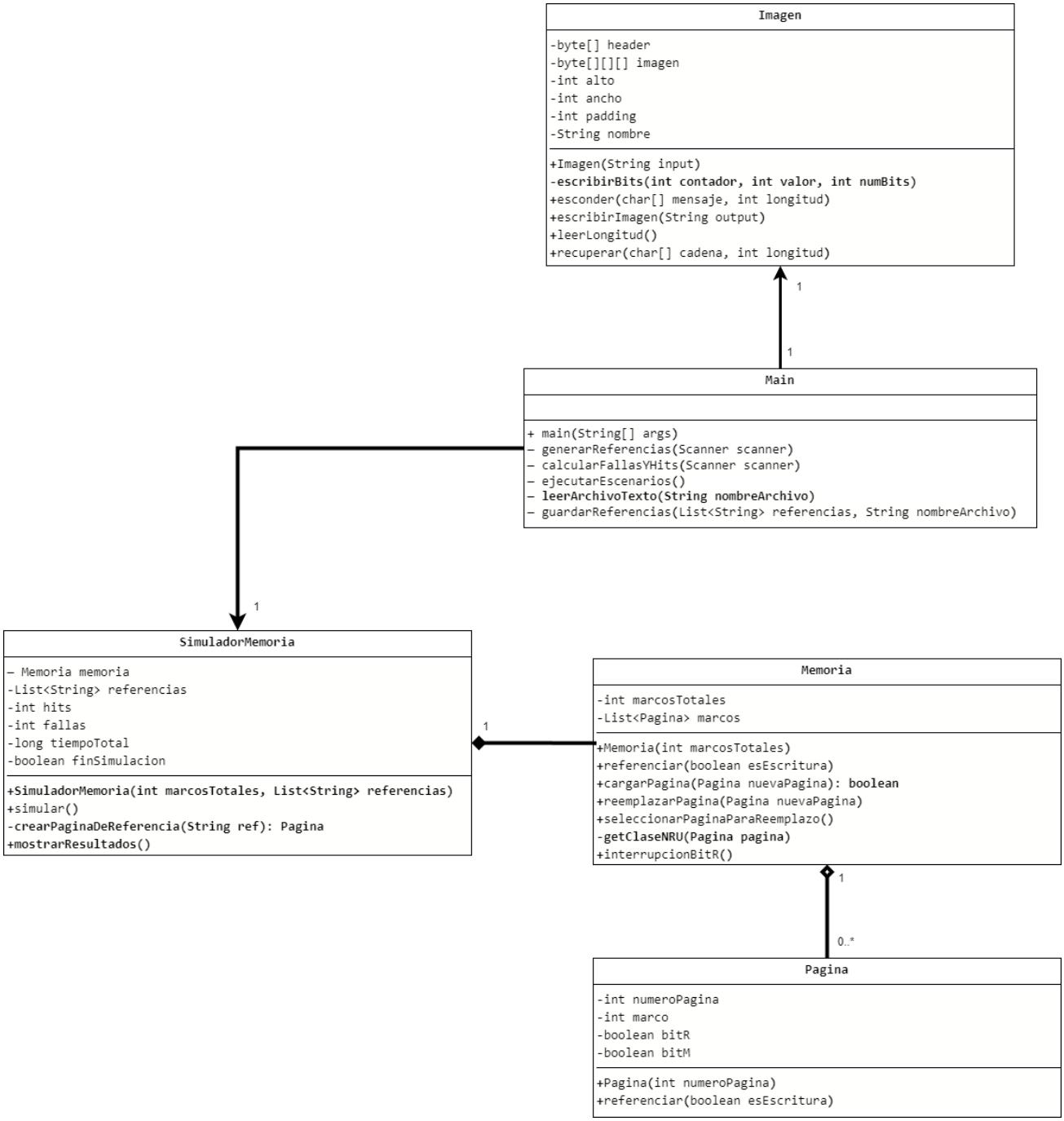
Informe Caso 2

Equipo #4

- María Alejandra Pinzón 202213956
- Manuel Santiago Prieto Hernández 202226947
- Alison Daiana Aristizabal Garcia 202125179

Contenido:

Diagrama de Clases



Descripción del algoritmo usado para generar referencias de página

Lectura de parámetros

Tamaño de página (tamanoPagina): El usuario ingresa el tamaño de la página en bytes, con esto se divide la memoria en páginas y como se calcularán las referencias de página.
Archivo de la imagen (nombreImagen): El usuario ingresa el nombre del archivo de la imagen que se utilizara para esconder el mensaje.
Archivo de mensaje (nombreMensaje): El usuario ingresa el nombre del archivo que contiene el mensaje que se va a esconder.

Carga de la imagen y del mensaje

Creación de la Imagen: Se instancia un objeto de la clase Imagen, donde se carga la imagen BMP y se almacena en una matriz tridimensional (imagen[fila][columna][color])
Lectura del mensaje: Se lee el mensaje del archivo de texto y se convierte en un array de caracteres (char[] mensajeChars).

Esconder el mensaje en la imagen

Se invoca el método imagen.esconder(mensajeChars, tamanoMensaje) donde se modifica la matriz de la imagen para esconder el mensaje en los bits menos significativos de los componentes de color de los píxeles.

Cálculo de parámetros para las referencias

Antes de poder generar las referencias se calculan algunos parametros que van a ser necesarios más adelante.
Primero se necesitan las dimensiones de la imagen para esto se llaman las funciones imagen.alto() e imagen.ancho(), estos se guardan en las variables nf (número de filas) y en nc (número columnas). En segundo lugar, se extrae la longitud del mensaje escondido con la función imagen.leerLongitud(). Luego se calcula el número de referencias (nr) que van a ser necesarias, esto se calculó con la formula (longitudMensaje *17) + 16, que representa el número total de operaciones que deben realizarse para procesar el mensaje. Esta fórmula tiene en cuenta que por cada carácter del mensaje se necesita 1 operación que inicialice el carácter, luego se realizan 8 iteraciones por carácter ya que para su recuperación se trabaja bit a bit, dentro de la iteración se realizan dos operaciones más que son lectura de imagen y escritura en el mensaje, lo que da como resultado el 17. Finalmente se tienen en cuenta los 16 bytes iniciales. Después con los datos de nc y nf calcula el tamaño de la imagen en bytes, con este tamaño y con la longitud del mensaje se calcula el tamaño total de datos, para finalmente calcular el número total de páginas dividiendo el tamaño total con el tamaño de página y redondeándolo hacia arriba para que sea un entero.

Generación de referencias de página

Para la generación de referencias de página primero se procesan los primero 16 Bytes de inicio de la imagen del encabezado, luego ya se inicia con las referencias del mensaje para esto se calculan las páginas que ocupara la imagen para saber desde que punto el mensaje empieza a ocupar páginas. Se inicia una iteración hasta que se llega a la longitud del del mensaje, por cada carácter se genera una referencia de escritura, cada carácter está compuesto por 8 bits para los cuales se hace uso de un for para tener en cuenta cada uno de estos bits, dentro de este for se calculan las posiciones dentro de la imagen, primero el byte de posición

en el que se encuentra, la fila en la que esta, la columna y en que componente de color se encuentra, si es R,G o B. luego se revisa en que página se encuentra y el desplazamiento que se debe realizar entre páginas. Para las referencias lee primero la imagen y da sus datos de referencia y luego añade la referencia del carácter bit a bit, cuando termina con los bits de carácter incrementa el numBytes,. Luego se asegura que a la página tenga suficiente espacio para guardar otra referencia con el desplazamiento, el cual es calculado tanto para la imagen como para el mensaje, en caso de que no se avanza a la siguiente página y se incrementa la posición del carácter para revisar el siguiente caracter.

Descripción de las estructuras de datos usadas para la paginación

1. Clase Memoria:

```java
public class Memoria {
    int marcosTotales;
    List<Pagina> marcos;

    public Memoria(int marcosTotales) {
        this.marcosTotales = marcosTotales;
        this.marcos = new ArrayList<>();
    }
```

- List<Pagina> marcos: Lista que representa los marcos de página actualmente cargados en la memoria. Cada elemento en esta lista es una instancia de la clase Pagina, que representa una página cargada en memoria.
- int marcosTotales: Número total de marcos de página disponibles en la memoria. Este valor determina la capacidad de la memoria en términos de cuántas páginas se pueden mantener en la memoria al mismo tiempo.

Actualización:

1. Cuando se accede a una página (lectura o escritura), el programa verifica si la página ya está en la lista de marcos (hit) o si no está presente (falla).

```java
public synchronized boolean cargarPagina(Pagina nuevaPagina) {
    for (Pagina pagina : marcos) {
        if (pagina.numeroPagina == nuevaPagina.numeroPagina) {
            pagina.referenciar(nuevaPagina.bitM); //Se actualiza bitR y bitM si es escritura
            return true; // Hit
        }
    }
    //Falla
    if (marcos.size() < marcosTotales) {
        marcos.add(nuevaPagina);
    } else {
        reemplazarPagina(nuevaPagina);
    }
    return false;
}
```

2. En caso de un "hit", se actualiza el estado de la página con la función referenciar, que ajusta el bit R (referenciado) y el bit M (modificado) según sea una lectura o escritura.

3. En caso de una "falla", si hay espacio disponible en la lista de marcos, se agrega la nueva página. Si no hay espacio, se selecciona una página para ser reemplazada usando el algoritmo de NRU (Not Recently Used)

```
private synchronized Pagina seleccionarPaginaParaReemplazo() { // utiliza NRU
    List<List<Pagina>> clases = new ArrayList<>(initialCapacity:4);

    for(int i = 0; i<4;i++){
        clases.add(new ArrayList<>()); // Se crea 1 array por cada clase
    }
    // paginas dentro de su clase
    for(Pagina pagina: marcos){
        int clase = getClaseNRU(pagina);
        clases.get(clase).add(pagina);
    }

    for (List<Pagina> clase : clases) {
        if (!clase.isEmpty()) {
            return clase.get(index:0); // como es un ciclo se toma la clase 0, luego 2 l luego 3  y por ultimo 4
        }
    }
    return null;
}
```

```
private int getClaseNRU(Pagina pagina){
    if(!pagina.bitR && !pagina.bitM ) return 0; // R=0, M=0
    else if(!pagina.bitR && pagina.bitM ) return 1; // R=0, M=1
    else if(pagina.bitR && !pagina.bitM ) return 2; // R=1, M=0
    else { return 3; } // R=1, M=1

}
```

4. El método reemplazarPagina se encarga de eliminar la página seleccionada para reemplazo y agregar la nueva página.

```
private synchronized void reemplazarPagina(Pagina nuevaPagina) {
    Pagina paginaReemplazo = seleccionarPaginaParaReemplazo();
    marcos.remove(paginaReemplazo);
    marcos.add(nuevaPagina);
}
```

5. El método interrupcionBitR se llama periódicamente para reiniciar el bit R de todas las páginas, siguiendo el algoritmo NRU. El encargado de llamarlo es el threadActualizacionBitR

```
public synchronized void interrupcionBitR(){
    for(Pagina pagina: marcos){
        pagina.bitR = false;
    }

}
```

2. Clase Main:

Opción 1:

```
private static List<String> generarReferenciasDesdeImagen(Imagen imagen, int tamanoPagina, int longitudMensaje, i
    List<String> referencias = new ArrayList<>();
    String[] RGB = {"R","G","B"};
    int ancho = imagen.ancho;
    int alto = imagen.alto;
    int bytesPorFila = ancho * 3;
    int contadorBytes = 0; // Para el inicio donde solo contamos los 16 primeros bytes de Imagen
```

Opción 2:

```
private static void calcularFallasYHits(Scanner scanner) {
 try {

     System.out.println(x:"Ingrese el número de marcos de página: ");
     int marcosTotales = scanner.nextInt();
     scanner.nextLine(); // Limpiar buffer


     System.out.println(x:"Ingrese el nombre del archivo de referencias: ");
     String nombreArchivo = scanner.nextLine();

     List<String> referencias = leerReferencias(nombreArchivo);

     SimuladorMemoria simulador = new SimuladorMemoria(marcosTotales, referencias);
     simulador.simular();
     simulador.mostrarResultados();


 } catch (Exception e) {
     e.printStackTrace();
 }
```

List<String> referencias: Es una lista que almacena las referencias de páginas, es decir, las acciones de lectura y escritura sobre las posiciones de la imagen y el mensaje escondido. Se utiliza tanto en la generación de referencias (generarReferenciasDesdeImagen) como en la simulación de memoria (calcularFallasYHits).

Escenarios de prueba:

```
private static void ejecutarEscenarios() {
    String[] imagenes = {"imagenArdilla.bmp", "imagenTucan.bmp"};
    String[] mensajes = {"mensaje_100", "mensaje_1000", "mensaje_2000", "mensaje_4000", "mensaje_8000"};
    int[] marcos = {4, 8};
    int[] tamanosPagina = {512,1024,2048};
    for (String i : imagenes) {
        for (String m : mensajes) {
            for (int numMarcos : marcos) {
                for (int tamanoP : tamanosPagina){
```

Actualización:

Método generarReferenciasDesdeImagen()

- Aquí se generan las referencias de páginas para las operaciones de lectura y escritura. La lista referencias se llena con entradas que representan operaciones sobre la imagen y el mensaje.
- La lista referencias se actualiza en dos pasos:
    - **Lectura de los primeros 16 bytes de la imagen (para la longitud del mensaje)**:

- Se recorre la imagen para añadir referencias de lectura, que corresponden a la longitud del mensaje.
- Cada referencia indica la posición en la imagen (fila, columna, color) y la operación ("R" para lectura).

- **Procesamiento del mensaje escondido**:
  - Para cada carácter del mensaje, se generan referencias de inicialización, lectura y escritura.
  - Las referencias de inicialización se agregan con la operación "W" (escritura) para el mensaje.
  - Luego, se generan referencias de lectura para cada bit del mensaje escondido en la imagen, seguidas de referencias de escritura para actualizar el mensaje.

```
private static List<String> generarReferenciasDesdeImagen(Imagen imagen, int tamanoPagina, int longitudMensaje, int tamanoImagen) {
    List<String> referencias = new ArrayList<>();
    String[] RGB = {"R","G","B"};
    int ancho = imagen.ancho;
    int alto = imagen.alto;
    int bytesPorFila = ancho * 3;
    int contadorBytes = 0; // Para el inicio donde solo contamos los 16 primeros bytes de Imagen

    //recorrer 16 bytes de inicio
        for (int fila = 0; fila < alto && contadorBytes < 16; fila++) {
            for (int col = 0; col < ancho && contadorBytes < 16; col++) {
                for (int color = 0; color < 3 && contadorBytes < 16; color++) {
                    int pagina = contadorBytes / tamanoPagina;
                    int desp = contadorBytes % tamanoPagina;
                    String referencia = "Imagen[" + fila + "][" + col + "]." + RGB[color] + "," + pagina + "," + desp + ",R";
                    referencias.add(referencia);
                    contadorBytes++;
                }
            }
        }

    int numBytes = 16;// inicia en 16 debido a los bytes de inicio (de longitud) de arriba
    int posCaracter = 0; // es igual al desplazamiento del mesaje
    int paginaMensaje = tamanoImagen/tamanoPagina ; // El mensaje empieza después de las páginas ocupadas por la imagen, ej. la imagen ocupa 1152

    while (posCaracter < longitudMensaje) {

        // Inicializar el byte (mensaje)
        String referenciaInicializacion = "Mensaje[" + posCaracter + "]," + (paginaMensaje) + "," + (posCaracter%tamanoPagina)+ ",W";
        referencias.add(referenciaInicializacion);
```

```
        // Cada caracter del mensaje tiene 8 bits, o sea van a haber 8 mensaje[], pag, desplazamiento, W + el de inicializacion
        for (int i = 0; i < 8; i++) {
            int bytePos = numBytes;
            int fila = bytePos / bytesPorFila;
            int col = (bytePos % bytesPorFila) / 3;
            int color = (bytePos % 3);
            int paginaImagen = bytePos / tamanoPagina;
            int despImagen = bytePos % tamanoPagina;

            // Leer imagen
            String referenciaLectura = "Imagen[" + fila + "][" + col + "]." + RGB[color] + "," + paginaImagen + "," + despImagen + ",R";
            referencias.add(referenciaLectura);

            int despMensaje = (posCaracter * 8 + i) % tamanoPagina;
            // Escribir mensaje
            String referenciaEscritura = "Mensaje[" + posCaracter + "]," + (paginaMensaje) + "," + (despMensaje) + ",W";
            referencias.add(referenciaEscritura);

            numBytes++;
        }
        if ((posCaracter+1) % tamanoPagina == 0) {
            paginaMensaje++;  // Cambia de página cuando hemos llenado la actual
        }
        // Avanzar al siguiente carácter del mensaje después de procesar sus 8 bits
        posCaracter++;
    }

    return referencias;
}
```

Simulación de fallas y hits (calcularFallasYHits)

- **Método calcularFallasYHits()**
  - En este método, se leen las referencias desde un archivo y se almacenan en la lista referencias.
  - Luego, se crea un objeto SimuladorMemoria que recibe la lista de referencias para simular el comportamiento del sistema de paginación.
  - Durante la simulación, se incrementan los contadores de hits y fallas según si la página estaba ya en memoria o no.

```
private static void calcularFallasYHits(Scanner scanner) {
 try {

    System.out.println(x:"Ingrese el número de marcos de página: ");
    int marcosTotales = scanner.nextInt();
    scanner.nextLine(); // Limpiar buffer


    System.out.println(x:"Ingrese el nombre del archivo de referencias: ");
    String nombreArchivo = scanner.nextLine();

    List<String> referencias = leerReferencias(nombreArchivo);

    SimuladorMemoria simulador = new SimuladorMemoria(marcosTotales, referencias);
    simulador.simular();
    simulador.mostrarResultados();


 } catch (Exception e) {
    e.printStackTrace();
 }
```

3. Clase SimuladorMemoria:

```java
public class SimuladorMemoria {
    Memoria memoria;
    List<String> referencias;
    int hits;
    int fallas;
    long tiempoTotal;
    long tiempoH;
    long tiempoF;
    long tiempoRAM;
    long tiempoFallas;
    boolean finSimulacion;

    public SimuladorMemoria(int marcosTotales, List<String> referencias) {
        this.memoria = new Memoria(marcosTotales);
        this.referencias = referencias;
        this.hits = 0;
        this.fallas = 0;
        this.tiempoTotal = 0;
        this.tiempoH = 0;
        this.tiempoF = 0;
        this.tiempoRAM = 0;
        this.tiempoFallas =0;
        this.finSimulacion = false;
    }
```

List<String> referencias: Es una lista de cadenas que contiene las referencias de acceso a la memoria, que indican qué páginas se deben cargar y si la operación es de lectura o escritura. Esta lista se procesa secuencialmente en la simulación.

Actualización:

Simulación del acceso a la memoria (simular() método)

En el método simular(), se utilizan dos hilos (threads) para simular de manera concurrente:

1. **threadProcesamiento**:
    a. Este hilo procesa cada referencia en la lista referencias, simulando la carga de una página en la memoria.
    b. Para cada referencia:
        i. Crea una instancia de Pagina llamando a crearPaginaDeReferencia(), que interpreta la referencia y determina si es una operación de lectura o escritura.
        ii. Llama a memoria.cargarPagina() para simular el acceso a la memoria. La función cargarPagina() en la clase Memoria determina si es un "hit" (la página ya estaba en memoria) o una "falla" (la página no estaba en memoria y se necesita cargar o reemplazar).
        iii. Según el resultado, incrementa el contador de hits o fallas.
    c. El hilo se detiene por 1 milisegundo (Thread.sleep(1)) para simular un tiempo de procesamiento.
    d. Cuando todas las referencias han sido procesadas, establece finSimulacion en true para indicar que la simulación ha finalizado.

```
public void simular() {

    Thread threadPaginacion = new Thread(new Runnable() { // thread que actualiza marcos de pagina de a cuerdo a hits y fallas
        public void run() {
            for (String ref : referencias) {
                Pagina pagina = crearPaginaDeReferencia(ref);
                boolean hit = memoria.cargarPagina(pagina); // hit o falla

                if (hit) {
                    hits++;
                    tiempoH+=25;}
                else {
                    fallas++;
                    tiempoF+=10000000;
                }
                try{
                    Thread.sleep(millis:1);
                } catch (InterruptedException e){
                    e.printStackTrace();
                }
            }
            finSimulacion=true;

        }
    }
    );
```

2. **threadActualizacion**:
    a. Este hilo actualiza el bit R de las páginas en la memoria.
    b. Llama al método memoria.interrupcionBitR(), que resetea el bit R de todas las páginas.

```
  Thread threadActualizacion = new Thread(new Runnable() { // thread que actualiza bit R
      public void run (){
          while(!finSimulacion){
              try{
                  Thread.sleep(millis:2);
              } catch (InterruptedException e){
                  e.printStackTrace();
              }

              memoria.interrupcionBitR();


          }
      }

  }
  );
```

```
    threadProcesamiento.start();
    threadActualizacion.start();

    try {
        threadProcesamiento.join();  // Espera hasta que el thread haya terminado
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    finSimulacion = true;

    long endTime = System.nanoTime();
    tiempoTotal = endTime - startTime;

}
```

Esquema de sincronización usado

Clase Memoria:

Método cargarPagina:

```
public synchronized boolean cargarPagina(Pagina nuevaPagina) {
    for (Pagina pagina : marcos) {
        if (pagina.numeroPagina == nuevaPagina.numeroPagina) {
            pagina.referenciar(nuevaPagina.bitM); //Se actualiza bitR y bitM si es escritura
            return true; // Hit
        }
    }
    //Falla
    if (marcos.size() < marcosTotales) {
        marcos.add(nuevaPagina);
    } else {
        reemplazarPagina(nuevaPagina);
    }
    return false;
}
```

La sincronización es necesaria aquí para evitar condiciones de carrera cuando varios hilos acceden o modifican la lista de marcos simultáneamente.

La operación de carga de una página implica la verificación de si ya está en memoria (hit) o si se necesita cargar una nueva página (falla). Sin sincronización, dos hilos podrían intentar agregar o reemplazar páginas al mismo tiempo, lo que podría corromper la estructura de datos.

La actualización de los bits de referencia y modificación (bitR y bitM) también debe estar protegida para asegurar que los cambios sean consistentes.

Método remplazarPagina:

```
private synchronized void reemplazarPagina(Pagina nuevaPagina) {
    Pagina paginaReemplazo = seleccionarPaginaParaReemplazo();
    marcos.remove(paginaReemplazo);
    marcos.add(nuevaPagina);
}
```

El método reemplazarPagina es crítico, ya que implica eliminar una página y agregar otra en la lista de marcos. Sin sincronización, un hilo podría eliminar una página mientras otro hilo intenta acceder a ella, lo que resultaría en una excepción de concurrencia o datos inconsistentes.

Metodo seleccionarPaginaParaRemplazo:

```
private synchronized Pagina seleccionarPaginaParaReemplazo() { // utiliza NRU
    List<List<Pagina>> clases = new ArrayList<>(initialCapacity:4);

    for(int i = 0; i<4;i++){
        clases.add(new ArrayList<>()); // Se crea 1 array por cada clase
    }
    // paginas dentro de su clase
    for(Pagina pagina: marcos){
        int clase = getClaseNRU(pagina);
        clases.get(clase).add(pagina);
    }

    for(int i = 0; i<clases.size();i++){
        if(!clases.get(i).isEmpty()){ // como es un ciclo se toma la clase 0, luego 2 1 luego 3  y por ultimo 4 , para ir en orden de prioridad
           return clases.get(i).get(index:0);
        }

    }
    return null;
}
```

Este método debe estar sincronizado porque accede a la lista de marcos para clasificar las páginas en función de sus bits de referencia y modificación (bitR y bitM), lo cual es una operación concurrente sensible.

El algoritmo NRU (Not Recently Used) clasifica las páginas en diferentes clases y selecciona una página para el reemplazo. Si no se sincroniza, varios hilos podrían intentar clasificar las páginas simultáneamente, lo que podría llevar a errores en la elección de la página a reemplazar.

Metodo interrupcionBitR:

```
public synchronized void interrupcionBitR(){
    for(Pagina pagina: marcos){
        pagina.bitR = false;
    }

}
```

Este método restablece el bit de referencia (bitR) de todas las páginas. La sincronización es necesaria para evitar que otro hilo modifique el bit de referencia mientras se está actualizando, lo que podría provocar inconsistencias en el estado de las páginas.

Tabla datos Recopilados

Tablas imagenArdilla.bmp

Mensaje de 100 caracteres				
Tamaño de Página (bytes)	**Marcos Asignados**	**Total Referencias**	**% Hits**	**% Fallas**
512	4	1716	99,83%	0,17%
512	8	1716	99,83%	0,17%
1024	4	1716	99,88%	0,12%
1024	8	1716	99,88%	0,12%
2048	4	1716	99,88%	0,12%

2048	8	1716	99,88%	0,12%

Mensaje de 1000 caracteres				
Tamaño de Página (bytes)	**Marcos Asignados**	**Total Referencias**	**% Hits**	**% Fallas**
512	4	17016	99,89%	0,11%
512	8	17016	99,89%	0,11%
1024	4	17016	99,95%	0,05%
1024	8	17016	99,95%	0,05%
2048	4	17016	99,97%	0,03%
2048	8	17016	99,97%	0,03%

Mensaje de 2000 caracteres				
Tamaño de Página (bytes)	**Marcos Asignados**	**Total Referencias**	**% Hits**	**% Fallas**
512	4	34016	99,89%	0,11%
512	8	34016	99,89%	0,11%
1024	4	34016	99,95%	0,05%
1024	8	34016	99,95%	0,05%
2048	4	34016	99,97%	0,03%
2048	8	34016	99,97%	0,03%

Mensaje de 4000 caracteres				
Tamaño de Página (bytes)	**Marcos Asignados**	**Total Referencias**	**% Hits**	**% Fallas**
512	4	68016	99,90%	0,10%
512	8	68016	99,89%	0,11%
1024	4	68016	99,95%	0,05%
1024	8	68016	99,95%	0,05%
2048	4	68016	99,97%	0,03%
2048	8	68016	99,97%	0,03%

Mensaje de 8000 caracteres

Tamaño de Página (bytes)	Marcos Asignados	Total Referencias	% Hits	% Fallas
512	4	136016	99,90%	0,10%
512	8	136016	99,90%	0,10%
1024	4	136016	99,95%	0,05%
1024	8	136016	99,95%	0,05%
2048	4	136016	99,97%	0,03%
2048	8	136016	99,97%	0,03%

Tablas imagenTucan.bmp

Mensaje de 100 caracteres				
Tamaño de Página (bytes)	**Marcos Asignados**	**Total Referencias**	**% Hits**	**% Fallas**
512	4	1716	99,83%	0,17%
512	8	1716	99,83%	0,17%
1024	4	1716	99,88%	0,12%
1024	8	1716	99,88%	0,12%
2048	4	1716	99,88%	0,12%
2048	8	1716	99,88%	0,12%

Mensaje de 1000 caracteres				
Tamaño de Página (bytes)	**Marcos Asignados**	**Total Referencias**	**% Hits**	**% Fallas**
512	4	17016	99,89%	0,11%
512	8	17016	99,89%	0,11%
1024	4	17016	99,95%	0,05%
1024	8	17016	99,95%	0,05%
2048	4	17016	99,97%	0,03%
2048	8	17016	99,97%	0,03%

Mensaje de 2000 caracteres				
Tamaño de Página (bytes)	**Marcos Asignados**	**Total Referencias**	**% Hits**	**% Fallas**
512	4	34016	99,89%	0,11%
512	8	34016	99,89%	0,11%
1024	4	34016	99,95%	0,05%

1024	8	34016	99,95%	0,05%
2048	4	34016	99,97%	0,03%
2048	8	34016	99,97%	0,03%

Mensaje de 4000 caracteres				
Tamaño de Página (bytes)	**Marcos Asignados**	**Total Referencias**	**% Hits**	**% Fallas**
512	4	68016	99,89%	0,11%
512	8	68016	99,90%	0,10%
1024	4	68016	99,95%	0,05%
1024	8	68016	99,95%	0,05%
2048	4	68016	99,97%	0,03%
2048	8	68016	99,97%	0,03%

Mensaje de 8000 caracteres				
Tamaño de Página (bytes)	**Marcos Asignados**	**Total Referencias**	**% Hits**	**% Fallas**
512	4	136016	99,90%	0,10%
512	8	136016	99,90%	0,10%
1024	4	136016	99,95%	0,05%
1024	8	136016	99,95%	0,05%
2048	4	136016	99,97%	0,03%
2048	8	136016	99,97%	0,03%

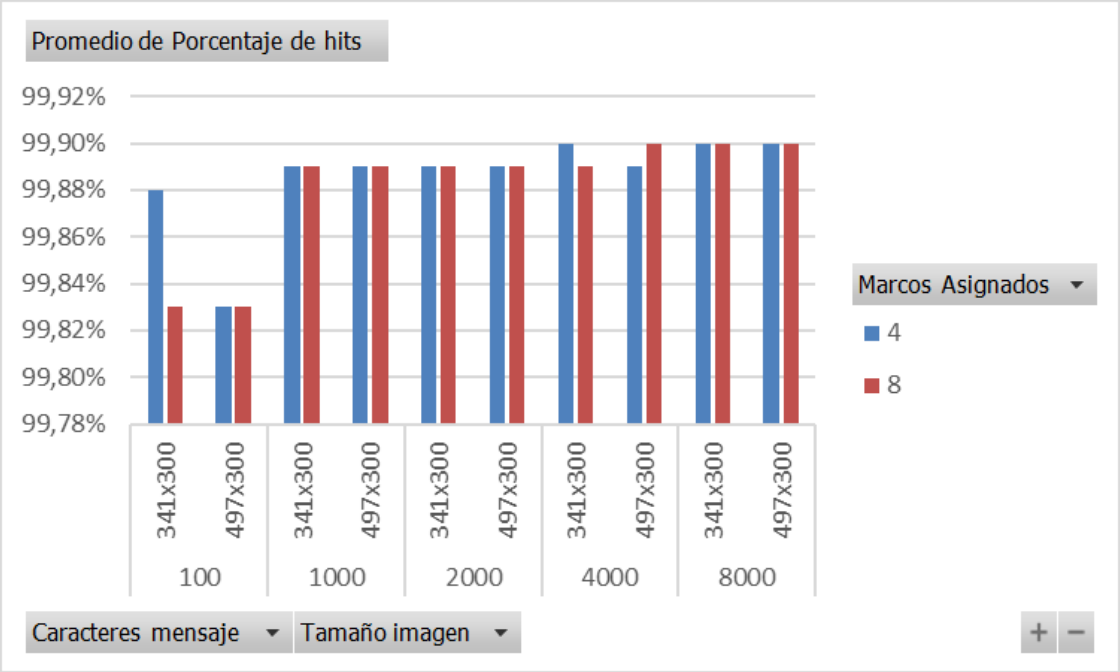
Graficas comportamiento del sistema

• Gráficas donde fije tamaño de página y grafique Tamaño de imagen vs. Marcos asignados vs. Porcentaje de hits. La gráfica al final del enunciado ilustra el tipo de gráfica que buscamos.

Tamaño de página 512:

Tamaño imagen	Marcos Asignados	Porcentaje de hits	Caracteres mensaje
341x300	4	99,88%	100
341x300	4	99,89%	1000
341x300	4	99,89%	2000
341x300	4	99,90%	4000

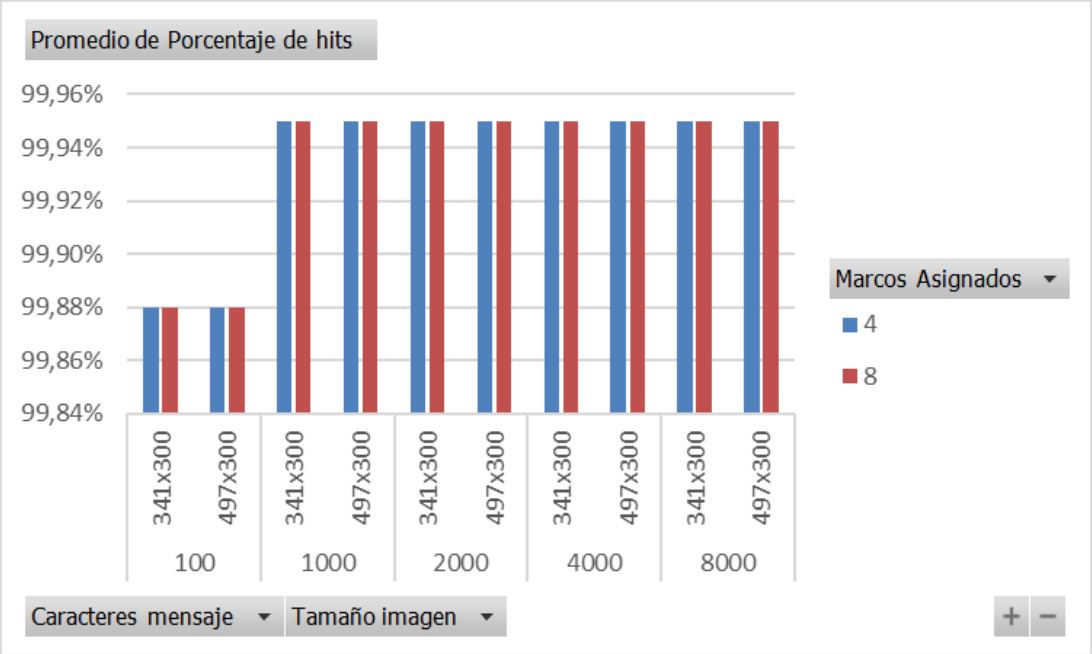
341x300	4	99,90%	8000
341x300	8	99,83%	100
341x300	8	99,89%	1000
341x300	8	99,89%	2000
341x300	8	99,89%	4000
341x300	8	99,90%	8000
497x300	4	99,83%	100
497x300	4	99,89%	1000
497x300	4	99,89%	2000
497x300	4	99,89%	4000
497x300	4	99,90%	8000
497x300	8	99,83%	100
497x300	8	99,89%	1000
497x300	8	99,89%	2000
497x300	8	99,90%	4000
497x300	8	99,90%	8000



Tamaño de página 1024:

Tamaño imagen	Marcos Asignados	Porcentaje de hits	Caracteres mensaje
341x300	4	99,88%	100
341x300	4	99,95%	1000
341x300	4	99,95%	2000
341x300	4	99,95%	4000
341x300	4	99,95%	8000

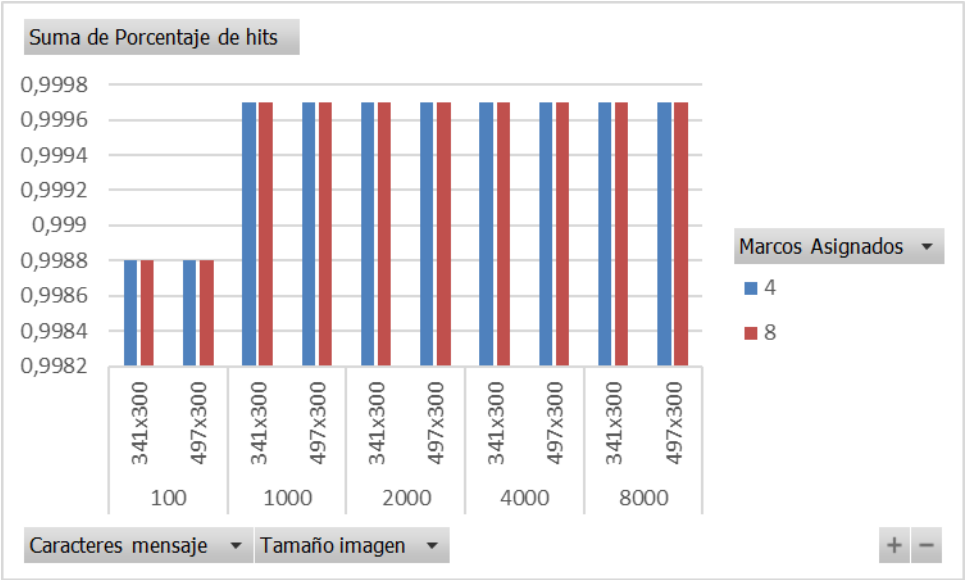
341x300	8	99,88%	100
341x300	8	99,95%	1000
341x300	8	99,95%	2000
341x300	8	99,95%	4000
341x300	8	99,95%	8000
497x300	4	99,88%	100
497x300	4	99,95%	1000
497x300	4	99,95%	2000
497x300	4	99,95%	4000
497x300	4	99,95%	8000
497x300	8	99,88%	100
497x300	8	99,95%	1000
497x300	8	99,95%	2000
497x300	8	99,95%	4000
497x300	8	99,95%	8000



Tamaño de página 2048:

Tamaño imagen	Marcos Asignados	Porcentaje de hits	Caracteres mensaje
341x300	4	99,88%	100
341x300	4	99,97%	1000
341x300	4	99,97%	2000
341x300	4	99,97%	4000
341x300	4	99,97%	8000
341x300	8	99,88%	100

341x300	8	99,97%	1000
341x300	8	99,97%	2000
341x300	8	99,97%	4000
341x300	8	99,97%	8000
497x300	4	99,88%	100
497x300	4	99,97%	1000
497x300	4	99,97%	2000
497x300	4	99,97%	4000
497x300	4	99,97%	8000
497x300	8	99,88%	100
497x300	8	99,97%	1000
497x300	8	99,97%	2000
497x300	8	99,97%	4000
497x300	8	99,97%	8000



Graficas diferentes escenarios

• Los 60 escenarios se corrieron y por medio de los prints que generaba el método, obtuvimos los datos e hicimos las anteriores graficas con el tamaño de página fijo, porcentaje de hits, caracteres de mensaje, tamaño de imagen y marcos asignados. Estas graficas recopilan casi todos los datos que obtuvimos a excepción de las fallas y el tiempo, para las cuales decidimos emplear otras graficas que explican su comportamiento. En los escenarios utilizamos una imagen de 341x300 y una de 497x300, siguiendo la recomendación de no usar imágenes mas grandes de 500x300. Para la anterior se nos pregunto ¿Cuál es la longitud máxima de mensaje que puede almacenar una imagen de este tamaño?, y así fue como lo calculamos:

Número total de píxeles: 500×300=150,000pixeles

Numero toral de bits disponibles: 150,000 pixeles x 3 bits por pixel = 450000

Caracteres: 450000 bits / 8 bits por caracter = 56250

La longitud máxima del mensaje que puede almacenar una imagen de 500 x 300 píxeles es de 56,250 caracteres.

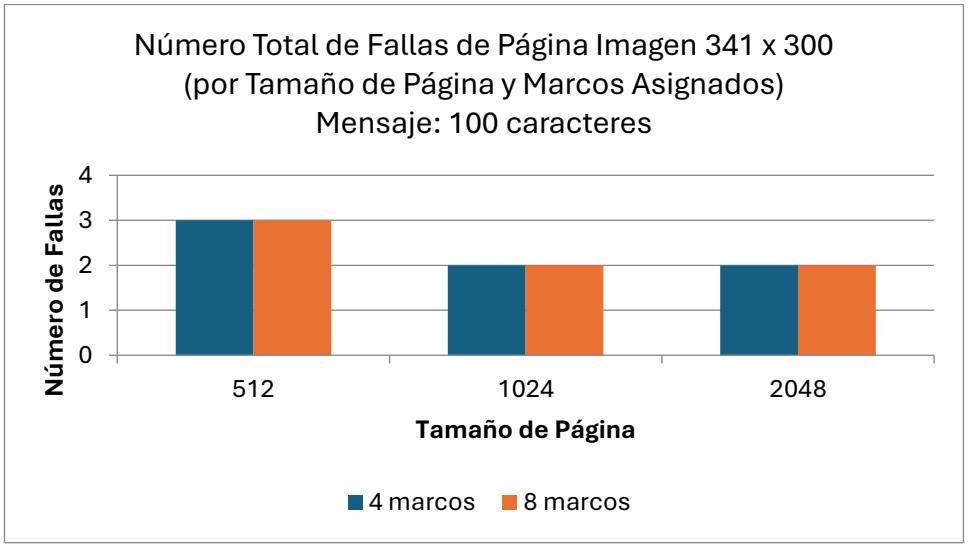
Otras configuraciones que pueden afectar el desempeño

• Además de los escenarios definidos, consideramos otras configuraciones que nos permiten entender cómo afecta la memoria virtual el desempeño del programa. En este caso nos centraremos en comparar el numero de fallas entre los diferentes marcos de página, para todos los tamaños de pagina que tenemos, y con cada longitud de mensaje que tenemos

Imagen 341 x 300 (imagenArdilla.bmp):

Mensaje 100 caracteres:

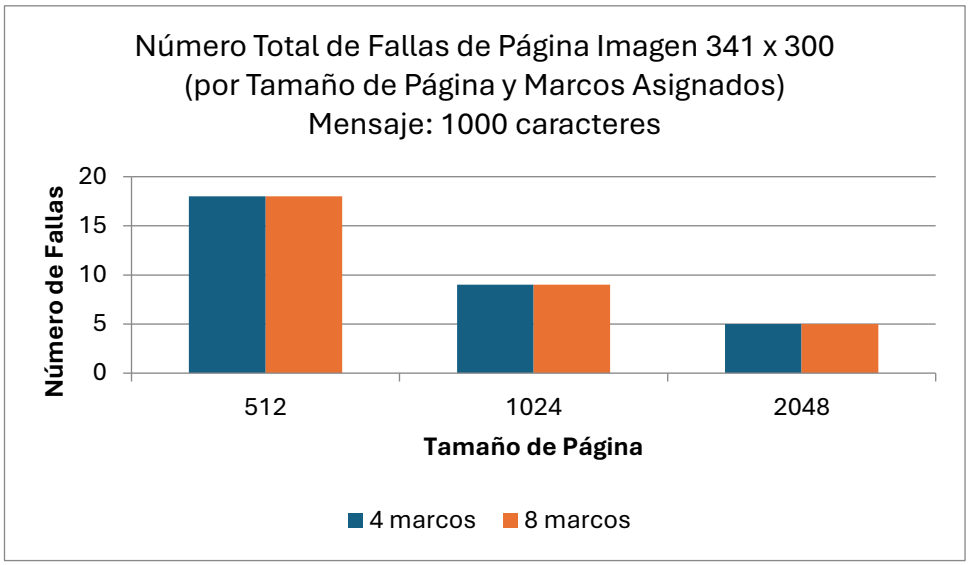
Tamano pagina	Marcos Asignados	Num Fallas
512	4	3
1024	4	2
2048	4	2
512	8	3
1024	8	2
2048	8	2



Mensaje 1000 caracteres:

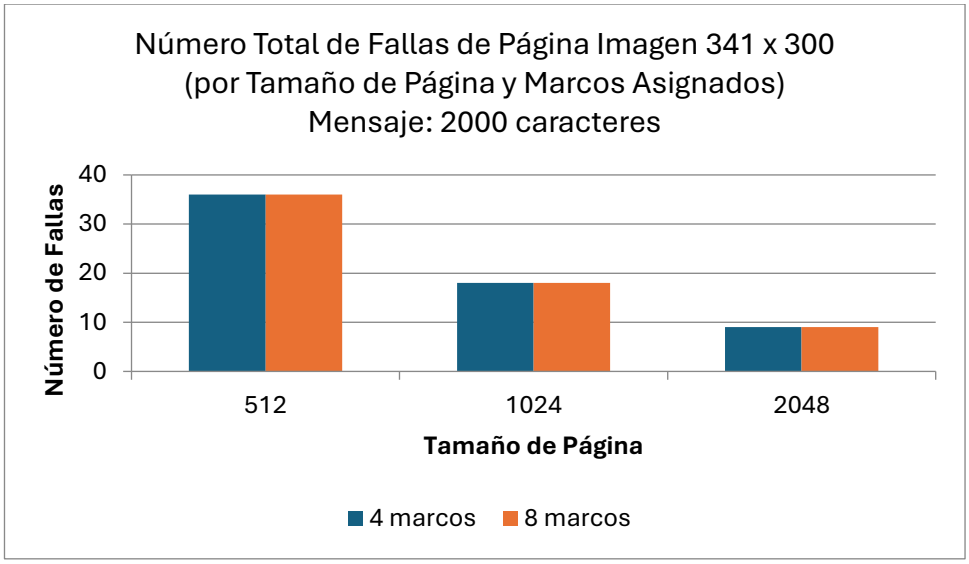
Tamano Pagina	Marcos Asignados	Num Fallas
512	4	18

1024	4	9
2048	4	5
512	8	18
1024	8	9
2048	8	5



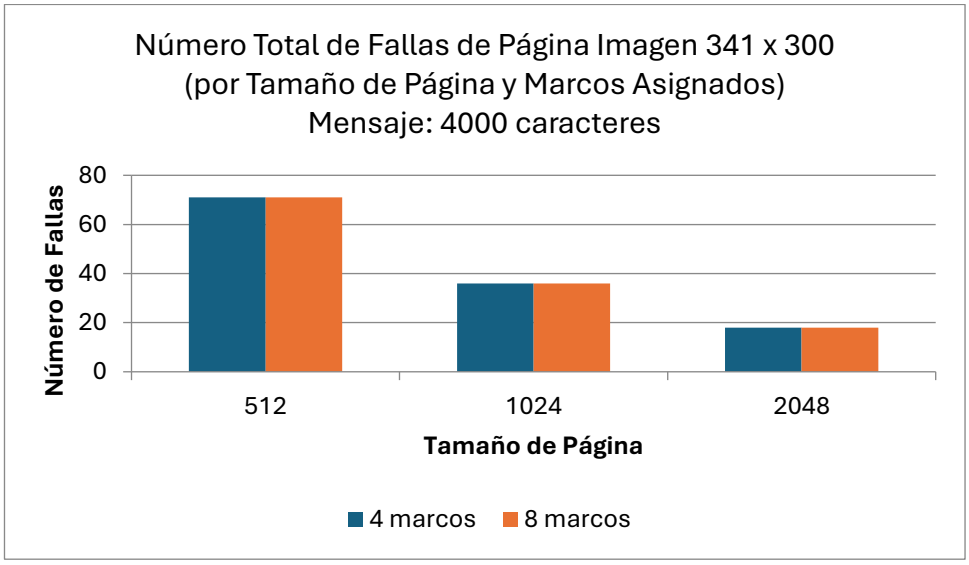
Mensaje 2000 caracteres:

Tamano Pagina	Marcos Asignados	Num Fallas
512	4	36
1024	4	18
2048	4	9
512	8	36
1024	8	18
2048	8	9



Mensaje 4000 caracteres:

Tamano Pagina	Marcos Asignados	Num Fallas
512	4	71
1024	4	36
2048	4	18
512	8	72
1024	8	36
2048	8	18



Mensaje 8000 caracteres:

Tamano Pagina	Marcos Asignados	Num Fallas
512	4	142
1024	4	71
2048	4	36
512	8	142
1024	8	71
2048	8	36

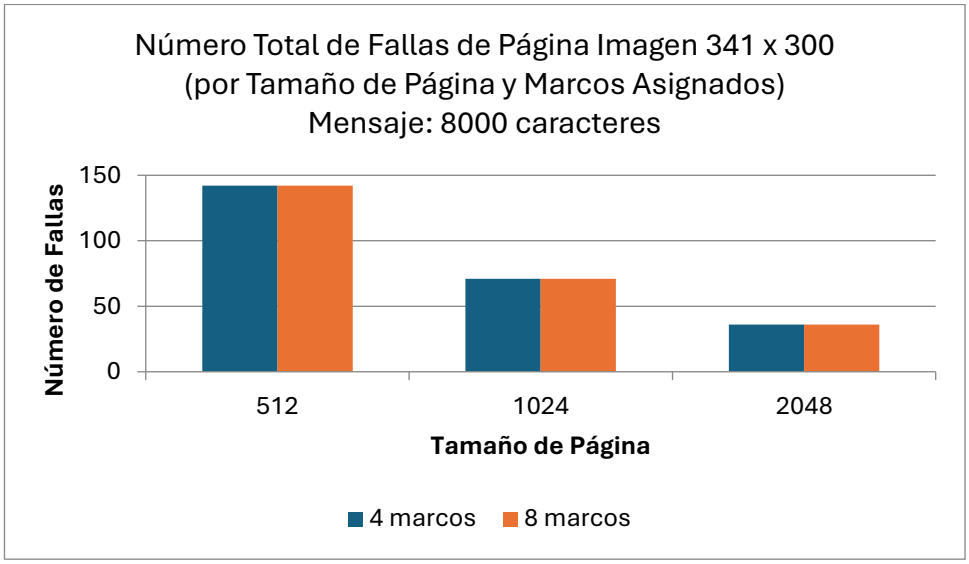
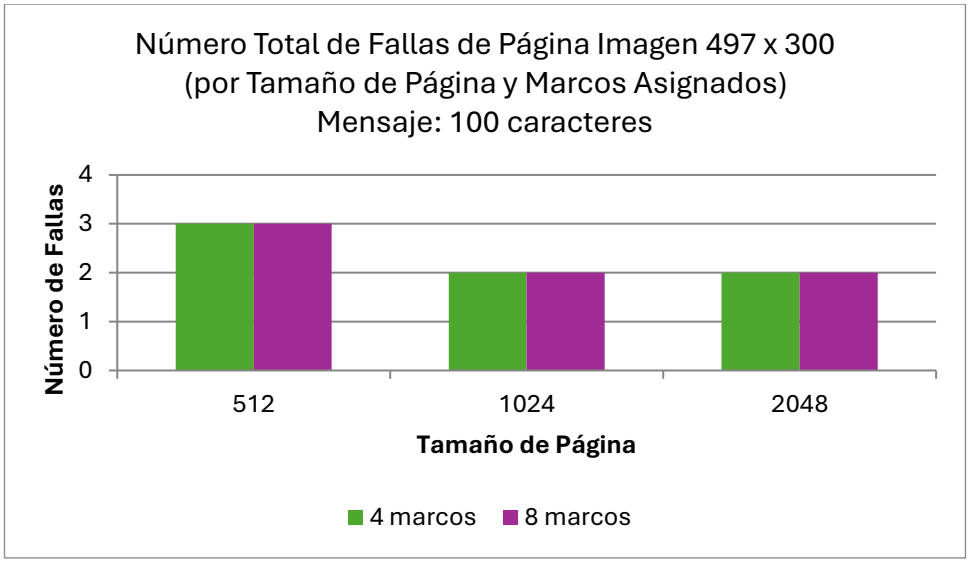


Imagen 497 x 300 (imagenTucan.bmp):

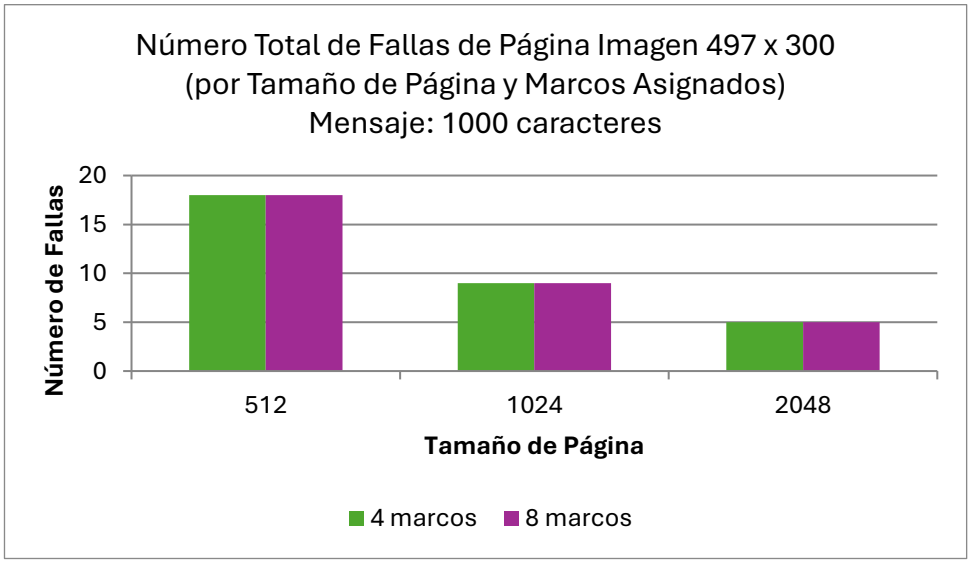
Mensaje 100 caracteres:

Tamano pagina	Marcos Asignados	Num Fallas
512	4	3
1024	4	2
2048	4	2
512	8	3
1024	8	2
2048	8	2



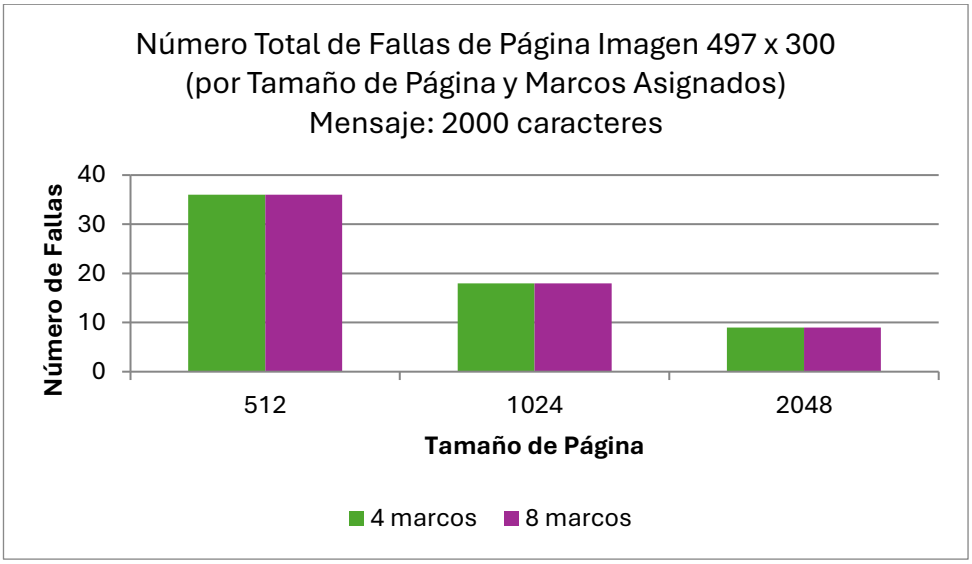
Mensaje 1000 caracteres:

Tamano pagina	Marcos Asignados	Num Fallas
512	4	18
1024	4	9
2048	4	5
512	8	18
1024	8	9
2048	8	5



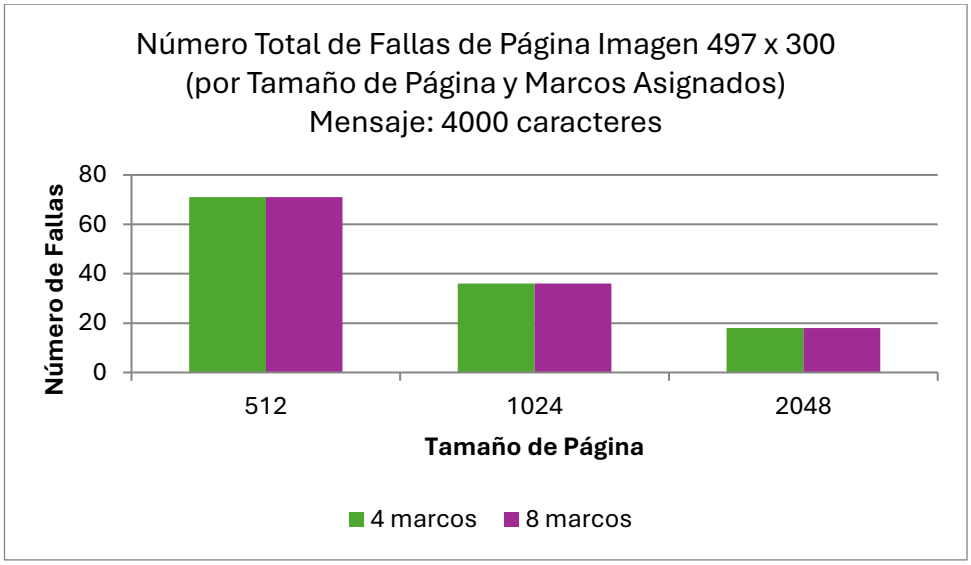
Mensaje 2000 caracteres:

Tamano pagina	Marcos Asignados	Num Fallas
512	4	36
1024	4	18
2048	4	9
512	8	36
1024	8	18
2048	8	9



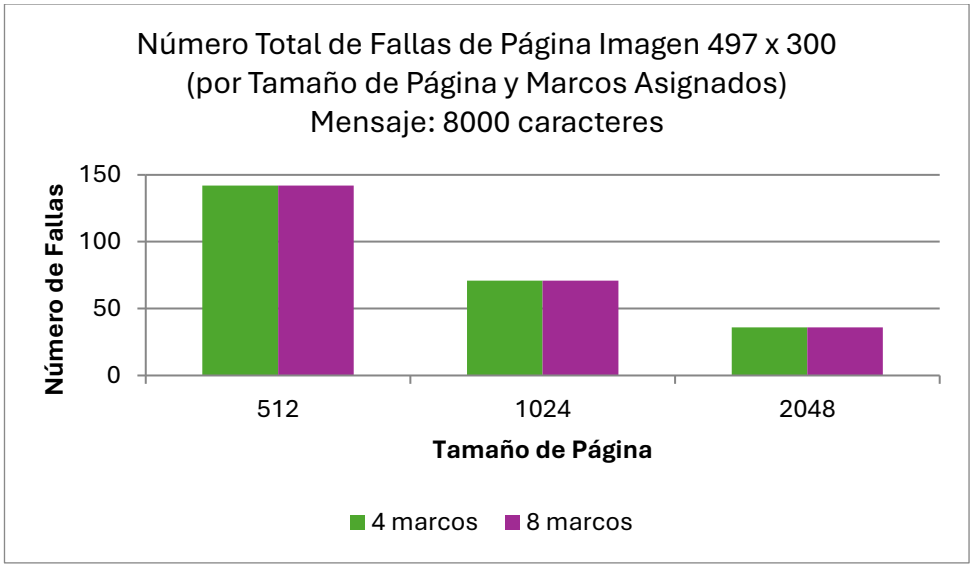
Mensaje 4000 caracteres:

Tamano pagina	Marcos Asignados	Num Fallas
512	4	71
1024	4	36
2048	4	18
512	8	72
1024	8	36
2048	8	18



Mensaje 8000 caracteres:

Tamano pagina	Marcos Asignados	Num Fallas
512	4	142
1024	4	71
2048	4	36
512	8	142
1024	8	71
2048	8	36

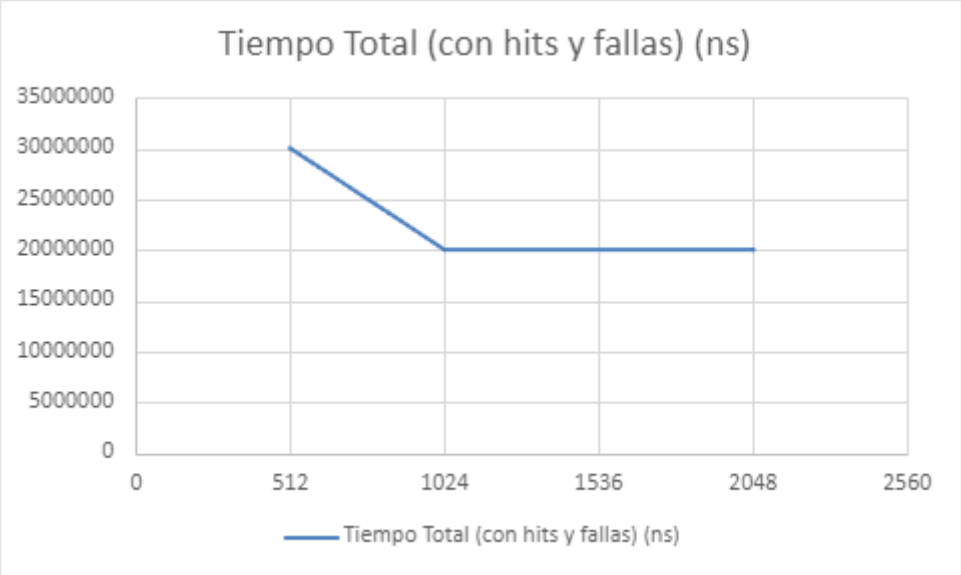


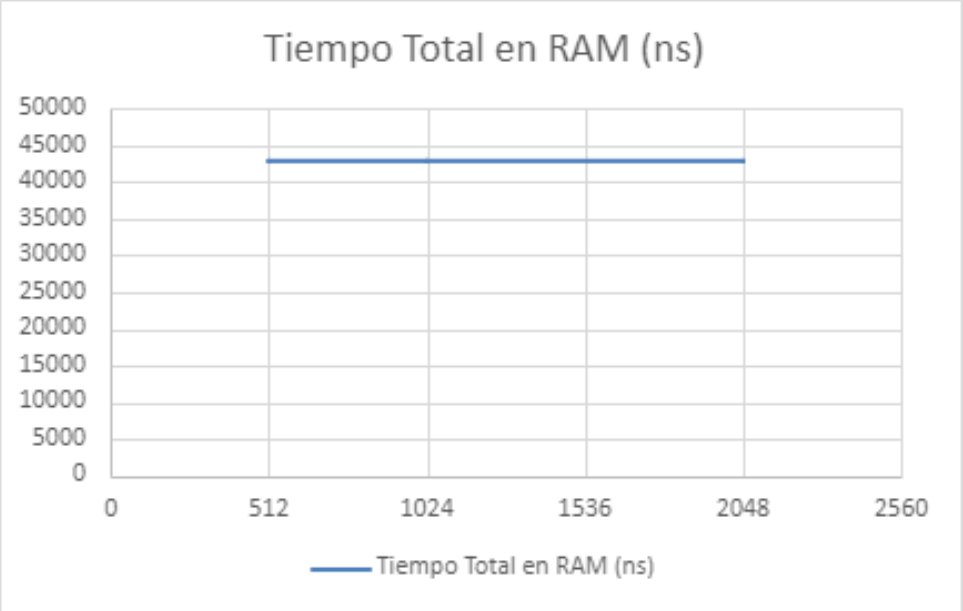
Graficas tiempo, Hits, Misses y total

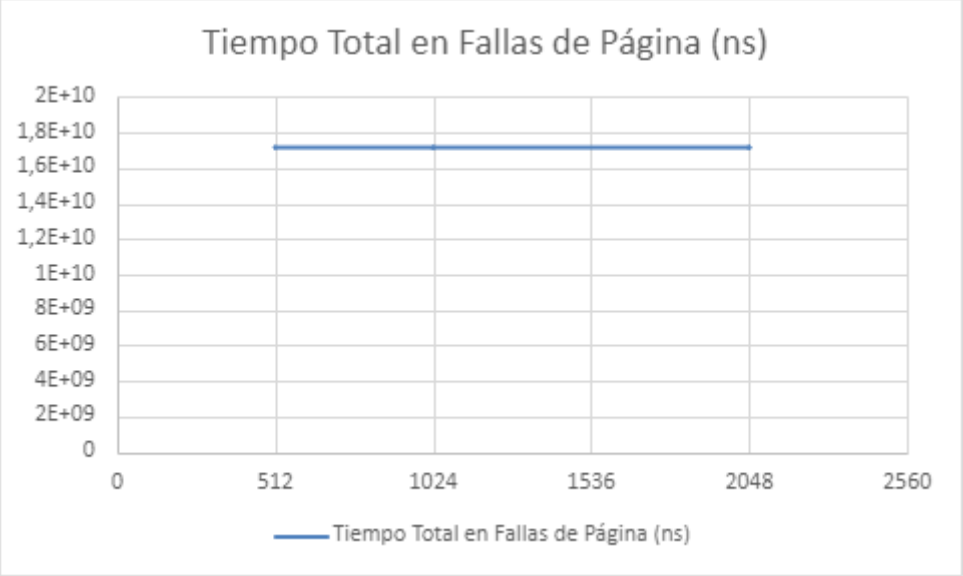
Graficas imagenArdilla.bmp

Mensaje de 100 caracteres

Tamaño Página	Tiempo Total (con hits y fallas) (ns)	Tiempo Total en RAM (ns)	Tiempo Total en Fallas de Página (ns)
512	30042825	42900	17160000000
512	30042825	42900	17160000000
1024	20042850	42900	17160000000
1024	20042850	42900	17160000000
2048	20042850	42900	17160000000
2048	20042850	42900	17160000000

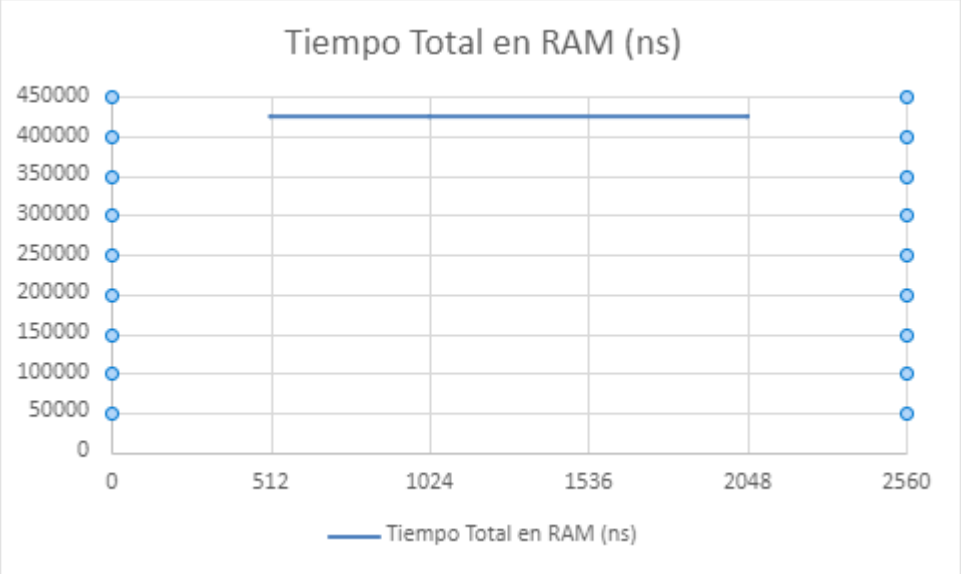


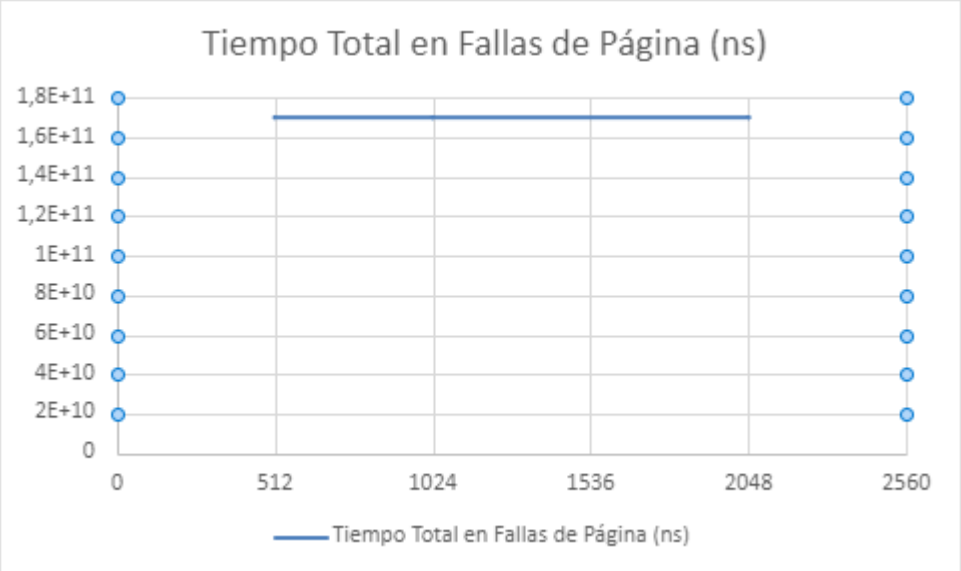




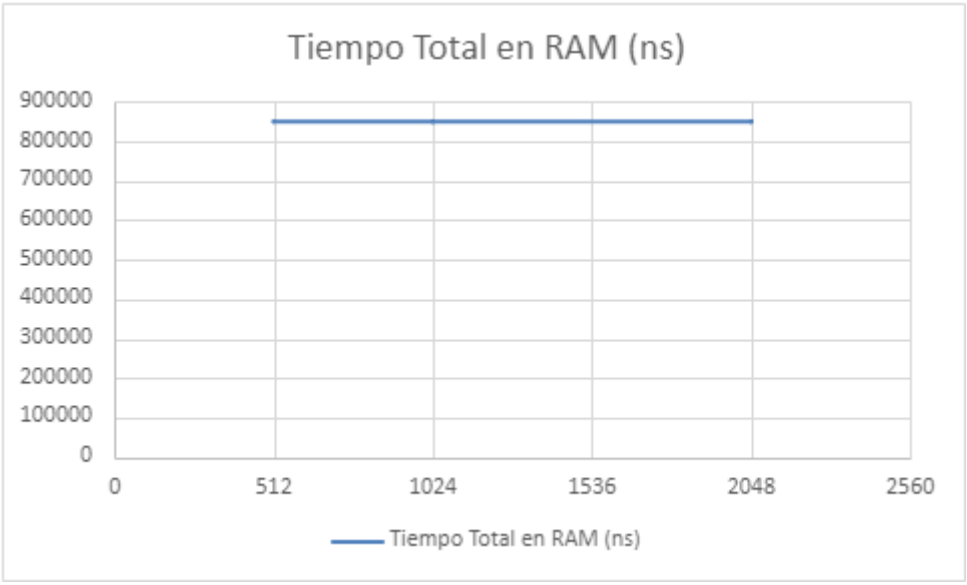
Mensaje de 1000 caracteres			
Tamaño Página	**Tiempo Total (con hits y fallas) (ns)**	**Tiempo Total en RAM (ns)**	**Tiempo Total en Fallas de Página (ns)**
512	180424950	425400	1,7016E+11
512	180424950	425400	1,7016E+11
1024	90425175	425400	1,7016E+11
1024	90425175	425400	1,7016E+11
2048	50425275	425400	1,7016E+11
2048	50425275	425400	1,7016E+11

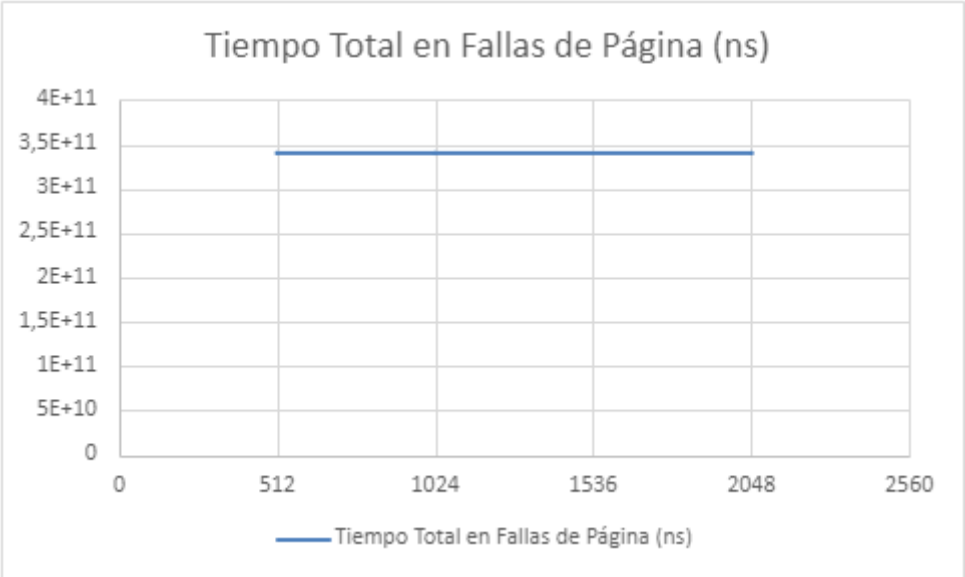

Tiempo Total (con hits y fallas) (ns)
200000000
180000000
160000000
140000000
120000000
100000000
80000000
60000000
40000000
20000000
0
0
512
1024
1536
2048
2560
Tiempo Total (con hits y fallas) (ns)


Tiempo Total en RAM (ns)
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
0
512
1024
1536
2048
2560
Tiempo Total en RAM (ns)

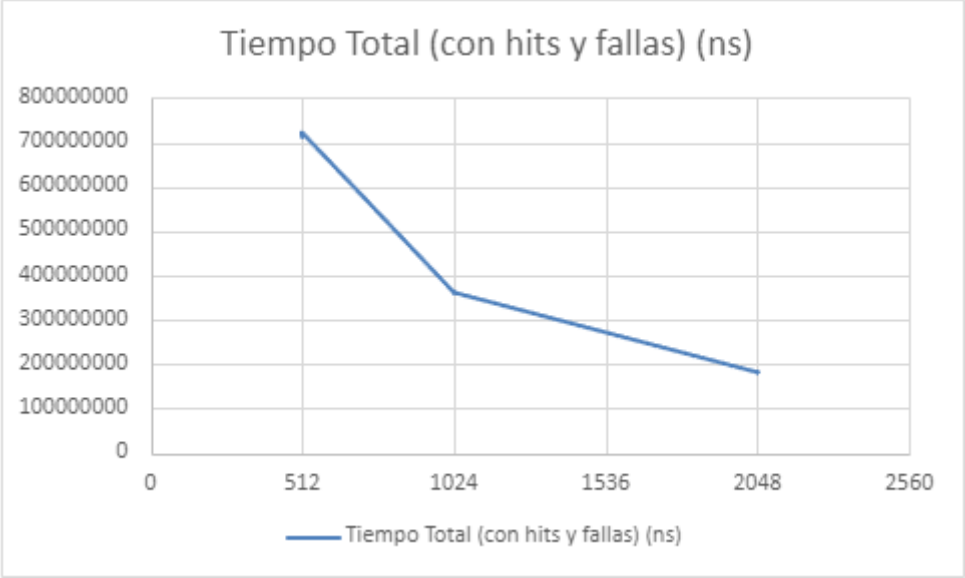

Tiempo Total en Fallas de Página (ns)
1,8E+11
1,6E+11
1,4E+11
1,2E+11
1E+11
8E+10
6E+10
4E+10
2E+10
0
0
512
1024
1536
2048
2560
Tiempo Total en Fallas de Página (ns)

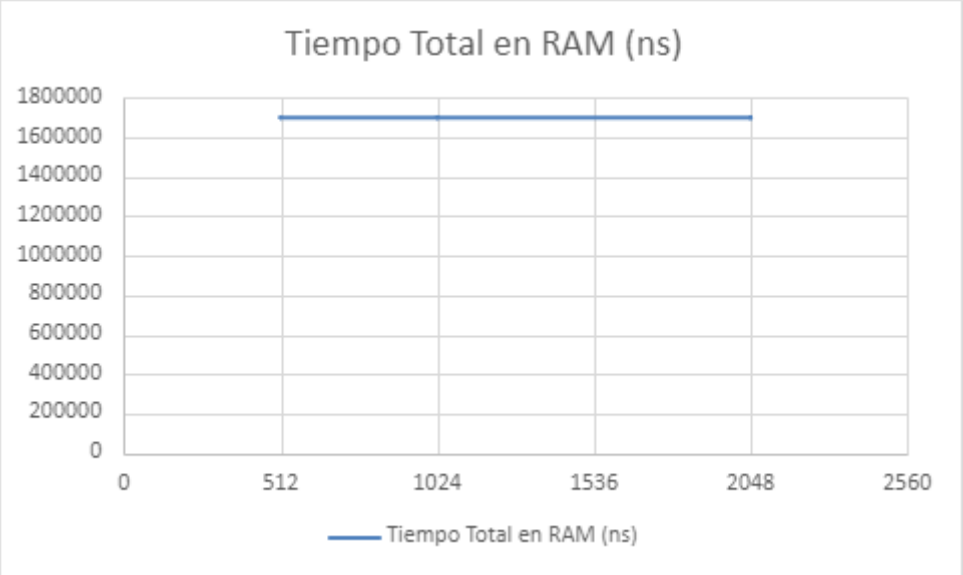
Mensaje de 2000 caracteres			
Tamaño Página	**Tiempo Total (con hits y fallas) (ns)**	**Tiempo Total en RAM (ns)**	**Tiempo Total en Fallas de Página (ns)**
512	360849500	850400	3,4016E+11
512	360849500	850400	3,4016E+11
1024	180849950	850400	3,4016E+11
1024	180849950	850400	3,4016E+11
2048	90850175	850400	3,4016E+11
2048	90850175	850400	3,4016E+11

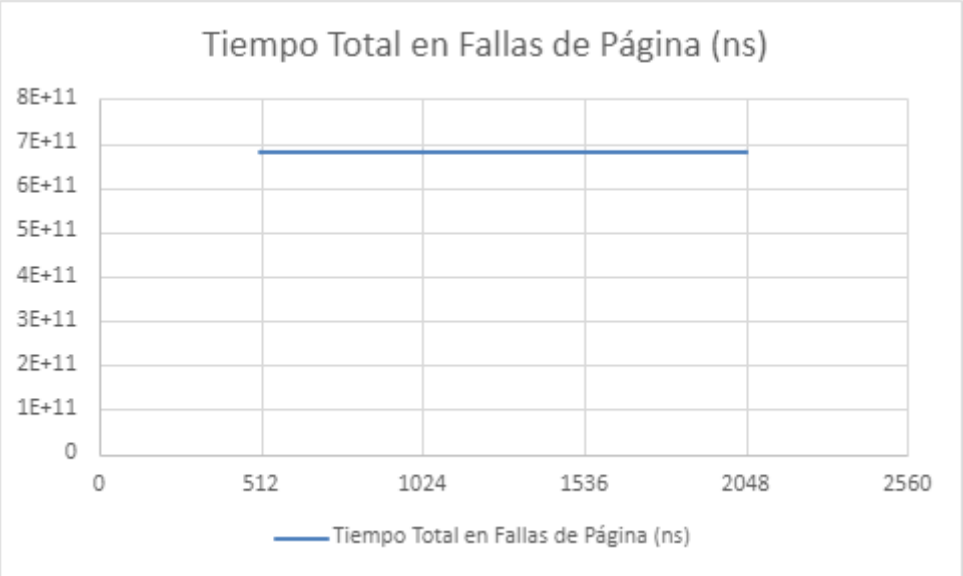




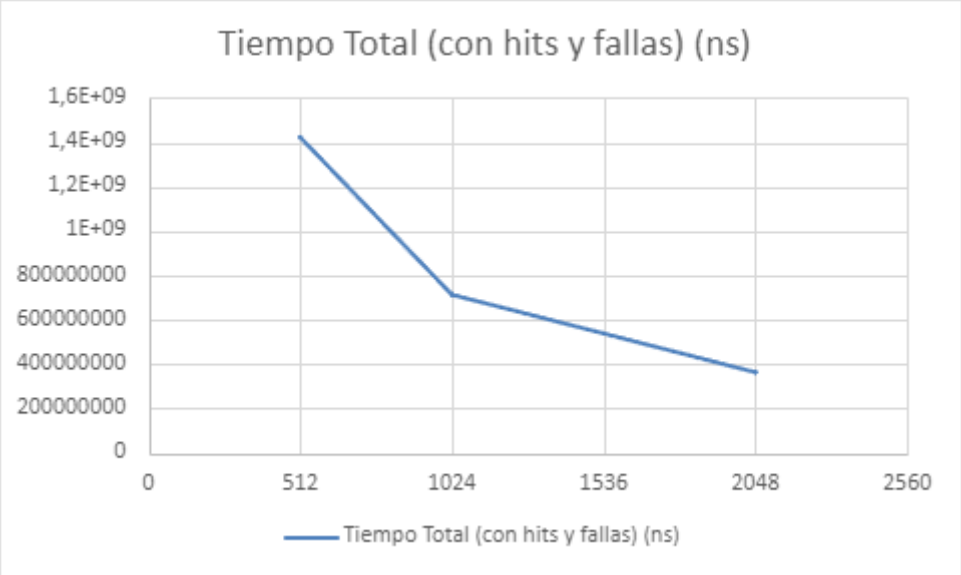
Mensaje de 4000 caracteres			
Tamaño Página	**Tiempo Total (con hits y fallas) (ns)**	**Tiempo Total en RAM (ns)**	**Tiempo Total en Fallas de Página (ns)**
512	711698625	1700400	6,8016E+11
512	721698600	1700400	6,8016E+11
1024	361699500	1700400	6,8016E+11
1024	361699500	1700400	6,8016E+11
2048	181699950	1700400	6,8016E+11
2048	181699950	1700400	6,8016E+11

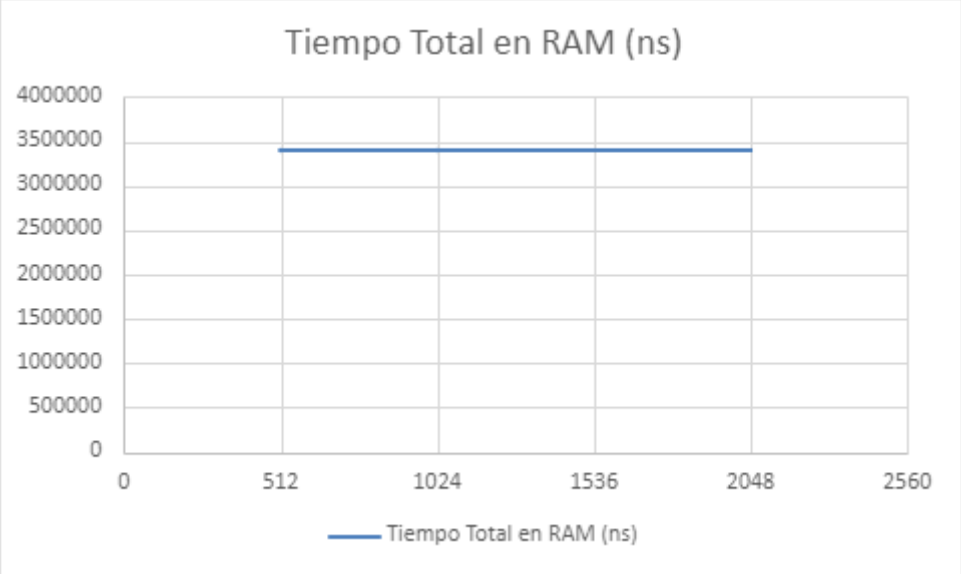


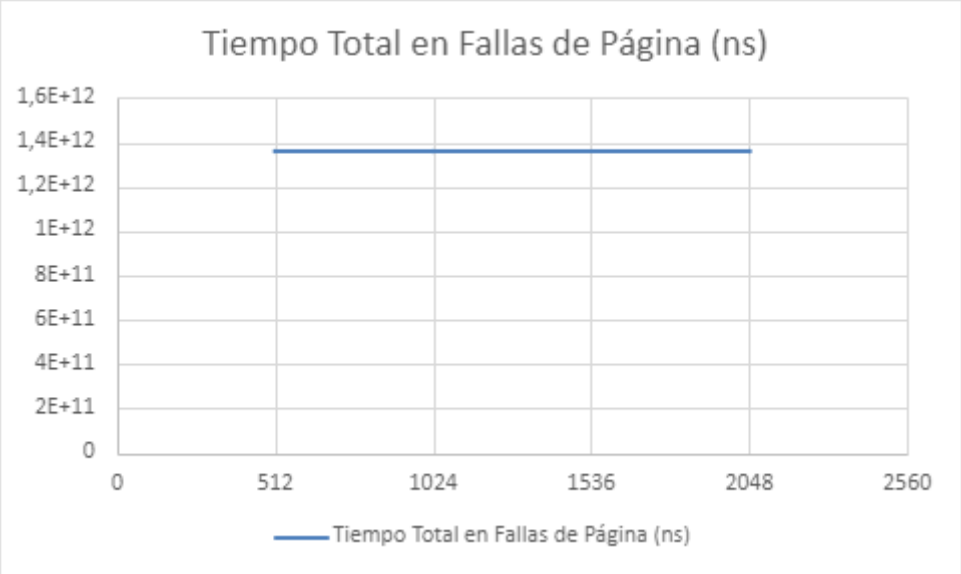




Mensaje de 8000 caracteres			
Tamaño Página	**Tiempo Total (con hits y fallas) (ns)**	**Tiempo Total en RAM (ns)**	**Tiempo Total en Fallas de Página (ns)**
512	1423396850	3400400	1,36016E+12
512	1423396850	3400400	1,36016E+12
1024	713398625	3400400	1,36016E+12
1024	713398625	3400400	1,36016E+12
2048	363399500	3400400	1,36016E+12
2048	363399500	3400400	1,36016E+12

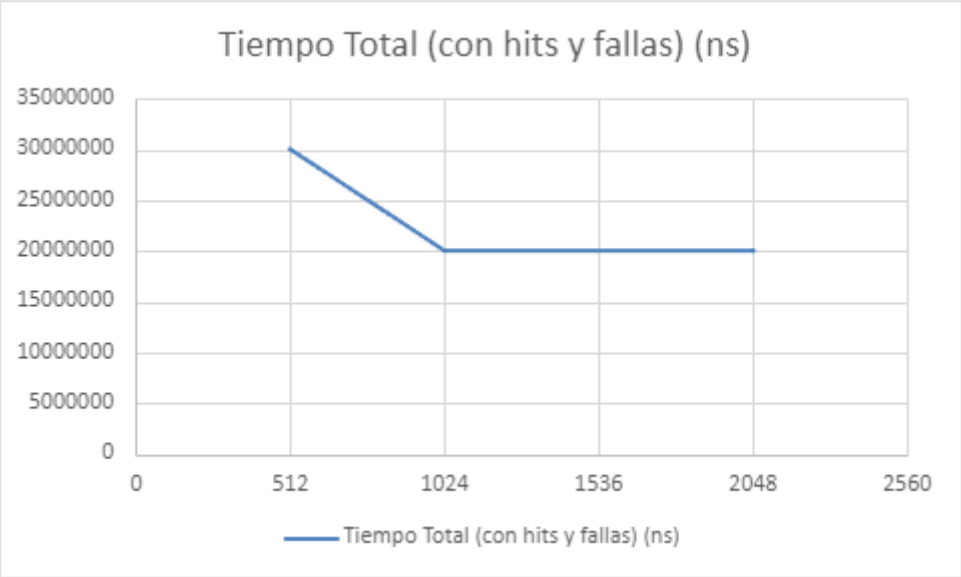

Tiempo Total (con hits y fallas) (ns)
1,6E+09
1,4E+09
1,2E+09
1E+09
800000000
600000000
400000000
200000000
0
0
512
1024
1536
2048
2560
Tiempo Total (con hits y fallas) (ns)

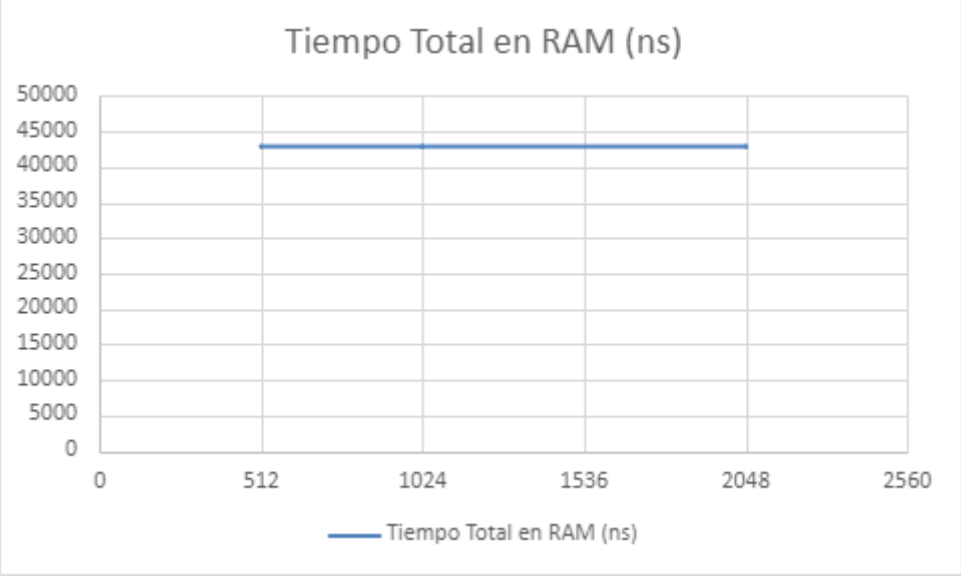

Tiempo Total en RAM (ns)
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
0
512
1024
1536
2048
2560
Tiempo Total en RAM (ns)

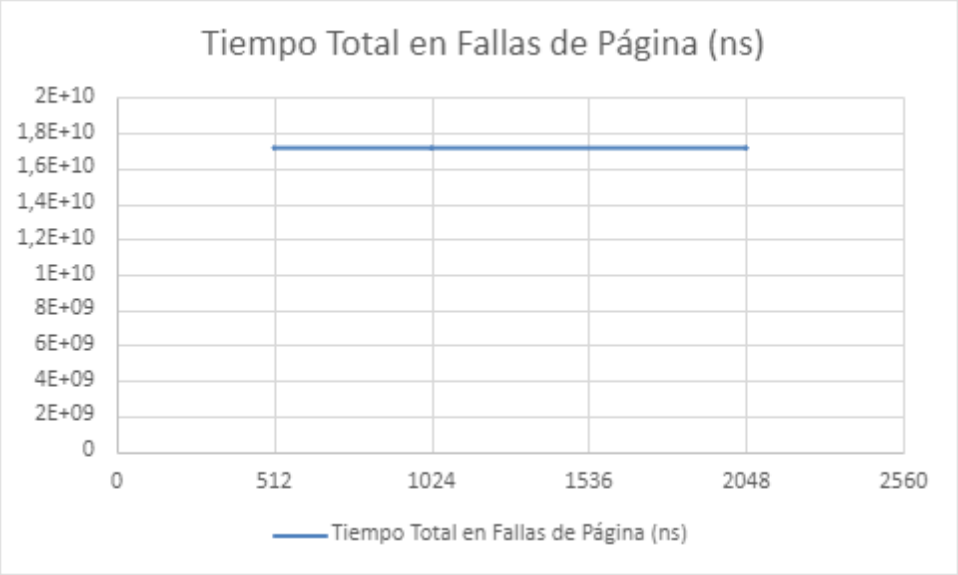

Tiempo Total en Fallas de Página (ns)
1,6E+12
1,4E+12
1,2E+12
1E+12
8E+11
6E+11
4E+11
2E+11
0
0
512
1024
1536
2048
2560
Tiempo Total en Fallas de Página (ns)

Graficas imagenTucan.bmp

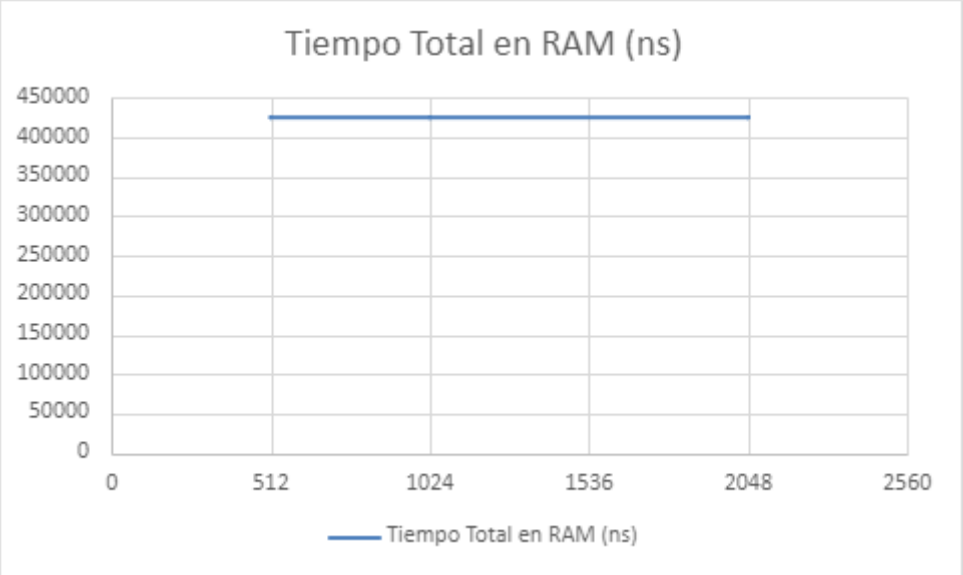
Mensaje de 100 caracteres			
Tamaño Página	**Tiempo Total (con hits y fallas) (ns)**	**Tiempo Total en RAM (ns)**	**Tiempo Total en Fallas de Página (ns)**
512	30042825	42900	17160000000
512	30042825	42900	17160000000
1024	20042850	42900	17160000000
1024	20042850	42900	17160000000
2048	20042850	42900	17160000000
2048	20042850	42900	17160000000

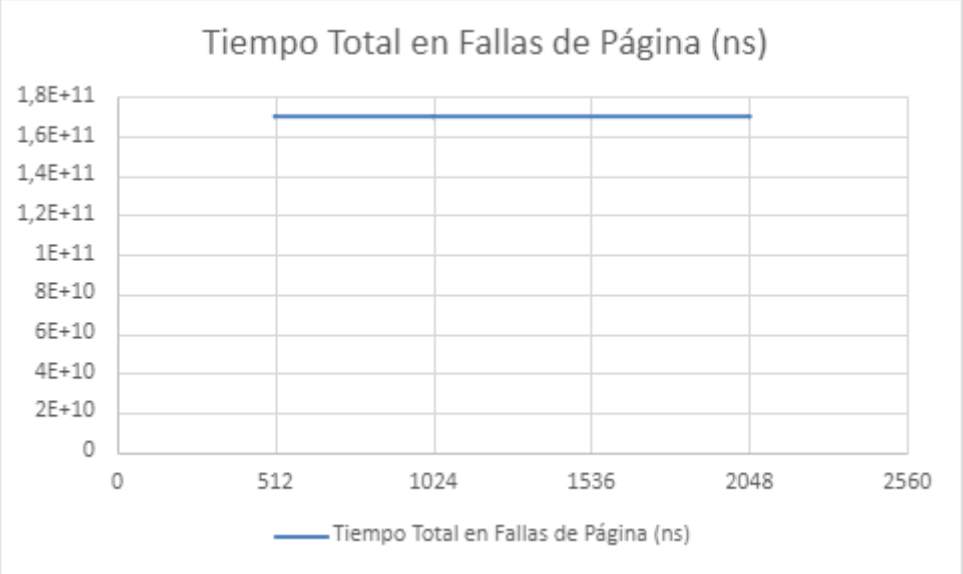




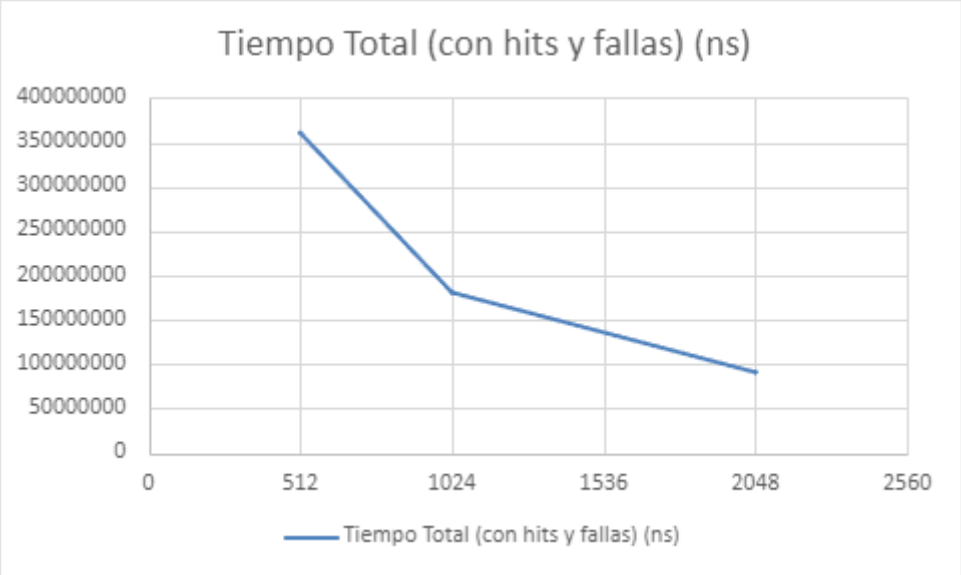


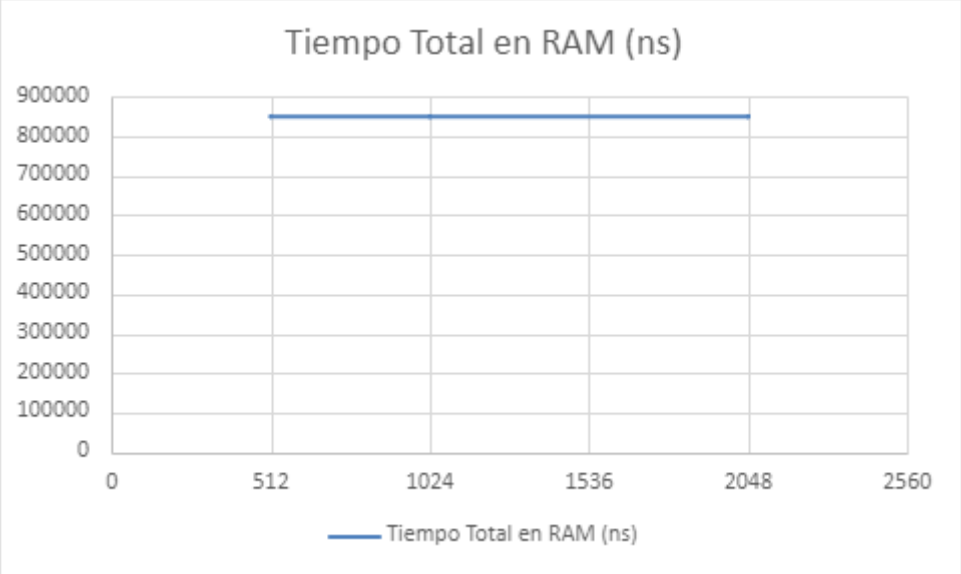
Mensaje de 1000 caracteres			
Tamaño Página	**Tiempo Total (con hits y fallas) (ns)**	**Tiempo Total en RAM (ns)**	**Tiempo Total en Fallas de Página (ns)**
512	180424950	425400	1,7016E+11
512	180424950	425400	1,7016E+11
1024	90425175	425400	1,7016E+11
1024	90425175	425400	1,7016E+11
2048	50425275	425400	1,7016E+11
2048	50425275	425400	1,7016E+11

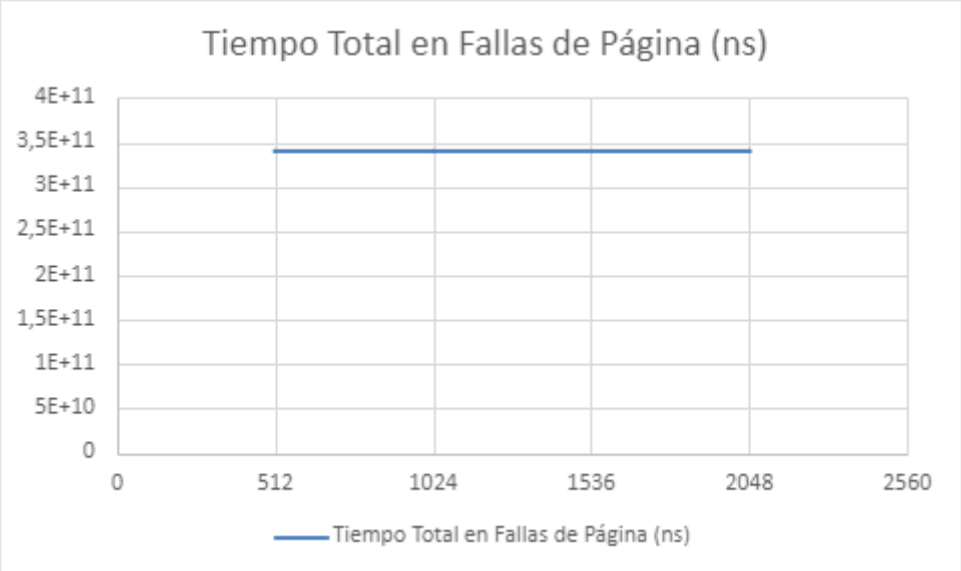




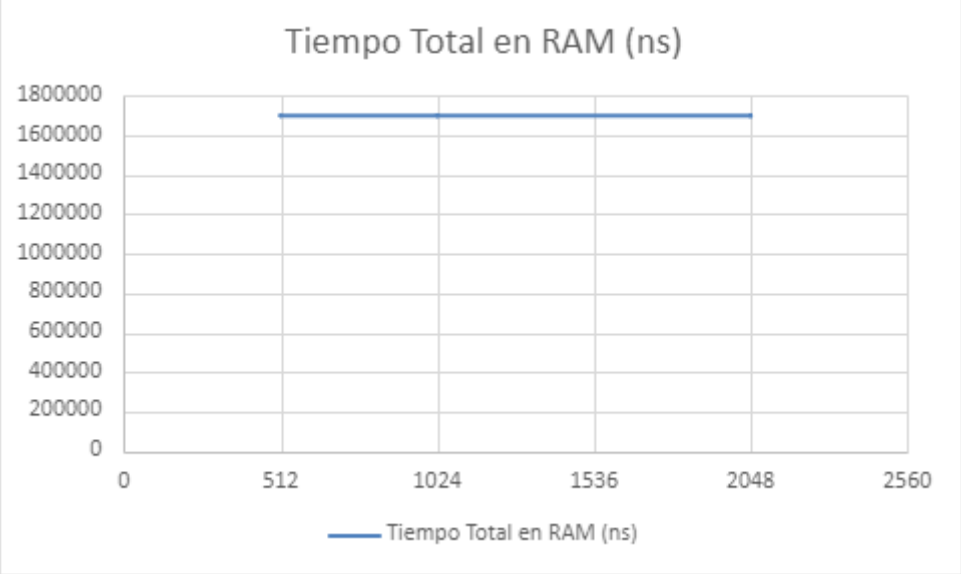
Mensaje de 2000 caracteres			
Tamaño Página	**Tiempo Total (con hits y fallas) (ns)**	**Tiempo Total en RAM (ns)**	**Tiempo Total en Fallas de Página (ns)**
512	360849500	850400	3,4016E+11
512	360849500	850400	3,4016E+11
1024	180849950	850400	3,4016E+11
1024	180849950	850400	3,4016E+11
2048	90850175	850400	3,4016E+11
2048	90850175	850400	3,4016E+11

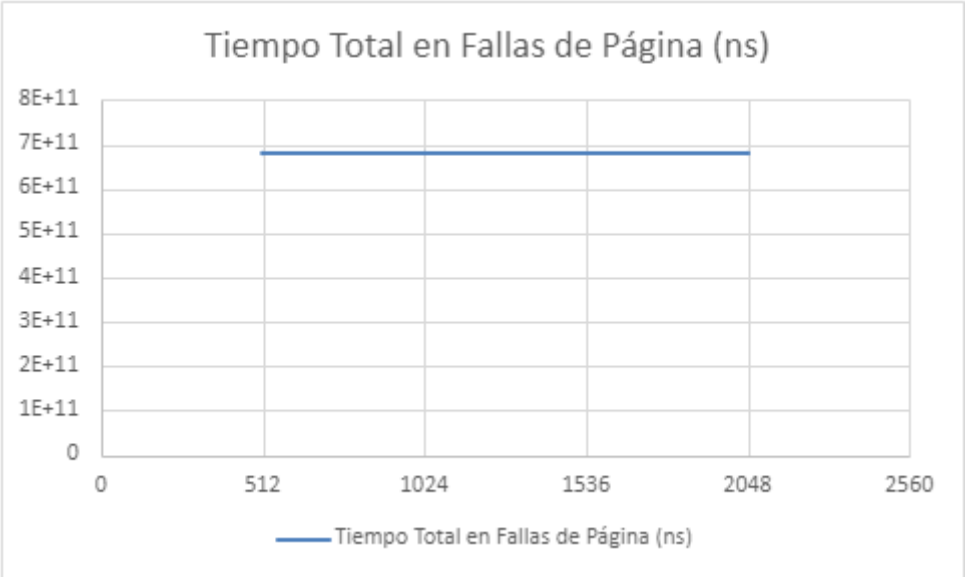

Tiempo Total (con hits y fallas) (ns)
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
0
512
1024
1536
2048
2560
Tiempo Total (con hits y fallas) (ns)


Tiempo Total en RAM (ns)
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
512
1024
1536
2048
2560
Tiempo Total en RAM (ns)

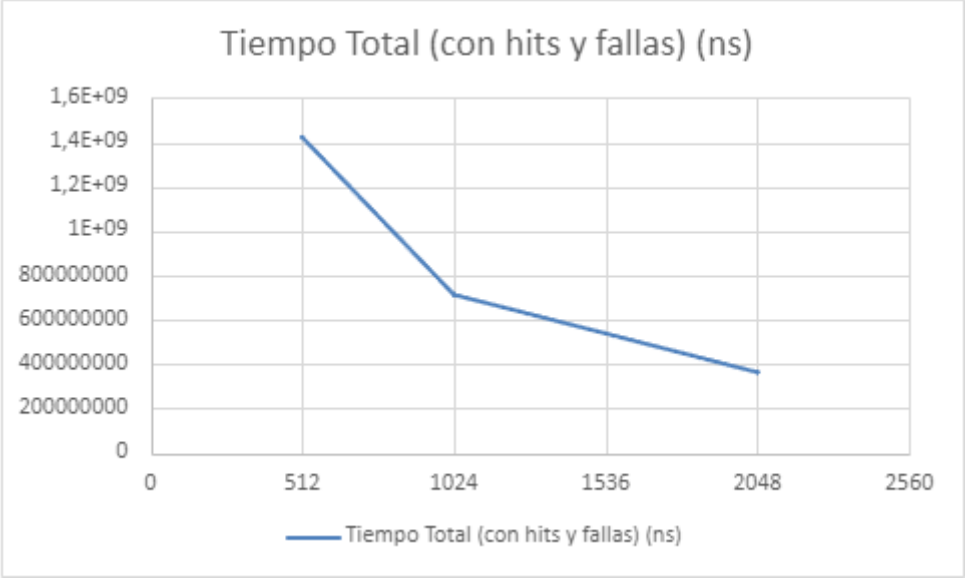

Tiempo Total en Fallas de Página (ns)
4E+11
3,5E+11
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
0
512
1024
1536
2048
2560
Tiempo Total en Fallas de Página (ns)

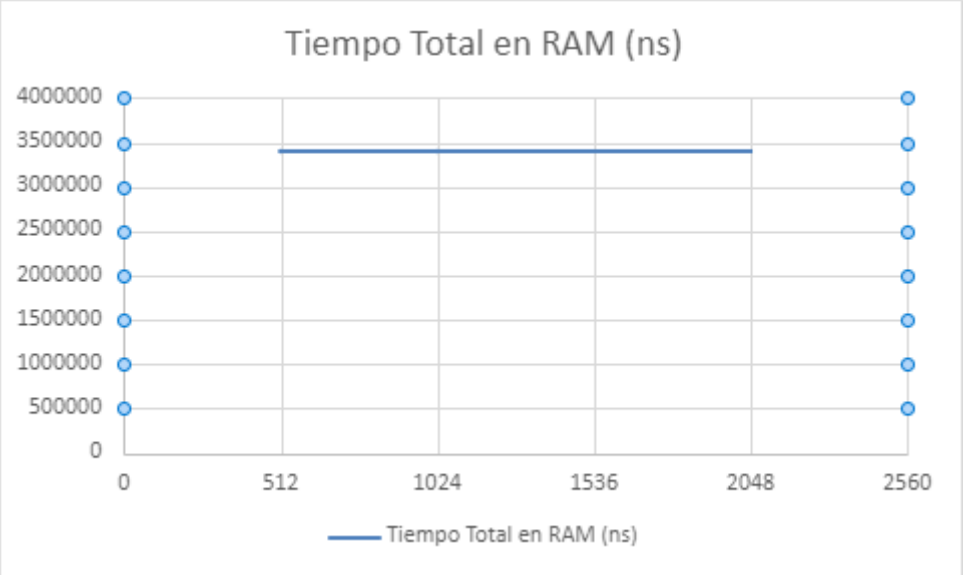
Mensaje de 4000 caracteres			
Tamaño Página	**Tiempo Total (con hits y fallas) (ns)**	**Tiempo Total en RAM (ns)**	**Tiempo Total en Fallas de Página (ns)**
512	711698625	1700400	6,8016E+11
512	721698600	1700400	6,8016E+11
1024	361699500	1700400	6,8016E+11
1024	361699500	1700400	6,8016E+11
2048	181699950	1700400	6,8016E+11
2048	181699950	1700400	6,8016E+11

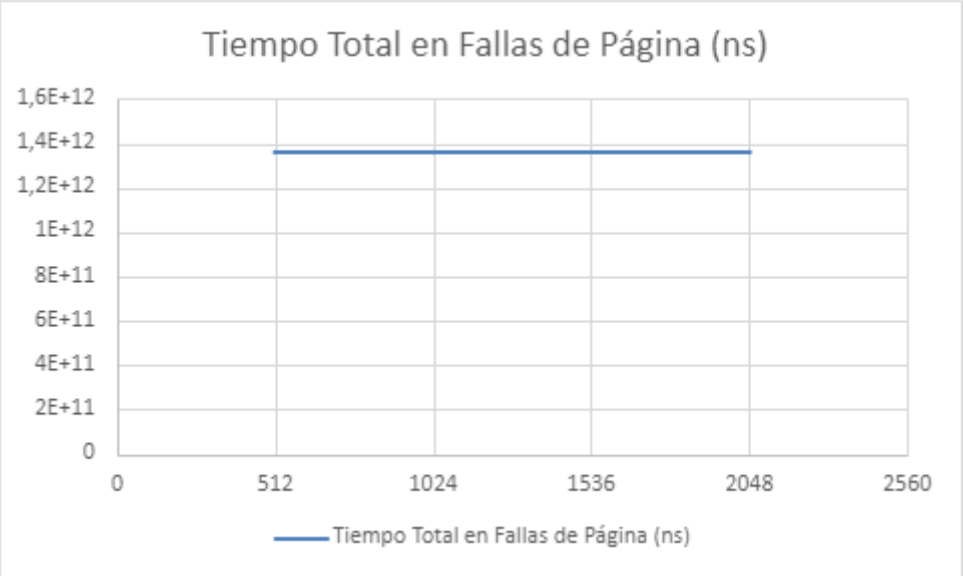




Mensaje de 8000 caracteres			
Tamaño Página	**Tiempo Total (con hits y fallas) (ns)**	**Tiempo Total en RAM (ns)**	**Tiempo Total en Fallas de Página (ns)**
512	1423396850	3400400	1,36016E+12
512	1423396850	3400400	1,36016E+12
1024	713398625	3400400	1,36016E+12
1024	713398625	3400400	1,36016E+12
2048	363399500	3400400	1,36016E+12
2048	363399500	3400400	1,36016E+12







Interpretación de Resultados

En las pruebas realizadas, los resultados del programa muestran cómo el porcentaje de hits y fallas de página varían según el número de marcos de página asignados y el tamaño de las páginas. A continuación, se hace una interpretación de los resultados obtenidos teniendo en cuenta cada combinación de imagen (imagen Ardilla e imagen Tucan), mensaje (100, 1000, 2000, 4000, 8000), marcos asignados (4,8) y tamaños de página (512, 1024, 2048):

En algunos escenarios, aumentar los marcos asignados no muestra una reducción significativa en las fallas de página. Por ejemplo:

Para el mensaje de 1000 caracteres con tamaño de página de 512 bytes, el número de fallas se mantiene en 18 tanto con 4 como con 8 marcos. Esto puede deberse a que la cantidad de marcos, aunque aumenta, ya es suficiente para manejar las referencias frecuentes del mensaje en ambos casos, por lo que no se

observa una mejora sustancial. Otro factor es que el tamaño del mensaje (1000 caracteres) no es lo suficientemente grande para saturar la memoria, incluso con solo 4 marcos disponibles. Por eso, la diferencia entre 4 y 8 marcos no es significativa en términos de reducción de fallas.

En las pruebas, se utilizaron diferentes tamaños de página (512, 1024 y 2048 bytes). Este cambio influye en el comportamiento del sistema, ya que un mayor tamaño de página puede reducir el número total de fallas. Por ejemplo, con mensaje de 2000 caracteres y 4 marcos, las fallas disminuyen de 36 con 512 bytes a 9 con 2048 bytes. Esto se debe a que, con páginas más grandes, cada página contiene más datos, lo que reduce el reemplazo.

Los mensajes más largos, como los de 8000 caracteres, incrementan significativamente el número de referencias y, por lo tanto, el número de fallas. Por ejemplo, para el mensaje de 8000 caracteres y 512 bytes con 4 marcos, se observan 142 fallas, que se reducen a 36 con 2048 bytes. Este comportamiento indica que, para mensajes más largos, es crucial tanto un tamaño de página mayor como un número adecuado de marcos para evitar la sobrecarga de reemplazos de página.

El tiempo total de simulación puede verse influenciado por el tamaño de la página y el número de fallas de página. Menos fallas llevan a tiempos de simulación más bajos, ya que se evita cargar páginas desde el disco constantemente y se necesita reemplazar las páginas con menos frecuencia, lo cual mejora el rendimiento temporal del sistema. Los mejores tiempos se logran cuando hay un balance adecuado entre el tamaño de la página y los marcos disponibles, permitiendo reducir tanto las referencias como las fallas.

Manejo diferente de la localidad del problema

Si la Localidad Fuera Mayor:

Cuando la localidad es alta, se incrementa la probabilidad de reutilización frecuente de las mismas páginas, lo que disminuye significativamente las fallas de página. Esto implica que el sistema funciona más eficientemente, ya que se minimizan los accesos al disco y los cambios de páginas en la RAM. El tiempo de acceso efectivo también se reduce, dado que los accesos a memoria RAM son más frecuentes que los accesos a disco. Este comportamiento evita problemas mejorando la utilización de CPU y asegurando un flujo de trabajo estable. Cuando hay más localidad (temporal y espacial), los datos que se necesitan con mayor frecuencia ya están cargados en la memoria, por lo que se reducen los fallos de página. Si sólo hubiera algunos pocos fallos de página, el tiempo total de simulación disminuiría, ya que el sistema no necesita cargar tanto desde el almacenamiento secundario.

Si la Localidad Fuera Menor:

En escenarios con baja localidad, se requiere cargar nuevas páginas con mayor frecuencia, incrementando las fallas de página. Esto provoca un aumento en los accesos al disco, afectando negativamente el rendimiento del sistema. Además, el tiempo de acceso efectivo se incrementa debido a las constantes operaciones de reemplazo de páginas. Como resultado, se requiere un número mayor de marcos para reducir la frecuencia de las fallas.

Con una menor localidad, las páginas referenciadas cambian con mayor frecuencia, lo que aumenta la probabilidad de que las páginas solicitadas no estén en memoria en un momento dado. Esto incrementa las fallas de página y reduce el porcentaje de hits. Debido a que habría más fallas de página, el tiempo

total de simulación sería mayor, ya que el sistema tendría que acceder más frecuentemente al almacenamiento secundario para cargar las páginas necesarias.