

I chose to use a hash table for storing contacts because it makes looking up names really fast. In a list, I would have to go through each contact one by one, which takes a lot longer as the list gets bigger. With a hash table, I can convert the contact's name into an index using a hash function, and then I can access that index almost instantly. This is really helpful if I want to search, add, or update contacts quickly.

Collisions were something I had to think about because two names might end up at the same index. I used separate chaining to solve this problem, which means each index can hold a linked list of contacts. If the same name is added twice, the hash table updates the number instead of creating a duplicate. This approach keeps the data organized and avoids overwriting contacts.

I think hash tables are better than lists or even trees for situations where you need fast access by a key, like a name or ID. Trees are useful if you want to keep things sorted, but I didn't need that here. Lists are simpler, but they become slow when the dataset grows. Using a hash table is a good balance because it keeps lookups fast and handles collisions efficiently.

Overall, I learned that hash tables are really practical for managing large sets of data where speed matters more than order. Writing the insert and search functions helped me understand how collisions work and why it's important to check for duplicates. I also saw how a simple hash function can impact the placement of data in the table.