1.- HOJA DE PRESENTACION:

MATERIA: ESTRUCTURA DE DATOS

INGIENERIA INFORMATICA

T1EDEQUIPO1 (UNIDAD 1)

EQUIPO #1: Castro Ramón David

Alejandro 20010329

Martínez Ramos Rodrigo 20010347

GRUPO: 4PM – 5PM HRS CLAVE: 3a3A

Fecha de entrega: 22-ABRIL-2023

INDICE DEL REPORTE

2) INTRODUCCIÓN (RESUMEN):

En esta unidad se tratarán y verán los temas de como la recursividad tanto en forma iterativa como en recursiva además de que haremos y presentaremos nuestros trabajaos, actividades y programas con códigos de lo que se realizo mediante practicas en el aula del salón de clases o laboratorio.

Además, veremos parte de lo que los trabajos en la memoria así como lo es la memoria estática y la memoria dinámica y lo que tiene que ver con estructura de datos.

MEMORIA ESTÁTICA: Las estructuras de datos estáticas son aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa

MEMORIA DINÁMICA: una **estructura de datos dinámica** es aquella en la que el tamaño ocupado en memoria puede modificarse durante la ejecución del programa.

Cada tipo de estructura dependerá del tipo de aplicación que se requiera. Una típica dentro de las **estructuras de datos estáticas** son los **arrays**:

ESTRUCTURA DE DATOS DINAMICOS ARRAYS (ARREGLOS)

En programación existen **estructuras de datos dinámicas**, es decir, una colección de elementos -nodos- que normalmente se utilizan para dejar asentados registros. A diferencia de un **array** que contiene espacio para almacenar un número fijo de elementos, una **estructura dinámica de datos** se amplía y contrae durante la ejecución del programa. Veamos algunos casos:

LISTAS ENLAZADAS

En las **estructuras de datos**, las listas enlazadas se construyen con elementos que están ubicados en una secuencia. Aquí, cada elemento se conecta con el siguiente a través de un enlace que contiene la posición del siguiente elemento. De este modo, teniendo la referencia del principio de la lista podemos acceder a todos los elementos de la misma.

RECURSIVIDAD:

Recursividad no es una estructura de datos, sino que es una técnica de programación que nos permite que un bloque de instrucciones se ejecute n veces. Remplaza en ocasiones a estructuras repetitivas.

En Java los métodos pueden llamarse a sí mismos. Si dentro de un método existe la llamada a sí mismo decimos que el método es recursivo.

Cuando un método se llama a sí mismo, se asigna espacio en la pila para las nuevas variables locales y parámetros.

Al volver de una llamada recursiva, se recuperan de la pila las variables locales y los parámetros antiguos y la ejecución se reanuda en el punto de la llamada al método.

3) COMPETENCIA ESPECIFICA:

ESPECÍFICA(S):

Aplica la recursividad en la solución de problemas valorando su pertinencia en el uso eficaz de los recursos.

GENÉRICAS:

- Redactar una definición propia del concepto de recursividad después de consultar en diferentes fuentes bibliográficas y comentarla en trinas.
- Enumerar las ventajas y desventajas del uso de la recursividad en una plenaria.

- Habilidad para buscar y analizar información proveniente de fuentes diversas.
- Capacidad de análisis y síntesis
- Habilidad en el manejo de equipo de cómputo
- Capacidad para trabajar en equipo.
- Capacidad de aplicar los conocimientos en la práctica.

- Trasladar un catálogo de problemas iterativos a recursivos, donde distinga el

segmento recursivo y la condición de salida, elaborar un reporte de práctica de ejercicios.

- Desarrollar programas en los cuales se aplique la recursividad y entregar informe.

4) MARCO TEÓRICO:

Los métodos iterativos y recursivos son dos enfoques comunes utilizados en programación para resolver problemas matemáticos y algorítmicos. Los métodos iterativos se basan en la repetición de un proceso o una fórmula matemática para acercarse a una solución, en lugar de calcularla directamente. Este enfoque es útil cuando la solución exacta no es necesaria y se desea una aproximación. Por otro lado, los métodos recursivos se basan en la resolución de un problema a través de la división en subproblemas más pequeños y sucesivos, hasta que se llega a un caso base que se puede resolver directamente. Este enfoque es útil para problemas que se pueden dividir en subproblemas similares y se pueden resolver de forma más elegante y eficiente que con otros métodos. Ambos enfoques tienen sus ventajas y desventajas, y la elección depende del problema y de la eficiencia y complejidad requeridas en su solución.

5) MATERIAL Y EQUIPO:

✓ ***Computadora:*** Cualquier dispositivo electrónico (Computadora o Laptop) que tenga buen funcionamiento en sistema y que pueda ser compatible con un lenguaje de programación para que se pueda instalar, además de una buena memoria de almacenamiento.

✓ ***Software y versión usados:*** En la creación de los programas se utilizó el programa/aplicación de Eclipse que se instala por medio de su JDK Y el IDE que se puede encontrar en la plataforma del lenguaje de programación

✓***Materiales de apoyo para el desarrollo de la practica:*** Documentos de apoyo para la parte teórica presentados por la maestra el cual contenía los códigos y ejemplos de los programas que vemos y pasarlos con nuestros conocimientos al lenguaje de programación que elegimos en este caso Eclipse Versión 2023

6) DESARROLO DE LA PRACTICA:

BINARIO-DECIMAL

PilaC.java | PilaB.java | Ejercicios.java | *DatoSimple.... | *Tools.java | *BinarioDec

```
1  package BinarioDecimal;
2  import Tools.ToolsPanel;
3
4  public class BinarioDecimal {
5
6      public static void BinarioDes(String num) {
7          int res = 0;
8          int base = 1;
9          for (int i = num.length()-1; i>=0; i--) {
10             if (num.charAt(i) == '1') {
11                 res+=base;
12             }
13             base *= 2;
14         }
15         ToolsPanel.imprimePantalla(""+res);
16     }
17 }
```

1. En la línea 1 se crea el paquete BinarioDecimal en la línea 2 se importa el paquete Tools y la clase ToolsPanel, en la línea 4 se crea la clase BinarioDecimal.

2. En la línea 6 se crea un método estático BinarioDes y una cadena caracteres de tipo Sting.

3. En la línea 7 se crea la variable res de tipo int (entero) y esta va a ser o contener el valor de 0.

4. E la línea 8 se crea la variable base de tipo int (entero) y esta va a ser o contener el valor de 1.

5. En la línea 9 se utilizará un ciclo for el cual contendrá que el entero i va a ser igual al tamaño de la cadena después nos marca que el ciclo durara hasta que i sea mayor o igual que 0 y este va a ir decrementando de 1 en 1.

6. Después se va a asegurar que mediante un if las letras o CharAt de i sean igual a 1.

7. En la línea 11 se pondrá el resultado + el resultado de la base.

8. En la línea 13 se mostrará que la base correspondiente será multiplicada *2.

9. En la línea 15 por medio de ToolsPanel se imprimirá un mensaje con la base el cual se le sumara lo que tenga res e imprimirá el cambio de numero binario a número decimal.

Decimal-Octal

```
1  package Practicas;
2
3  import Tools.ToolsPanel;
4
5  public class DecimalOctal  {
6
7      public static void Octal()  {
8
9          int num=0;
10         num = 200;
11         String octal="";
12
13         while(num>0){
14
15             octal=(num%8)+octal;
16             num/=8;
17
18         }
19
20         ToolsPanel.imprimePantalla(octal);
21
```

10. En la línea 1 se crea el paquete Practicas en la línea 3 se importa el paquete Tools y la clase ToolsPanel, en la línea 5 se crea la clase Decimal-Octal.

11. En la línea 7 se crea el método public static void Octal.

12. En la línea 9 se crea num de tipo int que vale o es igual a 0.

13. En la línea 10 num se iguala o vale 200.

14. En la línea 11 s pone el Sting o la cadena que se ingresara como octal.

15. En la línea 13 se pone un ciclo while (mientras) que significa que el programa durara hasta que numero sea menor que 0.

16. En la línea 15 significa que Octal va a ser igual a la división en modulo de num%8 y a este resultado se le sumara lo que octal tenga hasta que se realice este paso.

17. E la línea 16 dice que num se va a dividir entre el numero 8 ya que es octal.

18. En la línea 20 se imprime en la pantalla el octal correspondiente al número que en ese entonces octal tiene.

FACTORIAL

```
1  package Prueba;
2
3  public class Factorial {
4
5      public static double factorial(int dato) {
6
7          int f=1, c=1;
8          while(c<=dato) {
9              f*=c;
10             c++;
11         }
12         return f;
13     }
14
```

19. En la línea 1 se crea el paquete Prueba en la línea 3 se crea la clase Factorial.

20. En la línea 5 se crea un método public static de tipo double llamado factorial y una variable de tipo entero llamada dato.

21. En la línea 7 se crea una variable de tipo entero que es f y vale 1 y otra que es c e igual vale 1.

22. En la línea 8 se crea un ciclo (while) que dice que si c es mayor o igual que 0 se mete al ciclo.

23. dentro del ciclo en la línea 9 se dice que lo que tenga f se va a multiplicar por lo que valga c, además cada vez que el ciclo se cumpla la condición a c se le incrementara 1 esto sucede en la línea 10.

24. Una vez que el ciclo termine en la línea 12 se retornará lo que tenga f y se imprimirá el resultado de la factorial.

FIBONACCI

```
1  package Prueba;
2  import Tools.ToolsPanel;
3  public class Fibonacci {
4
5      public static void fibonacci(int n) {
6          int a = 0;
7          int b = 1;
8          String res="";
9          for (int i = 0; i<n; i++) {
10             res=res+","+a;
11             int aux = a;
12             a = b;
13             b = aux + b;
14         }
15         ToolsPanel.imprimePantalla(res);
16     }
17
18     public static String recFibo(int n, String res, int aux, int a, int b) {
19         if(n>0) {
20             return recFibo(n-1, res=res+","+a, aux=a, a=b, b=aux+b);
21         }else {
22             return res;
23         }
24     }
25
26 }
27
```

25. En la línea 1 se crea el paquete Prueba en la línea 2 se importa el paquete Tools y la clase ToolsPanel, en la línea 3 se crea la clase Fibonacci.
26. En la línea 5 se crea el método public static void fibonacci y n de tipo entero (int).
27. En las líneas 6 y 7 se crean las variables a con valor de 0 y b con valor de 1.y en la línea 8 res que es una cadena de tipo Sting.
28. En la línea 9 se pone un ciclo for con un entero que es i y que vale 0, después si i es menor que n hasta ahí se terminara el ciclo y se va incrementando de 1 en 1.
29. En la línea 10 se hace que res (resultado) va a ser lo que contenga res al final del ciclo + lo que tenga a.
30. En la línea 11 aux va a ser igual a a.

31. En la línea 12 a va a ser o valer lo que valga b.

32. En la línea 13 se muestra lo que vale b que es igual a lo que valga aux + lo que tenga b.

33. En la línea 15 se muestra que se imprimirá o mostrará lo que valdrá res al terminar el ciclo.

34. En la línea 18 se crea un método que se llama recFibo de tipo Cadena o Sting y con las variables n, aux a y b de tipo int y res de tipo String.

35. En la línea 19 se pone un ciclo for que diga que se ejecutara hasta que n sea menor que 0.

36. En la línea 20 se retornará o mandará a llamar a la recFibo el cual será cando a n se le quite 1 hasta que ya no haya más números, igual mostrara el resultado ósea res + lo que tenga que tiene aux=a, a=b y b=aux+b.

37. En la línea 21 habrá un else que diga que sino entonces en la línea 22 solo se retornara res y se mostrara la secuencia de números fibonacci final.

Potencia

```
1  package Practicas;
2
3  import Tools.ToolsPanel;
4
5  public class Potencia {
6
7      public static void potencia(int num, int pot) {
8          int aux=num;
9          for(int i=1; i<pot; i++)
10             aux*=num;
11
12         ToolsPanel.imprimePantalla(""+aux);
13
14     }
15
16     public static int recPot(int num, int pot, int aux) {
17         if(pot!=0) {
18             aux*=num;
19             return recPot(num, pot-1, aux);
20         }
21         return aux;
22     }
23
24 }
```

Esta clase será la de potencia en la cual tenemos el iterativo y el recursivo.

Iterativo:

Creamos un método estático llamado potencia la cual recibirá dos parámetros, recibiremos num que será el número que queremos elevar y la pot que será la potencia a lo que lo elevaremos, también ocuparemos una variable aux que guardare el valor de num para poder manipularlo libremente, iniciamos un for el cual tendrá como condición lo siguiente, creamos una variable int i que iniciara con 1, mientras i sea menor que pot, a i le iremos aumentando de uno en uno, mientras esta condición se cumpla multiplicaremos lo que ya tenga auxiliar por num, una vez terminada la condición se imprimirá el valor de aux.

Recursivo:

Creamos un método estático llamado recPot el cual recibirá 3 parámetros enteros, el numero a elevar (num), la potencia (pot) y el auxiliar (aux), como es un método recursivo se ocupara la memoria stack en la cual cada vez que se llame a este método se apilara, uno sobre otro mientras se cumpla la condición de que pot sea diferente de 0 (pot!=0) mientras haga esto se multiplicara lo que tenga aux por num y se volverá a llamar a el método recPot al cual le mandamos num, pot pero le disminuimos uno y aux, esto se repetirá hasta que la condición ya no se cumpla, en este momento se irán eliminando los espacios en el stack y se regresaran los resultados, el principio.

Lista Factorial

Este método se llama lista factorial, el cual es un método recursivo, en la cual ocuparemos el método de factorialR para que funcione correctamente este método recibe un parámetro de tipo entero que llamaremos k, primero comparamos que k sea menor igual que 15, mientras esta condición se cumpla retornara lo que le traiga factorialR al cual le enviaremos la variable k más listafactorial a esta le mandaremos k+1 asi haciendo este un método

```
16    public static String listaFactoriales(int k) {
17        if(k<=15) {
18            return "\n"+k+" Factorial "+ factorialR(k, 1)+listaFactoriales(k+1);
19        }else {
20            return "";
21        }
22    }
23
24 }
```

recursivo el cual se ira apilando en el stack, cuando k llegue a 15 empezará a regresare y concatenar todos los resultados

Lista Octal

```
13    public static String listaOctal(int k) {
14        if(k<=20) {
15            return "\n"+k+" Octal "+decimalOctalR(k)+listaOctal(k+1);
16        }else {
17            return "";
18        }
19    }
20
21 }
22
```

Este método se llama lista octal, el cual es un método recursivo, en la cual ocuparemos el método de decimalOctalR para que funcione correctamente este método recibe un parámetro de tipo entero que llamaremos k, primero comparamos que k sea menor igual que 20, mientras esta condición se cumpla retornara lo que le traiga decimalOctalR al cual le enviaremos la variable k mas lista octal a esta le mandaremos k+1 asi haciendo este un método recursivo el cual se ira apilando en el stack, cuando k llegue a 20 empezará a regresare y concatenar todos los resultado.

Matrices

```java
package Iterativo;

import ToolsPanel.Tools;

public class Matrises {

    //iterativo

    public static void leerMatris(int a[][]) {

        int i,j;
        for(i=0; i<a.length; i++) {
            for(j=0; j<a[0].length; j++) {
                a[i][j]=Tools.leerInt("Inserte un numero");
            }
        }
        verMatris(a);

    }

    public static void verMatris(int a[][]) {

        int i,j;
        String cad="";
        for(i=0; i<a.length; i++) {
            for(j=0; j<a[0].length; j++) {
                cad+="["+a[i][j]+"] ";
            }
            cad+="\n";
        }
        Tools.imprimePantalla(cad);
    }
```

A esta clase funcionara para agregar y ver datos a una matriz de forma iterativa y recursiva.

Iterativa

Creamos un método estático que no retornara nada, llamamos a la clase leer Matriz, la cual traerá un parámetro, el cual será una matriz de tipo entero, creamos dos variables de tipo int, iniciamos un for el cual tendrá como condición lo siguiente, i iniciara con 0, mientras i sea menor que el tamaño de las filas nuestro arreglo, aumentaremos i uno por uno (i=0; i<a.length; i++), iniciamos otro for el cual será j iniciara con 0, mientras j sea menor al tamaño de las columnas de nuestro arreglo, j aumentara de uno por uno (j=0; j<a[0].length; j++), así lograremos recorrer toda nuestra matriz, cada ves que j incremente un numero le pediremos al usuario que nos digite el número que quiera insertar, al pasar a la siguiente columna ósea cuando i aumente j volverá a iniciar con 0, así recorreremos la columna siguiente y se vuelve a repetir hasta que terminamos con toda la matriz.

Creamos un método estático que no retornara nada, llamado ver Matriz el cual traerá como parámetro una matriz de tipo int, iniciamos dos variables de tipo int y una variable llamada cad de tipo String la cual se encargara de guardar los datos de la matriz para poder imprimirlos después, iniciamos un for el cual tendrá como condición lo siguiente, i iniciara con 0, mientras i sea menor que el tamaño de las filas nuestro arreglo, aumentaremos i uno por uno (i=0; i<a.length; i++), iniciamos otro for el cual será j iniciara con 0, mientras j sea menor al tamaño de las columnas de nuestro arreglo, j aumentara de uno por uno (j=0; j<a[0].length; j++), cada vez que j aumente concatenaremos lo que ya tenia cad guardado mas el dato que se encuentre en la matriz en la posición i y j, cada que i aumente le daremos un salto de línea para que la información salga ordenadamente, al final cuando toda la matriz se termine de recorrer, se imprimirá todo lo que guardo la variable cad.

```
34      //recursivo
35
36      public static String leerMatrisR(int a[][], int i, int j) {
37          if(j<a[0].length) {
38              int n=Tools.leerInt("Imprime un numero");
39              a[i][j]=n;
40              return " ["+a[i][j]+"]={"+decimalOctalR(n)+"} "+leerMatrisR(a, i, j+1);
41          }
42          return "";
43      }
44
45      public static String leerMatrisRi(int a[][], int i, int j) {
46          String cad="";
47          if(i<a.length) {
48              return cad+="\n"+leerMatrisR(a, i, 0)+leerMatrisRi(a, i+1, j);
49          }else {
50              return "";
51          }
52      }
53
54      //llenar una matris de 3*4 con valores enteros, he imprimir el octal de cada uno de los elementos
55
56      public static String decimalOctalR(int n) {
57          if(n!=0) {
58              return decimalOctalR(n/8)+n%8;
59          }else {
60              return "";
61          }
```

Ahora pasemos con el método recursivo, utilizando la forma recursiva no necesitaremos tener un método de impresión, ya que ira guardando en una variable para su impresión al mismo tiempo que se guarda en nuestra matriz.

Creamos un método estático que regresara una cadena así que le datos el tipo String llamamos a nuestro método leer Matriz R el cual recibirá 3 parámetros, recibiremos la matriz de tipo int y dos variables mas del mismo tipo, para lograr recorrer la matriz completa necesitaremos hacer la forma recursiva indirecta, asi que necesitaremos crear otro método con las mismas características del anterior pero llamándolo leer Matriz Ri, el método leer Matriz R se encargara de ir recorriendo las columnas mientras que leer Matriz Ri se encargara de ir recorriendo las filas.

Método leer Matriz R:

En este método será donde iremos guardando cada dato en nuestra matriz, empezamos verificando que j sea menor que el tamaño de nuestra matriz en las columnas (j<a[0].length) mientras esta condición se siga cumpliendo le pedirá al usuario un numero el cual lo guardaremos en una variable, una ves insertado el numero lo guardaremos en la matriz, un paso extra es sacar el octal de cada numero así que llamamos a la función decimalOctalR y lo concatenamos igualmente, volvemos a llamar a la función leer Matriz R pero esta vez a j le aumentamos uno y la variable i y el arreglo los mandamos iguales, una vez que la condición ya no se cumpla simplemente se regresaran los resultados.

Método leer Matriz Ri:

Este método será donde iremos manejando las filas, primero crearemos una variable de tipo String llamada cad en la cual se guardaran todos los resultados ordenadamente para mostrarlos una ves terminado de llenar la matriz, comparamos si i es menor que el tamaño de nuestra matriz en las filas (i<a.length), mientras esta condición se cumpla a cad le concatenaremos los resultados que traiga leer Matriz R y los colocare en una columna, una vez hecho esto, se volverá a llamar a leer Matriz Ri pero esta ves le aumentaremos uno a i para pasar a la siguiente, una vez que esta condición ya no se cumpla todos los resultados se almacenaran en cad ordenadamente y se imprimirán.

Lista Ordenada

```java
package Proyecto;

import Tools.ToolsPanel;

public class buscar {

    private int datos[];
    private byte p;

    public buscar(byte tam) {

        datos= new int[tam];
         p=-1;

    }

    public void imprimeDatos()
     {
         String cad="";
         for (int i = 0; i <=p; i++) {
             cad+=i+"["+datos[i]+"]"+"\n";

        }
         ToolsPanel.imprimePantalla("datos del array\n"+cad);
     }

    public boolean validaVacio()
     {
        return (p==-1);
     }
```

Esta clase gestionara un arreglo ordenado a la cual llame buscar, para esto cada dato insertado se comparará y se colocara en su lugar indicado de la lista, al eliminar se tendrá que recorrer los dígitos posteriores a este una casilla antes, para modificarlo se tendrá que verificar que el numero no sea mayor o menor a los dígitos a su alrededor.

Primero creamos nuestro constructor el cual recibirá un parámetro que será el tamaño de nuestro arreglo de tipo byte el cual guardaremos en la variable llamada tam, iniciamos nuestro arreglo con el tamaño dado anterior mente y p lo igualamos a -1, este será nuestro apuntador, al inicio se inicia con este numero ya que no existe ningún dato dentro del arreglo.

Creamos nuestro método imprimeDatos el cual no regresara nada, creamos una variable de tipo String llamada cad en la cual guardaremos todos los dígitos que ay en la pila en una cadena para lograr imprimirla, iniciamos un for iniciamos una variable (i) con 0 la condición será que mientras i sea menor o igual a nuestro apuntador(p) esta se ira incrementando en 1, mientras esta condición se cumpla se estará repitiendo lo siguiente, a nuestra variable cad se le concatenara el número que tiene i mas lo que

tenga datos en la posición i, al final de este bucle se imprimirá todo lo que guardo la variable cad.

Creamos un método llamado validaVacio el cual retornara un true si nuestro arreglo esta vacío o un false si tiene algo dentro ya que nuestro apuntador (p) es el que lleva el conteo de cuantos dígitos hay en nuestro arreglo, si es -1 significa que esta vacío.

```
32    public byte buscarSecOrdenado(int dato) {
33
34            byte i=0;
35            while(i<=p && datos[i] < dato) {
36                i++;
37            }
38
39            return (byte) ((i>p || datos[i] > dato)?-i:i);
40
41    }
```

Creamos nuestro método buscarSecOrdenado el cual recibe un parámetro de tipo entero que guardaremos en una variable dato, creamos una variable de tipo byte que se llamara i he iniciara con 0, creamos un bucle while el cual tendrá como condición que mientras nuestra variable i sea menor a nuestro apuntador (p) y datos en la posición i sea menor que el dato recibido se repita la siguiente condición, i aumentara en 1, una vez terminado este bucle retornara como byte si i sobrepaso a p o dato en la posición i sobrepaso a el dato recibido retornara la posición donde debe de ir y para saber cunado es donde debe de ir a donde esta se manda como numero negativo, si fue encontrado el numero simplemente se manda su posición.

```
43    public void AgregaOr() {
44
45        if(validaVacio()) {
46
47            datos[p+1]=ToolsPanel.leerInt("Ingrese el numero");
48            p++;
49
50
51        }else {
52
53            int dato=ToolsPanel.leerInt("Ingrese el numero");
54            int pos=buscarSecOrdenado(dato);
55            if(pos>=0) {
56
57                ToolsPanel.imprimePantalla("Ya existe");
58
59            }else {
60                pos*=-1;
61                for(int k=p; k>=pos; k--) {
62                    datos[k+1]=datos[k];
63                }
64                p++;
65
66            }
67            datos[pos]=dato;
68        }
69
70    }
```

Creamos nuestro método AgregarOr primero verificamos que nuestra lista no este vacía, en caso de que así sea simplemente se agregara el numero a la lista y se incrementara nuestro apuntador (p), en caso de que contenga datos primero le pediremos el nuevo dato al usuario el cual guardaremos en una variable llamada dato de tipo entero, después buscamos la posición en donde debe de ir o si existe ese digito, para eso llamamos al método buscarSecOrdenado y le mandamos el dato que nos dio el usuario, si nos regresa un numero positivo significa que ese numero ya existe así que simplemente le mandamos al usuario el mensaje de que ese dato ya existe en la lista, si nos regresa un numero negativo significa que ese dato no existe en la lista pero va en esa posición, así que convertimos el numero negativo que nos mando en positivo, creamos un bucle for el cual tendrá como condición que iniciamos una variable (k) con el numero que tenga nuestro apuntador (p), mientras k sea menor o igual a la posición en la que debe de ir disminuiremos k uno por uno, mientras esto se cumpla moveremos cada dato, desde la posición donde va el nuevo digito una casilla después así tendremos el espacio libre donde al final simplemente lo insertaremos.

Creamos un método llamado eliminar el cual primero validara si nuestro arreglo esta vacío, si esta vacío simplemente mandaremos un mensaje que diga arreglo vacío, en caso contrario le pediremos al usuario que ingrese el digito que quiere eliminar, y lo buscaremos a través del método buscarSecOrdenado si nos regresa un numero positivo crearemos un bucle for el cual tendrá como condición lo siguiente, iniciaremos una variable (k) y la igualaremos a la posición donde esta el dato por el usuario (pos), mientras k sea menor o igual a nuestro apuntador (p), k le aumentaremos 1, mientras se cumpla esta condición iremos recorriendo todos los dígitos desde la posición donde esta el numero a eliminar hacia una posición atrás, así se sobre escribirán y se eliminara el numero elegido, una ves terminado

```
72    public void eliminar() {
73
74        if(validaVacio()) {
75            ToolsPanel.imprimeError("Array vacio");
76        }else {
77            int dato = ToolsPanel.leerInt("Que numero quiere eliminar");
78            int pos = buscarSecOrdenado(dato);
79            if(pos>=0) {
80                for(int k=pos; k<=p; k++) {
81                    datos[k]=datos[k+1];
82                }
83                p--;
84            }else {
85                ToolsPanel.imprimeError("El numero no existe");
86            }
87        }
88
89    }
```

le restaremos uno a nuestro apuntador, en caso de que el numero que nos regresó la búsqueda del numero sea negativo le mandaremos un mensaje al usuario que diga el numero no existen en la lista.

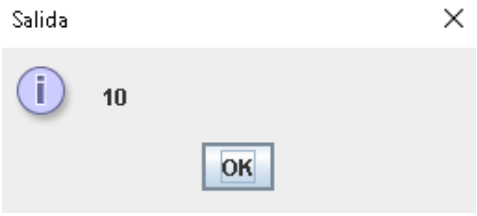
```
91  public void modificar(int dato) {
92      
93      int pos = buscarSecOrdenado(dato);
94      boolean ban=true;
95      if(pos>=0) {
96          do {
97              int dato2=ToolsPanel.leerInt("A que numero lo desa cambiar");
98              if(pos!=0 && pos!=p) {
99                  if(dato2>=datos[pos+1] || dato2<=datos[pos-1]) {
100                     ToolsPanel.imprimeError("El numero tiene que estar entre "+datos[pos-1]+" y "+ datos[pos+1]);
101                 }
102                 else {
103                     datos[pos]=dato2;
104                     ban = false;
105                 }
106             }else{
107                 if(pos>=p && dato2<=datos[pos-1]) {
108                     ToolsPanel.imprimeError("El numero no puede ser menor que "+datos[pos-1]);
109                 }else {
110                     if(pos==0 && dato2>=datos[1]) {
111                         ToolsPanel.imprimeError("El numero no puede ser mayor que "+datos[1]);
112                     }
113                     else {
114                         datos[pos]=dato2;
115                         ban = false;
116                     }
117                 }
118             }
119         }while(ban);
120     }else {
121         ToolsPanel.imprimeError("Este numero no existe");
122     }
```

Creamos nuestro método modificar el cual recibirá un parámetro llamado dato que será de tipo entero, este dato será el numero que queremos modificar, primero buscamos este número en la lista con el método buscarSecOrdenado, creamos una bandera será de tipo booleana que inicia con true, si el número que nos regresó la búsqueda es negativo significa que no existe así que solo le mandamos al usuario un mensaje de que este numero no existe, si nos regresa un numero positivo quiere decir que si existe así que primero crearemos un do el cual contendrá lo siguiente, primero le pedimos al usuario a que numero lo desea cambiar y compararemos que este numero es diferente a la primera pisicion de la lista o y a la ultima, si es verdadero se comparara que el nuevo numero sea menor al de la siguiente posición y mayor al de la anterior si esto se cumple se modificara sino se mandara un mensaje que le indique lo anterior, si el numero esta en la primera posición o en la ultima entonces entrara en el segundo else, y comparara que el numero sea mayor o menor dependiendo de donde se encuentre si la condición se cumple se modificara el dato sino se mandara un mensaje de error, si se llega a modificar el dato la variable bandera cambiara a false indicando que se completó la condición y saldrá del bucle.

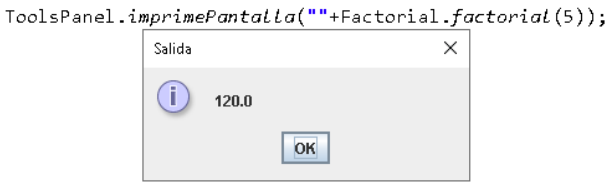
7) RESULTADOS:

Resultado de Binario Decimal dandole el valor de “1010”.

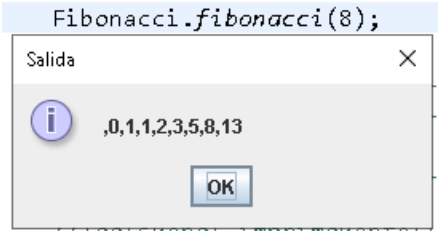
Resultado de Decimal Octal dandole el valor de 8.



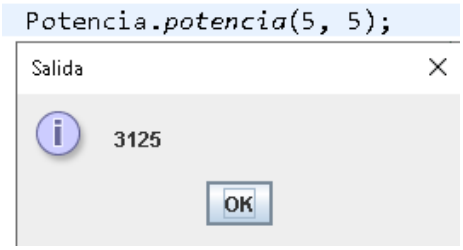
Resultado de Factorial dandole el valor de 5.



Resultado de Factorial dandole el valor de 8.



Resultado de Potencia dandole el valor de 5, 5.

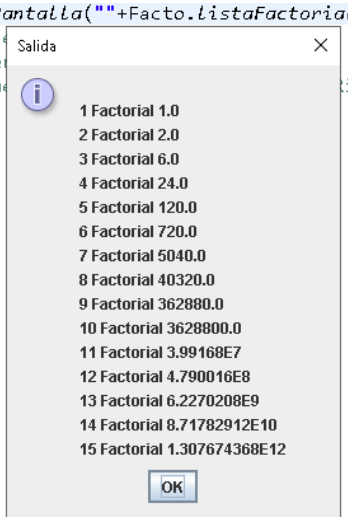


Resultado de lista octal epezando desde 1 hasta 10.

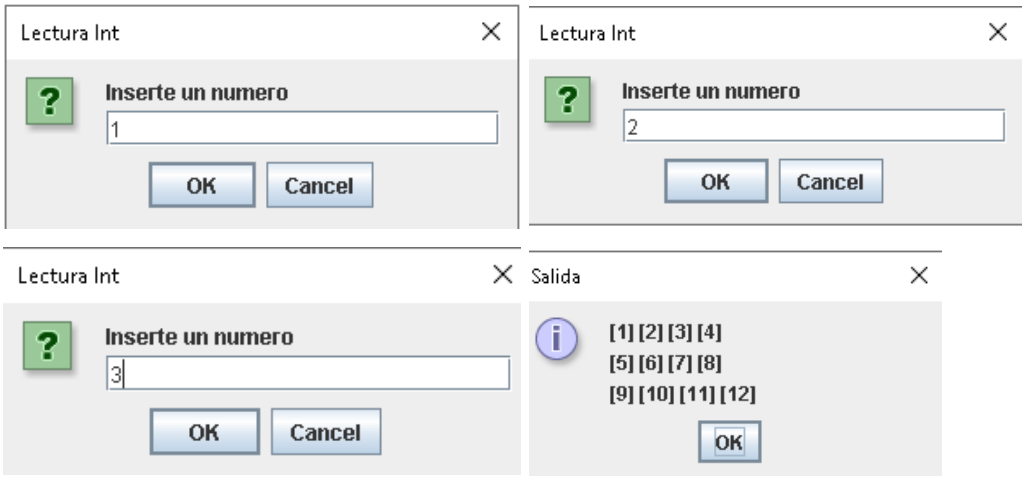
```
ToolsPanel.imprimePantalla(""+ListaOctal.listaOctal(1));
```

Resultado de lista factorial empezando desde 1 hasta 15.

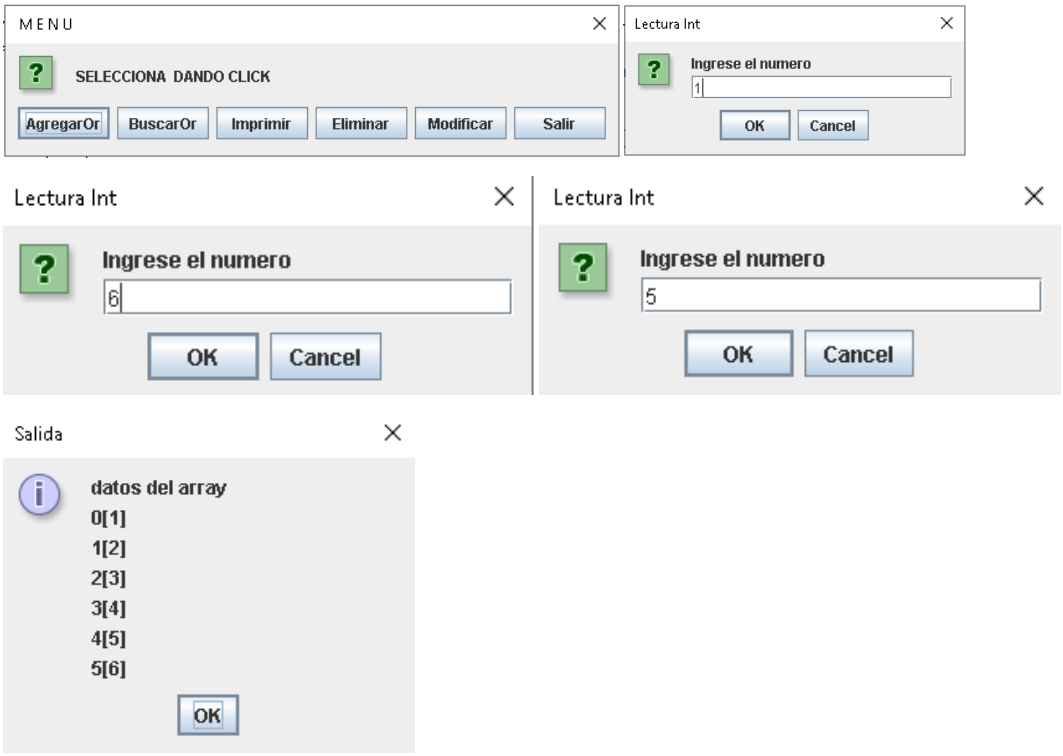
```
Tools.imprimePantalla(""+Facto.listaFactoriales(1));
//int a[][]= n
//Matrises.lee
//Tools.imprim
```



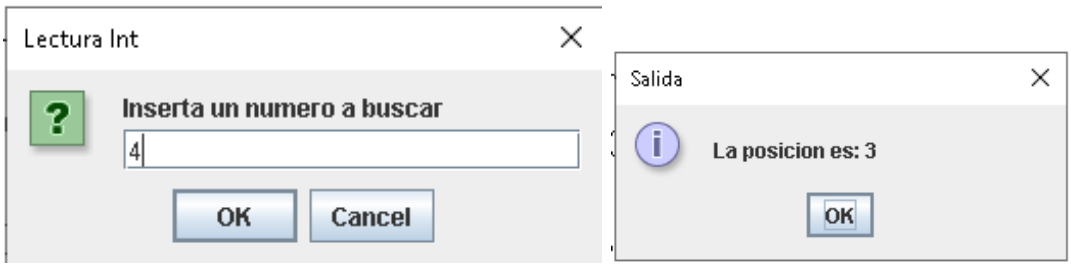
Resultados de Matrices.



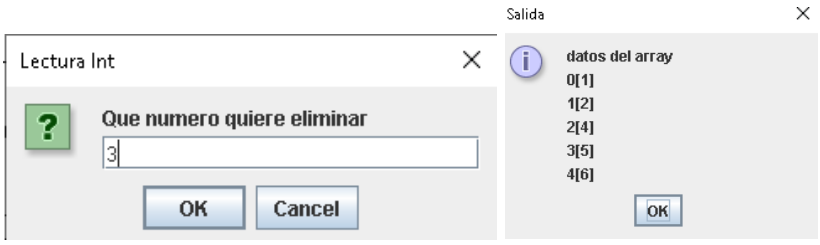
Resultado de agregar ordenado



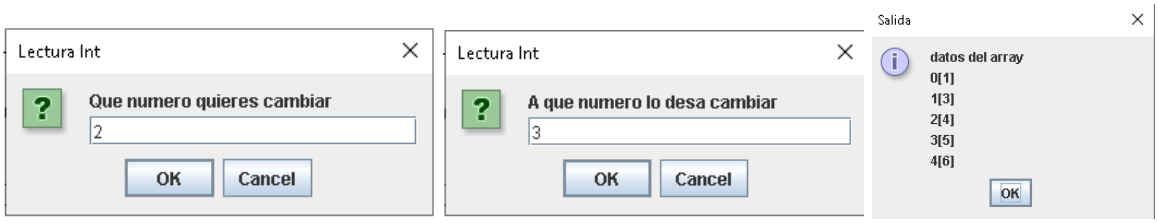
Resultado de buscar ordenado



Resultado de eliminar ordenado



Resultado de modificar ordenado



8) CONCLUSIONES:

En conclusión, en esta unidad aprendimos mucho sobre los tipos de memoria, listas Enlazadas y Recursividad.

Aprendimos que la memoria dinámica se utiliza porque es la forma más sencilla de almacenar una variable y no se modifica su tamaño, al menos en el tiempo que dura la ejecución.

La memoria dinámica es un espacio de almacenamiento.

Podemos solicitar espacios de almacenamientos, pero también liberarlos. Se acomodan a las necesidades de cada proceso solicitante.

Cuando un proceso termine o sea retirado de memoria el gestor de memoria puede devolver el espacio vacante.

La asignación dinámica de memoria resulta ser muy ventajosa a la hora de evitar el desperdicio de memoria,

* Gracias al uso de la asignación dinámica de memoria el procesamiento de información resulta mucho más rápido y eficiente con lo cual se logra optimizar el uso del computador.

* Administrar la memoria es una de las tareas más complejas que lleva a cabo el sistema operativo

LISTAS ELAZADAS:

Las listas enlazadas permiten agregar y eliminar nodos en cualquier punto de la lista en tiempo constante, siempre y cuando estén ya definidos o se puedan localizar; lo que no permite es un acceso aleatorio.

Las listas enlazadas es que hasta cierto punto son las más prácticas entre las colas y pilas dado que gracias a sus características que posee es posible en todo momento estar interaccionando con todos los datos o incluso ir eliminando datos o agregar en cualquiera de sus posiciones todo basta nomas con saber cómo acomodar sus nodos para que estén bien apuntados, que vaya que tiene mucha gracia el saber cómo ir acomodando los nodos correctamente.

Métodos Recursivos:

En definitiva, no hay una norma establecida acerca de cuándo implementar una solución recursiva y cuándo una iterativa. Existen varios factores que ya hemos mencionado a lo largo del artículo y que hay que tener en cuenta respecto a la recursividad, que se resumen en:

La recursividad consume mucha memoria y tiempo de ejecución.
La recursividad puede dar lugar a la redundancia (resolver el mismo problema más de una vez)
A veces es más sencillo encontrar una solución recursiva que una iterativa

9) Bibliografía:

- https://blog.soyhenry.com/que-es-una-estructura-de-datos-en-programacion/#:~:text=Las%20estructuras%20de%20datos%20est%C3%A1ticas,ocupado%20en%20memoria%20puede%20modificarse

https://prezi.com/hoguxp55jpcl/asignacion-dinamica-de-memoria-y-administracion-de-memoria-c/#:~:text=Conclusiones%3A,optimizar%20el%20uso%20del%20computador.