OBSERVACIONES DE LA PRÁCTICA

Estudiante 1 Cod 202520818
Estudiante 2 Cod XXXX
Estudiante 3 Cod XXXX

Ambientes de pruebas

	Máquina 1	Máquina 2	Máquina 3
Procesadores	M1, 2020	Intel Core i7	
Memoria RAM (GB)	8	16	
Sistema Operativo	macOS	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]	Tamaño de la muestra (ARRAY_LIST)	enqueue (Array List)	dequeue (Array List)
0.50%	50.00	**0.061**	**0.071**
5.00%	500.00	**0.39**	**0.783**
10.00%	1000.00	**0.808**	**1.541**
20.00%	2000.00	**1.727**	**4.388**
30.00%	3000.00	**2.633**	**8.354**
50.00%	5000.00	**3.809**	**16.011**
80.00%	8000.00	**4.394**	**32.7**
100.00%	10000.00	**6.061**	**43.649**

Resultados para Stack con Array List

push (Array List)	pop (Array List)	top(Array List)
0.051	**0.054**	**0.003**
0.554	**0.546**	**0.003**
1.34	**1.268**	**0.001**
4.434	**3.64**	**0.002**
7.227	**5.609**	**0.001**
13.954	**9.742**	**0.001**
25.516	**16.302**	**0.001**
32.669	**21.157**	**0.001**

Resultados para Queue con Linked List

Porcentaje de la muestra [pct]	Tamaño de la muestra (LINKED_LIST)	enqueue (Linked List)	dequeue (Linked List)
0.50%	50.00	**0.094**	**0.054**
5.00%	500.00	**0.792**	**0.595**
10.00%	1000.00	**1.451**	**1.228**
20.00%	2000.00	**2.387**	**2.379**
30.00%	3000.00	**3.423**	**3.242**
50.00%	5000.00	**4.605**	**3.715**
80.00%	8000.00	**8.313**	**6.233**
100.00%	10000.00	**10.736**	**7.033**

Resultados para Stack con Linked List

push (Linked List)	pop (Linked List)	top(Linked List)
0.05	**0.041**	**0.007**
0.641	**0.005**	**0.511**
1.292	**1.059**	**0.003**
1.806	**1.502**	**0.002**
3.135	**2.441**	**0.002**
3.565	**2.452**	**0.002**
6.108	**4.217**	**0.002**
6.345	**5.597**	**0.002**

Máquina 2

Resultados para Queue con Array List

Porcentaje de la muestra	enqueue (Array List)	dequeue (Array List)	peek (Array List)
0.50%	**0,044**	**0,027**	**0,006**
5.00%	**0,185**	**0,312**	**0,007**
10.00%	**0,343**	**0,713**	**0,002**
20.00%	**0,361**	**1,186**	**0,002**
30.00%	**1,128**	**3,538**	**0,002**
50.00%	**1,609**	**8,253**	**0,002**

80.00%	1,182	10,753	0,001
100.00%	1,685	15,103	0,001

Resultados para Stack con Array List

Porcentaje de la muestra	push (Array List)	pop (Array List)	top(Array List)
0.50%	0,029	0,022	0,002
5.00%	0,223	0,272	0,002
10.00%	0,537	0,683	0,001
20.00%	0,861	0,975	0,001
30.00%	2,786	3,08	0,001
50.00%	4,045	3,796	0
80.00%	7,669	10,038	0,001
100.00%	11,7	15,853	0,001

Resultados para Queue con Linked List

Porcentaje de la muestra	enqueue (Linked List)	dequeue (Linked List)	peek Linked List)
0.50%	0,064	0,028	0,006
5.00%	0,388	0,22	0,003
10.00%	0,586	0,449	0,002
20.00%	1,156	1,009	0,002
30.00%	1,618	1,404	0,002
50.00%	2,745	2,322	0,002
80.00%	4,588	3,615	0,002
100.00%	6,104	4,079	0,006

Resultados para Stack con Linked List

Porcentaje de la muestra	push (Linked List)	pop (Linked List)	top(Linked List)
0.50%	0,029	0,023	0,002
5.00%	0,259	0,206	0,002
10.00%	0,511	0,497	0,001
20.00%	1,025	0,801	0,001
30.00%	1,556	1,276	0,001
50.00%	2,688	2,239	0,001
80.00%	3,268	2,307	0,001

100.00%	**3,283**	**2,445**	**0,004**

Máquina 3

Resultados para Queue con Array List

Porcentaje de la muestra	enqueue (Array List)	dequeue (Array List)	peek (Array List)
0.50%			
5.00%			
10.00%			
20.00%			
30.00%			
50.00%			
80.00%			
100.00%			

Resultados para Stack con Array List

Porcentaje de la muestra	push (Array List)	pop (Array List)	top(Array List)
0.50%			
5.00%			
10.00%			
20.00%			
30.00%			
50.00%			
80.00%			
100.00%			

Resultados para Queue con Linked List

Porcentaje de la muestra	enqueue (Linked List)	dequeue (Linked List)	peek Linked List)
0.50%			
5.00%			
10.00%			
20.00%			
30.00%			
50.00%			
80.00%			

100.00%			

Resultados para Stack con Linked List

Porcentaje de la muestra	push (Linked List)	pop (Linked List)	top(Linked List)
0.50%			
5.00%			
10.00%			
20.00%			
30.00%			
50.00%			
80.00%			
100.00%			

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?** Las funciones con una diferencia significativa son enqueue, dequeue, push y pop. En enqueue es más eficiente array_list, ya que la complejidad de add_last en array_list es de O(1). En dequeue es más eficiente linked_list, ya que la complejidad de remove_first en linked_list es de O(1). En push es más eficiente linked_list ya que la complejidad de add_first en linked list es de O(1). En pop es más eficiente linked_list ya que la complejidad de remove_first en linked_list_list es de O(1).

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?**
   Únicamente para enqueue el array_list debe ser utilizado (sin tener una gran diferencia en comparación con linked list), mientras tanto el resto de funciones de insertar o eliminar deberían ser usadas con linked list. En cambio, si bien peek y top son más eficientes con array_list, la diferencia es casi mínima.

3. **Durante la ejecución de las pruebas ¿Se presentan anomalías en los tiempos de ejecución que no se explican con la teoría?** Si se presentan anomalías en la ejecución sin razón.
   Si, aunque según la teoría indica que la complejidad O(1) para ciertas operaciones en LinkedList y Arrays, los resultados experimentales muestran ciertos crecimientos en los tiempos debido a factores prácticos (como la memoria por ejemplo y el entorno de ejecución). Estas variaciones

más que contradecir la teoría, reflejan el comportamiento real de las implementaciones. Algunos ejemplos de estos son:

a. Por ejemplo, en Maquina 2 / enqueue (Array) el valor cuando hay 8000 datos disminuye con respecto a la toma anterior de los datos, el tiempo cae casi 0,5ms
b. En peek (ArrayList), los valores a pesar de que deberían ser constantes pues es un O(n), en la toma de datos obtenemos un 0,001 y un 0,007; ´por lo tanto, a pesar de que solo hay una fluctuación de 0,006, si lo ponemos en contexto es una variación notable.

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		La complejidad de add_last en array_list es de O(1), aunque también es O(1) en linked_list.
	Dequeue()		X	La complejidad de remove_first en linked_list_list es de O(1) mientras en array_list es O(N).
	Peek()	X		La diferencia es muy poquita ya que tanto para array_list y linked_list la complejidad es de O(1)
STACK	**Push()**		X	La complejidad de add_first en linked list es de O(1) mientras en array_list es O(N).
	Pop()		X	La complejidad de remove_first en linked_list_list es de O(1) mientras en array_list es O(N).
	Top()	X		La diferencia es muy poquita ya que tanto para array_list y linked_list la complejidad es de O(1)