

**UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS**

**TRABAJO FINAL**

**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 el lenguaje 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. 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,almacenar contraseñas y fechas de cumpleaños, etc.

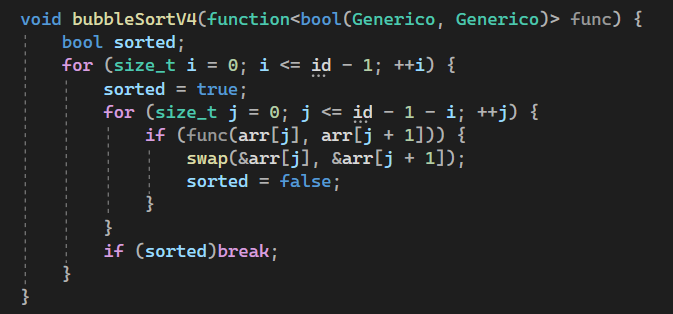
**Explicación del Caso de Estudio:**

En el presente trabajo se implementa los conocimientos de estructuras de datos y algoritmos de ordenamiento en el lenguaje de programación C++ donde se creará un "Gestor de Datos", inspirado en la app "Microsoft To Do",donde se usará estructuras de datos como: listas,listas dobles,listas circular doblemente enlazadas,arreglos unidimensionales, hashtable y arboles binarios que ayudarán en la inserción, desplazamiento, búsqueda y eliminación de datos.Además, de usar Templates para la creación de un programa genérico que almacene datos,funciones lambda y funciones recursivas,todo bajo el paradigma de POO en C++.la codificación del "Gestor de Datos" se ejecutará en una interfaz de consola y estos datos se guardarán en archivos .txt o .csv . Para la creación de listas de horarios, eventos, notas,tareas,compras y cumpleños se estará usando las estructuras de dato de lista simple y dobles, por otro lado para el almacenamiento de contraseñas se usará las estructuras hashtable para facilitar su búsqueda, las estructuras de datos vector se usarán en la inserción de músicas, comidas y lugares.Por último se usará arboles binarios para priorizar tareas.

**Explicación de las estructuras :**

Para el correcto desarrollo del proyecto “Gestor de tareas” que se nos fue asignado, nos pusimos de acuerdo de manera grupal al escoger las distintas estructuras de datos a implementar para conseguir que el programa funcione de forma óptima. Es así que optamos por usar las siguientes estructuras de datos: listas simplemente enlazadas, listas doblemente enlazadas, listas circulares doblemente enlazadas, arreglos unidimensionales, Hashtable y árboles binarios. En primer lugar, para el caso de las listas consideramos su uso para datos que queríamos ordenar de manera lineal (para listas simples y dobles) o de manera circular (para listas dobles circulares) y así facilitar tareas como insertar, imprimir o eliminar nodos según se requiera para entidades como Horario, Notas, Tareas, Compras, Eventos y Cumpleaños. Por otro lado, usamos arreglos unidimensionales para almacenar las entidades Comida y Música de manera dinámica según se ingresen o eliminen elementos a petición del usuario y también implementamos un método de búsqueda avanzado. Finalmente, consideramos el uso de las estructuras Hashtable y Árboles binarios para almacenar entidades como Contrasenia y TareasDePrioridad porque estas estructuras nos permiten manejar grandes cantidades de datos con mayor facilidad y eficiencia al momento de buscar y leer elementos contenidos en sus nodos.

**Big O Principal de la Estructura Vector:**

****

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

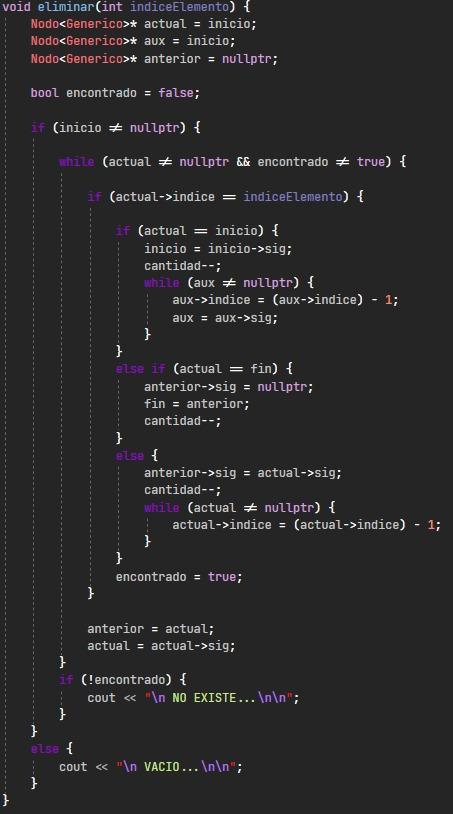
}

} if(sorted) break }

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

**Tiempo Asintótico**:

**Big O Principal de la Estructura Lista:**



Nodo<Generico>\* actual = inicio; →1

Nodo<Generico>\* aux = inicio; →1

Nodo<Generico>\* anterior = nullptr; →1

bool encontrado = false; →1

if (inicio != nullptr) { →1

while (actual != nullptr && encontrado != true) { →O(n)

**if (actual->indice == indiceElemento) { →1**

**if (actual == inicio) { →1**

**inicio = inicio->sig; →1**

**cantidad--; →2 1+3+2n =4+2n**

**while (aux != nullptr) { → O(n)**

**aux->indice = (aux->indice) - 1; →1**

**aux = aux->sig; →1**

**}**

}

else **if (actual == fin) { →1**

**anterior->sig = nullptr; →1**

**fin = anterior; →1 1+4=5**

**cantidad--; →2**

}

else {

anterior->sig = actual->sig; →1

cantidad--; →2 3+2n

while (actual != nullptr) { →O(n)

actual->indice = (actual->indice) - 1; →2

}

}

encontrado = true; →1

}

anterior = actual; →1

actual = actual->sig; →1

}

if (!encontrado) { →1

cout << "\n NO EXISTE...\n\n"; →1

}

}

else {

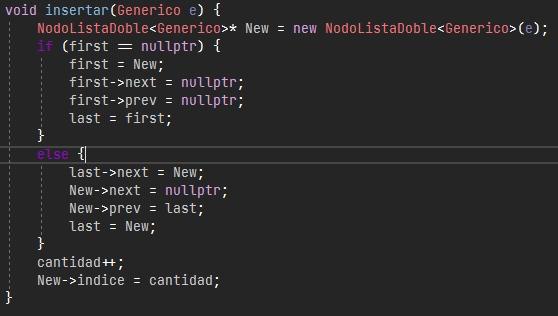
cout << "\n VACIO...\n\n"; →1

}

**Tiempo detallado: 4+ 1+ (1+4+2n+1+2)n +1 =**

**Tiempo Asintótico**:

**Big O Principal de la Estructura ListaDoble:**



NodoListaDoble<Generico>\* New = new NodoListaDoble<Generico>(e); →2

if (first == nullptr) { →1

first = New; → 1

first->next = nullptr; →1 4

first->prev = nullptr; →1

last = first; →1

}

else {

last->next = New; →1

New->next = nullptr; →1 4

New->prev = last; →1

last = New; →1 }

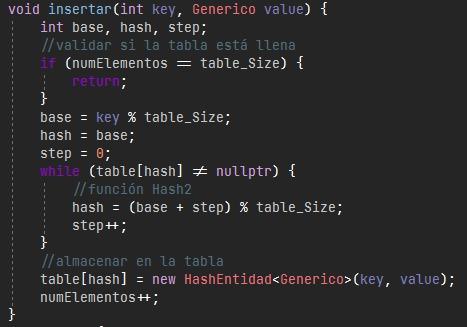
cantidad++; →2

New->indice = cantidad; →1

**Tiempo detallado: 2+1+4+2+1 =10**

**Tiempo Asintótico**:

**Big O Principal de la Estructura HashTable:**



int base, hash, step; →3

//validar si la tabla está llena

if (numElementos == table\_Size) { →1

return;

}

base = key % table\_Size; →2

hash = base; →1

step = 0; →1

while (table[hash] != nullptr) { → O(n)

hash = (base + step) % table\_Size; →3

step++; →2

}

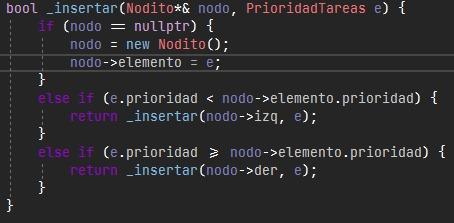
table[hash] = new HashEntidad<Generico>(key, value); →3

numElementos++; →2

**Tiempo detallado: 3+1+4+ 5n+5 =13+5n**

**Tiempo Asintótico**:

**Big O Principal de la Estructura Arbol Binario:**



if (nodo == nullptr) { →1

nodo = new Nodito(); →2 3

nodo->elemento = e; →1

}

else if (e.prioridad < nodo->elemento.prioridad) { →1

return \_insertar(nodo->izq, e); →1 1

}

else if (e.prioridad >= nodo->elemento.prioridad) { →1

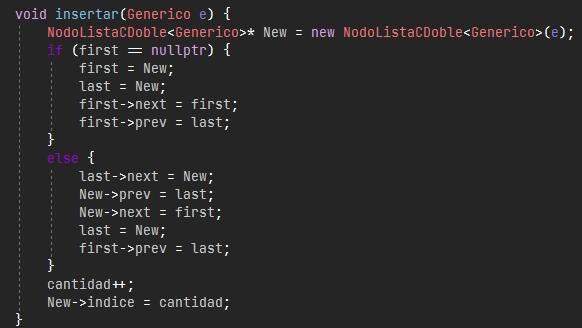
return \_insertar(nodo->der, e); →1 1

}

**Tiempo detallado: 3+1 =4**

**Tiempo Asintótico**:

**Big O Principal de la Estructura Lista Circular Doblemente Enlazada:**



NodoListaCDoble<Generico>\* New = new NodoListaCDoble<Generico>(e); →2

if (first == nullptr) { →1

first = New; →1

last = New; →1

first->next = first; →1 4

first->prev = last; →1

}

else {

last->next = New; →1

New->prev = last; →1

New->next = first; →1 5

last = New; →1

first->prev = last; →1

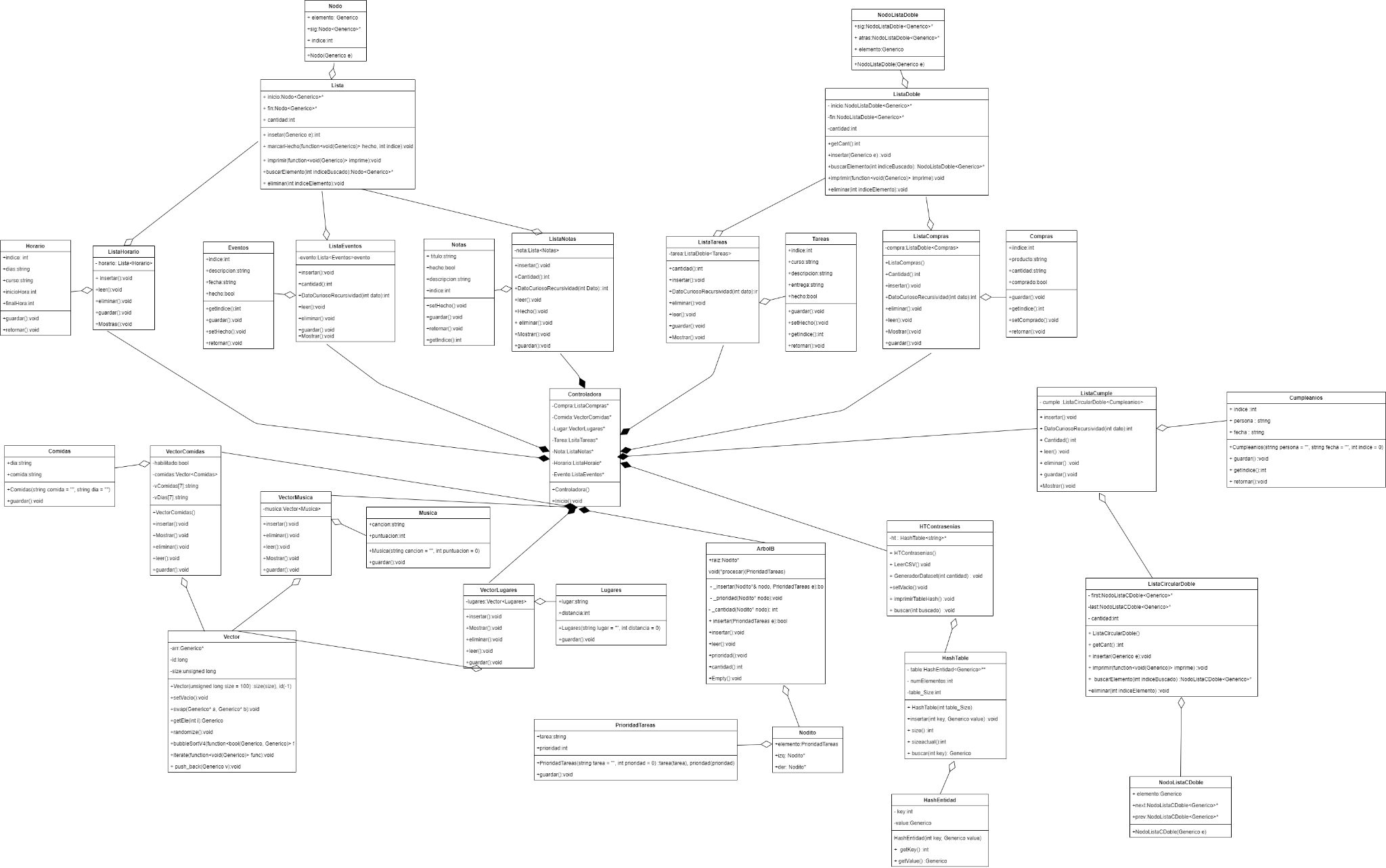
}

cantidad++; →2

New->indice = cantidad; →1

**Tiempo detallado: 2+5+1+2+1=11** **Tiempo Asintótico**:

**Diagrama de clases:**

****

**Link del diagrama:** <https://drive.google.com/file/d/159fVGgT1RsQ7TG-DHJxiwftfOJsoXm-N/view?usp=sharing>

**Link del video de exposicion**: -

**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 |
| Christian Alonso Roque Ponce | Implementación de Hashtable | 3 horas |
| 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 |
| André Dario Pilco Chiuyare | Implementación de la clase ÁrbolBinario | 3 horas |
| André Dario Pilco Chiuyare | Implementación Lista Circular Doblemente Enlazada | 3 horas |
| André Dario Pilco Chiuyare | Generador de Dataset | 1 hora |
| 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 |

| Nombre | Autovaloración |
| --- | --- |
| André Dario Pilco Chiuyare | 40% |
| Christian Alonso Roque Ponce | 35% |
| Ruben Genaro Velasquez Chambi | 25% |