

**UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS - UPC**

**FACULTAD DE INGENIERÍA**

**ALGORITMOS Y ESTRUCTURA DE DATOS - CC32**

**PROYECTO**

**TP - Busca Empleo**

**INTEGRANTES**

| Puglisevich Vergara, Eduardo Elias | u20201e850 |
| --- | --- |
| Pozo Campos, Rodrigo Jair | u20181e187 |
| Vargas Soto, Lennin Jhair | u20201e766 |

**Link de la exposición:**<https://youtu.be/M77mXeUnuMI>

**PROFESOR**

**Sopla Maslucán, Abraham**

**Grupo 5**

**CICLO**

**2022 - I**

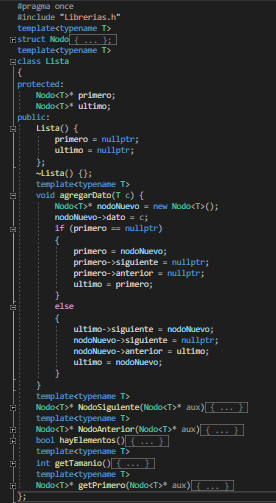
[**JUSTIFICACIÓN DE LAS ESTRUCTURAS IMPLEMENTADAS**](#_e07cwndiudmk) **3**

1. [Clase Vector Usuario](#_gg76vo6s37lc)3
2. [Clase Lista Amigos](#_ff18eoz3oow6) 5
3. [Clase ListaEmpleo](#_dileh9so2umv) 6
4. [Clase ListaCurso](#_hdg7o4b9vfp3) 7
5. [Clase ListaProducto](#_h2rdvs42rjc4) 8
6. [Clase ListaEvento](#_u2aqqx96hz56) 9

# JUSTIFICACIÓN DE LAS ESTRUCTURAS IMPLEMENTADAS

## Clase Lista

La clase Lista es una clase padre donde hereda a las demás listas sus atributos y métodos para ahorrar código. Esta ha sido creada con funciones que nos permiten manipular los nodos, como agregar uno por uno, verificar si existen los datos que están dentro de estos y saber cual es el dato siguiente o anterior al actual. Crear esta clase nos ayuda mucho a controlar en general los datos de todas las listas implementadas.



BIG(O) del método **agregarDato**:

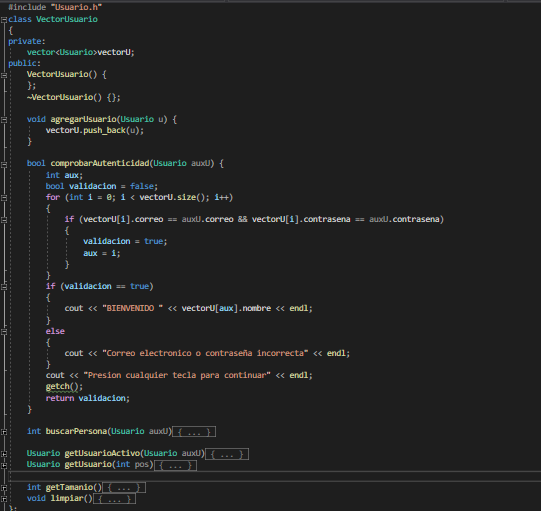
1. +1 por la asignación del NodoEmpleo\* nodoNuevo.
2. +1 por la asignación a nodoNuevo->empleo.
3. +1 por la comparación dentro del if (inicio == nullptr).
4. +4 por la expresión dentro del if.

Tiempo detallado: 1 + 1 + 1 + 4 = 7

Tiempo asintótico: O(1)

## Clase VectorUsuario

La clase Vector\_Usuario la creamos para que podamos guardar los datos registrados de los usuarios, además tiene una funcionalidad que nos permite reconocer si el correo y la contraseña con que se registro es la misma que se pone cuando se ingresa a la aplicación. Por último esta clase es de mucha importancia ya que, nos sirve para tener dentro de un vector, los distintos datos de un usuario al registrarse



BIG(O) del método **comprobarAutenticidad:**

1. por la asignación de false a la variable .
2. por la inicialización del y por la cantidad de iteraciones que realiza este.
3. por la comparación dentro del .
4. por la expresión dentro del , ya que esta es mayor a la del .
5. por el uso de dos operadores para la impresión.

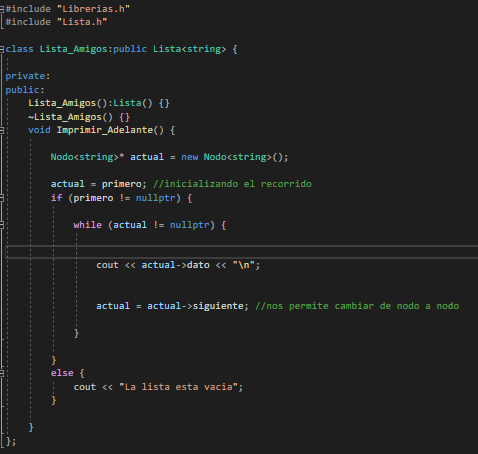
Tiempo detallado:

Tiempo asintótico: O(n)

## Clase Lista\_Amigos

La clase Lista\_Amigos lo implementamos debido a que necesitábamos un lugar donde se pueda guardar toda la información de un archivo .txt que contiene nombres de personas, las cuales vendrían a ser los amigos de cada usuario registrado.

Esto es una implementación de una lista con punteros inicio y fin, que nos ayuda a insertar los nombres que están en el .txt y que se puedan imprimir en pantalla.



BIG(O) del método **Inserta**:

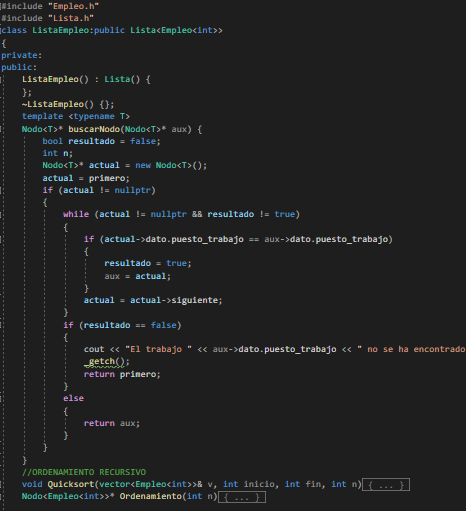
1. +1 por la asignación del NodoAmigos\* nuevo.
2. +1 por la asignación a nuevo->valor.
3. +1 por la comparación dentro del if (inicio == nullptr).
4. +4 por la expresión dentro del if.
5. +1 por el uso de un operador <<.

Tiempo detallado: 1 + 1 + 1 + 4 + 1 = 8

Tiempo asintótico: O(1)

## Clase ListaEmpleo

La clase Lista Empleo es una donde el valor que se guardará en cada nodo serán los datos del empleo. Esta lista ha sido creada con funciones que nos permitan manipular los nodos, como agregar uno por uno, verificar si existen los datos del empleo y saber cual es el dato siguiente al actual. Crear esta clase nos ayuda mucho a controlar en general los datos de la opción empleo.



BIG(O) del método **agregarEmpleo**:

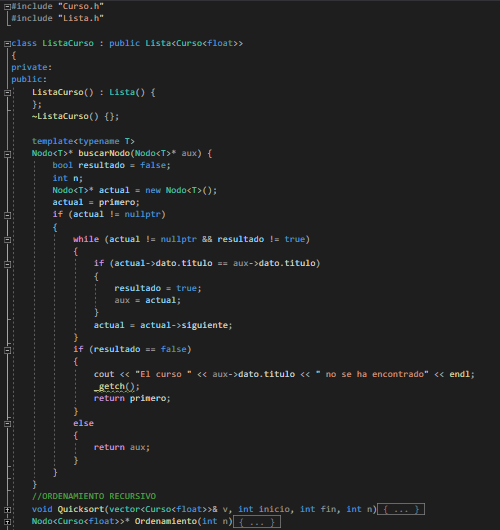
1. +1 por la asignación del NodoEmpleo\* nodoNuevo.
2. +1 por la asignación a nodoNuevo->empleo.
3. +1 por la comparación dentro del if (inicio == nullptr).
4. +4 por la expresión dentro del if.

Tiempo detallado: 1 + 1 + 1 + 4 = 7

Tiempo asintótico: O(1)

## Clase ListaCurso

Decidimos crear la clase ListaCurso para almacenar todos los cursos que ingresarán los usuarios en la función de publicar curso. Además, esta clase nos permitirá también mostrar los cursos en la función buscar cursos que desee el usuario. Y toda la información que se use en esta lista se guardará en un archivo .dat.



BIG(O) del método **agregarCurso**:

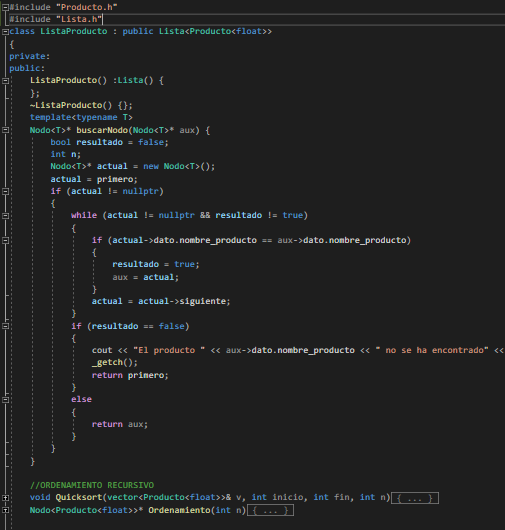
1. +1 por la asignación del NodoCurso\* nodoNuevo.
2. +1 por la asignación a nodoNuevo->curso.
3. +1 por la comparación dentro del if (inicio == nullptr).
4. +4 por la expresión dentro del if.

Tiempo detallado: 1 + 1 + 1 + 4 = 7

Tiempo asintótico: O(1)

## Clase ListaProducto

Implementamos la ListaProducto, ya que, en LinkedIn que es la aplicación de referencia se puede publicar y ver un producto. Por ello, creemos que es necesario almacenar los productos publicados en una lista para que el usuario luego si busca uno de ellos tenga acceso a él a través del nombre del producto. Además, todos los datos de la lista se guardan en un archivo .dat.



BIG(O) del método **agregarProfucto**:

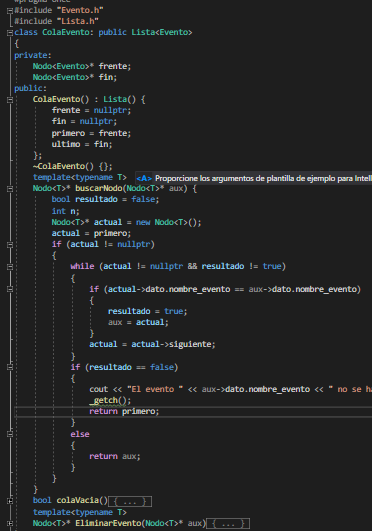
1. +1 por la asignación del NodoProducto\* nodoNuevo.
2. +1 por la asignación a nodoNuevo->prodcuto.
3. +1 por la comparación dentro del if (inicio == nullptr).
4. +4 por la expresión dentro del if.

Tiempo detallado: 1 + 1 + 1 + 4 = 7

Tiempo asintótico: O(1)

## Clase ColaEvento

Creamos la ColaEvento porque necesitábamos almacenar los eventos publicados por los usuarios y decidimos que esta era la forma más eficiente para ello. También para mostrar esos eventos si el usuario lo busca por el nombre del evento y porque con la cola podemos eliminar los eventos que han sido ingresados primero. Por último guardamos toda esta información en un archivo .dat.



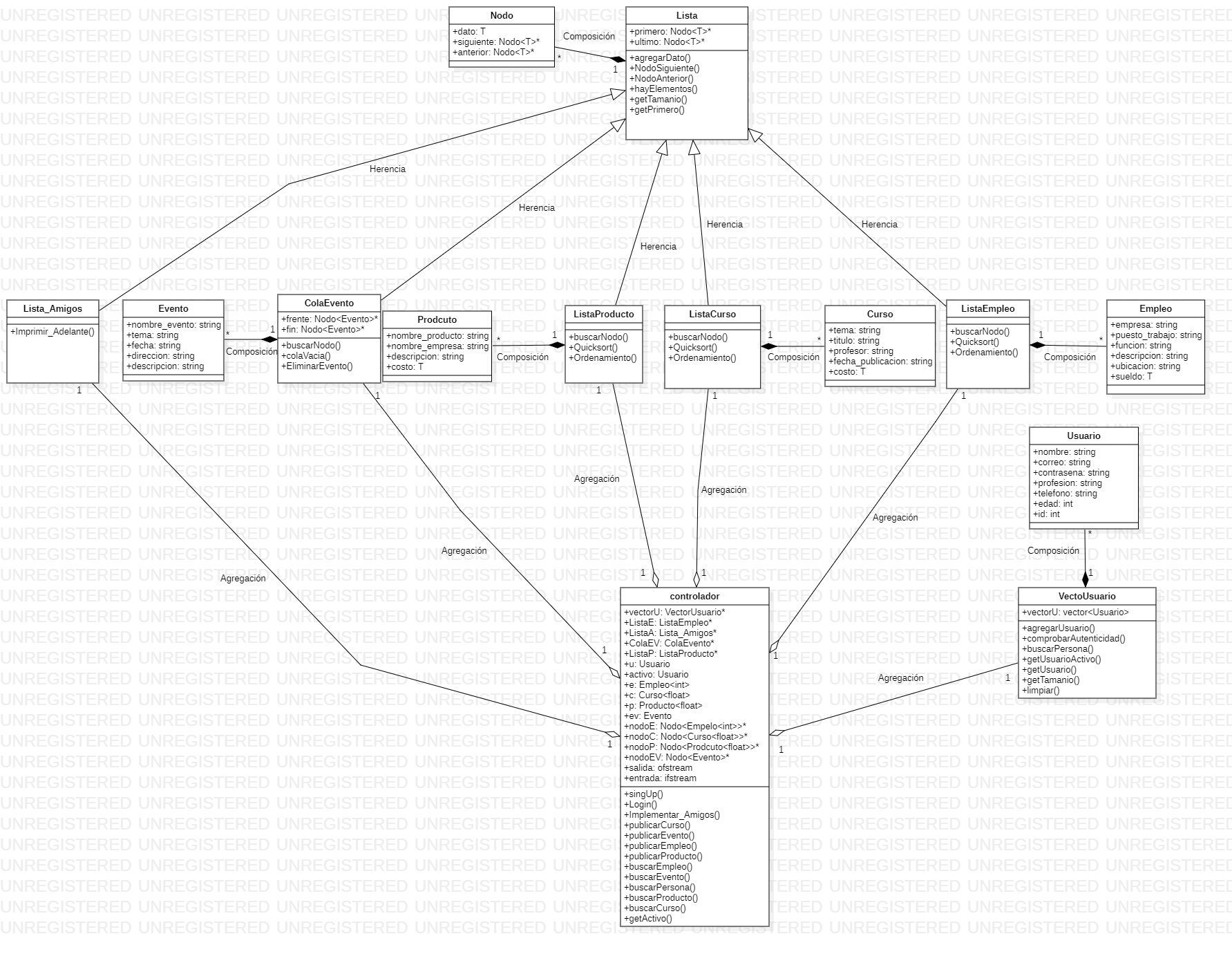
BIG(O) del método **agregarEvento**:

1. +1 por la asignación del NodoEvento\* nodoNuevo.
2. +1 por la asignación a nodoNuevo->evento.
3. +1 por la comparación dentro del if (inicio == nullptr).
4. +4 por la expresión dentro del if.

Tiempo detallado: 1 + 1 + 1 + 4 = 7

Tiempo asintótico: O(1)

# DIAGRAMA DE CLASES



# Participación de integrantes

| Integrantes | Participó | No Participó | Descripción de tareas que realizo |
| --- | --- | --- | --- |
| Puglisevich Vergara, Eduardo Elias | x |  | Implementó las estructuras de datos de la clase padre Lista y la clase hija Listaempleos. También implementó el vector usuarios. Asimismo, diseñó el ordenamiento recursivo Quicksort.  Por otro lado, elaboró el diagrama de clases. |
| Pozo Campos, Rodrigo Jair | x |  | Creó la lista para la clase hija lista\_Amigos y creó una función para leer datos de los amigos de un archivo de texto. Además ayudó a la creación para la implementación de una cola llamada eventos. |
| Vargas Soto, Lennin Jhair | x |  | Creó las listas y estructuras para las clases hijas Listacursos, Listaproductos y Listaeventos. |