ConjuntoTDA - Costos:

**getArray() - O(n)**

*La complejidad de este método es O(n) porque necesita recorrer cada elemento del arreglo para construir la representación en cadena del mismo. Aquí, n es la cantidad de elementos en el arreglo, en el peor caso sería el tamaño del arreglo array (15).*

**inciarConjunto() - O(1)**

*Este método tiene una complejidad constante porque solo se le asigna memoria para un arreglo de tamaño fijo (15) y establece cant a 0. Ambas operaciones son constantes en términos de tiempo.*

**agregar(int x) - O(n)**

*La complejidad de este método está en el método pertenece(x), que es O(n) en el peor caso porque puede necesitar recorrer todo el arreglo para verificar si el elemento ya está presente. Si el elemento no está presente, agregarlo al final del arreglo y actualizar cant son operaciones constantes O(1).*

**sacar(int x) - O(n)**

*La complejidad de este método es O(n) en el peor caso porque el bucle while puede necesitar recorrer todo el arreglo para encontrar el elemento x. Las operaciones posteriores (asignar array[i] y decrementar cant) son constantes O(1)*

**elegir() - O(1)**

*Este método tiene una complejidad constante O(1) porque simplemente devuelve el último elemento añadido en el arreglo, lo cual es una operación directa de acceso a un índice del arreglo.*

**pertenece(int x) - O(n)**

*La complejidad de este método es O(n) en el peor caso porque puede necesitar recorrer todo el arreglo para verificar si el elemento x está presente.*

**estaVacio() - O(1)**

*Este método tiene una complejidad constante O(1) porque simplemente compara el valor de cant con 0, lo cual es una operación constante.*

ColaPrioridadTDA - Costos:

**inicializar() - O(1)**

*La complejidad de este método es O(1) porque solo se le asigna memoria para un arreglo de tamaño fijo (15) y establece index a 0. Ambas operaciones son constantes en términos de tiempo.*

**acolarPrioridad(int x, int y) - O(n)**

*La complejidad de este método es O(n) porque en el peor caso necesita recorrer el arreglo completo para encontrar la posición donde insertar el nuevo elemento basado en la prioridad.*

**desacolar() - O(1)**

*La complejidad de este método es O(1) porque decrementa el índice index, lo cual es una operación constante.*

**primero() - O(1)**

*La complejidad de este método es O(1) porque se devuelve el número del último elemento añadido en el arreglo, lo cual es una operación directa de acceso a un índice del arreglo.*

**prioridad() - O(1)**

*La complejidad de este método es O(1) porque devuelve la prioridad del último elemento añadido en el arreglo, lo cual es una operación directa de acceso a un índice del arreglo.*

**colaVacia() - O(1)**

*La complejidad de este método es O(1) porque compara el valor de index con 0, lo cual es una operación constante.*

**printArray() - O(n)**

*La complejidad de este método es O(n) porque necesita recorrer todo el arreglo para imprimir cada elemento presente en él.*

**invertirCola() - O(n)**

*La complejidad de este método es O(n) porque necesita recorrer aproximadamente la mitad del arreglo (n/2) para invertirlo, esto se simplifica a O(n).*

Main - Costos:

**for(int i=1; i<=15; i++) - O(n)**

*El bucle principal itera hasta 15 veces y, dentro de cada iteración, se realizan operaciones que incluyen la verificación y adición de IDs y prioridades, lo que involucra métodos O(n) como pertenece, agregar, y acolarPrioridad. Resultando en una complejidad total de O(n).*