

MANUAL TÉCNICO

Ingreso de Jugadores y Fichas - $O(1)$:

Manipulación de fichas con Lista Enlazada - $O(n)$

```
#ifndef LISTA_OBJETOS_HPP #define LISTA_OBJETOS_HPP

#include <string>
#include <iostream>

template <typename T>
class ListaObjetos {
private:
    struct Nodo {
        T* objeto;
        Nodo* siguiente;
        Nodo(T* objeto) : objeto(objeto), siguiente(nullptr) {}
    };
    Nodo* cabeza;

public:
    ListaObjetos() : cabeza(nullptr) {}

    void agregar(T* objeto) {
        Nodo* nuevo = new Nodo(objeto);
        if (!cabeza) {
            cabeza = nuevo;
        } else {
            Nodo* actual = cabeza;
            while (actual->siguiente) {
                actual = actual->siguiente;
            }
            actual->siguiente = nuevo;
        }
    }

    T* buscar(int id) {
        Nodo* actual = cabeza;
        while (actual) {
            if (actual->objeto->getId() == id) {
                return actual->objeto;
            }
            actual = actual->siguiente;
        }
        return nullptr;
    }

    void eliminarElemento(int id) {
        Nodo* actual = cabeza;
        Nodo* anterior = nullptr;

        if (actual != nullptr && actual->objeto->getId() == id)
```

```

        {
            cabeza=actual->siguiente;
            delete actual;
            return;
        }

        while (actual != nullptr) {
            if (actual->objeto->getId() == id) {

                anterior->siguiente = actual->siguiente;
                delete actual;
                return;
            }

            anterior = actual;
            actual = actual->siguiente;
        }
    }

    void imprimir() {
        Nodo* actual = cabeza;
        while (actual) {
            std::cout << ", ID: " << actual->objeto->getId()
                << " | ";
            actual = actual->siguiente;
        }
        std::cout << "NULL" << std::endl;
    }

    void imprimirFichas() {
        Nodo* actual = cabeza;
        while (actual) {
            std::cout << ", ID: " << actual->objeto->getLetraAsignada()
                << " | ";
            actual = actual->siguiente;
        }
        std::cout << "NULL" << std::endl;
    }

    ~ListaObjetos() {
        Nodo* actual = cabeza;
        while (actual) {
            Nodo* temp = actual;
            delete actual->objeto;
            actual = actual->siguiente;
            delete temp;
        }
    }
};

#endif

```

Utilice, una lista enlazada para manejar a los jugadores, y sus fichas, ya que es una manera muy práctica y sencilla de manejar;

Gestión de turnos por Colas - O(1)

```
#include "../include/ColaCircular.hpp"
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

ColaCircular::ColaCircular(int* ids, int numJugadores) {
    capacidad = numJugadores;
    cola = new int[capacidad];
    frente = 0;
    final = capacidad - 1;
    elementos = capacidad;

    for (int i = 0; i < capacidad; i++) {
        cola[i] = ids[i];
    }

    mezclarCola();
}

ColaCircular::ColaCircular() {

}

ColaCircular::~ColaCircular() {
    delete[] cola;
}

void ColaCircular::mezclarCola() {
    srand(time(0));
    for (int i = capacidad - 1; i > 0; i--) {
        int j = rand() % (i + 1);
        swap(cola[i], cola[j]);
    }
}

void ColaCircular::encolar(int id) {
    final = (final + 1) % capacidad;
    cola[final] = id;
}
```

```
    elementos++;  
}  
  
int ColaCircular::desencolar() {  
    if (elementos == 0) {  
        cout << "se vacio la cosa" << endl;  
        return -1;  
    }  
  
    int idJugador = cola[frente];  
    frente = (frente + 1) % capacidad; //  
    elementos--;  
    return idJugador;  
}  
  
bool ColaCircular::estaVacia() {  
    return elementos == 0;  
}
```

Utilice una cola circular por así decirlo, que me permite encolar y desencolar a los jugadores para los turnos del juego,

Ordenación de palabras iniciales - $O(n^2)$

```
void CargaDeArchivo::ordenarPalabras() {  
    for (int i = 0; i < tamañoPalabras - 1; i++) {  
        for (int j = 0; j < tamaño Palabras - i - 1; j++) {  
            if (arr[j] > arr[j + 1]) {  
                std::string temp = arr[j];  
                arr[j] = arr[j + 1];  
                arr[j + 1] = temp;  
            }  
        }  
    }  
}
```

Método para ordenar las palabra ingresadas de forma alfabética, utilizando el método burbuja ya recorre el arreglo múltiples veces.

Ordenación de fichas por puntuación - $O(n \log n)$

Cálculo y ordenación de puntuaciones - $O((n \log n))$

```
#include "../include/QuickSort.hpp"
#include <iostream>

using namespace std;

void MergeSort::mergeSort(int* arreglo, int ini, int final) {

    if (ini < final) {
        int med=(ini+final)/2;
        mergeSort(arreglo, ini, med);
        mergeSort(arreglo, med+1, final);
        Mezclar(arreglo,ini,med,final);
    }
}

void MergeSort::Mezclar(int* arreglo, int ini, int medio, int final) {
    int izq = ini;
    int der = medio + 1;
    int ia = 0;
    int tamanoAux = final - ini + 1;
    int* listaAuxiliar = new int[tamanoAux];

    while (izq <= medio && der <= final) {
        if (arreglo[izq] < arreglo[der]) {
            listaAuxiliar[ia] = arreglo[izq];
            izq++;
        } else {
            listaAuxiliar[ia] = arreglo[der];
            der++;
        }
        ia++;
    }

    while (der <= final) {
        listaAuxiliar[ia] = arreglo[der];
        der++;
        ia++;
    }
}
```

```
while (izq <= medio) {  
    listaAuxiliar[ia] = arreglo[izq];  
    izq++;  
    ia++;  
}  
  
for (int i = 0; i < ia; i++) {  
    arreglo[ini + i] = listaAuxiliar[i];  
}  
}
```

He decidido usar merge sort ya que es muy eficiente, ya que no importa si esta en el peor, o mejor caso, siempre sera igual de eficiente.

Registro de palabras jugadas con pila - O(1)

```
#ifndef PILA_HPP
#define PILA_HPP
#include <string>

class Nodo{
public:
    std::string dato;
    Nodo* siguiente;

    Nodo(std::string d){
        dato=d;
        siguiente = nullptr;
    }
};

class Pila{
private:
    Nodo* frente;
    Nodo* final;

public:
    Pila() {
        frente = nullptr;
        final=nullptr;
    }

    void agregarDatos(std::string dato) {

        Nodo* nodoNuevo=new Nodo(dato);

        if (frente==nullptr)
        {
            frente=nodoNuevo;
            final=nodoNuevo;
        }else{
```



```

        nodoNuevo->siguiente=frente;
        frente=nodoNuevo;

    }

}

std::string desempila(){

    if (frente)
    {
        std::string dat=frente->dato;
        Nodo* nuevoFrente=frente->siguiente;
        frente=nuevoFrente;

        return dat;
    }

    return "";

}

};

#include <iostream>

#endif

```

Use una pila para almacenar las palabras, ya que era útil para almacenar las palabra conforme se encontraban.