Hoja de Trabajo 1

¿Qué es un TDA?

Es una abstracción de una estructura de dato, la cual especifica lo que se puede almacenar dentro de él y que operaciones se pueden realizar sobre el mismo. Las operaciones más comunes son: búsqueda, inserción, extracción, ordenamiento, entre otros.

¿Para qué se utiliza el nodo pivote?

El nodo pivote se utiliza para recorrer una estructura e indicar en qué posición se encuentra.

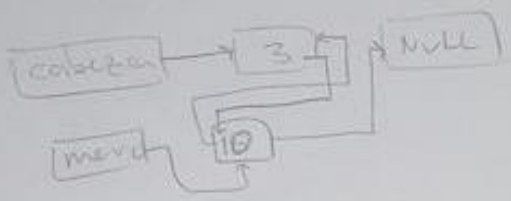
¿Para qué se utiliza el nodo cabeza?

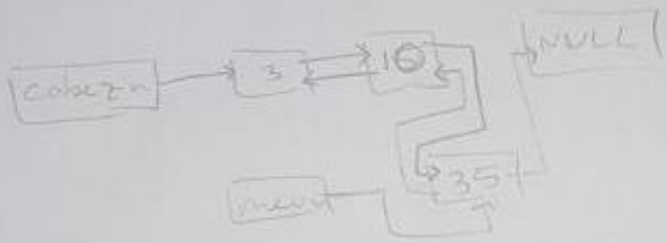
El nodo cabeza se utiliza para indicar el comienzo de una lista. Este no almacena ningún valor, solo apunta al primer elemento.

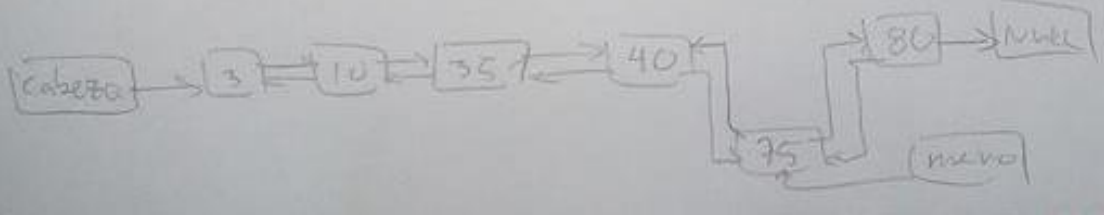
Estructuras dinámicas		**Estructuras estáticas**	
Ventajas	**Desventajas**	**Ventajas**	**Desventajas**
Se puede manejar más espacio en memoria.	Es más lento acceder al elemento. No hay acceso directo.	Permiten acceder más rápido al elemento de la lista. Hay acceso directo.	El espacio en memoria ya está definido y no se puede alterar

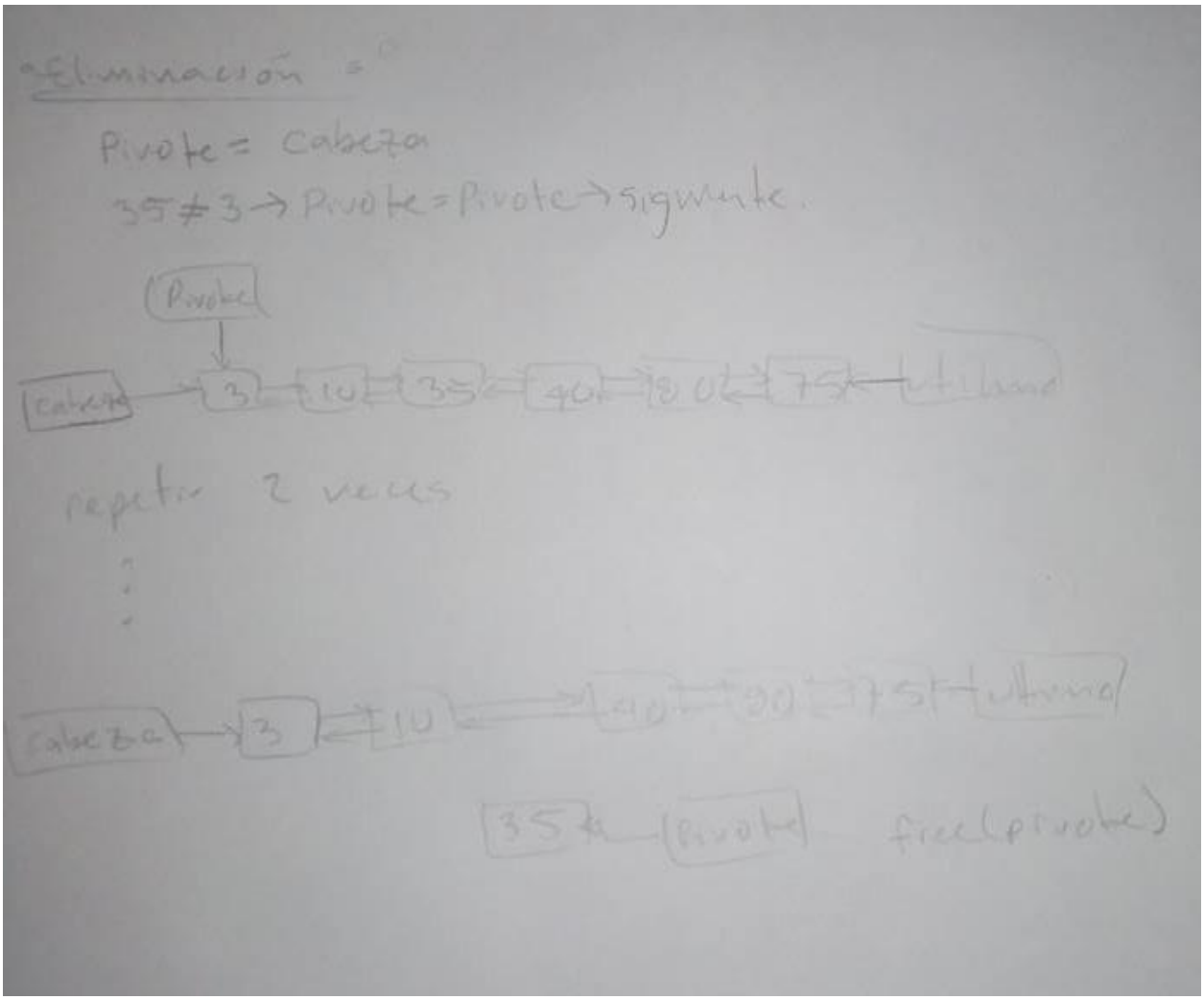
Lista doblemente enlazada:

- Inserción:



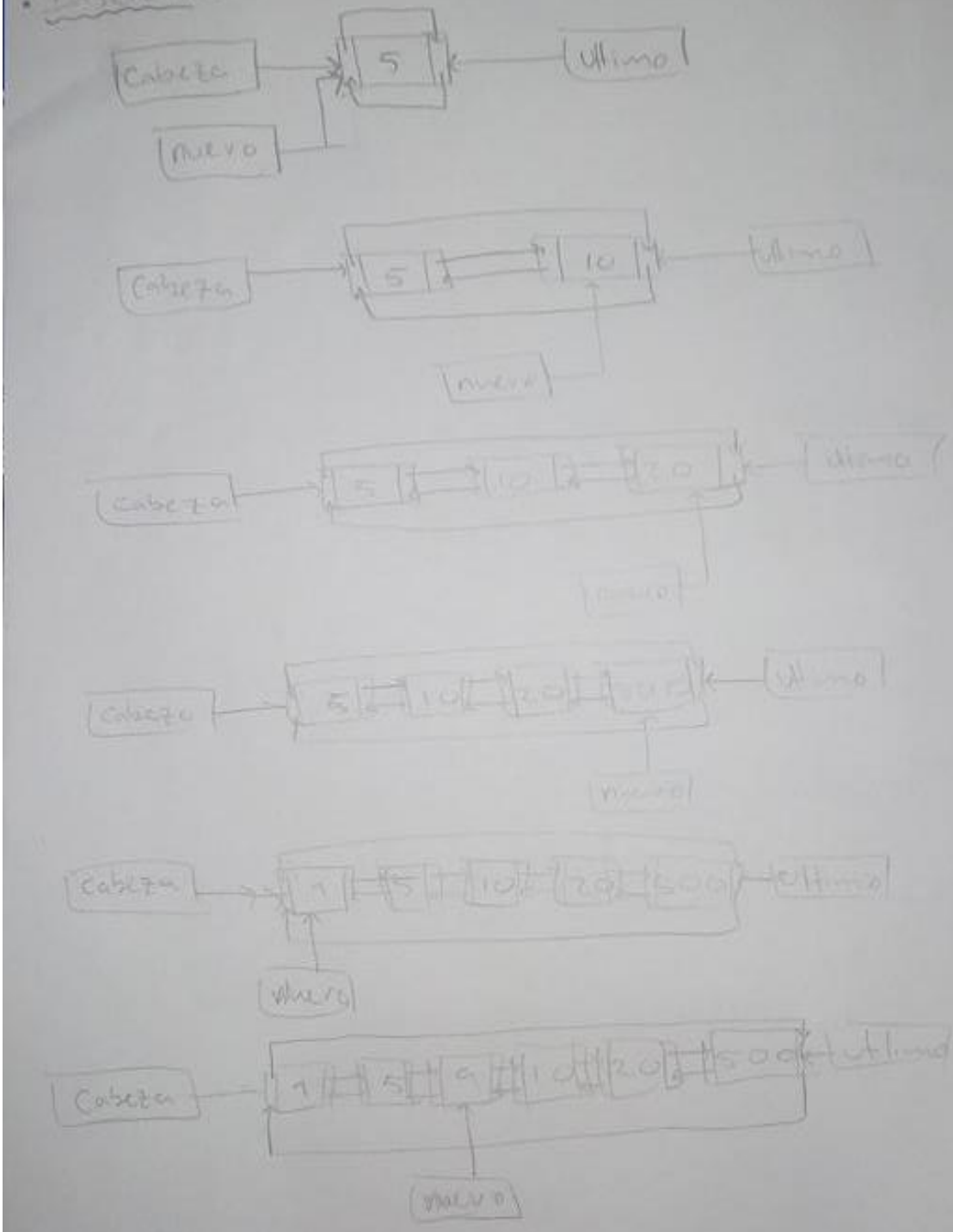


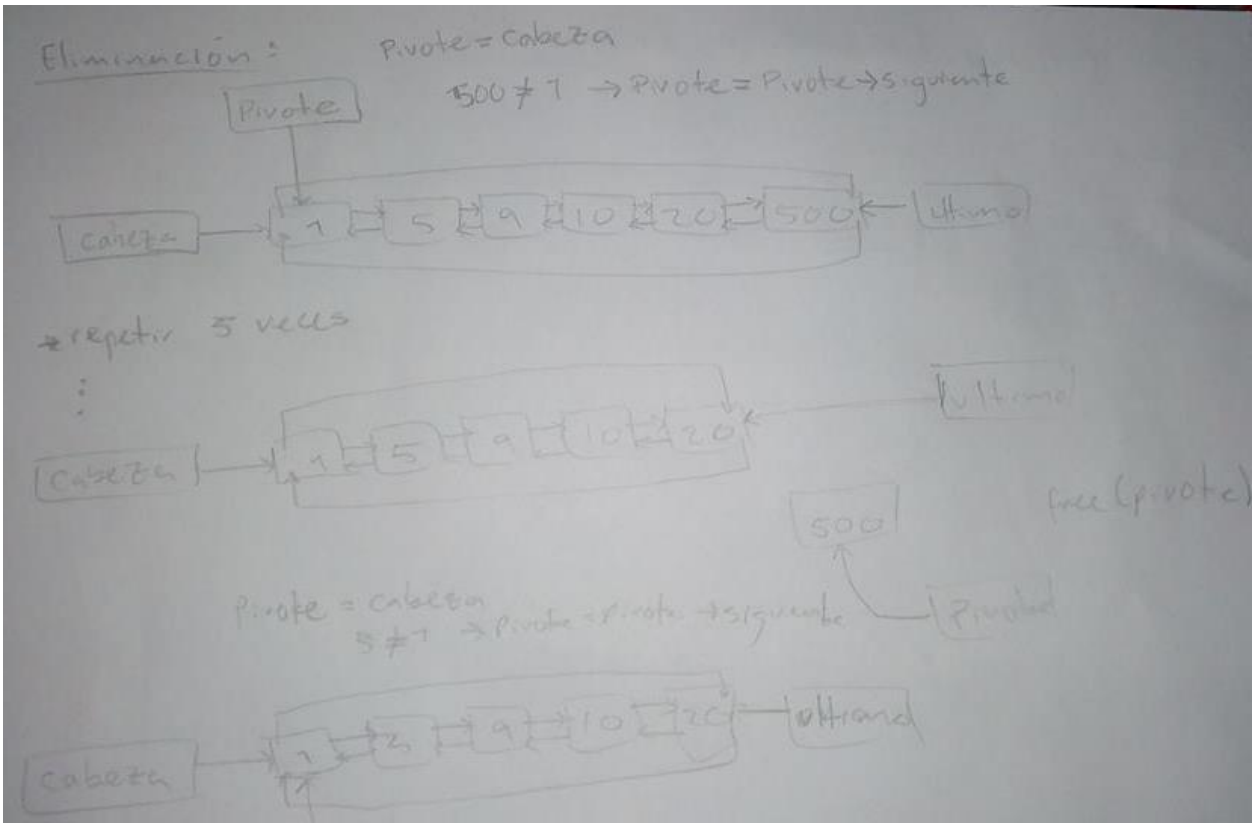


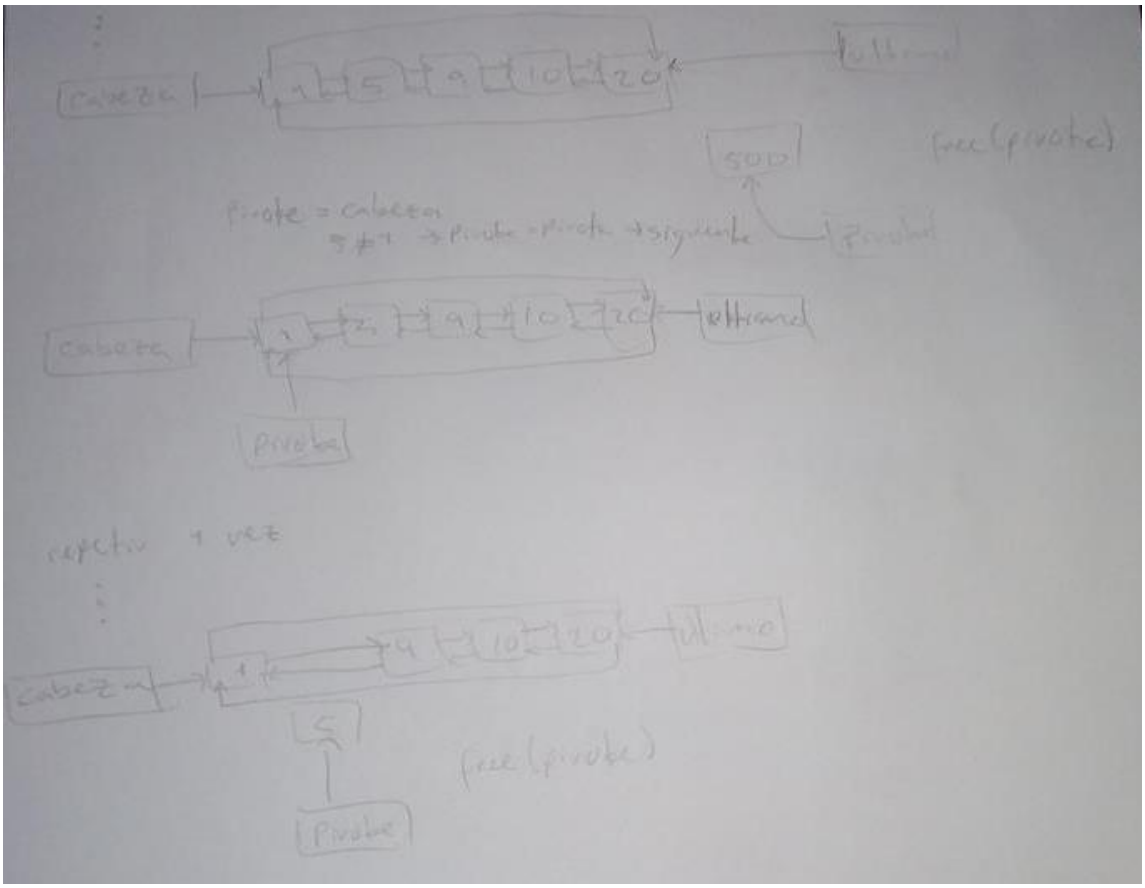

*Eliminación =
Pivote = cabeza
35 ≠ 3 → Pivote = Pivote → siguiente.
Pivote
cabeza
3
10
35
40
80
75
ultimo
repetir 2 veces
cabeza
3
10
ultimo
35
Pivote
free(pivote)

Lista circular doblemente enlazada

- Inserción:




Eliminación:
Pivote = cabeza
500 ≠ 1 → Pivote = Pivote → siguiente
Pivote
cabeza
1
5
9
10
20
500
último
repetir 5 veces
cabeza
último
500
free(pivote)
Pivote = cabeza
5 ≠ 1 → Pivote = Pivote → siguiente
Pivote
cabeza
último


cabeza
último
500
free(pivote)
Pivote = cabeza
5 ≠ 1 → Pivote = Pivote → siguiente
Pivote
cabeza
último
Pivote
repetir 1 vez
cabeza
último
5
free(pivote)
Pivote

Código

```
typedef struct nodo {
  int id;
  char valor;
  struct nodo *siguiente;
  struct nodo *anterior;
} nodo;

void insertarLSE()
{
   nodo *nuevo_nodo = (nodo*)malloc(sizeof(nodo));
   cout<<"\n   Ingrese un id:"<<endl;
   cin>>nuevo_nodo->id;
   cout<<"\n   Ingrese un valor:"<<endl;
   cin>>nuevo_nodo->valor;

   if(p == NULL){
      p = nuevo_nodo;
      p->siguiente = p;
      u = nuevo_nodo;
   }
   else if(p->siguiente == p){
      if(nuevo_nodo->id < p->id){
         nuevo_nodo->siguiente = u;
         p = nuevo_nodo;
      }
      else if(nuevo_nodo->id >= p->id){
         p->siguiente = nuevo_nodo;
```

```
        nuevo_nodo->siguiente = p;
        u = nuevo_nodo;
    }
}
else{
    nodo *aux = (nodo*)malloc(sizeof(nodo));
    aux = p;

    do{
        if(nuevo_nodo->id < p->id){
            nuevo_nodo->siguiente = p;
            p = nuevo_nodo;
            u->siguiente = p;
            break;
        }
        else if(nuevo_nodo->id >= aux->id && aux->siguiente == p){
            u->siguiente = nuevo_nodo;
            nuevo_nodo->siguiente = p;
            u = nuevo_nodo;
            break;
        }
        else if(nuevo_nodo->id >= aux->id && nuevo_nodo->id <= aux->siguiente->id){
            nuevo_nodo->siguiente = aux->siguiente;
            aux->siguiente = nuevo_nodo;
            break;
        }
        else{
            aux = aux->siguiente;
        }
```

```
        }while(aux != p);
    }
    cout<<"\n   Id: "<<nuevo_nodo->id<<"   Valor: "<<nuevo_nodo->valor<<endl;
}
```

```
Void imprimir(){
       nodo *aux = (nodo*)malloc(sizeof(nodo));
        aux = ultimo;

        while(aux != NULL){
             cout<<"\n   Id: "<<aux->id<<"   Valor: "<<aux->valor<<endl;
             aux = ultimo->anterior;
        }
}
```