# public class Compare { public void timeBubbleSort(int[] arr){ boolean isSorted; long timeStart = System.*nanoTime*() \* 2,4,5,6,7,8,9,10 : O(1) 1/ for(int i = 0; i < arr.length; i++){ \* 3 lặp 1+(n-i-1)+1 = n-i+1 lần 2/ isSorted = false; \* 3 : O(n-i+1) 3/ for (int j = 0; j < arr.length - i - 1; j++){ \* 1 chính là thời gian 4/ if (arr[j] > arr[j + 1]) { toàn phương thức // swap arr[j+1] and arr[j] 5/ int temp = arr[j]; T(n)= = (n(n-1))/2 = O(n^2) 6/ arr[j] = arr[j + 1]; \* best case : O(n^2) 7/ arr[j + 1] = temp; \* average case : O(n^2) 8/ isSorted = true; \* worst case : O(n^2) } } //If no two elements were swapped then break 9/ if (!isSorted) { 10/ break; } } long timeEnd = System.*nanoTime*(); long time = timeEnd - timeStart; System.*out*.println("Bubble Sort: " + time); } //Selection Sort public void timeSelectionSort(int[] arr){ int minIndex = 0; \* 2,3,5,6,7,8,9,10,11,12 : O(1) boolean isSorted; \* 4 lặp i+1+n+1 = n+i+2 lần long timeStart = System.*nanoTime*(); \* 1 chính là thời gian toàn 1/ for(int i = 0; i < arr.length; i++){ phương thức // Find the minimum element in unsorted array 2/ minIndex = i; T(n)= = (n(n-1))/2 = O(n^2) 3/ isSorted = false; \* best case : O(n^2) 4/ for (int j = i + 1; j < arr.length; j++){ \* average case : O(n^2) 5/ if(arr[j] < arr[minIndex]){ \* worst case : O(n^2) 6/ minIndex = j; 7/ isSorted = true; } } // Swap the found minimum element with the first element 8/ int temp = arr[minIndex]; 9/ arr[minIndex] = arr[i]; 10/ arr[i] = temp; //If no two elements were swapped then break 11/ if (!isSorted) { 12/ break; } } long timeEnd = System.*nanoTime*(); long time = timeEnd - timeStart; System.*out*.println("Selection Sort: " + time); } //insertion Sort public void insertionSort(int[] arr){ \* 2,3,4,6,7,8,9,10,11 : O(1) boolean isSorted; \* 5 lặp n lần long timeStart = System.*nanoTime*(); \* 1 chính là thời gian toàn 1/ for(int i = 1; i < arr.length; i++){ phương thức 2/ int key = arr[i]; 3/ int j = i - 1; T(n)= = (n(n-1))/2 = O(n^2) 4/ isSorted = false; /\* Move elements of arr[0..i-1], that are \* best case : O(n) greater than key, to one position ahead. \* average case : O(n^2) of their current position \*/ \* worst case : O(n^2) 5/ while ( j >= 0 && arr[j] > key){ 6/ arr[j + 1] = arr[j]; 7/ j--; 8/ isSorted = true; } 9/ arr[j + 1] = key; //If no two elements were swapped then break 10/ if (!isSorted) { 11/ break; } } long timeEnd = System.*nanoTime*(); long time = timeEnd - timeStart; System.*out*.println("insertion Sort: " + time); } }