OBSERVACIONES DE LA PRÁCTICA

Gabriel Sanchez Villa Cod 202420981

Alejandro Amaya Ardila Cod 202310236

Andrés Felipe Luengas Hernández Cod 201922576

# **Ambientes de pruebas**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Máquina 1 | Máquina 2 | Máquina 3 |
| Procesadores | Ryzen 7 3700U 4 Cores | 2,9GHz Dual-Core Intel Core i5 | 13th Gen Intel® Core™ i5-13500 2.5GHz |
| Memoria RAM (GB) | 6.9GB | 8GB | 15.7GB |
| Sistema Operativo | Windows 10 | macOS Monterey 12.7.6 | Windows 11 Enterprise |

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% | **0.161** | **0.109** | **0.008** |
| 5.00% | **0.642** | **0.651** | **0.005** |
| 10.00% | **1.072** | **0.94** | **0.004** |
| 20.00% | **3.33** | **4.481** | **0.004** |
| 30.00% | **2.713** | **3.498** | **0.004** |
| 50.00% | **4.53** | **7.542** | **0.003** |
| 80.00% | **26.711** | **42.892** | **0.007** |
| 100.00% | **25.402** | **31.55** | **0.007** |

## **Resultados para Stack con Array List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | push (Array List) | pop (Array List) | top(Linked List) |
| 0.50% | **0.05** | **0.048** | **0.003** |
| 5.00% | **0.336** | **0.328** | **0.003** |
| 10.00% | **0.68** | **0.669** | **0.002** |
| 20.00% | **1.352** | **1.304** | **0.003** |
| 30.00% | **3.183** | **4.05** | **0.006** |
| 50.00% | **2.732** | **3.276** | **0.011** |
| 80.00% | **8.171** | **5.916** | **0.004** |
| 100.00% | **12.374** | **12.146** | **0.011** |

## **Resultados para Queue con Linked List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | enqueue (Linked List) | dequeue (Linked List) | peek Linked List) |
| 0.50% | **0.581** | **0.069** | **0.006** |
| 5.00% | **1.865** | **0.855** | **0.008** |
| 10.00% | **5.863** | **2.049** | **0.007** |
| 20.00% | **7.029** | **3.822** | **0.006** |
| 30.00% | **6.496** | **6.854** | **0.008** |
| 50.00% | **26.297** | **11.32** | **0.007** |
| 80.00% | **18.029** | **12.721** | **0.008** |
| 100.00% | **38.558** | **14.442** | **0.007** |

## **Resultados para Stack con Linked List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | push (Linked List) | pop (Linked List) | top(Linked List) |
| 0.50% | **0.062** | **0.099** | **0.004** |
| 5.00% | **0.475** | **4.823** | **0.003** |
| 10.00% | **1.336** | **0.003** | **15.028** |
| 20.00% | **2.452** | **66.624** | **0.002** |
| 30.00% | **2.565** | **157.453** | **0.002** |
| 50.00% | **6.374** | **593.284** | **0.008** |
| 80.00% | **8** | **1302.546** | **0.002** |
| 100.00% | **60.96** | **1723.101** | **0.003** |

# **Máquina 2**

## **Resultados para Queue con Array List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | enqueue (Array List) | dequeue (Array List) | peek (Array List) |
| 0.50% | **0.138** | **0.144** | **0.005** |
| 5.00% | **0.918** | **1.012** | **0.018** |
| 10.00% | **1.587** | **1.422** | **0.003** |
| 20.00% | **2.855** | **3.061** | **0.003** |
| 30.00% | **4.035** | **4.804** | **0.004** |
| 50.00% | **5.879** | **7.751** | **0.003** |
| 80.00% | **15.815** | **22.255** | **0.003** |
| 100.00% | **15.596** | **35.435** | **0.013** |

## **Resultados para Stack con Array List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | push (Array List) | pop (Array List) | top(Array List) |
| 0.50% | **0.081** | **0.083** | **0.004** |
| 5.00% | **0.73** | **0.831** | **0.003** |
| 10.00% | **1.949** | **2.5** | **0.007** |
| 20.00% | **3.105** | **5.217** | **0.057** |
| 30.00% | **3.183** | **4.05** | **0.006** |
| 50.00% | **4.446** | **5.774** | **0.009** |
| 80.00% | **10.725** | **9.79** | **0.004** |
| 100.00% | **17.135** | **11.524** | **0.003** |

## **Resultados para Queue con Linked List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | enqueue (Linked List) | dequeue (Linked List) | peek Linked List) |
| 0.50% | **0.139** | **0.088** | **0.004** |
| 5.00% | **1.652** | **1.5** | **0.005** |
| 10.00% | **5.863** | **2.049** | **0.007** |
| 20.00% | **5.521** | **3.181** | **0.025** |
| 30.00% | **5.491** | **3.789** | **0.003** |
| 50.00% | **26.297** | **11.32** | **0.007** |
| 80.00% | **303.447** | **12.729** | **0.004** |
| 100.00% | **30.378** | **17.221** | **0.003** |

## **Resultados para Stack con Linked List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | push (Linked List) | pop (Linked List) | top(Linked List) |
| 0.50% | **0.193** | **0.277** | **0.006** |
| 5.00% | **1.038** | **7.978** | **0.038** |
| 10.00% | **2.184** | **46.207** | **0.003** |
| 20.00% | **3.654** | **111.711** | **0.003** |
| 30.00% | **9.454** | **233.434** | **0.025** |
| 50.00% | **11.329** | **690.111** | **0.003** |
| 80.00% | **21** | **1578.991** | **0.004** |
| 100.00% | **29.453** | **2838.418** | **0.003** |

# **Máquina 3**

## **Resultados para Queue con Array List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | enqueue (Array List) | dequeue (Array List) | peek (Array List) |
| 0.50% | **0.12** | **0.15** | **0.003** |
| 5.00% | **0.887** | **1.034** | **0.015** |
| 10.00% | **1.391** | **1.385** | **0.009** |
| 20.00% | **2.409** | **2.797** | **0.004** |
| 30.00% | **4.384** | **4.43** | **0.003** |
| 50.00% | **5.705** | **7.073** | **0.002** |
| 80.00% | **13.878** | **18.854** | **0.004** |
| 100.00% | **15.2** | **32.487** | **0.011** |

## **Resultados para Stack con Array List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | push (Array List) | pop (Array List) | top(Array List) |
| 0.50% | **0.063** | **0.049** | **0.005** |
| 5.00% | **0.73** | **0.809** | **0.004** |
| 10.00% | **2.118** | **1.862** | **0.006** |
| 20.00% | **3.3** | **4.132** | **0.015** |
| 30.00% | **3.542** | **4.654** | **0.004** |
| 50.00% | **4.203** | **5.214** | **0.005** |
| 80.00% | **9.864** | **8.87** | **0.02** |
| 100.00% | **15.178** | **12.324** | **0.004** |

## **Resultados para Queue con Linked List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | enqueue (Linked List) | dequeue (Linked List) | peek Linked List) |
| 0.50% | **0.131** | **0.049** | **0.003** |
| 5.00% | **1.479** | **1.299** | **0.01** |
| 10.00% | **4.368** | **1.928** | **0.005** |
| 20.00% | **5.743** | **3.274** | **0.007** |
| 30.00% | **5.864** | **4.142** | **0.003** |
| 50.00% | **23.341** | **10.811** | **0.013** |
| 80.00% | **36.825** | **12.933** | **0.008** |
| 100.00% | **42.567** | **16.558** | **0.005** |

## **Resultados para Stack con Linked List**

|  |  |  |  |
| --- | --- | --- | --- |
| Porcentaje de la muestra | push (Linked List) | pop (Linked List) | top(Linked List) |
| 0.50% | **0.634** | **0.298** | **0.008** |
| 5.00% | **1.342** | **5.821** | **0.004** |
| 10.00% | **2.873** | **33.234** | **0.007** |
| 20.00% | **3.899** | **72.249** | **0.004** |
| 30.00% | **8.675** | **159.974** | **0.009** |
| 50.00% | **10.009** | **570.007** | **0.011** |
| 80.00% | **16.234** | **1320.761** | **0.002** |
| 100.00% | **28.987** | **2400.699** | **0.006** |

# **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?

Caso Queue:

1. En el caso del queue, ambas implementaciones tienen ventaja y desventajas según los resultados de las regresiones lineares. Por ejemplo, enqueue aparenta ser mejor si se usa un array\_list, esta inserción al final es O(1) amortizada y si fuese implementada en linked\_list sería O(1). Sin embargo, los tiempos son relativamente similares en ambos casos.
2. En la siguiente función, dequeue es mejor con linked\_list. Esto se debe al hecho de que al implementar dequeue con array\_list, la eliminación al inicio es O(n) mientras que con linked\_list es O(1) ya que el “first” se desplaza 1 unidad a la derecha.
3. Peek no tiene diferencias muy significativas debido a que ambas implementaciones serían O(1).

Caso Stack:

1. En el stack, la función push tiene mejor rendimiento cuando se implementa con un array\_list. Esto es debido al hecho de que push añade al final y en array\_list es O(1) amortizada y en linked\_list también es O(1). Sin embargo, array\_list tiene un desempeño ligeramente más rápido.
2. La función pop tiene un rendimiento significativamente más rápido con una implementación array\_list. Pop se encarga de eliminar al final. La eliminación al final de un array\_list es O(1) mientras que en linked\_list es O(n) ya que se debe acceder al penúltimo nodo desde el inicio y asignar este como last.
3. Top no tiene diferencias muy significativas debido a que ambas implementaciones serían O(1).
4. ¿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 de las colas, es mejor implementarlas con linked\_list, ya que la eliminación al inicio y la inserción al final son frecuentes, por lo tanto todos los casos serían O(1) con linked\_list, a excepción del acceso aleatorio que en este caso si sería mejor con array\_list.

En el caso de las pilas, es mejor usar un array\_list, ya que la inserción y eliminación al final son recurrentes, usando un array\_list los casos serían O(1) amortizados a excepción de top que si es O(1). Además, acceder de manera aleatoria también sería O(1) en este caso.

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

Si se presentan algunas anomalías como el hecho de que O(1) amortizado tenga un mejor rendimiento que O(1), sin embargo, quizás se pueda deber al hecho de que cambiaba entre aplicaciones para poder registrar los tiempos y al volver a correr los tiempos estos se reducían significativamente. Un poco curioso pero podría ser una explicación.

1. 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 |  | No concuerda, ya que O(1) amortizado debería de mostrar de vez en cuando algún redimensionamiento, sin embargo es ligeramente mejor que linked\_list en los tiempos. |
| **Dequeue()** |  | x | Si concuerda, ya que la eliminación al inicio es más efectiva con linked\_list. |
| **Peek()** | x |  | Si concuerda, debido a que acceder a este elemento es O(1) tanto en array\_list como en linked\_list |
| STACK | **Push()** | X |  | Si concuerda porque en ambos casos es O(1) |
| **Pop()** | X |  | Si concuerda porque la eliminación del final es más rápida con array\_list |
| **Top()** |  | x | Si concuerda, debido a que acceder a este elemento es O(1) tanto en array\_list como en linked\_list |