1.Configurações do computador:

Processador: AMD Ryzen 5 5500 3.80 GHz 6 núcleos e 12 processadores lógicos
Memória: XPG 16 RAM 3200 MHz dual channel
Sistema Operacional: 64 bits, Windows 10 pro

OBS: Meu sistema é modificado para não ficar rodando processos em background e economizar memória/processador

2.Casos de teste:

ArrayList IndexOf

	Marcações P e N	Tempo em ms	Tempo em s
1	P = 2.500.000, N = 20.000	24757,82 ms	24,7578 s
2	P = 5.000.000, N = 20.000	51336,12 ms	51,3361 s
3	P = 10.000.000, N = 20.000	101121,14 ms	101,1211 s
4	P = 2.500.000 e N = 40.000	47864,34 ms	47,8643 s

ArrayList Loop

	Marcações P e N	Tempo em ms	Tempo em s
1	P = 2.500.000, N = 20.000	32698,00 ms	32,6980 s
2	P = 5.000.000, N = 20.000	64807,71 ms	64,8077 s
3	P = 10.000.000, N = 20.000	128284,99 ms	128,2850 s
4	P = 2.500.000 e N = 40.000	64556,74 ms	64,5567 s

HashMap Loop

	Marcações P e N	Tempo em ms	Tempo em s
1	P = 2.500.000, N = 20.000	139,74 ms	0,1397 s
2	P = 5.000.000, N = 20.000	225,52 ms	0,2255 s
3	P = 10.000.000, N = 20.000	456,86 ms	0,4569 s
4	P = 2.500.000 e N = 40.000	0,1446 s	0,1446 s

3.Considerações casos de teste:

ArrayList IndexOf e Loop

Acesso: O(1);
Busca: O(N);
Inserção/Delegação no final: O(1) amortizado; O(N) no pior caso(redimensionar array);

Análise dos Resultados:

Para as operações com ArrayList, tanto o método de busca por índice (indexOf) quanto o loop manual mostram um aumento significativo no tempo de execução à medida que o tamanho da lista (P) aumenta. Isso ocorre porque a busca em um ArrayList tem complexidade de tempo O(n), o que significa que o tempo necessário para encontrar um elemento cresce linearmente com o número de elementos na lista. Esse comportamento é evidente nos resultados, onde dobrar o tamanho de P mais que dobra o tempo de execução, indicando um crescimento linear.

Sobre os diferentes resultados de indexOf x Loop:
O método indexOf utiliza o método equals para verificar a igualdade dos objetos.A Implementação de equals pode ser otimizada para casos específicos do objeto Pessoa, enquanto o loop manual pode não utilizar dessa otimização, tendo em vista que ao escrever um loop manual para percorrer o ArrayList, o código pode incluir verificações ou operações adicionar que não estão relacionados a busca em si, que podem adicionar sobrecarga extra ao tempo, mas no fim, ambos percorrem linearmente. Métodos como indexOf podem tirar vantagem de instruções de máquina específicas ou de execução de código nativo que é mais rápido do que o código interpretado ou compilado Just-In-Time (JIT) de um loop escrito pelo usuário em Java. Isso significa que, embora a operação seja linear, a velocidade de execução por elemento pode ser maior.

HashMap

Acesso/Busca/Inserção/Delegação: O(1) no médio; O(N) no pior caso;

Análise dos Resultados:

O HashMap, por outro lado, apresenta um comportamento bastante diferente. A busca por chave (containsKey) em um HashMap tem complexidade de tempo O(1) na média, o que significa que o tempo necessário para encontrar um elemento é constante, independentemente do tamanho do mapa. Isso é evidenciado pelos resultados muito mais rápidos para as operações de busca, mesmo quando P é aumentado. A leve variação no tempo conforme P e N aumentam pode ser atribuída ao gerenciamento interno do HashMap, como o redimensionamento e a colisão de hash, mas mesmo assim, esses tempos são ordens de magnitude menores do que aqueles observados com ArrayList..

Fontes:

- https://pt.stackoverflow.com/questions/56836/defini%c3%a7%c3%a3o-da-nota%c3%a7%c3%a3o-big-o/56868#56868
- https://stackoverflow.com/questions/21388466/is-arraylist-indexof-complexity-n
- https://cursos.alura.com.br/forum/topico-velocidade-da-busca-104920#:~:text=HashSet%20possui%20uma%20complexidade%20de,depende%20do%20n%C3%BAmero%20de%20elementos.
- https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
- https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html