Tabela Hash(Hash table):
Complexidade de tempo: O(1) em média, pode ser O(n) em casos de colisão.
Eficiência em listas ordenadas: Não se aplica, pois não é uma estrutura ordenada.
Método de busca: Usa uma função de hash para calcular a posição do elemento.
Espaço de armazenamento: Requer espaço adicional para a estrutura de hash.

Melhor caso: No melhor caso, a busca em uma tabela de hash encontra o elemento desejado em O(1) tempo. Isso acontece quando não há colisões, e a função de hash mapeia diretamente a chave para uma posição na tabela.

Pior caso: No pior caso, a busca em uma tabela de hash encontra o elemento em O(n) tempo, que ocorre quando há muitas colisões e a lista ligada precisa ser percorrida para encontrar o elemento desejado. Isso pode acontecer quando muitas chaves produzem o mesmo valor de função de hash.

Busca Binária(Sorted list):
Complexidade de tempo: O(logn)
Eficiência em listas ordenadas: Altamente eficiente, requer lista ordenada.
Método de busca: Divide repetidamente o espaço de busca pela metade.
Espaço de armazenamento: Requer apenas a lista ordenada.

Melhor caso: No melhor caso, a busca binária encontra o elemento desejado em O(1) tempo. Isso acontece quando o elemento procurado está exatamente no meio da lista, e portanto é encontrado na primeira iteração.

Pior caso: No pior caso, a busca binária encontra o elemento em O(logn) tempo. Isso ocorre quando o elemento não está na lista, ou está no extremo oposto da lista em relação ao ponto de início da busca. Neste caso, a busca binária divide repetidamente o espaço de busca pela metade até que o intervalo de busca seja reduzido a zero.

Busca Sequencial(Unsorted list):
Complexidade de tempo: O(n)
Eficiência em listas ordenadas: Não requer lista ordenada.
Método de busca: Examina cada elemento sequencialmente.
Espaço de armazenamento: Requer apenas a lista.

Melhor caso: No melhor caso, a busca sequencial encontra o elemento desejado em O(1) tempo. Isso ocorre quando o elemento está na primeira posição da lista, e portanto é encontrado na primeira comparação.

Pior caso: No pior caso, a busca sequencial encontra o elemento em O(n) tempo. Isso ocorre quando o elemento não está presente na lista ou está no final da lista. Neste caso, a busca sequencial precisa examinar cada elemento da lista até encontrar o elemento desejado ou determinar que ele não está presente.

Gráfico comparativo do tempo de inserção em 4 gerações de dados em escala reduzida:

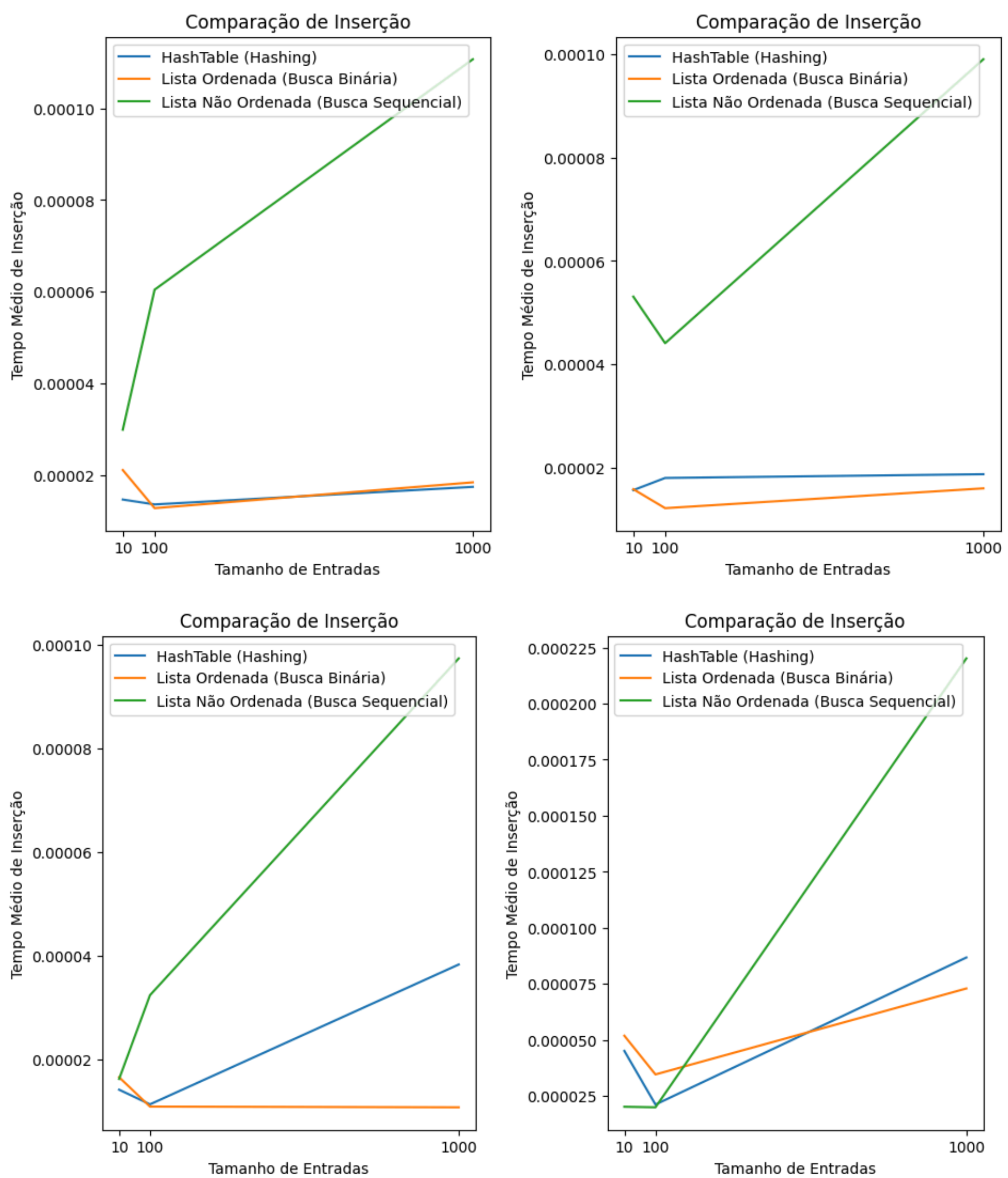
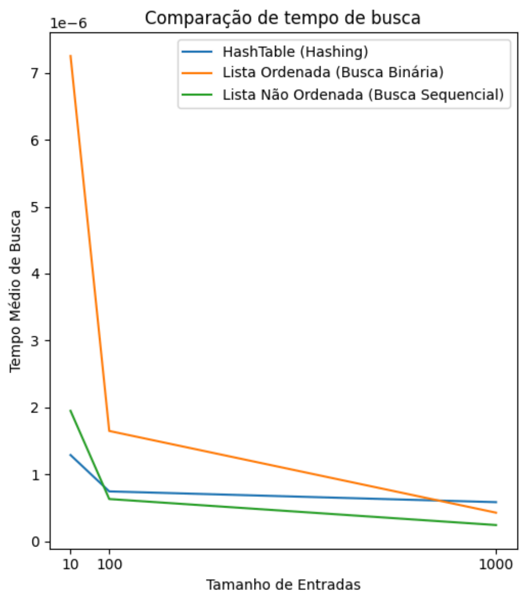
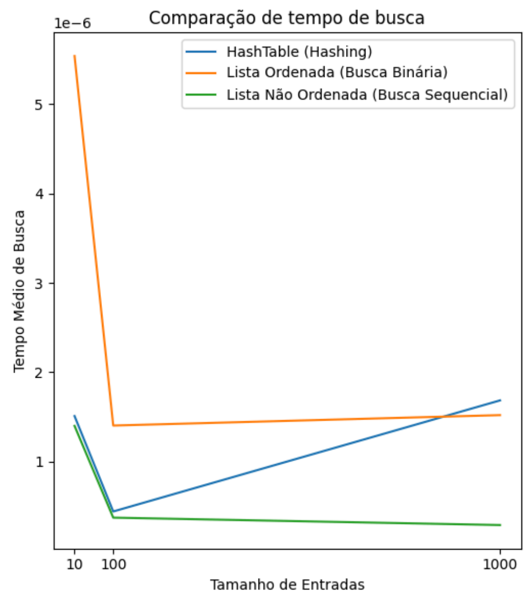
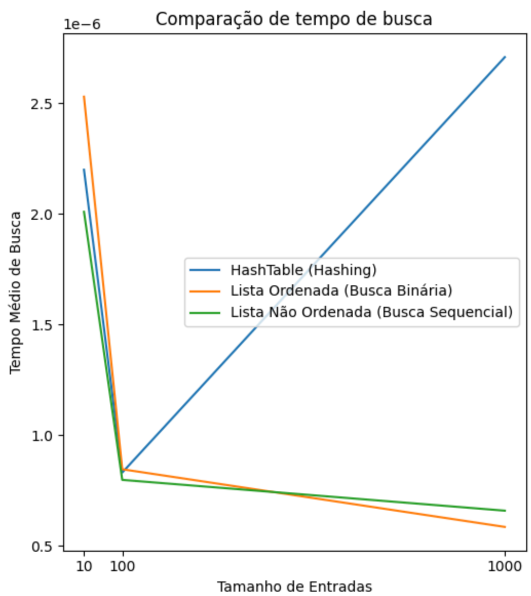
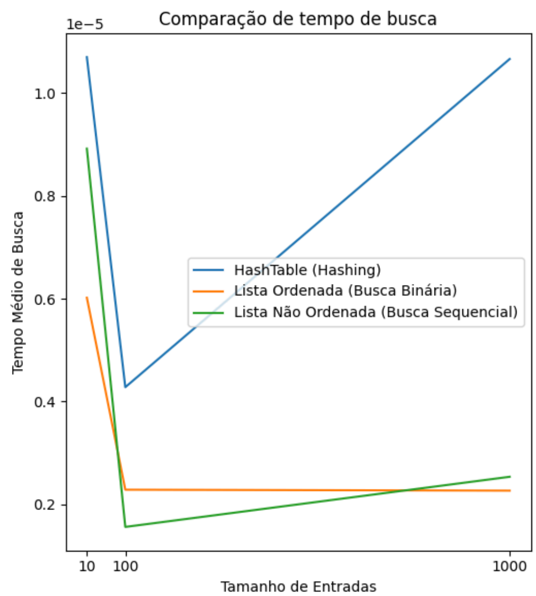


Gráfico comparativo do tempo de busca em 4 gerações de dados em escala reduzida:









Visão geral:

Ao se analisar qual dos 3 métodos é o mais adequado para a implementação desse projeto, devemos levar em conta o volume de dados que será registrado e a carga que será atribuída ao mesmo, para assim avaliarmos o quão difícil será lidar com as colisões do conjunto.

ao fazermos testes com a quantidade proposta de de registros obtivemos os seguintes resultados:

Para 10.000 registros: Todos os métodos são adequados, incluindo busca sequencial, busca binária e tabela de hash.
Para 100.000 registros: A busca sequencial pode começar a mostrar lentidão. A busca binária e a tabela de hash são preferíveis.
Para 1.000.000 de registros: A busca sequencial se torna impraticável. A busca binária é viável, mas a tabela de hash se destaca como a opção mais robusta e eficiente.

Tendo isso em vista as melhores soluções são a busca binária e a tabela hash, dando preferência pela tabela hash por sua versatilidade e robustez de acordo com sua qualidade de implementação, contudo vale dar crédito a busca binária que entre os volumes de 10.000 e 100.000 se mostrou muito eficaz e com o uso mínimo de memória exigindo somente a ordenação da lista.