OBSERVACIONES DE LA PRÁCTICA

1. Juan David Forero Huerfano, jd.foreroh1@uniandes.edu.co, 202411707.
2. David Felipe Mendoza Carrillo, d.mendozacarrillo@uniandes.edu.co, 202421689.
3. Juan José Hernández Vera, jj.hernandezve1@uniandes.edu.co, 202423465.

Ambientes de pruebas

	Máquina 1	Máquina 2	Máquina 3
Procesadores	Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz 1.19 GHz	Intel(R) evo i7	Intel(R) core i5
Memoria RAM (GB)	8,00 Gb	32,00 Gb	16,00 Gb
Sistema Operativo	Windows	Windows	Windows

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

Máquina 1

Resultados para Queue con Array List

Porcentaje de la muestra [pct]	enqueue (Array List)	dequeue (Array List)	peek (Array List)
0,50%	**0.112**	**0.074**	**0.004**
5,00%	**1.552**	**0.709**	**0.007**
10,00%	**3.577**	**1.221**	**0.003**
20,00%	**4.099**	**2.341**	**0.003**
30,00%	**5.364**	**3.656**	**0.003**
50,00%	**12.589**	**12.500**	**0.003**
80,00%	**16.585**	**16.033**	**0.004**
100,00%	**19.871**	**10.667**	**0.003**

Resultados para Stack con Array List

Porcentaje de la muestra [pct]	push (Array List)	pop (Array List)	top(Array List)
0,50%	**0.056**	**0.051**	**0.003**
5,00%	**1.631**	**0.648**	**0.006**
10,00%	**1.500**	**1.261**	**0.003**
20,00%	**2.488**	**5.676**	**0.002**
30,00%	**3.299**	**14.082**	**0.002**
50,00%	**5.379**	**37.930**	**0.002**
80,00%	**5.332**	**85.559**	**0.001**
100,00%	**6.426**	**128.989**	**0.002**

Resultados para Queue con Linked List

Porcentaje de la muestra [pct]	enqueue (Linked List)	dequeue (Linked List)	peek (Linked List)
0,50%	**0.285**	**0.069**	**0.004**
5,00%	**9.373**	**0.433**	**0.002**
10,00%	**38.178**	**0.497**	**0.001**
20,00%	**117.509**	**1.015**	**0.001**
30,00%	**324.894**	**1.541**	**0.001**
50,00%	**960.308**	**2.621**	**0.002**
80,00%	**2.269.160**	**5.811**	**0.002**
100,00%	**3.636.512**	**7.238**	**0.002**

Resultados para Stack con Linked List

Porcentaje de la muestra [pct]	push (Linked List)	pop (Linked List)	top(Linked List)
0,50%	**0.209**	**0.066**	**0.003**
5,00%	**10.562**	**0.480**	**0.002**
10,00%	**25.813**	**0.510**	**0.001**
20,00%	**95.561**	**4.437**	**0.002**
30,00%	**291.575**	**6.613**	**0.001**
50,00%	**985.616**	**37.401**	**0.002**
80,00%	**2.186.212**	**91.668**	**0.002**
100,00%	**3.623.251**	**180.501**	**0.002**

Máquina 2

Resultados para Queue con Array List

Porcentaje de la muestra [pct]	enqueue (Array List)	dequeue (Array List)	peek (Array List)
0,50%	**0.116**	**0.029**	**0.003**
5,00%	**0.400**	**0.250**	**0.002**
10,00%	**0.402**	**0.267**	**0.001**
20,00%	**0.964**	**0.654**	**0.001**
30,00%	**2.911**	**1.616**	**0.001**
50,00%	**3.776**	**2.526**	**0.001**
80,00%	**5.699**	**3.698**	**0.001**
100,00%	**8.145**	**6.337**	**0.001**

Resultados para Stack con Array List

Porcentaje de la muestra [pct]	push (Array List)	pop (Array List)	top(Array List)
0,50%	**0.033**	**0.048**	**0.002**
5,00%	**0.270**	**0.271**	**0.001**
10,00%	**0.263**	**0.262**	**0.000**
20,00%	**0.598**	**3.303**	**0.001**
30,00%	**1.471**	**10.892**	**0.001**
50,00%	**2.368**	**26.603**	**0.001**
80,00%	**2.254**	**76.078**	**0.003**
100,00%	**2.792**	**98.385**	**0.001**

Resultados para Queue con Linked List

Porcentaje de la muestra [pct]	enqueue (Linked List)	dequeue (Linked List)	peek (Linked List)
0,50%	**0.182**	**0.034**	**0.003**
5,00%	**6.783**	**0.179**	**0.002**
10,00%	**24.694**	**0.265**	**0.001**
20,00%	**71.132**	**0.629**	**0.001**
30,00%	**149.653**	**0.790**	**0.001**
50,00%	**383.299**	**2.045**	**0.001**
80,00%	**1023,66**	**2.469**	**0.001**
100,00%	**1578,348**	**3.166**	**0.001**

Resultados para Stack con Linked List

Porcentaje de la muestra [pct]	push (Linked List)	pop (Linked List)	top(Linked List)
0,50%	**0.118**	**0.031**	**0.003**
5,00%	**4.607**	**0.125**	**0.001**
10,00%	**15.989**	**0.265**	**0.001**
20,00%	**56.834**	**3.229**	**0.001**
30,00%	**137.259**	**7.897**	**0.001**
50,00%	**384.597**	**25.094**	**0.001**
80,00%	**999.772**	**63.032**	**0.002**
100,00%	**1532,23**	**93.206**	**0.002**

Máquina 3

Resultados para Queue con Array List

Porcentaje de la muestra [pct]	enqueue (Array List)	dequeue (Array List)	peek (Array List)
0,50%	**0,108**	**0,052**	**0,006**
5,00%	**0,657**	**0,307**	**0,003**
10,00%	**0,925**	**0,545**	**0,003**
20,00%	**2,134**	**1,496**	**0,003**
30,00%	**2,69**	**1,414**	**0,002**
50,00%	**4,566**	**3,12**	**0,001**
80,00%	**4,091**	**2,474**	**0,001**
100,00%	**8,761**	**3,893**	**0,001**

Resultados para Stack con Array List

Porcentaje de la muestra [pct]	push (Array List)	pop (Array List)	top(Array List)
0,50%	**0,055**	**0,049**	**0,005**
5,00%	**0,332**	**0,282**	**0,001**
10,00%	**0,589**	**0,572**	**0,002**
20,00%	**1,515**	**8,732**	**0,002**
30,00%	**1,518**	**11,144**	**0,001**
50,00%	**3,845**	**28,752**	**0,007**
80,00%	**2,283**	**86,458**	**0,001**
100,00%	**3,801**	**112,947**	**0,001**

Resultados para Queue con Linked List

Porcentaje de la muestra [pct]	enqueue (Linked List)	dequeue (Linked List)	peek (Linked List)
0,50%	**0,138**	**0,045**	**0,002**
5,00%	**6,118**	**0,234**	**0,002**
10,00%	**24,061**	**0,622**	**0,002**
20,00%	**95,462**	**1,232**	**0,003**
30,00%	**158,759**	**1,039**	**0,002**
50,00%	**450,675**	**1,673**	**0,001**
80,00%	**1106,635**	**2,979**	**0,001**
100,00%	**1789,626**	**3,413**	**0,001**

Resultados para Stack con Linked List

Porcentaje de la muestra [pct]	push (Linked List)	pop (Linked List)	top(Linked List)
0,50%	**0,118**	**0,028**	**0,002**
5,00%	**5,399**	**0,24**	**0,002**
10,00%	**17,431**	**0,424**	**0,002**
20,00%	**80,395**	**3,649**	**0,004**
30,00%	**172,356**	**11,971**	**0,003**
50,00%	**474,269**	**26,903**	**0,001**
80,00%	**1131,379**	**90,998**	**0,007**
100,00%	**1768,797**	**121,139**	**0,001**

Preguntas de análisis

1. ¿Se observan diferencias significativas entre las implementaciones con ArrayList y LinkedList para las funciones de Queue y Stack? ¿Cuál es más eficiente en cada operación? ¿Por qué una implementación es más rápida en ciertos casos?
   Se pueden observar diferencias significativas entre las implementaciones de ArrayList y SingleLinkedList para Queue y Stack. Cuando hay grandes tamaños de datos en queue.enqueue, ArrayList es mucho más eficiente comparado con SingleLinkedList. En cambio, queue. dequeue es más eficiente con SingleLinkedList, porque son Array es necesario desplazar las posiciones de todos los elementos.
   En Stack, push y pop, la eficiencia es mucho menor con SingleLinkedList a medida que aumenta el número de elementos. En este caso, array maneja de mejor manera los tiempos de operación gracias al acceso directo por índices. Por último, en ambos casos Peek y Top tienen una eficiencia rápida sin importar el numero de elementos, gracias a que ambas estructuras permiten acceder rápidamente al primer elemento.
   LinkedList resultó ser mas eficiente para eliminaciones al inicio, pero mucho más ineficiente en inserciones y eliminaciones al final, gracias al manejo de referencias.

2. ¿Cuándo es preferible usar ArrayList o LinkedList? Si insertamos y eliminamos con frecuencia, ¿qué estructura conviene más? Si accedemos aleatoriamente a elementos, ¿cuál es más eficiente?
   En el caso del queue, lo mejor es usar una SingleLinkedList ya que al tener nodos y referencias en su interior asemeja más en su funcionamiento a una cola, donde el primero en llegar es el primero en irse. Por otro lado, la implementación de array listas es mejor para Stacks dado que al tener posibilidad de acceso por posición, permite facilidad para interactuar entre los elementos de la pila sin tener que hacer recorridos completos por la misma. A la hora de insertar y eliminar con frecuencia, lo mejor es usar una lista enlazada simple dado que no requiere un desplazamiento

de elementos y el inicio es de fácil acceso. Para el acceso aleatorio de elementos lo mejor es usar un ArrayList debido a que, al poder acceder por índice, no necesita un recorrido de la lista para acceder al elemento buscado. Por esta razón, es importante entender la funcionalidad y el objetivo de la función a realizar.

3. Durante la ejecución de las pruebas ¿Se presentan anomalías en los tiempos de ejecución que no se explican con la teoría?

   Durante la ejecución de las pruebas, se identificaron anomalías en tres de las seis funciones utilizadas. La primera anomalía se encuentra evidentemente en la función Enqueue para la gráfica de Single-linked list. Se evidencia que el tiempo de ejecución tiene una tendencia a crecer exponencialmente, lo cual contrasta con la complejidad temporal esperada, siendo esta O(n) en lugar de O(1). Del mismo modo, para Single-linked list se encontraron anomalías en la función push, donde la teoría narra que se espera que la complejidad temporal de la función stack sea de O(1). Al evaluar los datos obtenidos, se presenta una gráfica con tendencia exponencial, la cual va en contra de los valores teóricos. Finalmente, la función pop para Array, la cual crece exponencialmente conforme se aumenta el tamaño de la muestra. Esto no coincide con la teoría que propone una complejidad temporal lineal O(1) para la función pop.

4. Complete la siguiente tabla de acuerdo con qué operación es más eficiente en cada implementación (marque con una x la que es más eficiente). Adicionalmente, explique si este comportamiento es acorde con lo enunciado en la teoría. Justifique las respuestas.

		Array List	Linked List	Justificación
QUEUE	**Enqueue()**	X		En ArrayList, la inserción al final es O (1), en cambio en LinkedList aumenta de acuerdo al número de elementos.
	Dequeue()		X	LinkedList permite eliminar el primer elemento sin tener que modificar las posiciones de los otros elementos y es O (1), mientras que ArrayList tiene que modificar las posiciones del resto de elementos.
	Peek()	X	X	Es igual de eficiente en ambos casos. El tiempo de respuesta no varía por el número de elementos.
STACK	**Push()**	X		ArrayList es mas eficiente porque agregar al final es de temporalidad O (1), mientras que en LinkedList, el tiempo crece con respecto al número de elementos.
	Pop()	X		En ArrayList se puede eliminar el ultimo elemento sin modificar las posiciones del resto de elementos. En cambio LinkedList tiene que recorrer todos los elementos para actualizar la referencia.
	Top()	X	X	Es igual de eficiente en ambos casos. El tiempo de respuesta no varía por el número de elementos.