OBSERVACIONES DE LA PRÁCTICA

Ambientes de pruebas

	Máquina 1
Procesadores	12th Gen Intel® Core™ i5-12450H × 12
Memoria RAM (GB)	16
Sistema Operativo	Ubuntu 25.04

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

Máquina 1

Resultados para Queue con Array List

Porcentaje de la muestra	enqueue (Array List)	dequeue (Array List)	peek (Array List)
0.50%	0,026	0,028	0,003
5.00%	0,149	0,222	0,002
10.00%	0,257	0,480	0,002
20.00%	0,547	1,066	0,002
30.00%	0,883	1,658	0,002
50.00%	1,391	3,000	0,002
80.00%	2,089	6,687	0,002
100.00%	2,639	11,776	0,002

Resultados para Stack con Array List

Porcentaje de la muestra	push (Array List)	pop (Array List)	top(Linked List)
0.50%	0,017	0,030	0,001
5.00%	0,138	0,204	0,001
10.00%	0,296	0,444	0,001
20.00%	0,712	0,907	0,001
30.00%	1,239	1,572	0,001
50.00%	2,849	3,069	0,001
80.00%	10,007	6,801	0,001
100.00%	11,458	14,505	0,002

Resultados para Queue con Linked List

Porcentaje de la muestra	enqueue (Linked List)	dequeue (Linked List)	peek (Linked List)
0.50%	0,032	0,030	0,001
5.00%	0,239	0,294	0,002
10.00%	0,503	0,549	0,001
20.00%	0,957	1,119	0,001
30.00%	1,533	1,808	0,002
50.00%	2,496	2,759	0,001
80.00%	4,531	4,892	0,004
100.00%	4,885	6,782	0,002

Resultados para Stack con Linked List

Porcentaje de la muestra	push (Linked List)	pop (Linked List)	top(Linked List)
0.50%	0,021	0,028	0,001
5.00%	0,193	0,284	0,001
10.00%	0,423	0,552	0,001
20.00%	0,833	1,052	0,001
30.00%	1,347	1,802	0,001
50.00%	2,376	3,377	0,001
80.00%	4,694	5,707	0,004
100.00%	6,104	6,016	0,003

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?

   R// Se observa que tanto el tiempo de ejecución como la tasa de crecimiento son considerablemente mayores, por lo general, en el caso de ArrayList. El orden de crecimiento de las operaciones top (pila), peek (cola) y enqueue (cola) sobre n elementos es el mismo en ambas

implementaciones; sin embargo, las operaciones push (pila), pop (pila) y dequeue (cola) sobre n elementos presentan un orden de crecimiento menor en LinkedList.

La equivalencia en el orden de crecimiento para top (pila) y peek (cola) sobre n elementos se debe a que ambas operaciones acceden al primer elemento de la estructura, lo cual tiene complejidad O(1) en ambas implementaciones y, en consecuencia, orden O(n) al realizar la operación sobre todos los elementos. Para las operaciones restantes, al procesar todos los elementos mediante push (pila), pop (pila) y dequeue (cola), la complejidad resultante es O(n) en LinkedList, dado que la eliminación del primer elemento en una lista enlazada es O(1). En contraste, estas mismas operaciones tienen complejidad $O(n^2)$ sobre todos los elementos en ArrayList, debido a que la eliminación del primer elemento requiere el desplazamiento de todos los elementos subsiguientes, resultando en una operación O(n).

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?

   R// El uso de ArrayList resulta conveniente cuando se requiere acceso frecuente a elementos en posiciones arbitrarias, así como para operaciones de inserción y eliminación al final de la estructura. En contraste, LinkedList es más eficiente para operaciones de acceso, inserción y eliminación en las primeras posiciones de la lista.

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

   R// En ocasiones se observaron desviaciones considerables respecto a las predicciones teóricas; sin embargo, al repetir las pruebas después de una desviación, generalmente se obtenían resultados coherentes con la teoría. Este fenómeno se atribuye a procesos o eventos que ocasionalmente se presentan durante la ejecución de las pruebas y que no están considerados en el modelo teórico.

   Entre los efectos no contemplados se encuentran: (i) los punteros se implementan mediante tablas hash (diccionarios de Python), cuyo acceso tiene complejidad O(1) únicamente en promedio; (ii) las listas dinámicas de Python presentan un crecimiento amortizado, donde la operación de inserción tiene complejidad constante amortizada. A pesar de estas limitaciones, el modelo teórico constituye una aproximación adecuada para el análisis.

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.

R//

		Array List	Linked List	Justificación
QUEUE	**Enqueue()**	x	x	Agregar en última posición es O(1) en LinkedList y ArrayList. Ligeramente más eficiente en ArrayList.
	Dequeue()		x	Eliminar en primera posición es O(1) y en LinkedList y O(n) en ArrayList.
	Peek()	x	x	Acceder a primera posición es O(1) en ambas. Eficiencia similar.
STACK	**Push()**		x	Agregar en primera posición es O(1) en LinkedList y O(n) en ArrayList.
	Pop()		x	Eliminar en primera posición es O(1) en LinkedList y O(n) en ArrayList.
	Top()	x	x	Acceder a primera posición es O(1) en ambas. Eficiencia similar.

Por último, se observa que la teoría concuerda aproximadamente con el comportamiento experimental, aunque se presentan pequeñas desviaciones como consecuencia de que algunas complejidades son O(1) solo en promedio.