UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ESTRUCTURA DE DATOS

BÚSQUEDA BINARIA Y HASH

NOMBRE: QUISHPE CHAVEZ DENISSE PAULINA

NRC: 29852

FECHA: 08/12/2025

DOCENTE: SOLIS ACOSTA EDGAR FERNANDO

Contenido

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

BÚSQUEDA BINARIA Y HASH

EJERCICIO 1 – LIBRO DISEÑO DE ALGORITMOS Y SU CODIFICACION EN LENGUAJE C

Implementar la función de búsqueda binaria en Lenguaje C para localizar un número específico (nbus) dentro de un arreglo de números enteros (v), aprovechando la eficiencia del algoritmo divide y vencerás.

- La búsqueda se basa en el conocido método divide y vencerás.
- Búsqueda donde se requiere que los elementos estén ordenados.

```
#include <iostream>
using namespace std;

// Función de búsqueda binaria
int busquedaBinaria(int v[], int n, int nbus) {
    int izq = 0;
    int der = n - 1;
    int iteracion = 1;

    cout << "\n=== Proceso de Búsqueda Binaria ===" << endl;
    cout << "Buscando el número: " << nbus << endl;
    cout << "Tamaño del arreglo: " << n << endl;

    while (izq <= der) {
        int medio = (izq + der) / 2;

        cout << "\n--- Iteración " << iteracion << " ---" << endl;
        cout << "Límites: izq = " << izq << ", der = " << der << endl;
        cout << "Índice medio: medio = (" << izq << " + " << der << ") / 2 = " << medio <<
endl;
        cout << "Valor en v[" << medio << "] = " << v[medio] << endl;

        // Elemento encontrado
        if (v[medio] == nbus) {
            cout << "\n¡Elemento encontrado!" << endl;
```

```
            cout << "v[" << medio << "] == " << nbus << endl;
            return medio;
        }
        // El elemento está en la mitad derecha
        else if (v[medio] < nbus) {
            cout << "Decisión: " << v[medio] << " < " << nbus << endl;
            cout << "El número está en la mitad derecha" << endl;
            cout << "Ajustando: izq = medio + 1 = " << medio << " + 1 = " << (medio + 1) <<
endl;
            izq = medio + 1;
        }
        // El elemento está en la mitad izquierda
        else {
            cout << "Decisión: " << v[medio] << " > " << nbus << endl;
            cout << "El número está en la mitad izquierda" << endl;
            cout << "Ajustando: der = medio - 1 = " << medio << " - 1 = " << (medio - 1) <<
endl;
            der = medio - 1;
        }

        iteracion++;
    }

    cout << "\n¡Elemento no encontrado en el arreglo!" << endl;
    return -1;
}

int main() {
    int n, nbus;

    cout << "=== BÚSQUEDA BINARIA ===" << endl;
    cout << "Ingrese el tamaño del arreglo: ";
    cin >> n;

    int v[n];

    cout << "\nIngrese " << n << " números ORDENADOS de forma ASCENDENTE:" <<
endl;
    for (int i = 0; i < n; i++) {
        cout << "v[" << i << "] = ";
        cin >> v[i];
```

```
    }

    cout << "\nArreglo ingresado: { ";
    for (int i = 0; i < n; i++) {
        cout << v[i];
        if (i < n - 1) cout << ", ";
    }
    cout << " }" << endl;

    cout << "\nIngrese el número a buscar: ";
    cin >> nbus;

    int resultado = busquedaBinaria(v, n, nbus);

    cout << "\n=== RESULTADO FINAL ===" << endl;
    if (resultado != -1) {
        cout << "El número " << nbus << " se encuentra en la posición: " << resultado <<
endl;
    } else {
        cout << "El número " << nbus << " NO se encuentra en el arreglo." << endl;
    }

    return 0;
}
```

Datos iniciales:

- N = 11
- Arreglo = {1, 2, 4, 5, 7, 9, 11, 16, 21, 25, 34}
- Posiciones: 0 1 2 3 4 5 6 7 8 9 10
- Número a buscar = 7

Nota: Este ejemplo usa índices desde 0 (como en el primer código que te mostré)

ITERACIÓN 1:
Límites actuales:

- izq = 0
- der = 10

Cálculo del medio:
medio = (izq + der) / 2
medio = (0 + 10) / 2
medio = 10 / 2

medio = 5
Comparación:
- v[5] = 9
- nbus = 7

Decisión:
- ¿9 == 7? → NO
- ¿9 > 7? → SÍ
- El valor 7 está en la mitad IZQUIERDA
- Nuevo rango: izq = 0, der = 4

ITERACIÓN 2:
Límites actuales:
- izq = 0
- der = 4

Cálculo del medio:
medio = (izq + der) / 2
medio = (0 + 4) / 2
medio = 4 / 2
medio = 2
Comparación:
- v[2] = 4
- nbus = 7

Decisión:
- ¿4 == 7? → NO
- ¿4 < 7? → SÍ
- El valor 7 está en la mitad DERECHA
- Nuevo rango: izq = 3, der = 4

ITERACIÓN 3:
Límites actuales:
- izq = 3
- der = 4

Cálculo del medio:
medio = (izq + der) / 2
medio = (3 + 4) / 2
medio = 7 / 2
medio = 3 (división entera)
Comparación:
- v[3] = 5
- nbus = 7

Decisión:

- ¿5 == 7? → NO
- ¿5 < 7? → SÍ
- El valor 7 está en la mitad DERECHA
- Nuevo rango: izq = 4, der = 4

ITERACIÓN 4:
Límites actuales:

- izq = 4
- der = 4

Cálculo del medio:
medio = (izq + der) / 2
medio = (4 + 4) / 2
medio = 8 / 2
medio = 4
Comparación:

- v[4] = 7
- nbus = 7

Decisión:

- ¿7 == 7? → ¡SÍ!
- ¡ENCONTRADO!

Resultado final:
El número 7 se encuentra en la posición 4

Resumen visual del proceso:
Iteración 1: [1, 2, 4, 5, 7, | 9, 11, 16, 21, 25, 34]
izq=0 der=10, medio=5 → 9>7, ir a izquierda

Iteración 2: [1, 2, | 4, 5, 7]
izq=0 der=4, medio=2 → 4<7, ir a derecha

Iteración 3: [5, | 7]
izq=3 der=4, medio=3 → 5<7, ir a derecha

Iteración 4: [7]
izq=4 der=4, medio=4 → 7==7, ¡ENCONTRADO!
El algoritmo necesitó 4 iteraciones para encontrar el número 7 en la posición 4.

EJERCICIO 2 – FUNDAMENTOS DE PROGRAMACIÓN ALGORITMOS, ESTRUCTURA DE DATOS Y OBJETOS

Una clase de estudiantes ha registrado sus estaturas en un vector denominado ESTATURAS. Se sabe que los datos en este vector ya han sido ordenados de menor a mayor estatura.

Se te pide realizar las siguientes tareas algorítmicas utilizando estas estaturas:

1. Implementar el algoritmo de búsqueda binaria para encontrar rápidamente la posición (índice) de una estatura específica (E_objetivo) que se introduce por teclado.
2. Si la estatura se encuentra en el algoritmo debe escribir la posición (índice) donde se encuentra.
3. Si la estatura no se encuentra en el algoritmo debe escribir un mensaje indicando que no se encontró la estatura.

Consideraciones:

- Asume que el vector ESTATURAS ya está cargado con N valores reales y ordenado.
- El límite inferior del es 1 y el límite superior es N.
- La búsqueda binaria es apropiada porque está ordenado.

```
#include <iostream>
using namespace std;

int busquedaBinaria(double estaturas[], int N, double E_objetivo) {
    int inferior = 1;
    int superior = N;

    cout << "\nBuscando estatura: " << E_objetivo << endl;

    while (inferior <= superior) {
```

```
        int medio = (inferior + superior) / 2;

        cout << "\nIteracion:" << endl;
        cout << "inferior = " << inferior << ", superior = " << superior << endl;
        cout << "medio = (" << inferior << " + " << superior << ") / 2 = " << medio <<
endl;
        cout << "ESTATURAS[" << medio << "] = " << estaturas[medio] << endl;

        if (estaturas[medio] == E_objetivo) {
            cout << "\nEstatura encontrada en posicion: " << medio << endl;
            return medio;
        }
        else if (estaturas[medio] < E_objetivo) {
            cout << estaturas[medio] << " < " << E_objetivo << " -> Buscar en mitad
superior" << endl;
            inferior = medio + 1;
        }
        else {
            cout << estaturas[medio] << " > " << E_objetivo << " -> Buscar en mitad
inferior" << endl;
            superior = medio - 1;
        }
    }

    cout << "\nEstatura NO encontrada" << endl;
    return -1;
}

int main() {
    int N;
    double E_objetivo;

    cout << "Ingrese cantidad de estudiantes: ";
    cin >> N;

    double estaturas[N + 1];

    cout << "\nIngrese estaturas ordenadas de menor a mayor:\n";
    for (int i = 1; i <= N; i++) {
        cout << "ESTATURAS[" << i << "] = ";
        cin >> estaturas[i];
```

```
    }

    cout << "\nIngrese estatura a buscar: ";
    cin >> E_objetivo;

    int resultado = busquedaBinaria(estaturas, N, E_objetivo);

    cout << "\n--- RESULTADO ---" << endl;
    if (resultado != -1) {
        cout << "Posicion: " << resultado << endl;
    } else {
        cout << "Estatura no encontrada en el vector" << endl;
    }

    return 0;
}
```

Datos iniciales:

- N = 7
- ESTATURAS = {1.55, 1.62, 1.68, 1.72, 1.78, 1.83, 1.90}
- Posiciones: 1 2 3 4 5 6 7
- E_objetivo = 1.72

ITERACIÓN 1:
Límites actuales:

- inferior = 1
- superior = 7

Cálculo del medio:
medio = (inferior + superior) / 2
medio = (1 + 7) / 2
medio = 8 / 2
medio = 4
Comparación:

- ESTATURAS[4] = 1.72
- E_objetivo = 1.72

Decisión:

- ¿1.72 == 1.72? → SÍ
- ¡ENCONTRADO!

Resultado: La estatura 1.72 está en la posición 4

EJERCICIO 3 – METODOLOGIA DE LA PROGRAMACION ALGORITMOS, DIAGRAMAS DE FLUJO Y PROGRAMAS

La búsqueda binaria consiste en dividir el intervalo de búsqueda en dos partes y comparar el elemento buscado con el elemento central del arreglo. En caso de ser diferentes, se deben redefinir los extremos del intervalo considerando si el elemento buscado es mayor o menor que el elemento que se encuentra en la posición central. De esta forma, el proceso de búsqueda se repite hasta que el elemento es encontrado o bien cuando el espacio de búsqueda se anula. Esto último ocurre cuando el elemento buscado no se encuentra en el arreglo. Es importante destacar que el método funciona únicamente con arreglos ordenados. En cada paso del método el espacio de búsqueda se reduce a la mitad, por lo que el número de comparaciones a realizar disminuye considerablemente. Esta disminución en el número de comparaciones será más notoria cuando más grande sea el tamaño del arreglo.

Datos: [1..N], X $1 \leq N \leq 50$Donde:

VECTOR: es un arreglo unidimensional de tipo entero.

X: es una variable de tipo entero que representa el dato que se va a buscar.

Explicación de las variables

N: Variable de tipo entero. Indica el total de componentes que tendrá el arreglo unidimensional.

I: Variable de tipo entero. Se utiliza como índice del arreglo y como variable de control de diferentes ciclos.

VECTOR: Arreglo unidimensional de tipo entero.

X: Variable de tipo entero. Representa el dato que se va a buscar.

IZQ: Variable de tipo entero. Se utiliza para almacenar el extremo izquierdo del intervalo.

DER: Variable de tipo entero. Se utiliza para almacenar el extremo derecho del intervalo.

CEN: Variable de tipo entero. Se utiliza para almacenar el centro del intervalo.

BAN: Variable de tipo entero. Se utiliza para interrumpir el ciclo si se localiza el elemento buscado.

```
#include <iostream>
using namespace std;

int main() {
    int N, X;
    int IZQ, DER, CEN;
    int BAN = 0;
    int iteracion = 1;

    cout << "=== BUSQUEDA BINARIA ===" << endl;
    cout << "Ingrese la cantidad de elementos (N): ";
    cin >> N;

    int VECTOR[N];

    cout << "\nIngrese " << N << " numeros ORDENADOS de menor a mayor:\n";
    for (int i = 0; i < N; i++) {
        cout << "VECTOR[" << i << "] = ";
        cin >> VECTOR[i];
    }

    cout << "\nVECTOR ingresado: { ";
    for (int i = 0; i < N; i++) {
        cout << VECTOR[i];
        if (i < N - 1) cout << ", ";
    }
    cout << " }" << endl;
```

```
    cout << "\nIngrese el numero a buscar (X): ";
    cin >> X;

    // Inicializar límites
    IZQ = 0;
    DER = N - 1;

    cout << "\n--- PROCESO DE BUSQUEDA ---" << endl;
    cout << "Buscando X = " << X << endl;
    cout << "Limites iniciales: IZQ = " << IZQ << ", DER = " << DER << endl;

    // Búsqueda binaria
    while (IZQ <= DER && BAN == 0) {
        cout << "\n** Iteracion " << iteracion << " **" << endl;

        // Calcular centro
        CEN = (IZQ + DER) / 2;

        cout << "CEN = (IZQ + DER) / 2" << endl;
        cout << "CEN = (" << IZQ << " + " << DER << ") / 2" << endl;
        cout << "CEN = " << (IZQ + DER) << " / 2" << endl;
        cout << "CEN = " << CEN << endl;

        cout << "VECTOR[" << CEN << "] = " << VECTOR[CEN] << endl;

        // Comparar
        if (VECTOR[CEN] == X) {
            cout << "Comparacion: " << VECTOR[CEN] << " == " << X << " ->
VERDADERO" << endl;
            cout << "Elemento encontrado!" << endl;
            BAN = 1;
        }
        else if (X < VECTOR[CEN]) {
            cout << "Comparacion: " << X << " < " << VECTOR[CEN] << " ->
VERDADERO" << endl;
            cout << "Buscar en mitad IZQUIERDA" << endl;
            cout << "DER = CEN - 1 = " << CEN << " - 1 = " << (CEN - 1) << endl;
            DER = CEN - 1;
        }
        else {
```

```
            cout << "Comparacion: " << X << " > " << VECTOR[CEN] << " ->
VERDADERO" << endl;
            cout << "Buscar en mitad DERECHA" << endl;
            cout << "IZQ = CEN + 1 = " << CEN << " + 1 = " << (CEN + 1) << endl;
            IZQ = CEN + 1;
        }

        if (BAN == 0) {
            cout << "Nuevos limites: IZQ = " << IZQ << ", DER = " << DER << endl;
        }

        iteracion++;
    }

    // Resultado
    cout << "\n=== RESULTADO ===" << endl;
    if (BAN == 1) {
        cout << "El numero " << X << " se encuentra en la posicion: " << CEN << endl;
        cout << "BAN = " << BAN << " (elemento encontrado)" << endl;
    }
    else {
        cout << "El numero " << X << " NO se encuentra en el vector." << endl;
        cout << "BAN = " << BAN << " (elemento no encontrado)" << endl;
        cout << "Condicion de salida: IZQ > DER (" << IZQ << " > " << DER << ")" <<
endl;
    }

    return 0;
}
```

Buscar X = 2

Datos:

- N = 11
- N = {1, 2, 4, 5, 7, 9, 11, 16, 21, 25, 34}
- Posiciones: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- X = 2

ITERACIÓN 1:

Límites: IZQ = 0, DER = 10

Cálculo del centro:

CEN = (IZQ + DER) / 2
CEN = (0 + 10) / 2 = 5
Comparación:

- 5 = 9
- X = 2
- ¿2 < 9? → SÍ → Buscar en mitad IZQUIERDA

Ajuste: DER = 5 - 1 = 4
Nuevos límites: IZQ = 0, DER = 4

ITERACIÓN 2:
Límites: IZQ = 0, DER = 4
Cálculo del centro:
CEN = (0 + 4) / 2 = 2
Comparación:

- 2 = 4
- X = 2
- ¿2 < 4? → SÍ → Buscar en mitad IZQUIERDA

Ajuste: DER = 2 - 1 = 1
Nuevos límites: IZQ = 0, DER = 1

ITERACIÓN 3:
Límites: IZQ = 0, DER = 1
Cálculo del centro:
CEN = (0 + 1) / 2 = 0
Comparación:

- 0 = 1
- X = 2
- ¿2 > 1? → SÍ → Buscar en mitad DERECHA

Ajuste: IZQ = 0 + 1 = 1
Nuevos límites: IZQ = 1, DER = 1

ITERACIÓN 4:
Límites: IZQ = 1, DER = 1
Cálculo del centro:
CEN = (1 + 1) / 2 = 1
Comparación:

- 1 = 2
- X = 2
- ¿2 == 2? → ¡SÍ! → BAN = 1

RESULTADO: El número 2 se encuentra en la posición 1 (4 iteraciones)

Diferencias clave:

1. X = 2: Está al inicio → 4 iteraciones buscando hacia la izquierda

EJERCICIO 4 – LIBRO ALGORITMOS COMPUTACIONESLES INTRODUCCIÓN AL ANÁLISIS Y DISEÑO

Sea S un conjunto de m enteros. Sea E un arreglo de n enteros distintos ($n \leq m$). Sea K un elemento escogido al azar de S. En promedio, ¿cuántas comparaciones efectuará Búsqueda Binaria (algoritmo 1.4) con E, 0, n − 1, y K como entrada? Exprese su respuesta en función de n y m.

Sea:

- S: conjunto de m enteros
- E: arreglo ordenado de n enteros distintos ($n \leq m$)
- K: elemento aleatorio de S

Casos posibles:

1. K está en E: Probabilidad = n/m
    - Búsqueda exitosa
    - Comparaciones promedio = $\log_2(n) + 1$
2. K no está en E: Probabilidad = (m-n)/m
    - Búsqueda fallida
    - Comparaciones promedio = $\log_2(n) + 1$

Fórmula del promedio total:

Comparaciones promedio = $\log_2(n) + 1$

Aunque parezca contraintuitivo, tanto búsquedas exitosas como fallidas en búsqueda binaria realizan aproximadamente el mismo número de comparaciones: $\lfloor \log_2(n) \rfloor + 1$

Por lo tanto, el número promedio de comparaciones es:

$C(n,m) = \lfloor \log_2(n) \rfloor + 1$

Este resultado es independiente de m, ya que la búsqueda binaria siempre recorre la misma altura del árbol binario de búsqueda, sin importar si el elemento se encuentra o no.

```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>
using namespace std;

// Búsqueda binaria que cuenta comparaciones
int busquedaBinaria(int E[], int inicio, int fin, int K, int &comparaciones) {
    comparaciones = 0;

    while (inicio <= fin) {
        comparaciones++;
        int medio = (inicio + fin) / 2;

        if (E[medio] == K) {
            return medio;  // Encontrado
        }

        if (E[medio] < K) {
            inicio = medio + 1;
        } else {
            fin = medio - 1;
        }
    }

    return -1;  // No encontrado
```

```
}

int main() {
    srand(time(0));

    // Parámetros
    int n, m;
    cout << "Ingrese n (tamaño del arreglo E): ";
    cin >> n;
    cout << "Ingrese m (tamaño del conjunto S): ";
    cin >> m;

    if (n > m) {
        cout << "Error: n debe ser <= m" << endl;
        return 1;
    }

    // Crear arreglo E con n elementos ordenados
    int *E = new int[n];
    for (int i = 0; i < n; i++) {
        E[i] = i * 2;  // Elementos pares: 0, 2, 4, 6, ...
    }

    // Crear conjunto S con m elementos
    int *S = new int[m];
    for (int i = 0; i < n; i++) {
        S[i] = E[i];  // Primeros n elementos son los de E
    }
    for (int i = n; i < m; i++) {
        S[i] = i * 2 + 1;  // Resto son impares: no están en E
    }

    // Realizar experimento: 10000 búsquedas aleatorias
    int numPruebas = 10000;
    int totalComparaciones = 0;
    int comparaciones;

    for (int i = 0; i < numPruebas; i++) {
        int K = S[rand() % m];  // Elegir K al azar de S
        busquedaBinaria(E, 0, n - 1, K, comparaciones);
        totalComparaciones += comparaciones;
```

```
    }

    double promedioExperimental = (double)totalComparaciones / numPruebas;
    double promedioTeorico = floor(log2(n)) + 1;

    // Resultados
    cout << "\n=== RESULTADOS ===" << endl;
    cout << "n = " << n << ", m = " << m << endl;
    cout << "\nPromedio experimental de comparaciones: " << promedioExperimental <<
endl;
    cout << "Promedio teórico (⌊log₂(n)⌋ + 1): " << promedioTeorico << endl;
    cout << "\nFórmula: C(n,m) = ⌊log₂(" << n << ")⌋ + 1 = " << promedioTeorico << endl;

    // Liberar memoria
    delete[] E;
    delete[] S;

    return 0;
}
```

Generación de datos:
Arreglo E: primeros n números pares ordenados
Conjunto S: contiene los n elementos de E más (m-n) elementos adicionales
Experimento: Realiza 10,000 búsquedas con elementos K elegidos aleatoriamente de S
Resultados: Compara el promedio experimental con el teórico
Ejemplo de ejecución:
Para n=100, m=1000:
Promedio teórico = ⌊log₂(100)⌋ + 1 = 6 + 1 = 7 comparaciones
El promedio experimental será muy cercano a 7
Nota importante: El resultado es prácticamente independiente de m, solo depende de n.

EJERCICIO 5 HASH:

Detalle los procedimientos de direccionamiento abierto para buscar, insertar y borrar claves en una tabla de dispersión. ¿En qué difieren las condiciones en las que los ciclos terminan en cada uno de estos procedimientos? Tome en cuenta la posibilidad de que haya celdas "obsoletas".

```
#include <iostream>
using namespace std;

const int TAM = 10;
const int VACIO = -1;
const int BORRADO = -2;

// Variables globales simples
int tabla[TAM];

// Función de dispersión (hash)
int funcionHash(int clave) {
    return clave % TAM;
}

// Inicializar tabla
void inicializar() {
    for(int i = 0; i < TAM; i++) {
        tabla[i] = VACIO;
    }
}

// Mostrar tabla
void mostrar() {
    cout << "\n=== TABLA DE DISPERSION ===\n";
    for(int i = 0; i < TAM; i++) {
        cout << "Posicion [" << i << "]: ";
        if(tabla[i] == VACIO)
            cout << "VACIO";
        else if(tabla[i] == BORRADO)
            cout << "BORRADO";
        else
            cout << tabla[i];
        cout << endl;
    }
    cout << "============================\n";
}

// INSERTAR con sondeo lineal
bool insertar(int clave) {
    int pos = funcionHash(clave);
```

```
    int posInicial = pos;
    int intentos = 0;

    cout << "\n--- INSERTANDO " << clave << " ---\n";
    cout << "Hash inicial: " << pos << endl;

    // Buscar posición vacía o borrada
    while(intentos < TAM) {
        cout << "Intento " << (intentos + 1) << ": Posicion " << pos;

        if(tabla[pos] == VACIO || tabla[pos] == BORRADO) {
            tabla[pos] = clave;
            cout << " -> INSERTADO\n";
            return true;
        }

        if(tabla[pos] == clave) {
            cout << " -> YA EXISTE\n";
            return false;
        }

        cout << " -> OCUPADA (valor: " << tabla[pos] << ")\n";
        pos = (posInicial + intentos + 1) % TAM;
        intentos++;
    }

    cout << "TABLA LLENA - NO SE PUDO INSERTAR\n";
    return false;
}

// BUSCAR con sondeo lineal
int buscar(int clave) {
    int pos = funcionHash(clave);
    int posInicial = pos;
    int intentos = 0;

    cout << "\n--- BUSCANDO " << clave << " ---\n";
    cout << "Hash inicial: " << pos << endl;

    while(intentos < TAM) {
        cout << "Intento " << (intentos + 1) << ": Posicion " << pos;
```

```
        // Si encontramos la clave
        if(tabla[pos] == clave) {
            cout << " -> ENCONTRADO\n";
            return pos;
        }

        // Si encontramos celda vacía, la clave no existe
        if(tabla[pos] == VACIO) {
            cout << " -> VACIO (clave no existe)\n";
            return -1;
        }

        // Si está borrada o tiene otra clave, seguir buscando
        cout << " -> ";
        if(tabla[pos] == BORRADO)
            cout << "BORRADO";
        else
            cout << "Otro valor (" << tabla[pos] << ")";
        cout << " (continuar)\n";

        pos = (posInicial + intentos + 1) % TAM;
        intentos++;
    }

    cout << "NO ENCONTRADO (tabla recorrida completa)\n";
    return -1;
}

// BORRAR con sondeo lineal
bool borrar(int clave) {
    int pos = funcionHash(clave);
    int posInicial = pos;
    int intentos = 0;

    cout << "\n--- BORRANDO " << clave << " ---\n";
    cout << "Hash inicial: " << pos << endl;

    while(intentos < TAM) {
        cout << "Intento " << (intentos + 1) << ": Posicion " << pos;
```

```
        // Si encontramos la clave
        if(tabla[pos] == clave) {
            tabla[pos] = BORRADO;
            cout << " -> BORRADO\n";
            return true;
        }

        // Si encontramos celda vacía, la clave no existe
        if(tabla[pos] == VACIO) {
            cout << " -> VACIO (clave no existe)\n";
            return false;
        }

        // Si está borrada o tiene otra clave, seguir buscando
        cout << " -> ";
        if(tabla[pos] == BORRADO)
            cout << "BORRADO";
        else
            cout << "Otro valor (" << tabla[pos] << ")";
        cout << " (continuar)\n";

        pos = (posInicial + intentos + 1) % TAM;
        intentos++;
    }

    cout << "NO ENCONTRADO (tabla recorrida completa)\n";
    return false;
}

int main() {
    inicializar();
    int opcion, clave;

    do {
        cout << "\n========== MENU ==========\n";
        cout << "1. Insertar clave\n";
        cout << "2. Buscar clave\n";
        cout << "3. Borrar clave\n";
        cout << "4. Mostrar tabla\n";
        cout << "0. Salir\n";
        cout << "Opcion: ";
```

```
        cin >> opcion;

        switch(opcion) {
            case 1:
                cout << "Ingrese clave a insertar: ";
                cin >> clave;
                insertar(clave);
                mostrar();
                break;
            case 2:
                cout << "Ingrese clave a buscar: ";
                cin >> clave;
                buscar(clave);
                break;
            case 3:
                cout << "Ingrese clave a borrar: ";
                cin >> clave;
                borrar(clave);
                mostrar();
                break;
            case 4:
                mostrar();
                break;
            case 0:
                cout << "Saliendo...\n";
                break;
            default:
                cout << "Opcion invalida\n";
        }
    } while(opcion != 0);

    return 0;
}
```

Fórmula: posición = número ÷ 10, tomar el RESIDUO
INSERTAR 25
Cálculo:
25 ÷ 10 = 2 y sobran 5
Posición = 5
Resultado:

Tabla antes: [_] [_] [_] [_] [_] [_] [_] [_] [_] [_] Tabla después: [_] [_] [_] [_] [_] [25] [_] [_] [_] [_] ↑ posición 5
BUSCAR 25 Cálculo: 25 ÷ 10 = sobra 5 Empezar en posición 5 Proceso: Revisar posición 5: tabla[5] = 25 ✓ ¡ENCONTRADO!
BORRAR 25 Cálculo: 25 ÷ 10 = sobra 5 Empezar en posición 5 Proceso: Revisar posición 5: tabla[5] = 25 → Marcar como BORRADO Resultado: Tabla después: [_] [_] [_] [_] [_] [BORRADO] [_] [_] [_] [_] ↑ posición 5
Ahora BUSCAR 25 otra vez (después de borrado) Cálculo: 25 ÷ 10 = sobra 5 Empezar en posición 5 Proceso: Revisar posición 5: tabla[5] = BORRADO (no es 25) Revisar posición 6: tabla[6] = VACÍO → PARAR, NO EXISTE ✗
RESUMEN DEL CÁLCULO Solo necesitas hacer: número ÷ 10 = tomar el RESIDUO Ejemplo: 25 ÷ 10 = residuo 5 → va en posición 5

EJERCICIO 6 HASH:

El tipo de tabla de dispersión H bajo direccionamiento cerrado es un arreglo de referencias a listas, y bajo direccionamiento abierto es un arreglo de claves. Suponga que una clave

requiere una "palabra" de memoria y un nodo de lista ligada requiere dos palabras, una para la clave y una para una referencia a una lista. Considere cada uno de estos factores de carga con direccionamiento cerrado: 0.25, 0.5, 1.0, 2.0. Sea hc el número de celdas de la tabla de dispersión con direccionamiento cerrado.

- a. Estime el espacio total requerido, incluido espacio para listas, con direccionamiento cerrado. Luego, suponiendo que se usa la misma cantidad de espacio para una tabla de dispersión con direccionamiento abierto, determine los factores de carga correspondientes si se usa direccionamiento abierto.
- b. Ahora suponga que una clave ocupa cuatro palabras y que un nodo de lista ocupa cinco palabras (cuatro para la clave y una para la referencia al resto de la lista), y repita la parte (a).

FÓRMULAS SIMPLES:

- $n = \alpha \times hc$
- CERRADO = hc + (n × palabras_por_nodo)
- ABIERTO_capacidad = CERRADO / palabras_por_clave
- α_abierto = n / ABIERTO_capacidad

```
#include <iostream>
using namespace std;

int main() {
    int hc; // Tamaño de la tabla
    float factorCarga;
    int tipoClave;

    cout << "=== CALCULADORA DE ESPACIO ===\n\n";

    // Entrada de datos
```

```
    cout << "Tamanio de la tabla (hc): ";
    cin >> hc;

    cout << "Factor de carga (0.25, 0.5, 1.0, 2.0): ";
    cin >> factorCarga;

    cout << "Tipo de clave:\n";
    cout << "1 = Clave de 1 palabra (nodo=2 palabras)\n";
    cout << "2 = Clave de 4 palabras (nodo=5 palabras)\n";
    cout << "Opcion: ";
    cin >> tipoClave;

    // Calcular número de claves
    int n = factorCarga * hc;

    cout << "\n========== RESULTADOS ==========\n";
    cout << "Numero de claves (n): " << n << "\n\n";

    // DIRECCIONAMIENTO CERRADO
    cout << "--- DIRECCIONAMIENTO CERRADO ---\n";

    int espacioTabla = hc;
    int espacioListas;
    int totalCerrado;

    if(tipoClave == 1) {
        espacioListas = n * 2; // cada nodo = 2 palabras
        cout << "Espacio tabla: " << hc << " palabras\n";
        cout << "Espacio listas: " << n << " nodos x 2 = " << espacioListas << "
palabras\n";
    } else {
        espacioListas = n * 5; // cada nodo = 5 palabras
        cout << "Espacio tabla: " << hc << " palabras\n";
        cout << "Espacio listas: " << n << " nodos x 5 = " << espacioListas << "
palabras\n";
    }

    totalCerrado = espacioTabla + espacioListas;
    cout << "TOTAL CERRADO: " << totalCerrado << " palabras\n";

    // DIRECCIONAMIENTO ABIERTO
```

```
    cout << "\n--- DIRECCIONAMIENTO ABIERTO ---\n";
    cout << "Usando el mismo espacio: " << totalCerrado << " palabras\n";

    int capacidadAbierto;
    float factorCargaAbierto;

    if(tipoClave == 1) {
        capacidadAbierto = totalCerrado; // 1 palabra por clave
        cout << "Capacidad: " << totalCerrado << " / 1 = " << capacidadAbierto << "
claves\n";
    } else {
        capacidadAbierto = totalCerrado / 4; // 4 palabras por clave
        cout << "Capacidad: " << totalCerrado << " / 4 = " << capacidadAbierto << "
claves\n";
    }

    factorCargaAbierto = (float)n / capacidadAbierto;
    cout << "Factor de carga: " << n << " / " << capacidadAbierto << " = " <<
factorCargaAbierto << "\n";

    cout << "\n========== RESUMEN ==========\n";
    cout << "Factor cerrado: " << factorCarga << " -> Factor abierto: " <<
factorCargaAbierto << "\n";

    return 0;
}
```

DATOS:

- Tabla de tamaño: hc = 10
- Factor de carga: α = 1.0
- Tipo: Clave de 1 palabra

PASO 1: ¿Cuántas claves tengo?
n = α × hc
n = 1.0 × 10
n = 10 claves

PASO 2: DIRECCIONAMIENTO CERRADO (con listas)
¿Cuánto espacio uso?
Espacio tabla = 10 palabras (solo referencias)

Espacio listas = 10 claves × 2 palabras/nodo = 20 palabras

TOTAL = 10 + 20 = 30 palabras
Dibujo:
TABLA (10 palabras) LISTAS (20 palabras)
[ref] ----------> [clave|ref] [clave|ref]
[ref] ----------> [clave|ref]
[ref] ----------> [clave|ref] [clave|ref] [clave|ref]
...

PASO 3: DIRECCIONAMIENTO ABIERTO con mismo espacio
Tengo 30 palabras para usar:
Cada clave usa 1 palabra
Capacidad = 30 / 1 = 30 claves

Pero solo tengo 10 claves
Factor de carga = 10 / 30 = 0.33
Dibujo:
TABLA (30 palabras, todas para claves)
[clave] [clave] [vacío] [clave] [vacío] [vacío] ...

RESULTADO:
CERRADO: Factor = 1.0 (10 claves en tabla de 10)
ABIERTO: Factor = 0.33 (10 claves en tabla de 30)
Conclusión: Con el mismo espacio, ABIERTO tiene tabla más grande y factor más bajo.

AHORA CON CLAVE DE 4 PALABRAS:
DATOS:

- Tabla: hc = 10
- Factor: α = 1.0
- Tipo: Clave de 4 palabras

PASO 1: Número de claves
n = 1.0 × 10 = 10 claves

PASO 2: CERRADO
Espacio tabla = 10 palabras
Espacio listas = 10 claves × 5 palabras/nodo = 50 palabras

TOTAL = 10 + 50 = 60 palabras

PASO 3: ABIERTO con 60 palabras
Cada clave usa 4 palabras
Capacidad = 60 / 4 = 15 claves
Factor = 10 / 15 = 0.67

RESULTADO:
CERRADO: Factor = 1.0 (10 claves)
ABIERTO: Factor = 0.67 (10 claves en tabla de 15)
TABLA FINAL (Todos los casos):
PARTE A (Clave = 1 palabra):

Factor Cerrado	Claves	Espacio Total	Factor Abierto
0.25	3	16	0.19
0.5	5	20	0.25
1.0	10	30	0.33
2.0	20	50	0.40

PARTE B (Clave = 4 palabras):

Factor Cerrado	Claves	Espacio Total	Factor Abierto
0.25	3	25	0.50
0.5	5	35	0.57
1.0	10	60	0.67
2.0	20	110	0.73

EJEMPLO 7 – LIBRO ALGORITMOS Y ESTRUCTURAS DE DATOS EN PYTHON

Se cuenta con una lista de Pokémons, de cada uno de estos se sabe su nombre, número y tipo (solo considerar uno el principal), con los cuales deberá resolver las siguientes tareas:

a. mostrar un listado de los Pokémons ordenados por números usando el método de ordenamiento por conteo;

b. realizar un listado ordenado por nombre utilizando el método de ordenamiento rápido;

c. mostrar toda la información de pokémon número 640;

d. listar todos los Pokémons que comienzan con la letra T;

e. determinar si existe el Pokémon Cobalion y mostrar toda su información;

f. realizar un listado de todos los Pokémon de tipo eléctrico y calcular cuántos son.

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    // Declarar cantidad de pokemones
    int n;
    cout << "Cuantos Pokemones vas a ingresar? ";
    cin >> n;

    // Declarar arrays para guardar datos
    string nombres[100];
    int numeros[100];
    string tipos[100];

    // PASO 1: Ingresar datos
    cout << "\n--- INGRESA LOS DATOS ---\n";
    for(int i = 0; i < n; i++) {
        cout << "\nPokemon " << (i+1) << ":\n";
        cout << "Nombre: ";
        cin >> nombres[i];
        cout << "Numero: ";
        cin >> numeros[i];
        cout << "Tipo: ";
        cin >> tipos[i];
    }

    // PASO 2: Encontrar el numero mas grande
    int maximo = numeros[0];
    for(int i = 1; i < n; i++) {
        if(numeros[i] > maximo) {
            maximo = numeros[i];
        }
    }

    cout << "\n--- ORDENAMIENTO POR CONTEO ---\n";
```

```
    cout << "Numero maximo encontrado: " << maximo << "\n\n";

    // PASO 3: Crear array de conteo
    int conteo[1000];
    // Inicializar todo en 0
    for(int i = 0; i <= maximo; i++) {
        conteo[i] = 0;
    }

    // PASO 4: Contar cuantas veces aparece cada numero
    cout << "Contando apariciones...\n";
    for(int i = 0; i < n; i++) {
        conteo[numeros[i]]++;
        cout << "Pokemon " << nombres[i] << " (num " << numeros[i] << ") -> conteo[" <<
numeros[i] << "]++\n";
    }

    // PASO 5: Acumular conteos (suma acumulada)
    cout << "\nAcumulando conteos...\n";
    for(int i = 1; i <= maximo; i++) {
        conteo[i] = conteo[i] + conteo[i-1];
    }

    // PASO 6: Crear arrays ordenados
    string nombresOrdenados[100];
    int numerosOrdenados[100];
    string tiposOrdenados[100];

    // PASO 7: Colocar cada pokemon en su posicion correcta
    cout << "\nColocando en posiciones correctas...\n";
    for(int i = n-1; i >= 0; i--) {
        int numero = numeros[i];
        int posicion = conteo[numero] - 1;

        cout << nombres[i] << " (num " << numero << ") va en posicion " << posicion <<
"\n";

        numerosOrdenados[posicion] = numeros[i];
        nombresOrdenados[posicion] = nombres[i];
        tiposOrdenados[posicion] = tipos[i];
```

```
        conteo[numero]--;
    }

    // PASO 8: Mostrar resultado ordenado
    cout << "\n=== POKEMONES ORDENADOS POR NUMERO ===\n";
    for(int i = 0; i < n; i++) {
        cout << numerosOrdenados[i] << " - " << nombresOrdenados[i] << " (" <<
tiposOrdenados[i] << ")\n";
    }

    return 0;
}
```

DATOS DE ENTRADA:
Pokemon 1: Pikachu, Numero: 25, Tipo: Electrico
Pokemon 2: Charmander, Numero: 4, Tipo: Fuego
Pokemon 3: Bulbasaur, Numero: 1, Tipo: Planta
Pokemon 4: Squirtle, Numero: 7, Tipo: Agua
Pokemon 5: Eevee, Numero: 4, Tipo: Normal

CÁLCULO MANUAL PASO A PASO
PASO 1: Datos originales
Posición: 0 1 2 3 4
Nombre: Pikachu Charmander Bulbasaur Squirtle Eevee
Numero: 25 4 1 7 4
Tipo: Electrico Fuego Planta Agua Normal

PASO 2: Encontrar el máximo
Comparar todos los números: 25, 4, 1, 7, 4
Máximo = 25

PASO 3: Crear array de conteo (tamaño = 26)
Posición: 0 1 2 3 4 5 6 7 8...25
Conteo: 0 0 0 0 0 0 0 0 0...0

PASO 4: Contar apariciones
Recorrer cada pokemon y contar:

- Pikachu (25) → conteo[25]++
- Charmander (4) → conteo[4]++

- Bulbasaur (1) → conteo[1]++
- Squirtle (7) → conteo[7]++
- Eevee (4) → conteo[4]++

Resultado:
Posición: 0 1 2 3 4 5 6 7 8...25
Conteo: 0 1 0 0 2 0 0 1 0...1
↑ ↑ ↑ ↑
Bulb Char+Eev Squir Pika

PASO 5: Acumular conteos (suma acumulada)
conteo[i] = conteo[i] + conteo[i-1]

conteo[1] = 1 + 0 = 1
conteo[2] = 0 + 1 = 1
conteo[3] = 0 + 1 = 1
conteo[4] = 2 + 1 = 3
conteo[5] = 0 + 3 = 3
conteo[6] = 0 + 3 = 3
conteo[7] = 1 + 3 = 4
...
conteo[25] = 1 + 4 = 5

Resultado:
Posición: 0 1 2 3 4 5 6 7 8...25
Conteo: 0 1 1 1 3 3 3 4 4...5
¿Qué significa esto?

- conteo[1] = 1 → hay 1 pokemon con número ≤ 1
- conteo[4] = 3 → hay 3 pokemones con número ≤ 4
- conteo[7] = 4 → hay 4 pokemones con número ≤ 7
- conteo[25] = 5 → hay 5 pokemones con número ≤ 25

PASO 6: Colocar en posiciones correctas
Procesar de DERECHA a IZQUIERDA (i=4 hasta i=0):
i=4: Eevee (numero 4)
posicion = conteo[4] - 1 = 3 - 1 = 2
Colocar Eevee en posicion 2
conteo[4]-- → conteo[4] = 2
i=3: Squirtle (numero 7)
posicion = conteo[7] - 1 = 4 - 1 = 3
Colocar Squirtle en posicion 3

conteo[7]-- → conteo[7] = 3
i=2: Bulbasaur (numero 1)
posicion = conteo[1] - 1 = 1 - 1 = 0
Colocar Bulbasaur en posicion 0
conteo[1]-- → conteo[1] = 0
i=1: Charmander (numero 4)
posicion = conteo[4] - 1 = 2 - 1 = 1
Colocar Charmander en posicion 1
conteo[4]-- → conteo[4] = 1
i=0: Pikachu (numero 25)
posicion = conteo[25] - 1 = 5 - 1 = 4
Colocar Pikachu en posicion 4
conteo[25]-- → conteo[25] = 4

RESULTADO FINAL ORDENADO:

Posición:	0	1	2	3	4
Nombre:	Bulbasaur	Charmander	Eevee	Squirtle	Pikachu
Numero:	1	4	4	7	25
Tipo:	Planta	Fuego	Normal	Agua	Electrico

VERIFICACIÓN
Orden correcto: 1 → 4 → 4 → 7 → 25
Los pokemones quedaron ordenados de menor a mayor número!

EJERCICIO 8

Un vendedor de “Fuko Pop”, tiene la lista de todos los funko que tiene disponibles en su tienda

con la siguiente información: número, nombre, colección y precio; para lo cual nos solicita le desarrollemos un algoritmo que contemple los siguiente requerimientos:

a. realizar un listado ordenado por número;

b. realizar un listado de los funko pop de una colección;

c. determinar si tiene disponible el funko pop 130 de la colección de Star Wars y mostrar toda

la información;

d. mostrar todos los funko pop disponible número 295;

e. listar todos los modelos de Darth Vader y Capitana Marvel;

f. determinar si existen funko pop de Red Skull, Thanos y Galactus, además mostrar la información de estos;

g. calcular el costo de todos los modelos de Tony Stark e Iron Man disponibles;

h. calcular el promedio de costo de todos los funko pop y el costo total de las colecciones de Rocks y Harry Potter.

- Ordenar = comparar números y cambiarlos
- Buscar = si coincide, mostrarlo
- Sumar = hacer precio1 + precio2 + precio3...
- Promedio = sumar todo ÷ cantidad

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    int cantidad;

    cout << "Cuantos Funko Pop vas a ingresar? ";
    cin >> cantidad;

    // 4 arrays simples
    int numero[100];
    string nombre[100];
    string coleccion[100];
    float precio[100];
```

```
// INGRESAR DATOS
for(int i = 0; i < cantidad; i++) {
    cout << "\n--- Funko " << (i+1) << " ---" << endl;
    cout << "Numero: ";
    cin >> numero[i];
    cin.ignore();
    cout << "Nombre: ";
    getline(cin, nombre[i]);
    cout << "Coleccion: ";
    getline(cin, coleccion[i]);
    cout << "Precio: ";
    cin >> precio[i];
}

// A) ORDENAR POR NUMERO
cout << "\n\n--- A) ORDENADOS POR NUMERO ---" << endl;
for(int i = 0; i < cantidad; i++) {
    for(int j = i+1; j < cantidad; j++) {
        if(numero[i] > numero[j]) {
            // cambiar numero
            int temp1 = numero[i];
            numero[i] = numero[j];
            numero[j] = temp1;

            // cambiar nombre
            string temp2 = nombre[i];
            nombre[i] = nombre[j];
            nombre[j] = temp2;

            // cambiar coleccion
            string temp3 = coleccion[i];
            coleccion[i] = coleccion[j];
            coleccion[j] = temp3;

            // cambiar precio
            float temp4 = precio[i];
            precio[i] = precio[j];
            precio[j] = temp4;
        }
    }
```

```
    }

    for(int i = 0; i < cantidad; i++) {
        cout << numero[i] << " - " << nombre[i] << " - " << coleccion[i] << " - $" <<
precio[i] << endl;
    }

    // B) BUSCAR POR COLECCION
    cout << "\n\n--- B) BUSCAR POR COLECCION ---" << endl;
    string buscar;
    cin.ignore();
    cout << "Que coleccion quieres buscar? ";
    getline(cin, buscar);

    for(int i = 0; i < cantidad; i++) {
        if(coleccion[i] == buscar) {
            cout << numero[i] << " - " << nombre[i] << " - $" << precio[i] << endl;
        }
    }

    // C) BUSCAR NUMERO 130 DE STAR WARS
    cout << "\n\n--- C) BUSCAR NUMERO 130 DE STAR WARS ---" << endl;
    for(int i = 0; i < cantidad; i++) {
        if(numero[i] == 130 && coleccion[i] == "Star Wars") {
            cout << "SI EXISTE!" << endl;
            cout << "Nombre: " << nombre[i] << endl;
            cout << "Precio: $" << precio[i] << endl;
        }
    }

    // D) BUSCAR TODOS LOS NUMERO 295
    cout << "\n\n--- D) TODOS LOS NUMERO 295 ---" << endl;
    for(int i = 0; i < cantidad; i++) {
        if(numero[i] == 295) {
            cout << nombre[i] << " - " << coleccion[i] << " - $" << precio[i] << endl;
        }
    }

    // E) BUSCAR DARTH VADER Y CAPITANA MARVEL
    cout << "\n\n--- E) DARTH VADER Y CAPITANA MARVEL ---" << endl;
    for(int i = 0; i < cantidad; i++) {
```

```
        if(nombre[i] == "Darth Vader" || nombre[i] == "Capitana Marvel") {
            cout << numero[i] << " - " << nombre[i] << " - " << coleccion[i] << " - $" <<
precio[i] << endl;
        }
    }

    // F) BUSCAR RED SKULL, THANOS Y GALACTUS
    cout << "\n\n--- F) RED SKULL, THANOS Y GALACTUS ---" << endl;
    for(int i = 0; i < cantidad; i++) {
        if(nombre[i] == "Red Skull" || nombre[i] == "Thanos" || nombre[i] == "Galactus") {
            cout << nombre[i] << " - " << numero[i] << " - " << coleccion[i] << " - $" <<
precio[i] << endl;
        }
    }

    // G) SUMAR PRECIOS DE TONY STARK E IRON MAN
    cout << "\n\n--- G) COSTO TONY STARK E IRON MAN ---" << endl;
    float suma = 0;
    for(int i = 0; i < cantidad; i++) {
        if(nombre[i] == "Tony Stark" || nombre[i] == "Iron Man") {
            cout << nombre[i] << " - $" << precio[i] << endl;
            suma = suma + precio[i];
        }
    }
    cout << "TOTAL: $" << suma << endl;

    // H) CALCULAR PROMEDIOS
    cout << "\n\n--- H) PROMEDIOS Y TOTALES ---" << endl;

    // Promedio de TODOS
    float todosSuma = 0;
    for(int i = 0; i < cantidad; i++) {
        todosSuma = todosSuma + precio[i];
    }
    float promedio = todosSuma / cantidad;
    cout << "Promedio de todos: $" << promedio << endl;

    // Total de ROCKS
    float rocksTotal = 0;
    for(int i = 0; i < cantidad; i++) {
        if(coleccion[i] == "Rocks") {
```

```
            rocksTotal = rocksTotal + precio[i];
        }
    }
    cout << "Total Rocks: $" << rocksTotal << endl;

    // Total de HARRY POTTER
    float hpTotal = 0;
    for(int i = 0; i < cantidad; i++) {
        if(coleccion[i] == "Harry Potter") {
            hpTotal = hpTotal + precio[i];
        }
    }
    cout << "Total Harry Potter: $" << hpTotal << endl;

    return 0;
}
```

a) ORDENAR POR NÚMERO
Simplemente comparo números:

- Si 50 > 30 → los cambio de posición
- Así quedan de menor a mayor

Ejemplo:
Antes: 50, 30, 100
Después: 30, 50, 100

b) BUSCAR POR COLECCIÓN
Veo si la colección coincide:

- Si dice "Marvel" → lo muestro
- Si no dice "Marvel" → lo ignoro

c) BUSCAR #130 DE STAR WARS
Dos condiciones:

1. ¿El número es 130?
2. ¿La colección es "Star Wars"? Si AMBAS son SÍ → lo muestro

d) BUSCAR TODOS LOS #295
Solo busco si el número es 295

**e) y f) BUSCAR POR NOMBRES
Pregunto: ¿el nombre es X o Y?

- Si SÍ → lo muestro

g) SUMAR TONY STARK E IRON MAN
CÁLCULO:
Suma = precio1 + precio2 + precio3...
Ejemplo real:
Iron Man = $15
Tony Stark = $20
Iron Man = $18

TOTAL = $53 (15+20+18)

h) PROMEDIOS Y TOTALES
PROMEDIO DE TODOS:
Promedio = (suma de todos los precios) ÷ (cantidad total)
Ejemplo:
Funko 1 = $10
Funko 2 = $20
Funko 3 = $30
Funko 4 = $40

Suma = 10+20+30+40 = 100
Promedio = 100 ÷ 4 = $25
TOTAL DE ROCKS:
Solo sumo los precios que dicen "Rocks"
Ejemplo:
Rocks #1 = $15
Rocks #2 = $25
Marvel = $30 (este NO se suma)

Total Rocks = 15+25 = $40
TOTAL DE HARRY POTTER:
Solo sumo los precios que dicen "Harry Potter"

EJERCICIO 9

A partir de una lista con todos los caballeros Jedi y los lores Sith, de los que conocemos su nombre y el de su maestro (considere solo uno), resuelva las siguientes actividades:

a. realizar un listado ordenado por nombre;

b. listar todos los Jedi ordenados por nombre;

c. listar todos los Sith ordenados por nombre;

d. mostrar los aprendices de Palpatine y de Obi-Wan kenobi, además contar cuantos aprendices tuvo cada uno;

e. determinar si Dath Malak está en la lista y mostrar su información;

f. mostrar la posición en la que se encuentra Yoda.

Ordenamiento: $O(n^2)$ = Si hay 10 elementos → máximo 100 operaciones

Búsqueda: O(n) = Si hay 10 elementos → máximo 10 comparaciones

Conteo: O(n) = Si hay 10 elementos → exactamente 10 recorridos

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    // Variables simples - Arrays paralelos
    string nombres[100];
    string maestros[100];
    string tipo[100]; // "Jedi" o "Sith"
    int total = 0;

    cout << "=== SISTEMA DE JEDI Y SITH ===" << endl;
    cout << "Cuantos personajes desea ingresar? ";
    cin >> total;
    cin.ignore();

    // ENTRADA DE DATOS
    for(int i = 0; i < total; i++) {
        cout << "\n--- Personaje " << (i+1) << " ---" << endl;
        cout << "Nombre: ";
        getline(cin, nombres[i]);
```

```
    cout << "Maestro: ";
    getline(cin, maestros[i]);
    cout << "Tipo (Jedi/Sith): ";
    getline(cin, tipo[i]);
}

// a. ORDENAR POR NOMBRE (Bubble Sort)
cout << "\n\n=== a. LISTA ORDENADA POR NOMBRE ===" << endl;
string tempN, tempM, tempT;
for(int i = 0; i < total-1; i++) {
    for(int j = 0; j < total-i-1; j++) {
        if(nombres[j] > nombres[j+1]) {
            // Intercambiar nombres
            tempN = nombres[j];
            nombres[j] = nombres[j+1];
            nombres[j+1] = tempN;
            // Intercambiar maestros
            tempM = maestros[j];
            maestros[j] = maestros[j+1];
            maestros[j+1] = tempM;
            // Intercambiar tipos
            tempT = tipo[j];
            tipo[j] = tipo[j+1];
            tipo[j+1] = tempT;
        }
    }
}

for(int i = 0; i < total; i++) {
    cout << i+1 << ". " << nombres[i] << " - Maestro: "
         << maestros[i] << " - Tipo: " << tipo[i] << endl;
}

// b. LISTAR SOLO JEDI ORDENADOS
cout << "\n\n=== b. LISTA DE JEDI ===" << endl;
int contJedi = 0;
for(int i = 0; i < total; i++) {
    if(tipo[i] == "Jedi" || tipo[i] == "jedi") {
        contJedi++;
        cout << contJedi << ". " << nombres[i]
             << " - Maestro: " << maestros[i] << endl;
```

```
    }
}
cout << "Total Jedi: " << contJedi << endl;

// c. LISTAR SOLO SITH ORDENADOS
cout << "\n\n=== c. LISTA DE SITH ===" << endl;
int contSith = 0;
for(int i = 0; i < total; i++) {
    if(tipo[i] == "Sith" || tipo[i] == "sith") {
        contSith++;
        cout << contSith << ". " << nombres[i]
             << " - Maestro: " << maestros[i] << endl;
    }
}
cout << "Total Sith: " << contSith << endl;

// d. APRENDICES DE PALPATINE Y OBI-WAN
cout << "\n\n=== d. APRENDICES ===" << endl;

cout << "\nAprendices de Palpatine:" << endl;
int contPalp = 0;
for(int i = 0; i < total; i++) {
    if(maestros[i] == "Palpatine" || maestros[i] == "palpatine") {
        contPalp++;
        cout << "  - " << nombres[i] << endl;
    }
}
cout << "Total: " << contPalp << " aprendices" << endl;

cout << "\nAprendices de Obi-Wan Kenobi:" << endl;
int contObi = 0;
for(int i = 0; i < total; i++) {
    if(maestros[i] == "Obi-Wan Kenobi" || maestros[i] == "Obi-Wan"
       || maestros[i] == "obi-wan kenobi" || maestros[i] == "obi-wan") {
        contObi++;
        cout << "  - " << nombres[i] << endl;
    }
}
cout << "Total: " << contObi << " aprendices" << endl;

// e. BUSCAR DARTH MALAK
```

```
    cout << "\n\n=== e. BUSQUEDA DE DARTH MALAK ===" << endl;
    bool encontrado = false;
    for(int i = 0; i < total; i++) {
        if(nombres[i] == "Darth Malak" || nombres[i] == "darth malak"
           || nombres[i] == "Dath Malak" || nombres[i] == "dath malak") {
            encontrado = true;
            cout << "ENCONTRADO!" << endl;
            cout << "Nombre: " << nombres[i] << endl;
            cout << "Maestro: " << maestros[i] << endl;
            cout << "Tipo: " << tipo[i] << endl;
            break;
        }
    }
    if(!encontrado) {
        cout << "Darth Malak NO esta en la lista" << endl;
    }

    // f. POSICION DE YODA
    cout << "\n\n=== f. POSICION DE YODA ===" << endl;
    bool yodaEncontrado = false;
    for(int i = 0; i < total; i++) {
        if(nombres[i] == "Yoda" || nombres[i] == "yoda") {
            yodaEncontrado = true;
            cout << "Yoda se encuentra en la posicion: " << (i+1) << endl;
            cout << "Informacion:" << endl;
            cout << "  Nombre: " << nombres[i] << endl;
            cout << "  Maestro: " << maestros[i] << endl;
            cout << "  Tipo: " << tipo[i] << endl;
            break;
        }
    }
    if(!yodaEncontrado) {
        cout << "Yoda NO esta en la lista" << endl;
    }

    return 0;
}
```

a) ORDENAMIENTO POR NOMBRE (Bubble Sort)
Ejemplo: Ordenar ["Luke", "Anakin", "Yoda"]

Pasada 1:
- Comparo "Luke" con "Anakin" → Luke > Anakin → INTERCAMBIO
  ["Anakin", "Luke", "Yoda"]
- Comparo "Luke" con "Yoda" → Luke < Yoda → NO cambio
  ["Anakin", "Luke", "Yoda"]

Pasada 2:
- Comparo "Anakin" con "Luke" → Anakin < Luke → NO cambio
  ["Anakin", "Luke", "Yoda"]

RESULTADO: ["Anakin", "Luke", "Yoda"]

Fórmula de comparaciones:

- Con n elementos: (n-1) + (n-2) + ... + 1 = n(n-1)/2
- Con 3 elementos: 3(2)/2 = 3 comparaciones
- Con 5 elementos: 5(4)/2 = 10 comparaciones

---

b) y c) CONTAR JEDI Y SITH

Ejemplo: Lista = ["Luke(Jedi)", "Vader(Sith)", "Yoda(Jedi)"]

Contador Jedi = 0

Contador Sith = 0

Posición 0: "Luke" es Jedi → Jedi++ → Jedi = 1

Posición 1: "Vader" es Sith → Sith++ → Sith = 1

Posición 2: "Yoda" es Jedi → Jedi++ → Jedi = 2

RESULTADO: 2 Jedi, 1 Sith

Fórmula: Recorridos = n (donde n = total de elementos)

---

d) CONTAR APRENDICES

Ejemplo: Buscar aprendices de "Palpatine"

Lista de maestros: ["Qui-Gon", "Palpatine", "Palpatine", "Yoda"]

Contador = 0

Posición 0: "Qui-Gon" ≠ "Palpatine" → No cuenta

Posición 1: "Palpatine" = "Palpatine" → Contador++ → 1

Posición 2: "Palpatine" = "Palpatine" → Contador++ → 2

Posición 3: "Yoda" ≠ "Palpatine" → No cuenta

RESULTADO: 2 aprendices

Fórmula: Comparaciones = n (n = total de elementos)

---

e) BÚSQUEDA DE "DARTH MALAK"

Ejemplo: Lista = ["Luke", "Yoda", "Darth Malak", "Anakin"]

Posición 0: "Luke" ≠ "Darth Malak" → Continuar

Posición 1: "Yoda" ≠ "Darth Malak" → Continuar

Posición 2: "Darth Malak" = "Darth Malak" → ¡ENCONTRADO!

→ Detenerse (break)

RESULTADO: Encontrado en posición 3 (índice 2)
Fórmula:

- Mejor caso: 1 comparación (primer elemento)
- Peor caso: n comparaciones (último elemento o no existe)
- Promedio: n/2 comparaciones

---

f) ENCONTRAR POSICIÓN DE "YODA"
Ejemplo: Lista ordenada = ["Anakin", "Luke", "Yoda", "Obi-Wan"]
Posición 0 (índice 0): "Anakin" ≠ "Yoda"
Posición 1 (índice 1): "Luke" ≠ "Yoda"
Posición 2 (índice 2): "Yoda" = "Yoda" → ENCONTRADO
RESULTADO: Posición 3 (mostramos índice+1 al usuario)

EJEMPLO COMPLETO CON 5 PERSONAJES
Entrada:
1. Luke Skywalker - Obi-Wan - Jedi
2. Darth Vader - Palpatine - Sith
3. Yoda - Ninguno - Jedi
4. Anakin Skywalker - Obi-Wan - Jedi
5. Darth Maul - Palpatine - Sith
Proceso de ordenamiento (Bubble Sort):
Original: [Luke, Vader, Yoda, Anakin, Maul]

Paso 1: [Luke, Vader, Yoda, Anakin, Maul]
Paso 2: [Luke, Vader, Anakin, Yoda, Maul]
Paso 3: [Luke, Anakin, Vader, Yoda, Maul]
Paso 4: [Anakin, Luke, Vader, Yoda, Maul]
Final: [Anakin, Darth Maul, Darth Vader, Luke, Yoda]
Conteo:

- Jedi: 3 (Anakin, Luke, Yoda)
- Sith: 2 (Maul, Vader)
- Aprendices de Palpatine: 2 (Vader, Maul)
- Aprendices de Obi-Wan: 2 (Luke, Anakin)

EJERCICIO 10 – HASH

Desarrollar un algoritmo que implemente una tabla hash para una guía de teléfono, los datos que se conocen son número de teléfono, apellido, nombre y dirección de la persona. El campo

clave debe ser el número de teléfono.

```
#include <iostream>
#include <string>
using namespace std;

const int TAMANO = 10;

// Variables globales para la tabla hash
int telefonos[TAMANO];
string nombres[TAMANO];
string apellidos[TAMANO];
string direcciones[TAMANO];
bool ocupado[TAMANO];

// Inicializar tabla
void inicializar() {
    for (int i = 0; i < TAMANO; i++) {
        ocupado[i] = false;
        telefonos[i] = 0;
        nombres[i] = "";
        apellidos[i] = "";
        direcciones[i] = "";
    }
}

// Función hash simple
int funcionHash(int telefono) {
    return telefono % TAMANO;
}

// Insertar contacto
void insertar() {
    int telefono;
    string nombre, apellido, direccion;

    cout << "\n--- INSERTAR CONTACTO ---" << endl;
    cout << "Telefono: ";
    cin >> telefono;
    cin.ignore();
    cout << "Nombre: ";
```

```
    getline(cin, nombre);
    cout << "Apellido: ";
    getline(cin, apellido);
    cout << "Direccion: ";
    getline(cin, direccion);

    int indice = funcionHash(telefono);
    int intentos = 0;

    // Buscar posición libre (sondeo lineal)
    while (ocupado[indice] && intentos < TAMANO) {
        indice = (indice + 1) % TAMANO;
        intentos++;
    }

    if (intentos == TAMANO) {
        cout << "ERROR: Tabla llena" << endl;
        return;
    }

    // Guardar datos
    telefonos[indice] = telefono;
    nombres[indice] = nombre;
    apellidos[indice] = apellido;
    direcciones[indice] = direccion;
    ocupado[indice] = true;

    cout << "Contacto guardado en posicion: " << indice << endl;
}

// Buscar contacto
void buscar() {
    int telefono;
    cout << "\n--- BUSCAR CONTACTO ---" << endl;
    cout << "Telefono a buscar: ";
    cin >> telefono;

    int indice = funcionHash(telefono);
    int intentos = 0;

    // Buscar el teléfono
```

```
    while (intentos < TAMANO) {
        if (ocupado[indice] && telefonos[indice] == telefono) {
            cout << "\nCONTACTO ENCONTRADO:" << endl;
            cout << "Posicion: " << indice << endl;
            cout << "Telefono: " << telefonos[indice] << endl;
            cout << "Nombre: " << nombres[indice] << endl;
            cout << "Apellido: " << apellidos[indice] << endl;
            cout << "Direccion: " << direcciones[indice] << endl;
            return;
        }
        indice = (indice + 1) % TAMANO;
        intentos++;
    }

    cout << "Contacto NO encontrado" << endl;
}

// Mostrar toda la tabla
void mostrarTodo() {
    cout << "\n--- TABLA HASH COMPLETA ---" << endl;
    for (int i = 0; i < TAMANO; i++) {
        cout << "Posicion " << i << ": ";
        if (ocupado[i]) {
            cout << telefonos[i] << " - " << nombres[i]
                 << " " << apellidos[i] << endl;
        } else {
            cout << "[VACIO]" << endl;
        }
    }
}

// Eliminar contacto
void eliminar() {
    int telefono;
    cout << "\n--- ELIMINAR CONTACTO ---" << endl;
    cout << "Telefono a eliminar: ";
    cin >> telefono;

    int indice = funcionHash(telefono);
    int intentos = 0;
```

```
    while (intentos < TAMANO) {
        if (ocupado[indice] && telefonos[indice] == telefono) {
            ocupado[indice] = false;
            cout << "Contacto eliminado de la posicion: " << indice << endl;
            return;
        }
        indice = (indice + 1) % TAMANO;
        intentos++;
    }

    cout << "Contacto NO encontrado" << endl;
}

int main() {
    inicializar();
    int opcion;

    do {
        cout << "\n========== GUIA TELEFONICA ==========" << endl;
        cout << "1. Insertar contacto" << endl;
        cout << "2. Buscar contacto" << endl;
        cout << "3. Eliminar contacto" << endl;
        cout << "4. Mostrar tabla completa" << endl;
        cout << "0. Salir" << endl;
        cout << "Opcion: ";
        cin >> opcion;

        switch(opcion) {
            case 1: insertar(); break;
            case 2: buscar(); break;
            case 3: eliminar(); break;
            case 4: mostrarTodo(); break;
            case 0: cout << "Saliendo..." << endl; break;
            default: cout << "Opcion invalida" << endl;
        }
    } while (opcion != 0);

    return 0;
}
```

Función Hash Simple
índice = clave % tamaño_tabla
Ejemplo:

- Tamaño tabla = 10
- Teléfono = 12345
- índice = 12345 % 10 = 5 → guardamos en posición 5

Manejo de Colisiones (Sondeo Lineal)
Si la posición está ocupada, buscamos la siguiente:
nuevo_índice = (índice + 1) % tamaño_tabla
Ejemplo de inserción:

1. Teléfono 12345 → índice = 5 → guardamos en [5]
2. Teléfono 54321 → índice = 1 → guardamos en [1]
3. Teléfono 11111 → índice = 1 → OCUPADO → probamos [2] → guardamos en [2]