Bases and Bricks

José Luis Ávila Guzmán

No. de equipo de trabajo: 7

I. INTRODUCCIÓN

En el mundo de LEGO, a veces se nos complica más encontrar la pieza indicada en el manual, que la misma construcción. Esto puede generar frustración al momento de construir o crear un proyecto, puesto es más el tiempo que se toma buscar una pieza, que ensamblar y terminar la construcción. Por lo tanto, si se tienen todas las piezas debidamente clasificadas por atributos y ubicadas en un almacenamiento organizado, la experiencia con LEGO se va a disfrutar mucho más. Mediante una base de datos en una aplicación web llamada "Bases and Bricks", se pretende mejorar la experiencia al momento de clasificar y organizar las fichas de LEGO que se posean, disponiendo también de un buscador, el cual encontrará fácilmente la pieza que se requiera, mediante la introducción de algunos atributos de la ficha.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

Cuando se tiene una colección de LEGO, el principal problema es encontrar las piezas adecuadas para la construcción de algún set predeterminado o un proyecto que se tenga en mente. Esto se debe a que la marca dispone de aproximadamente 43000 piezas diferentes, disponibles hasta en 58 colores distintos (en esta primera entrega solo se tendrán en cuenta 11 colores diferentes). Debido a esta gran variedad de piezas, si no se tienen correctamente clasificadas y ubicadas, al momento de construir cualquier proyecto que se tenga en mente, va a ser muy difícil encontrar las piezas que se deseen usar.

III. USUARIOS DEL PRODUCTO DE SOFTWARE

Este proyecto impacta principalmente a los amantes de los legos, puesto que como se ha mencionado anteriormente, los ayuda para organizar sus inventarios. Pero también puede dársele uso en las instituciones educativas, debido a que los lego son una excelente herramienta de aprendizaje para todas las edades, ya que con estas piezas se pueden lograr proyectos de construcción muy interesantes para una pedagogía más didáctica y atrayente. Otros usuarios de este producto son los constructores profesionales de LEGO, ya que manejan una gran cantidad de fichas para sus construcciones, por lo tanto tienen un inventario muy grande y diverso de piezas, y gracias a esta herramienta podrán organizar mas fácilmente las fichas a su disposición.

IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

Cargar la base de datos:

- Descripción:

  Mediante esta funcionalidad se cargan los datos de todas las partes y sets que tiene Lego.

- Acciones iniciadoras y comportamiento esperado:

  Al inicio de la aplicación se tienen estructuras de datos vacías que se van llenando. Si uno de los archivos que se quiere leer no existe entonces se muestra un mensaje que dice "`El archivo x no existe. Revise que esté bien escrita su ruta y vuelva a ejecutar`". También, si uno de los archivos existe pero hay otro programa que lo está utilizando entonces se muestra el mensaje "`Hay otro programa que está abriendo el archivo x. Ciérrelo y vuelva a ejecutar.`". Si ninguna de estas situaciones ocurre se muestra un mensaje que dice "`Se han cargado satisfactoriamente los datos de sets y de partes.`".

- Requerimientos funcionales:

  Esta funcionalidad necesita de archivos separados por comas (.csv) para poder cargar los valores en las estructuras de datos. Cuando se intentan cargar estos archivos al programa de C# se tiene una serie de excepciones que anticipan los fallos mencionados anteriormente donde no se encuentre el archivo o esté ocupado por otro programa.

Filtrar piezas de Lego:

- Descripción:

  Mediante esta funcionalidad se podrán consultar y filtrar todas las piezas de Lego dados los parámetros de color o id.

- Acciones iniciadoras y comportamiento esperado:

  En el código fuente escribir una de las consultas y asignarla a una variable. Si la lista de piezas de Lego está vacía entonces imprime un mensaje en pantalla que dice `"No hay piezas para buscar"`. Además, en esta entrega solo se consideraron los colores `string[] colors = { "Red", "Magenta", "Brown", "Purple", "Violet", "Orange", "Pink", "Green", "Yellow", "Black", "Blue" }; y si se ingresa un color diferente se imprime un mensaje en pantalla que dice Console.WriteLine("No hay piezas de color " + givenColor);`

- Requerimientos funcionales:

  El software realiza las consultas sobre las estructuras de datos dados los parámetros de color o de id y regresa una lista de las piezas que coincidan con estos parámetros. Para esta primera entrega el contenido de esta lista se puede consultar mediante la terminal con el depurador de C#.

Lista de deseos:

- Descripción:

  Mediante esta funcionalidad el usuario podrá tener una cola con los sets que quiera adquirir.

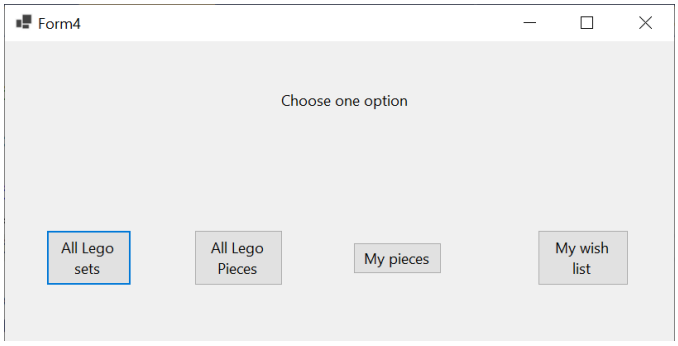
- Acciones iniciadoras y comportamiento esperado:

  En el código fuente escribir una de las consultas y asignarla a una variable.

- Requerimientos funcionales:

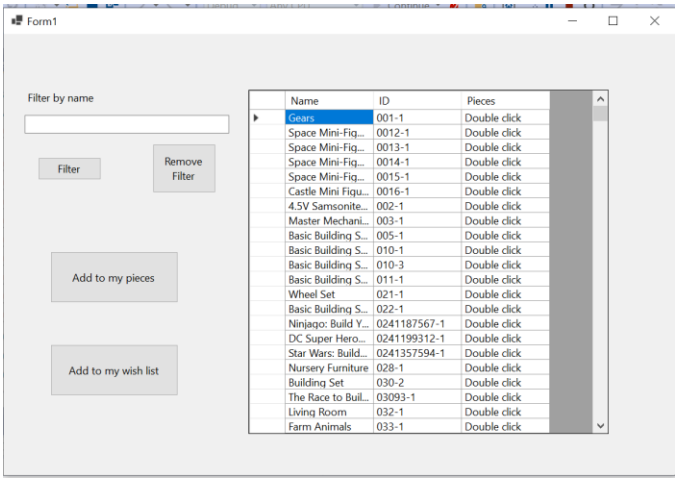
  Se ingresa un id de un set que se quiera construir (este id es el asignado por la empresa Lego) y este set se añade a una cola de sets por construir. Se puede consultar el primer elemento de esta cola para saber el set que se quiere adquirir primero. Para esta primera entrega el primer elemento de la cola se puede consultar mediante la terminal con el depurador de C#.

V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

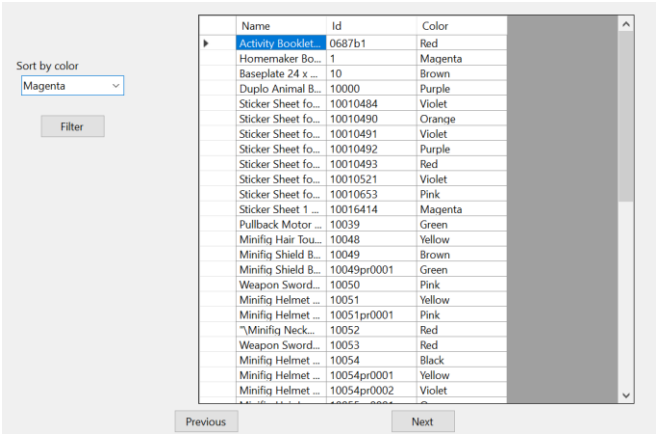


Ventana 1: Escoger una opción.

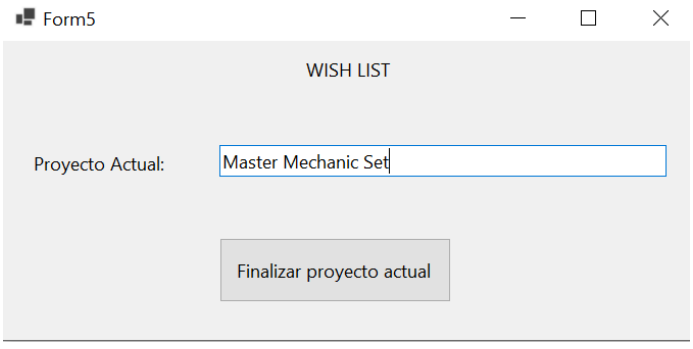
- All Lego sets: Lleva a la ventana 2
- All Lego Piece: Lleva a la ventana 3
- My pieces
- My wish List: Lleva a la ventana 5



Ventana 2: En esta ventana se tiene una vista de todos los sets de Lego. Adicionalmente se pueden filtrar por nombre, añadir un set a las piezas del usuario o añadir un set a la lista de deseos.



Ventanas 3 y 4: Muestra todas las piezas de Lego. Como son muchos registros se muestran en grupos de a 50 y se puede navegar entre ellos con los botones de Previous y Next. Adicionalmente se incluye el filtro por color mostrado anteriormente en los requerimientos funcionales. La diferencia entre las ventanas 3 y 4 es que la ventana 3 muestra todas las piezas de la empresa Lego, mientras que la ventana 4 muestra solo las piezas del usuario.



Ventana 5: Muestra el proyecto/set actual que el usuario quiere. Cuando se presiona el botón que dice “Finalizar proyecto actual” se hace dequeue y se muestra el siguiente set que está en la cola.

VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

Para esta primera entrega el programa será desarrollado en el lenguaje C# con Visual Studio 2019 para Windows. Los datos de todas las piezas de Lego se toman de la base de datos libre Rebrickable [1]. Estos datos luego son procesados con MySQL 8.0 para organizarlos y poder crear archivos .csv que serán usados en la aplicación de C#.

VII. PROTOTIPO DE SOFTWARE INICIAL

El prototipo de software inicial se encuentra en el repositorio de GitHub https://github.com/avila131/Estructuras-Lego.

VIII. IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

Listas:

Las listas encadenadas serán usadas para el almacenamiento de set existentes tanto en la marca LEGO como en el inventario personal de cada usuario.

Arreglos:

Los arreglos serán implementados en la lista de las piezas de marca LEGO. Esto porque solo se tienen cerca de 45 mil piezas en total y al agregarlas a un arreglo dinámico se puede aprovechar el hecho de que estén indexadas para reducir los tiempos de búsqueda.

Pilas:

Se utiliza una pila para organizar todas las piezas que tiene un set:

Set ID	Set Name	Piece ID	Quantity	Piece index
001-1	Gears	132a	4	2193
001-1	Gears	3020	4	9612
001-1	Gears	3062c	1	10504
001-1	Gears	3404bc01	4	13538
001-1	Gears	36	4	14068
001-1	Gears	7039	6	27684
001-1	Gears	7049b	4	27695
001-1	Gears	715	4	27801
001-1	Gears	741	4	28188
001-1	Gears	742	4	28198
001-1	Gears	743	1	28205
001-1	Gears	744	3	28206

El archivo .csv tiene varias líneas con el mismo set, entonces se utiliza una pila para agrupar las piezas en un solo objeto set: Se ingresa un set a la pila y la siguiente línea tiene el mismo “SET ID” entonces no crea un nuevo objeto sino que agrega las piezas al objeto que está en la parte superior de la pila. Cuando la siguiente lectura de “SET ID” sea diferente de stack.Peek() entonces se crea un nuevo objeto, se inserta al stack y se repite el proceso.

Colas:

Las colas se implementarán en la “Lista de deseos” que corresponde a los sets que el usuario desea armar, aprovechando sus características para usarlas como una lista de prioridades.

IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

Cargar la base de datos O(n)

```
1  public static void pushLegoSet(ref string[] line, ref LinkedList<LegoSet> list, ref Stack<LegoSet> objectStack)
2  {
3      LegoSet newLegoSet = new LegoSet();
4      newLegoSet.id_set = line[0];
5      newLegoSet.nameSet = line[1];
6      newLegoSet.addPiece(line[2], line[3], line[4]);
7      objectStack.Push(newLegoSet);                    O(1)
8      list.pushBack(newLegoSet);O(1)
9  }                              O(1)
10
11
12 string path = "C:/Users/josel/Documents/Estructuras/Database/Filtered queries/set_parts.csv";
13 string[] lines = System.IO.File.ReadAllLines(path);
14
15
16 // Load Sets_Parts
17 foreach (string line in lines)  n veces
18 {
19     string[] columns = line.Split(','); cte
20     if (myStack.Count == 0) // Stack is empty
21         pushLegoSet(ref columns, ref setsLinkedList, ref myStack); cte
22     else if (myStack.Peek().id_set != columns[0]) // Different set
23         pushLegoSet(ref columns, ref setsLinkedList, ref myStack); cte
24     else
25         myStack.Peek().addPiece(columns[2], columns[3], columns[4]); cte
26 }
27
```

Solo hay un ciclo en esta funcionalidad. Este ciclo recorre cada uno de los registros que se encuentran en un archivo .csv. De esta manera, la notación BigO es O(n).

Eliminar una Pieza dado su ID O(n)

```
1  public static void removePieceByID(ref DynamicArray<Piece> legoPiecesList, string givenID)
2  {
3      int removeIndex = legoPiecesList.arraySize - 1;
4      for (int i = 0; i < legoPiecesList.arraySize; i++) n - removeIndex
5          if (legoPiecesList.data[i].lego_id == givenID)
6              removeIndex = i;
7      for (int i = removeIndex; i < legoPiecesList.arraySize - 1; i++)remove Index
8          legoPiecesList.data[i] = legoPiecesList.data[i + 1];
9  }                                                                          } n
```

A pesar de que en esta funcionalidad se ven dos ciclos, el recorrido de ambos juntos es n. El procedimiento particular que se desea realizar en este caso es eliminar un elemento de un arreglo dinámico. El primer ciclo encuentra el índice del elemento que se quiere eliminar y el segundo ciclo mueve todos los elementos a la derecha de este índice una casilla a la izquierda. Como el recorrido de ambos ciclos es n, la notación BigO es O(n).

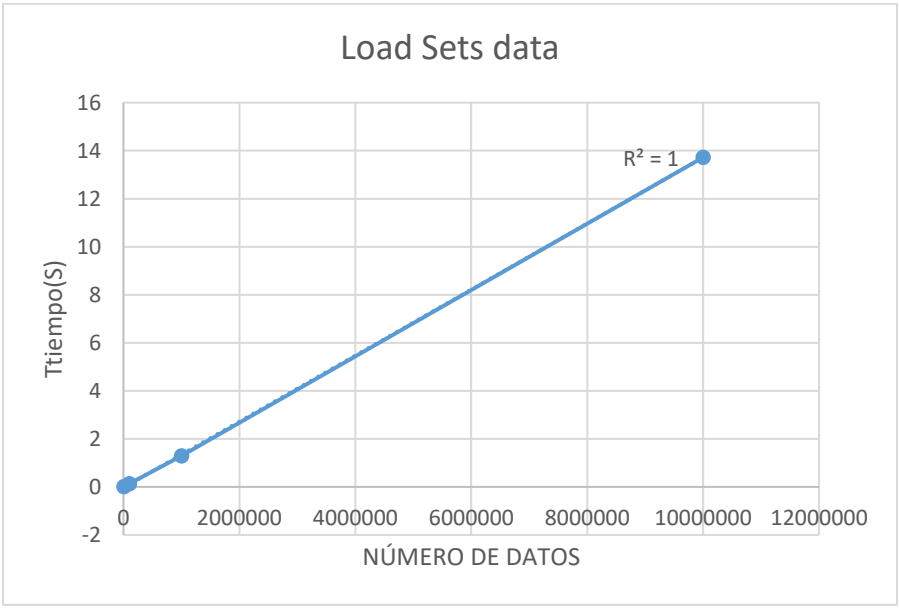
PopBack lista simplemente encadenada O(n)

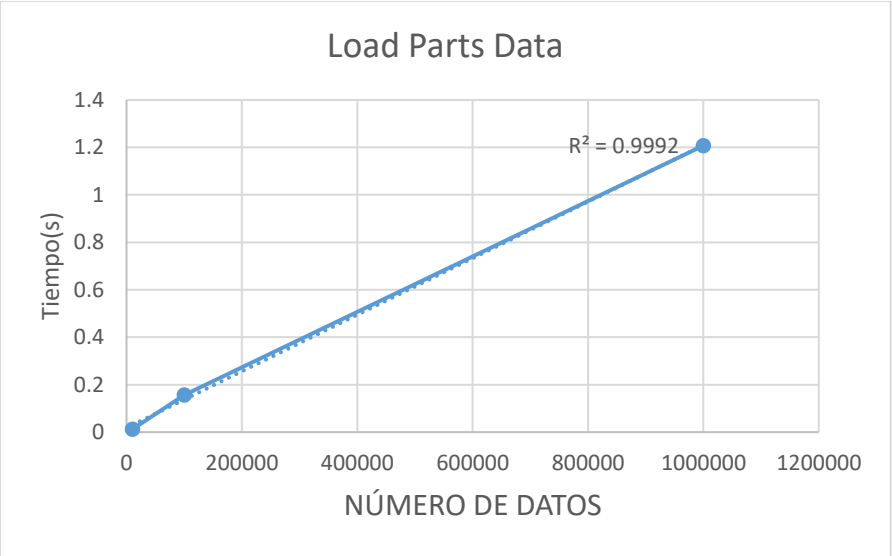
```
1   public T popBack()
2   {
3       try
4       {
5           if (head.next == null)
6           {
7               head = null;
8               tail = null;
9           }
10          Node<T> iteratorPointer = head;
11          while (iteratorPointer.next.next != null)
12              iteratorPointer = iteratorPointer.next;
13          iteratorPointer.next = null;
14          T returnValue = tail.key;
15          tail = iteratorPointer;
16          return returnValue;
17      }
18      catch (Exception e)
19      { throw new Exception("List is empty"); }
20  }
```

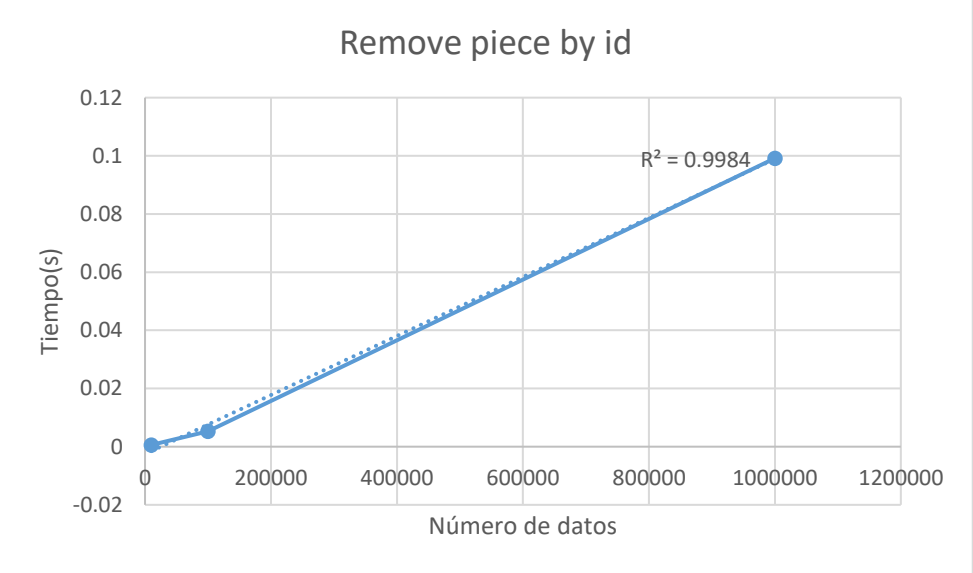
Otra funcionalidad interesante es eliminar de una lista encadenada. Es importante aclarar que esta es una lista simplemente encadenada, por eso debe recorrer todos los elementos de la lista para encontrar el penúltimo elemento y poder quitar el último elemento. Con este fin se tiene un ciclo while. Como solo hay un ciclo while, la notación Big O es O(n).

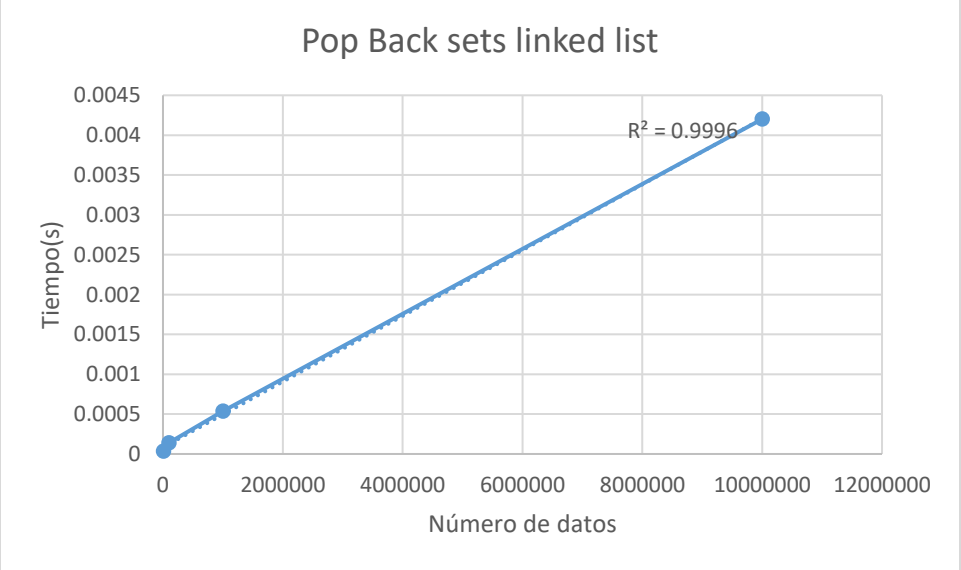
Resultados análisis comparativo

	Load Sets data	Load Parts Data	Remove piece by id	Pop Back sets linked list
10000	*0.009942*	*0.0118441*	*0.000596*	*0.000035*
100000	*0.134635*	*0.1568757*	*0.005274*	*0.0001387*
1000000	*1.28967*	*1.2076322*	*0.099151*	*0.0005387*
10000000	*13.72169*	*No data*	*No data*	*0.0042013*
100000000	*No data*	*No data*	*No data*	*No data*




Load Parts Data
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Tiempo(s)
R² = 0.9992
0
200000
400000
600000
800000
1000000
1200000
NÚMERO DE DATOS


Remove piece by id
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
Tiempo(s)
R² = 0.9984
0
200000
400000
600000
800000
1000000
1200000
Número de datos


Pop Back sets linked list
0.0045
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
Tiempo(s)
R² = 0.9996
0
2000000
4000000
6000000
8000000
10000000
12000000
Número de datos

El análisis comparativo muestra que la operación de cargar los datos de los sets es más eficiente que cargar los datos a las partes. Esto ocurre porque los sets se organizan en listas simplemente encadenadas (cuya dificultad de agregar un elemento al final es O(1)) mientras que las partes están organizadas en un arreglo dinámico. Este arreglo dinámico, a pesar de que el análisis amortizado sugiere que el costo de agregar elementos es O(1)[2], es comparativamente más lento que agregar a una lista encadenada. Incluso, el dato que se ve de las lecturas de los sets incluye reorganizar la información mediante el uso de una pila, así que el tiempo registrado en la primera columna incluye también estas operaciones de lectura y push sobre la pila. Aún así, el tiempo obtenido es menor que agregar a un arreglo dinámico.

Otra situación interesante que se observa de estas tablas es que no se tiene ningún tiempo para 100 millones de datos. Esto ocurre porque el programa obtiene sus datos a partir de un archivo .csv > 5Gb que leer, donde cada línea es un set diferente, por lo que debe crear un objeto por cada línea que lea e insertarlo a las estructuras de datos.

Además, se observa que no se tienen datos para carga de 10 millones de piezas. Esto ocurre porque se deben cargar 10 millones de piezas en un arreglo dinámico y esta operación tarda mucho. En nuestro diseño consideramos que, al solo existir cerca de 45 mil piezas diferentes en toda la historia de Lego, un arreglo dinámico era adecuado para reducir el tiempo de búsqueda y poder tener las piezas indexadas. Esto se confirma también con la tabla de resultados del análisis comparativo, donde cargar hasta 1 millón de datos se hacía en menos de 1 segundo.

En cuanto a las gráficas realizadas, se evidencia que estas corresponden a los análisis hechos a mano al inicio de esta sección de análisis comparativo. Se esperaba que las 4 operaciones propuestas fueran de complejidad O(n) lo cual corresponde con una función lineal. Este comportamiento esperado se reflejó en las gráficas realizadas, donde se evidencia un comportamiento lineal. Este dato puede confirmarse además porque coeficientes de correlación lineal R^2 son muy cercanos a 1.

Por último, en cuanto a factores que pueden afectar los resultados de la tabla de análisis comparativo: El programa se ejecutó a través del depurador del IDE Visual Studio 2019. Es necesario ejecutar el archivo C# de esta manera para poder tomar los tiempos de ejecución. Este IDE requería bastante memoria RAM para ejecutarse, y pudo haber afectado los resultados obtenidos. Además, la máquina en la que se prueban los datos tiene 4GB de memoria RAM, lo cual puede también limitar la cantidad de objetos que se pueden crear en tiempo de ejecución.

X. INFORMACIÓN DE ACCESO AL VIDEO DEMOSTRATIVO DEL PROTOTIPO DE SOFTWARE

Video Demostrativo

XI. ROLES Y ACTIVIDADES

Rol	Actividades fundamentales
Líder(esa)	Consultar a los otros miembros del equipo, atento que la información sea constante para todos. Aportar con la organización y plan de trabajo.
Coordinador(a)	Mantener el contacto entre todos, Programar y agendar reuniones; ser facilitador para el acceso a los recursos.
Experto(a)	Líder técnico que propende por coordina funciones y actividades operativas.
Investigador(a)	Consultar otras fuentes. Propender por resolver inquietudes comunes para todo el equipo.
Observador(a)	Siempre está atento en el desarrollo del proyecto y aporta en el momento apropiado cuando se requiera apoyo adicional por parte del equipo.
Animador(a)	Energía positiva, motivador en el grupo.
Secretario(a)	Se convierte en un facilitador de la comunicación en el grupo. Documenta (actas) de los acuerdos/compromisos realizados en las reuniones del equipo.
Técnico(a)	Aporta técnicamente en el desarrollo del proyecto.

INTEGRANTE	ROL(ES)	ACTIVIDADES REALIZADAS (Listado)
José Luis Ávila Guzman	Observador	Codificación de las estructuras
		Creación del repositorio
	Secretario	Creación de la interfaz
		Organización de tareas
José Luis Ávila Guzman	Líder	Documento escrito
		Presentación
	Investigador	Inquietudes y observaciones
		Revisión de calidad

XII. DIFICULTADES Y LECCIONES APRENDIDAS

Mis compañeros cancelaron la materia un día antes de la entrega y tuve que hacer todo el trabajo de manera individual.
En cuanto a la parte técnica, pensaba que el tiempo de agregar al final a una lista encadenada con cola y a un arreglo dinámico iba a ser igual porque en clase concluimos que ambas operaciones tenían una dificultad de O(1). Sin embargo, al realizar el análisis comparativo me di cuenta de que la lista encadenada es mucho más eficiente al agregar elementos.

XIII. BIBLIOGRAFÍA

[1] "Rebrickable Help Guide: Frequently Asked Questions | Rebrickable - Build with LEGO", *Rebrickable.com*. [Online]. Available: https://rebrickable.com/help/faq/. [Accessed: 28- Nov- 2021].

[2] N. Rhodes, "Basic Data Structures: Dynamic Arrays and Amortized Analysis", *Coursera*. [Online]. Available: https://d3c33hcgiwev3.cloudfront.net/_72d29db2f2280185e66f0a77ada6d61f_05_4_dynamic_arrays_and_amortized_analysis.pdf?Expires=1638230400&Signature=OCtloBnmfH2~jBlSkkE~VeYAPj8dnEOxRTZvOffbieC8sohBruCdtWozElXyR4~8VyomwX8Wq~iTtF2-OeKF2TcnlXpro6SM4kZCA8GGbuYcFGv66gR3bKfgnKw4X5SBK13L444ZKRK~2dT6QwvuIe90ht~BbRwrHYBnqaoz7sM_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A. [Accessed: 28- Nov- 2021].