

**UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS**

**2022 - 01**

**CURSO:** [**Algoritmos y Estructuras de Datos**](https://aulavirtual.upc.edu.pe/webapps/blackboard/execute/courseMain?course_id=_317376_1)

**DOCENTE:**  **Abraham Sopla Maslucán**

**SECCIÓN:** [**CC32**](https://aulavirtual.upc.edu.pe/webapps/blackboard/execute/courseMain?course_id=_317376_1)

**INTEGRANTES**:

| Alumno | Código | Carrera |
| --- | --- | --- |
| Pilco Chiuyare, André Dario | **u202110764** | Ciencias de la Computación |
| Roque Ponce, Christian Alonso | **u20201a917** | Ciencias de la Computación |
| Velasquez Chambi , Ruben Genaro | **u202117342** | Ingeniería de Software |

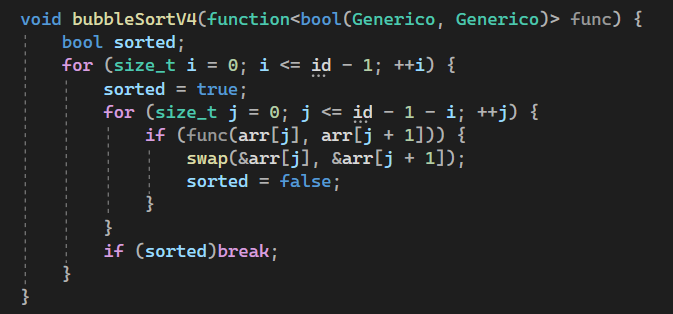
**Introducción y aplicación**

El presente trabajo se refiere a la implementación de las Estructura de Datos en C++ en un programa tipo “Gestor de Datos” (inspirado en la app “Microsoft To Do”). El uso de estas estructuras nos permite ordenar secuencialmente datos que, gracias al uso de Templates, recibe y almacena datos de tipo Class en Nodos. Para desarrollar el programa trabajamos con el paradigma de POO en C++ y codificamos una interfaz en consola para verificar la funcionalidad de nuestro “Gestor de Datos”. Este tipo de programa puede ser fácilmente utilizado para diversos ámbitos de la vida cotidiana. Entre ellos: Organizarse en las tareas, calificar la música favorita, generar horarios académicos, crear menús de comidas, etc.

**Explicación del Caso de Estudio:**

Empleamos tres estructuras de datos: listas simples, listas dobles y arreglos unidimensionales. Una lista simple únicamente tiene un apuntador al siguiente nodo, mientras que una lista doble tiene un apuntador al siguiente nodo y también uno al anterior. Estas estructuras se utilizan según el problema que se desee resolver. Por ejemplo, una lista simple la emplearemos para guardar datos que se requieran en secuencia y una lista doble se implementa cuando se debe tener la posibilidad de hacer secuencia hacia adelante o atrás a través de los nodos que conforman la lista.En el caso del arreglo, se maneja a través de índices, los cuales son muy útiles si se desea trabajar con elementos en específico.

**Big O Del Algoritmo Principal:**

****

for(size\_t i=0;i<= id-1; ++i){ —> 1+n(2+......+2)

sorted=true; →1

for(size\_t =0;j<=id-1-i;++j){ —-->1+n-1(3+1+2)

if( func(arr[j],arr[j+1])){

swap(&arr[j],&arr[j+1]);

sorted=false; —>1

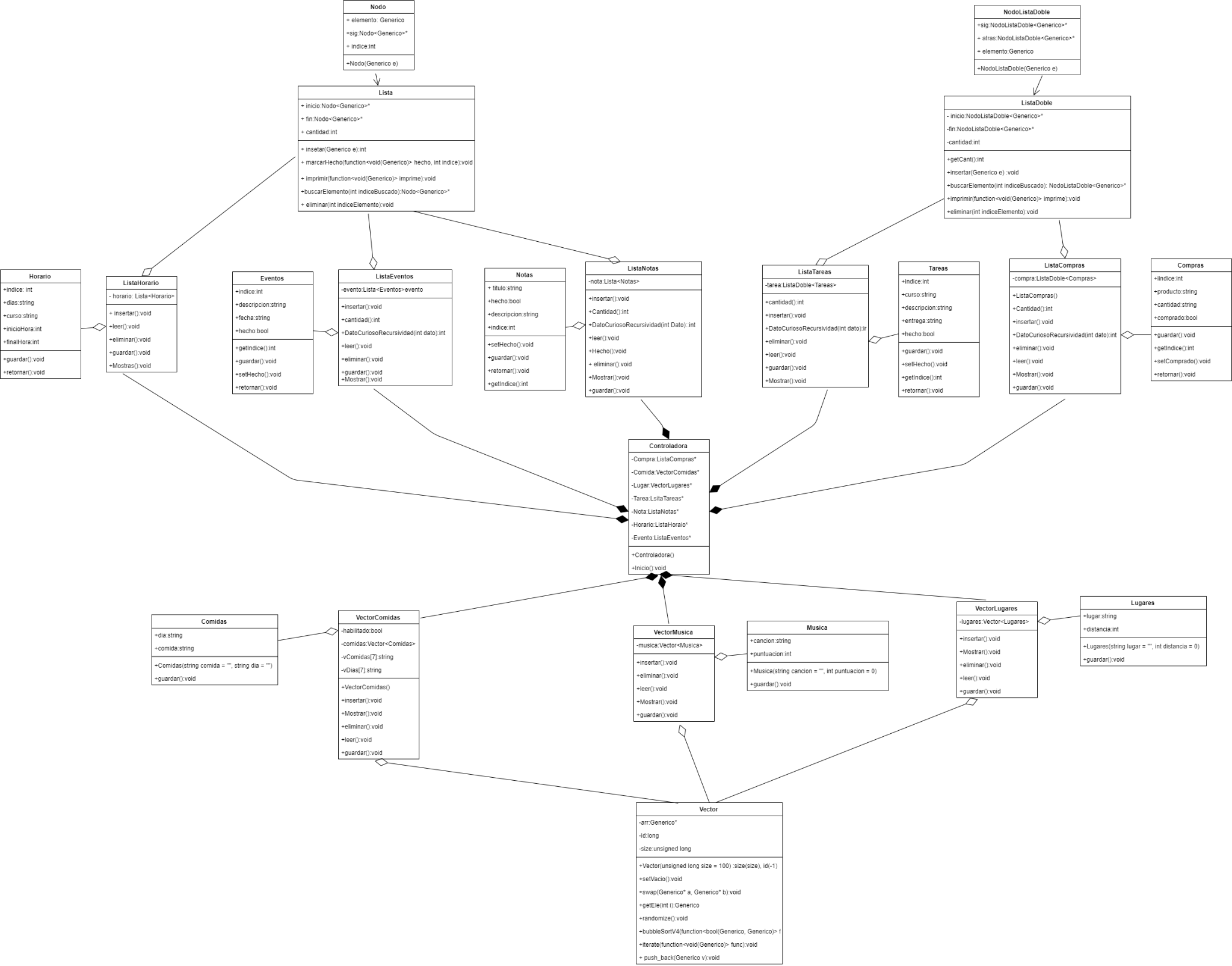
}

} if(sorted) break }

**Tiempo detallado:** 2+1+1+n-1(3+1+2))=

**Tiempo Asintótico**:

**Diagrama de clases:**

**Link del diagrama:** <https://drive.google.com/file/d/1cfbs1Ke7fdAe0nO96XOi1pCdVFZ8bSEC/view?usp=sharing>

**Link del video de exposicion**:<https://www.youtube.com/watch?v=TxA9coOcdbg>

**Descripción de las tareas realizadas:**

| **Nombres y apellidos:** | **Tarea realizada:** | **Tiempo** |
| --- | --- | --- |
| Christian Alonso Roque Ponce | Clase Lista y Nodo | 2 horas |
| Christian Alonso Roque Ponce | Métodos Insertar e Imprimir | 2 horas |
| Christian Alonso Roque Ponce | Constructores de Entidades | 30 minutos |
| Christian Alonso Roque Ponce | Implementar la Clase VectorLugar y la Clase Lugar | 1 hora |
| Christian Alonso Roque Ponce | Implementar las clases Vector en la Controladora | 30 minutos |
| André Dario Pilco Chiuyare | Métodos de recursividad y eliminar | 30 minutos |
| André Dario Pilco Chiuyare | Implementar la Clase Vector y VectorComida | 2 horas |
| André Dario Pilco Chiuyare | Interfaz y Listas de cada una de las Entidades | 3 horas |
| André Dario Pilco Chiuyare | Lectura y escritura de archivos | 2 horas |
| André Dario Pilco Chiuyare | Implementación del Algoritmo de ordenamiento bubblesort | 30 minutos |
| André Dario Pilco Chiuyare | Implementación de la clase Comida | 20 minutos |
| Ruben Genaro Velasquez Chambi | Implementar la clase lista doble | 1 hora |
| Ruben Genaro Velasquez Chambi | Implementar la Clases VectorMusica y Musica | 40 minutos |
| Ruben Genaro Velasquez Chambi | Diagrama de clases del proyecto | 1 hora 30 minutos |
| Ruben Genaro Velasquez Chambi | Análisis Big O | 20 min |