Day 17:

Assignment4: Rabin-Karp Substring Search

Implement the Rabin-Karp algorithm for substring search using a rolling hash. Discuss the impact of hash collisions on the algorithm's performance and how to handle them.

A)

Java code:

```
package Day17;
import java.util.ArrayList;
import java.util.List;
public class RabinKarpAlgorithm {
        public static void main(String[] args) {
    String text = "ABABDABACDABABCABAB";
    String pattern = "ABAB";
    searchPattern(text, pattern);
  }

  public static void searchPattern(String text, String pattern) {
    int textLength = text.length();
    int patternLength = pattern.length();
    int patternHash = hash(pattern, patternLength);
    int textHash = hash(text.substring(0, patternLength), patternLength);

    for (int i = 0; i <= textLength - patternLength; i++) {
      if (textHash == patternHash && checkEquals(text, i, pattern)) {
        System.out.println("Pattern found at index " + i);
      }
      if (i < textLength - patternLength) {
        textHash = recalculateHash(text, i, patternLength, textHash);
      }
    }
  }


  public static int hash(String str, int length) {
    int hash = 0;
    for (int i = 0; i < length; i++) {
      hash += str.charAt(i) * Math.pow(10, length - i - 1);
    }
    return hash;
  }

  public static int recalculateHash(String str, int oldIndex, int length, int oldHash) {
```

```
    int newHash = oldHash - str.charAt(oldIndex) * (int)Math.pow(10, length - 1);
    newHash = newHash * 10 + str.charAt(oldIndex + length);
    return newHash;
  }

  public static boolean checkEquals(String text, int startIndex, String pattern) {
    for (int i = 0; i < pattern.length(); i++) {
      if (text.charAt(startIndex + i) != pattern.charAt(i)) {
        return false;
      }
    }
    return true;
  }
}
```

Explanation of the algorithm:

Hashing: *The algorithm uses hashing to quickly compare the pattern with substrings of the text. It computes the hash value for both the pattern and the first substring of the text with the same length as the pattern.*

Rolling Hash: *As it iterates through the text, it calculates the hash value of the next substring using a rolling hash function. This avoids recalculating the hash value from scratch for each substring.*

Comparison: *It compares the hash values of the pattern and the current substring. If they match, it verifies the equality of the substrings character by character to handle potential hash collisions.*

Impact of Hash Collisions:

Performance: *Hash collisions can impact the performance of the Rabin-Karp algorithm, especially if they occur frequently. Collisions may result in false positives, leading to additional character comparisons to confirm the match, which can increase the algorithm's runtime.*

Handling Collisions: *To handle hash collisions, the algorithm must perform additional character comparisons whenever hash values match. These comparisons ensure that the pattern matches the substring correctly, mitigating the impact of collisions on the algorithm's accuracy.*

Hash Function Selection: *Choosing a good hash function can minimize the likelihood of collisions. A well-designed hash function distributes hash values evenly across different substrings, reducing the chances of collisions and improving the algorithm's performance.*

Overall, while hash collisions can affect the Rabin-Karp algorithm's performance, proper handling strategies and careful selection of hash functions can mitigate their impact and ensure efficient substring search.