Q1)

What would I prefer?

=> I would prefer option 1 because it is a straight-forward implementation with same time/space complexity.
=>The big O time/space complexity is same (O(n^2) and O(1)).
=> However, if we count individual operations Option1 is still better.
=> Option 2 could potentially use parallelism but it doesn't in the current implementation so it is not faster.
=> The array can easily fit in memory so a parallel implementation might not be necessary.

Option 1
Pros:

1. Easy to understand and write code
2. O(1) space complexity. 4 bytes for sum.

Cons

1. Sequential implementation. Will not be able to use multi-threading

Option 2:

Pros:

1. Could potentially use parallelism to compute sum faster by computing each sum[i] in a separate thread.

Cons:

1. Requires slightly more memory 16 bytes . But still O(1)

Q2)

=> If there was a username exists error, 200 should not be returned in the POST request. It should be 409 (Conflict).

=> PUT/PATCH is used to update an item in the collection. It seems here POST is used to update/rename. The format should be => `PUT/PATCH /users/1`

=> DELETE => The uri for delete should point to a user resource e.g DELETE /users/:id

Q3)

=> The code uses the same salt for each password.
=>The purpose for the salt is to add a random value to each password so that identical passwords do not hash to the same value.
=> Use different salt for a password and store the salt value in DB.

Q4)

=> There is just a tiny check I believe is missing.
=> I think there is a need to check for null each of the properties in user.transactions.history.deposits before running for loop to sum all values.
=> Any one of user, transactions, history, deposits could be NULL. It would be safer to check for these before going ahead with the sum operation.

Q5)

1. *db.game_accounts.findOne({"name": from},{"credits":1});*
   Bad primary key => Here the name of the user is being used to search for the record. FindOne only returns the first match. For all transactions involving John, it will only be the first user named John in the database. Using an account number would be much better.

2. *db.game_accounts.update({name: from}, {$set: {credits:fromAccount.credits - amt}});*
   db.game_accounts.update({name: to}, {$set: {credits: toAccount.credits + amt}});

   Handling concurrency => The above two transactions should be treated as a single atomic transaction to prevent race conditions.

Q6)

=> The only problem I see with this function is the lack of error checking and seeing if isCredit is a boolean and amount is an float/int.

=> Otherwise, the function does increase the credit/debit balance of the Account if the respective Account was initialized with the “new” keyword.
=> The new keyword makes sure that the prototype function applies to all Accounts.

Q7)

1. Using an index for faster lookups and efficient access of ordered records
2. Use partitioning to help accomplish multiple tasks in parallel. A single table could be mapped to multiple files.
3. Use a cache to reuse results from earlier queries
4. Use denormalization to achieve better query performance.