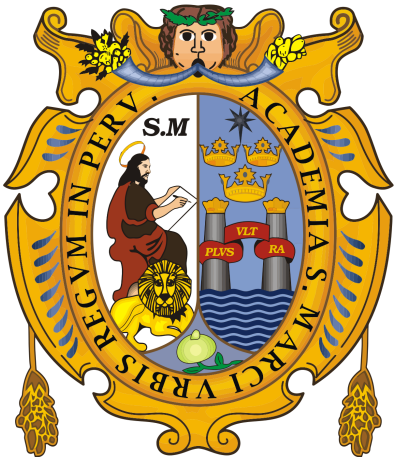
“UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS”

“Facultad de Ingeniería de Sistemas e Informática”
Escuela profesional de Ingeniería de Software



SISTEMA INTEGRADO DE TRÁMITE DOCUMENTARIO

Datos generales:

- **Asignatura:** Análisis y Diseño de Algoritmos
- **Grupo:** 3
- **Sección: 1**
- **Integrantes:**
  - Ancaya Martinez Benjamin
  - Bedia Gonzales, Anthony Alonso
  - Collado Jimenez Jhair Alejandro Ovenar
  - Landeo Cuentas, Sebastian Alonso
  - Alejandro Santaria Gian Franco
- **Profesor:** Prudencio Vidal Javier Antonio

2023

Índice

Cálculo de la complejidad algorítmica

1. Realizar Solicitud

<table>
<tr><th colspan="3">Cálculo T(n) de la función Realizar Solicitud</th></tr>
<tr><td>Código</td><td colspan="2">

```
case 1:{
    pushSolicitud(pila);
    cout<<"Mostrando pila"<<endl;
    VerSolicitud(pila);
    system("pause");
    break;
}
```

</td></tr>
<tr><td>Ubicación</td><td colspan="2">Función: <code>int main()</code></td></tr>
<tr><td colspan="3">Desarrollo</td></tr>
<tr><td></td><td><code>pushSolicitud(pila);</code> 1 Operación
La función tiene una complejidad de tiempo constante, ya que simplemente empuja un elemento a la pila.</td><td>Llamado a función.</td></tr>
<tr><td></td><td><code>cout<<"Mostrando pila"<<endl;</code>
1 Operación</td><td></td></tr>
<tr><td></td><td><code>VerSolicitud(pila);</code> N Operaciones
Muestra el contenido de la pila. Suponiendo que la pila contiene n elementos, esto requeriría un tiempo proporcional a n,</td><td>Llamado a función.</td></tr>
<tr><td></td><td><code>pushSolicitud(pila);</code> 1 Operación</td><td>Llamado a función.</td></tr>
<tr><td></td><td><code>break;</code>1 Operación</td><td>Llamado a función.</td></tr>
<tr><td></td><td colspan="2">Resultado:
T(N) = 1 + 1 + N + 1 = N + 3
F(N) = N
Demostración:
N + 3 =< cN
escogemos c = 2
N + 3 =< 2N
3 =< N y escogemos N0 = 3</td></tr>
</table>

2. Extraer Solicitud

<table>
<tr><th colspan="3">Cálculo T(n) de la función Extraer Solicitud</th></tr>
<tr><td>Código</td><td colspan="2">

```
temporal=popSolicitud(pila);
VerSolicitud(pila);
if(temporal->nacionalidad==1){
    cout<<"Peruano(a) pasando automaticamente a
lista"<<endl;
    temporal->estado=2;
    InsertarInicio(lista,temporal);
    cout<<"Mostrando DNI's"<<endl;
    MostrarDNI_Resumido(lista);
}else{
    cout<<"Extranjero(a) pasando a cola para
revision"<<endl;
    TrasladarCola(cola,temporal);
    cout<<"\nCola de solicitudes"<<endl;
    VerSolicitud(cola);
}
system("pause");
break;
```

</td></tr>
<tr><td>Ubicación</td><td colspan="2">Función: <code>int main()</code></td></tr>
<tr><td colspan="3">Desarrollo</td></tr>
<tr><td></td><td><code>temporal=popSolicitud(pila);</code>
2 Operaciones
La función simplemente elimina el elemento superior de la pila y luego se le asigna a la variable.</td><td>Llamado a función y asignación.</td></tr>
<tr><td></td><td><code>VerSolicitud(pila);</code>
N Operaciones</td><td>Llamado a función</td></tr>
<tr><td></td><td><code>if(temporal->nacionalidad==1)</code>
2 Operación</td><td>Comparación y acceso.</td></tr>
</table>

	`cout<<"Peruano(a) pasando automaticamente a lista"<<endl;`
	1 Operación
	`temporal->estado=2;`
	2 Operaciones
	`InsertarInicio(lista,temporal);`
	1 Operación
	`cout<<"Mostrando DNI's"<<endl;`
	1 Operación
	`MostrarDNI_Resumido(lista);`
	N Operaciones
		Acceso y asignación.
		
		Llamada a función
		
		Llamada a función

	`cout<<"Extranjero(a) pasando a cola para revision"<<endl;`
	1 Operación
	`TrasladarCola(cola,temporal);`
	Traslada la solicitud a la cola. Suponiendo que la cola contiene n elementos, esto requeriría un tiempo proporcional a n,
	N Operaciones
	`cout<<"\nCola de solicitudes"<<endl;`
	1 Operación
	`VerSolicitud(cola);`
	N Operaciones
		Llamado a función.
		
		Llamado a función.

	`system("pause");`
	`break;`
	2 Operaciones
		Llamado a función.

	Resultado: T(N) = 6 + N + max{N + 5, 2N + 2} = 3N + 8 F(N) = N **Demostración:** 3N + 8 =< cN escogemos c = 4 3N + 8 =< 4N 8 =< N y escogemos N0= 8	

3. Verificar solicitud (extranjeros)

<table>
<tr><th colspan="3">Cálculo T(n) de la función Verificar solicitud</th></tr>
<tr><td>Código</td><td colspan="2">

```
cout<<"Numero de solicitud: "<<cola->codigo;
cout<<" Extranjero"<<endl;
cout<<"Verficando anios de residencia"<<endl;
cout<<"Cuales son los anios de residencia del
solicitante"<<endl;
cin>>anios;
if(anios>=2){
    //Ver el primero de la cola
    cout<<"Aceptado para obtener su DNI"<<endl;
    //Extraer el temporal de la cola para pasarlo
a la lista
    temporal=ExtraerCola(cola);
    InsertarInicio(lista,temporal);
    cout<<"\nMostrando DNI's"<<endl;
    MostrarDNI_Resumido(lista);
}else{
    cout<<"Rechazando solicitud"<<endl;
    Desencolar(cola);
    cout<<"\nCola de solicitudes"<<endl;
    VerSolicitud(cola);
}
system("pause");
break;
}
```

</td></tr>
<tr><td>Ubicación</td><td colspan="2">Función: <code>int main()</code></td></tr>
<tr><td>Desarrollo</td><td colspan="2"></td></tr>
<tr><td></td><td>

```
cout<<"Numero de solicitud:
"<<cola->codigo;
cout<<" Extranjero"<<endl;
cout<<"Verficando anios de
residencia"<<endl;
cout<<"Cuales son los anios de
```

</td><td>Acceso</td></tr>
</table>

	residencia del solicitante"<<endl; cin>>anios; **6 Operaciones**	Asignación
	if(anios>=2) **1 Operación**	Comparación
	cout<<"Aceptado para obtener su DNI"<<endl; **1 Operación** temporal=ExtraerCola(cola); **2 Operación** InsertarInicio(lista,temporal); **1 Operación** cout<<"\n Mostrando DNIs"<<endl; **1 Operación** MostrarDNI_Resumido(lista); **N Operaciones**	Llamado a función y asignación. Llamado a función. Llamado a función.
	cout<<"Rechazando solicitud"<<endl; **1 Operación** Desencolar(cola); **1 Operación** cout<<"\nCola de solicitudes"<<endl; **1 Operación** VerSolicitud(cola); **N Operaciones**	Llamado a función. Llamado a función.
	system("pause"); break; **2 Operaciones**	
	Resultado: T(N) = 7 + max{N + 5, N + 3} = N + 12 F(N) = N **Demostración:** N + 12 =< cN escogemos c = 2 N + 12 =< 2N 12 =< N y escogemos N0 = 12	

4. Mostrar DNI registrados

<table>
<tr><th colspan="3">Cálculo T(n) de la función Mostrar DNI registrados</th></tr>
<tr><td>Código</td><td colspan="2"><pre>
MostrarDNI(lista);
cout << endl;
system("pause");
break;
</pre></td></tr>
<tr><td>Ubicación</td><td colspan="2">Función: <code>int main()</code></td></tr>
<tr><td colspan="3">Desarrollo</td></tr>
<tr><td></td><td><code>MostrarDNI(lista);</code>
N Operaciones</td><td>Llamado a función.</td></tr>
<tr><td></td><td><code>system("pause");</code>
<code>break;</code> 2 Operaciones</td><td></td></tr>
<tr><td></td><td colspan="2">Resultado:
T(N) = N + 2
F(N) = N
Demostración:
N + 2 =< cN
escogemos c = 2
N + 2 =< 2N
2 < =N y escogemos N0 = 2</td></tr>
</table>

5. Mostrar DNI registrado (resumido)

<table>
<tr><th colspan="3">Cálculo T(n) de la función Mostrar DNI registrado (resumido)</th></tr>
<tr><td>Código</td><td colspan="2"><pre>
     int i=0;
     while(lista != NULL){
cout<< " "<<i+1<<") "<<lista->numeroDNI<<" -
"<<lista->primer_Nombre;
cout<<" - "<<lista->primer_Apellido<<endl;
          lista=lista->sgte;
          i++;
</pre></td></tr>
<tr><td>Ubicación</td><td colspan="2">Función: <code>void MostrarDNI_Resumido(TpLista lista)</code></td></tr>
<tr><td colspan="3">Desarrollo</td></tr>
<tr><td></td><td><pre>int i=0;</pre>2 Operaciones</td><td>Asignación y declaración</td></tr>
<tr><td></td><td><pre>while(lista != NULL){</pre>N Operaciones
<pre>cout<<" "<<i+1<<")
"<<lista->numeroDNI<<"
-"<<lista->primer_Nombre;</pre>3N Operaciones
<pre>cout<<"
-"<<lista->primer_Apellido<<endl;
          lista=lista->sgte;</pre>2N Operaciones
<pre>   i++;</pre>2N Operaciones
<pre>   }</pre></td><td>Recorrer en la lista n veces

Operación (+) y 2 accesos n veces

2 accesos n veces

Declaración y suma n veces</td></tr>
<tr><td></td><td colspan="2">Resultado:
T(N) = 8N +2
F(N) = N</td></tr>
</table>

	Demostración: **8**N + 2 =< cN escogemos c = 9 8N + 2 =< 9N 2< =N y escogemos N0 = 2

6. Ver Solicitud

Cálculo T(n) de la función Ver Solicitud		

Código	`int i=0;`
	`while(solicitud != NULL){`
	`cout<< " "<<i+1<<")`
	`"<<solicitud->codigo<<" - ";`
	`if(solicitud->nacionalidad==1){`
	`cout<<"Peruano"<<" - ";`
	`}else{`
	`cout<<"Extranjero"<<" - ";`
	`}`
	
	`switch(solicitud->estado){`
	`case 1:cout<<"En espera"<<endl;break;`
	`case 2:cout<<"Aceptado"<<endl;break;`
	`case 3:cout<<"Rechazado"<<endl;`
	`}`
	`solicitud=solicitud->sgte;`
	`i++;`
	`}`	

Ubicación	Función: `void VerSolicitud(TpSoli solicitud)`	
Desarrollo		
	`int i=0;` **2 Operaciones**	Asignación y declaración
	`while(solicitud != NULL){` **N Operaciones**	Recorrer en la lista n veces

	cout<< " "<<i+1<<") "<<solicitud->codigo<<" - ";
	2N Operaciones
	if(solicitud->nacionalidad==1){
	N Operaciones
	cout<<"Peruano"<<" - ";
	}else{
	cout<<"Extranjero"<<" - ";
	}
	
	switch(solicitud->estado){
	case 1:cout<<"En espera"<<endl;break;
	case 2:cout<<"Aceptado"<<endl;break;
	case 3:cout<<"Rechazado"<<endl;
	}
	solicitud=solicitud->sgte;
	N Operaciones
	i++;
	2N Operaciones
	}
		Operación i+1 y acceso n veces
		
		Condicional n veces
		
		Asignación n veces
		
		Declaración y suma n veces

	Resultado: T(N) = 6N +2 F(N) = N **Demostración:** **6**N + 2 =< cN escogemos c = 7 6N + 2 =< 7N 2 < =N y escogemos N0 = 2	

7. Actualizar información de un DNI

<table>
<tr><th colspan="3">Cálculo T(n) de la función Actualizar información de un DNI</th></tr>
<tr><td>Código</td><td colspan="2">if(lista == NULL){
cout<<"\nERROR.. La lista esta vacia.. no permite esta opcion "<<endl;
}
else{
cout<<"\nIngresar el DNI: ";cin>>busca;
if(busquedaDNI(lista, busca)==true){
cout<<"Direccion de su vivienda: ";
cin>>reempD;
cout<<"Estado civil: ";
cin>> reempE;
cout<<"Nuevo nombre: ";
cin>> reempN;
Actualizar(lista, busca, reempN, reempD, reempE);
}
else{
cout<<"\nERROR: No esta permitido ese valor"<<endl;
}
}</td></tr>
<tr><td>Ubicación</td><td colspan="2">Función: <code>int main()</code></td></tr>
<tr><td colspan="3">Desarrollo</td></tr>
<tr><td></td><td>if(lista == NULL){
cout<<"\nERROR.. La lista esta vacia.. no permite esta opcion "<<endl;
}
Si la lista está vacía, la complejidad de tiempo sería O(1).</td><td>Comparación y salida</td></tr>
<tr><td></td><td>else{
cout<<"\nIngresar el DNI: ";cin>>busca;
if(busquedaDNI(lista, busca)==true){
Si no está vacía, la función busquedaDNI se llama para realizar una búsqueda lineal en la</td><td>Llamado a función de tiempo n.

Asignaciones y salidas.</td></tr>
</table>

	lista. La complejidad de búsqueda lineal es O(n).
	cout<<"Direccion de su vivienda: ";
	cin>>reempD;
	cout<<"Estado civil: ";
	cin>> reempE;
	cout<<"Nuevo nombre: ";
	cin>> reempN;
	Si la búsqueda encuentra el nodo, se realizan operaciones de entrada y salida de datos con un tiempo constante. O(1).
	Actualizar(lista, busca, reempN, reempD, reempE);
	Esta función busca el nodo linealmente y emplea el algoritmo de distancia de edición entre dos cadenas. Este algoritmo encuentra la distancia mínima entre el nombre del nodo buscado de tamaño k y el nuevo nombre reempN de tamaño t. Esta función tiene un tiempo de ejecución O(n*k*t)
	}
	else{
	cout<<"\nERROR: No esta permitido ese valor"<<endl;
	}
	Finaliza con una condición para un DNI no encontrado que tiene un tiempo de ejecución de O(1)
	
	El tiempo total de ejecución dentro del else es de O(n) + O(n*k*t). Resultando en un tiempo de ejecución total de O(n*k*t).
	}
	La complejidad de tiempo total del algoritmo se puede calcular como el máximo entre O(1) (cuando la lista está vacía) y O(n * k * t) (cuando la lista no está vacía y se realiza la actualización
		Llamada a función de tiempo n*k*t

	del nodo).	
	Por lo tanto, la complejidad de tiempo total del algoritmo es: $T(n) = O(n * k * t)$	

Cálculo de complejidad con Notación BigO

1. Programación dinámica: Algoritmo de distancia de edición entre dos cadenas

Cálculo BigO de la función minDistanciaEdicion	
Código	

```
int minDistanciaEdicion(const string& u, const
string& v) {
    int m = u.length();
    int n = v.length();

    int dp[m + 1][n + 1];

    for (int i = 0; i <= m; ++i) {
        dp[i][0] = i;
    }
    for (int j = 0; j <= n; ++j) {
        dp[0][j] = j;
    }

    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) {
            dp[i][j] = 0;
        }
    }
    int cost;
    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) {

            if (u[i - 1] == v[j - 1]) {
                cost = 0;
            } else {
                cost = 1;
            }
            // dp[i - 1][j - 1] + cost :
Sustitucion,
            //dp[i][j - 1] + 1 :Insercion,
            //dp[i - 1][j] + 1 : Eliminacion
```

<table>
<tr><td></td><td colspan="2">

```
            dp[i][j] = min(min(dp[i - 1][j - 1] +
cost, dp[i][j - 1] + 1), dp[i - 1][j] + 1);
        }
    }
    return dp[m][n];
}
```

</td></tr>
<tr><td>Ubicación</td><td colspan="2">Función: Función global del programa</td></tr>
<tr><td colspan="3">Desarrollo</td></tr>
<tr><td></td><td>

```
if (u[i - 1] == v[j - 1]) {
    cost = 0; // Sustitucion por
letra igual costo = 0
} else {
    cost = 1;// Sustitucion de letra
diferente costo = 1
}
// dp[i - 1][j - 1] + cost :
Sustitucion,
//dp[i][j - 1] + 1 :Insercion,
//dp[i - 1][j] + 1 : Eliminacion
dp[i][j] = min(min(dp[i - 1][j - 1]
+ cost, dp[i][j - 1] + 1), dp[i -
1][j] + 1);
```

El bucle anidado más interno tiene una complejidad de tiempo constante, ya que solo realiza asignaciones y comparaciones simples. Por lo tanto, el costo de tiempo dentro de este bucle es O(1).

</td><td>Asignación y comparación</td></tr>
<tr><td></td><td>

```
for (int i = 1; i <= m; ++i){
  for (int j = 1; j <= n; ++j){
      ...
  }
}
```

El bucle externo más cercano se ejecuta m veces, donde m es la longitud de la cadena u. Dentro de este bucle, el bucle anidado más externo se ejecuta n veces, donde n es la

</td><td>Bucles anidados</td></tr>
</table>

	longitud de la cadena v. Por lo tanto, el costo de tiempo del bucle anidado más externo es O(n). En general, el algoritmo tiene una complejidad de tiempo de O(m * n), donde m y n son las longitudes de las cadenas u y v, respectivamente.	
	Resultado: O(m * n)	

Ordenar DNI

Cálculo BigO de la función Ordenar DNI	

Código	`cout<<"Lista sin ordenar"<<endl;`
	`MostrarDNI_Resumido(lista);`
	`cout<<"Ordenando lista"<<endl;`
	
	`lista=quicksort(lista,getLastNode(lista));`
	`cout<<"Lista ordenada"<<endl;`
	`MostrarDNI_Resumido(lista);`
	`system("pause");`
	`break;`

Ubicación	**`quicksort(lista,getLastNode(lista));`**
Desarrollo	
	`quicksort(lista,getLastNode(lista));` `if (start == NULL || start == end)` **3 Operaciónes** `return start;` **1 Operación** `TpLista newStart = NULL;` **1 Operación** `TpLista newEnd = NULL;` **1 Operación**

	TpLista pivot = partition(start, end, &newStart, &newEnd); **En el caso común es (Log n) ya que el pivote partirá a la listas en 2 partes iguales**
	
	`if (newStart != pivot)` **1 Operación** `{`
	`TpLista temp = newStart;` **1 Operación**
	`while (temp->sgte != pivot)` **N +1 Operación** `{`
	`temp = temp->sgte;` **N Operaciónes**
	`}`
	`temp->sgte = NULL;` **1 Operación**
	
	`newStart = quicksort(newStart, temp);` **N Operaciónes**
	
	`temp = getLastNode(newStart);` **1 Operación**
	`temp->sgte = pivot;` **1 Operación**
	`}`
	
	`pivot->sgte = quicksort(pivot->sgte, newEnd);`
	
	`return newStart;` **1 Operación**

8. Búsqueda DNI

Cálculo T(n) de la función Búsqueda DNI	

Código	`bool busquedaDNI(TpLista lista, int b){`
	`TpLista p=lista;`
	`bool flag=false;`
	`while(p != NULL){`
	`if(p->numeroDNI == b){`

<table>
<tr><td></td><td colspan="2">

```
                flag=true;
            }
            p=p->sgte;
        }
        if(flag==false){
            cout<<"Valor buscado "<<b<<" no existe
en la lista"<<endl;
        }
        return flag;
}
```

</td></tr>
<tr><td>Ubicación</td><td colspan="2">Función: <code>int main()</code></td></tr>
<tr><td colspan="3">Desarrollo</td></tr>
<tr><td></td><td><code>pushSolicitud(pila);</code> 1 Operación
La función tiene una complejidad de tiempo constante, ya que simplemente empuja un elemento a la pila.</td><td>Llamado a función.</td></tr>
<tr><td></td><td><code>cout<<"Mostrando pila"<<endl;</code>
1 Operación</td><td></td></tr>
<tr><td></td><td><code>VerSolicitud(pila);</code> N Operaciones
Muestra el contenido de la pila. Suponiendo que la pila contiene n elementos, esto requeriría un tiempo proporcional a n,</td><td>Llamado a función.</td></tr>
<tr><td></td><td><code>pushSolicitud(pila);</code> 1 Operación</td><td>Llamado a función.</td></tr>
<tr><td></td><td><code>break;</code> 1 Operación</td><td>Llamado a función.</td></tr>
<tr><td></td><td colspan="2">Resultado:
T(N) = 1 + 1 + N + 1 = N + 3
F(N) = N
Demostración:
N + 3 =< cN
escogemos c = 2
N + 3 =< 2N
3 =< N y escogemos N0 = 3</td></tr>
</table>

9. Búsqueda Binaria

<table>
<tr><th colspan="2">Cálculo T(n) de la función BusquedaBinaria</th></tr>
<tr><td>Código</td><td>

```
 void BusquedaBinaria(TpLista lista, int low, int
high, int dni) {
    if (low > high) {
        cout << "El DNI ingresado no se encontr?
en la lista." << endl;
        return;
    }

    int mid = low + (high - low) / 2;
    TpLista p = lista;

    // Avanzar a la posici?n 'mid' en la lista
    for (int i = 0; i < mid; i++) {
        p = p->sgte;
    }

    if (p->numeroDNI == dni) {
        MostrarDNI(lista);
    } else if (dni < p->numeroDNI) {
        BusquedaBinaria(lista, low, mid - 1, dni);
    } else {
        BusquedaBinaria(lista, mid + 1, high,
dni);
    }
}
```

</td></tr>
<tr><td>Ubicación</td><td>void BusquedaBinaria(TpLista lista, int low, int high, int dni)</td></tr>
<tr><td colspan="2">Desarrollo
Se explicará una manera de como hallar el T(n). La búsqueda binaria normal, con vectores, tiene una complejidad de O(logn), esto sucede porque la cantidad de datos se divide. Pero nuestro programa trabaja con listas simplemente enlazadas, por lo que tiene que hacer un recorrido de n, para poder llegar hasta el elemento del medio. Por lo que daría como resultado una complejidad O(nlogn)</td></tr>
<tr><td>Resultado</td><td>El resultado que se obtuvo de manera formal fue,</td></tr>
</table>

	T(n)=log(n)C1 + nlog(n)C2 - n/n+1C2+1 Esto se cumple cuando se analiza el lado izquierdo de la lista. Con la lista derecha sería similar, pero con signo positivo. Por lo dicho anteriormente la complejidad es O(nlogn)

10.InsertarInicio

Cálculo T(n) de la función InsertarInicio		

Código	`void InsertarInicio(TpLista &lista, TpSoli temporal){`
	`  TpLista q=NULL;`
	`  //Crea el nodo q`
	`  q = Registrar(temporal);`
	`  //Lo uno al inicio`
	`  if(lista!=NULL)`
	`  q->sgte = lista;`
	`  lista=q;`
	`}`	

Ubicación	void InsertarInicio(TpLista &lista, TpSoli temporal)	
Desarrollo		
	`TpLista q=NULL;` **2 Operaciones**	Declaración y asignación
	`q = Registrar(temporal);` **2 Operaciones**	Asignación y llamado a la función
	`if(lista!=NULL)` **1 Operación**	1 condición
	`q->sgte = lista;` **2 Operaciones**	Acceso al campo y Asignación
	`lista=q;` **1 Operación**	Asignación.
Resultado	T(n)=7 O(1)	

11. Desencolar

<table>
<tr><td colspan="3">Cálculo T(n) de la función Desencolar</td></tr>
<tr><td>Código</td><td colspan="2"><pre>
 //Eliminar sobre de la cola
void Desencolar(TpSoli &cola){
     TpSoli t=cola;
     cola=cola->sgte;
     cout<<"Eliminando) "<<t->codigo<<" -
"<<"Extranjero"<<" - "<<"Rechazada";
     t->sgte=NULL;
     //Solicitud rechazada
     t->estado=3;
}
</pre></td></tr>
<tr><td>Ubicación</td><td colspan="2">void InsertarInicio(TpLista &lista, TpSoli temporal)</td></tr>
<tr><td colspan="3">Desarrollo</td></tr>
<tr><td></td><td><code>TpSoli t=cola;</code> 2 Operaciones</td><td>Declaración y asignación</td></tr>
<tr><td></td><td><code>cola=cola->sgte;</code>
2 Operaciones</td><td>Asignación y acceso al campo</td></tr>
<tr><td></td><td><code>cout<<"Eliminando) "<<t->codigo<<" -
"<<"Extranjero"<<" - "<<"Rechazada";</code>
2 Operaciones</td><td>Mostrar mensaje y acceso al campo</td></tr>
<tr><td></td><td><code>t->sgte=NULL;</code> 2 Operaciones</td><td>Acceso al campo y Asignación</td></tr>
<tr><td></td><td><code>t->estado=3;</code> 2 Operaciones</td><td>Acceso al campo y Asignación</td></tr>
<tr><td>Resultado</td><td colspan="2">T(n)=10
O(1)</td></tr>
</table>

12.Registrar(temporal)

Cálculo T(n) de la función Registrar	

Código	TpLista Registrar(TpSoli temporal){
	TpLista nuevo=NULL;
	nuevo=new(struct Dni);
	temporal->estado=3;
	/*nuevo->numeroSolicitud=temporal->codigo;*/
	
	cout<<"\nIngrese los datos solicitados\n";
	
	cout<<"1. Primer nombre: ";
	cin>>nuevo->primer_Nombre;
	cout<<"2. Segundo nombre: ";
	cin>>nuevo->segundo_Nombre;
	cout<<"3. Primer apellido: ";
	cin>>nuevo->primer_Apellido;
	cout<<"4. Segundo apellido: ";
	cin>>nuevo->segundo_Apellido;
	cout<<"5. Anio de nacimiento: ";
	cin>>nuevo->anioNacimiento;
	cout<<"6. Lugar nacimiento: ";
	cin>>nuevo->lugar_Nac;
	cout<<"7. Nacionalidad: ";
	
	if(temporal->nacionalidad==1){
	cout<<"Peruano"<<endl;
	nuevo->nacionalidad="Peruano";
	}else{
	cout<<"Extranjero"<<endl;

<table>
<tr><td></td><td colspan="2">

```
            nuevo->nacionalidad="Extranjero";
      }

      cout<<"8. Direccion de su vivienda: ";
      cin>>nuevo->direccion;
      cout<<"9. Indique su sexo: ";
      cin>>nuevo->sexo;
      cout<<"10. Indique su estado civil: ";
      cin>>nuevo->estado;

nuevo->numeroDNI=GenerarDNI(nuevo->anioNacimiento)
;
      cout<<"11. DNI: "<<nuevo->numeroDNI<<endl;
      cout<<endl;
      nuevo->sgte=NULL;
      return nuevo;
}
```

</td></tr>
<tr><td>Ubicación</td><td colspan="2">TpLista Registrar(TpSoli temporal)</td></tr>
<tr><td colspan="3">Desarrollo</td></tr>
<tr><td></td><td>TpLista nuevo=NULL; 2 Operaciones</td><td>Declaración y asignación</td></tr>
<tr><td></td><td>temporal->estado=3;; 2 Operaciones</td><td>Asignación y acceso al campo</td></tr>
<tr><td></td><td>nuevo=new(struct Dni); 2 Operaciones</td><td>Asignación y creación de una estructura en la memoria Heap</td></tr>
<tr><td></td><td>cout<<"1. Primer nombre: ";
cin>>nuevo->primer_Nombre;
3 operaciones</td><td>Mostrar mensaje, lectura y acceso al campo</td></tr>
<tr><td></td><td>En total hay 10 similares al análisi anterior, 1 por cada dato
30 operaciones</td><td>Mostrar mensaje, lectura y acceso al campo</td></tr>
</table>

	`if(temporal->nacionalidad==1)` **2 Operaciones**	Comparación y acceso al campo
	`nuevo->sgte=NULL;` **2 Operación**	Acceso al campo y asignación
	`return nuevo;` **1 Operación**	Un retorno
Resultado	T(n)=41 O(1)	