(1) + O(1) + O(1) + O(n) + O(1) + O(1)
O(n)

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)
// Encontrado
return i;
}}
// no Encontrado
} O(1) + O(n) + O(1) = O(n)

14. int binarySearch (int[] sortedArray, int target) { O(1)
int left = 0, rigth = SortedArray.length - 1; O(n)
while (left <= rigth) { O(1)
int mid = left + (rigth - left) / 2; O(log n)
if (SortedArray[mid] == target) { O(1)
return mid; // Indice del elemento encontrado O(1)
} else if (SortedArray[mid] < target) { O(1)
left = mid + 1; O(1)
} else {
rigth = mid - 1; O(1)
}}
return -1; // Elemento no encontrado O(1)
} O(1) + O(n) + O(1) + O(log n) + O(1) + O(1) + O(1) + O(1)

O(log n)

TEMA / MATERIA

DÍA

15. int factorial (int n) { O(1)

if (n == 0 || n == 1) { O(1)

return 1; O(1)

}

return n * factorial (n - 1); O(n-1)

}

~~O(1) + O(1) + O(1) + O(n)~~

O(n)

$\int 3x^3 \, dx = \int (x^4+14)^{-1/2} (3x^3)$ $u = x^4+14$