|  |  |
| --- | --- |
| **once** | **repeating** |
| int Counter = 1; | Counter >=1 && Counter<=2 |
| Int max=0; | nums[pointer2] = nums[max]; |
| int pointer1=0; | nums[max] = temp; |
| int pointer2 = nums.length – 1; | i++; |
| int max=-9999; | pointer2--; |
| int maxIndex= 0; | Counter++; |
| int i = pointer1; | nums[pointer1] = nums[max]; |
| return maxIndex; | nums[max] = temp; |
|  | pointer1++; |
|  | Counter++; |
|  | if (Counter > 4) |
|  | Counter = 1; |
|  | i <= pointer2; |
|  | i++ |
|  | if (nums[i] > max) |
|  | max = nums[i] |
|  | maxIndex = i; |
|  | Int temp = nums[pointer2]; |
|  | int temp = nums[pointer1]; |

sortOfSort and maxIndex

Analysis

The time complexity of this class can be broken down between the two methods I used. In the sortOfSort method I have five instructions that only happen once, and in the maxIndex method I have three. This makes a total of eight instructions that happen once. In the Sort I use a while loop that has a total of 15 instructions that will run for n amount of times. In the maxIndex I have a for loop that has 5 instructions that run a total of n times. The time complexity both equations is T(n)=20n+8, however the run time depends on the given array. In terms of Big O(n) notation the result will be O(n). The best case would be that the array is already sorted, which would reduce the run time.