# Payal Chavan

# Summer 2024

# CS 5800 Algorithms (Seattle)

# Assignment 9

# Date: 08/16/2024

# Time Travel Day: Used 3rd and 4th day

# Question 1: RANSAC Implementation

RANSAC, short for Random Sample Consensus, is an iterative technique used to estimate model parameters from noisy data that may contain outliers. In computer vision, it’s commonly employed for tasks such as fitting a line or a plane to a set of points.

The objective of this experiment is to find the best-fitting line for a dataset that contains noisy points, even when some of those points are outliers.

Below are the key concepts we use in RANSAC:

- **Inliers and Outliers**: Inliers refer to data points that align well with the model, while outliers are points that deviate significantly due to noise or errors.
- **Threshold**: This is a predefined distance used to distinguish between inliers and outliers.
- **Iterations**: The algorithm performs random sampling and model fitting a certain number of times, which we call iterations.

The parameters used in plotting are described as:

- **frac_noises:** This parameter controls the fraction of Gaussian noise added to the data points. Smaller values correspond to less noise.
- **slopes:** These values determine the slope of the true line within the data.
- **intersects:** These values correspond to variations in the y-intercept of the true line.

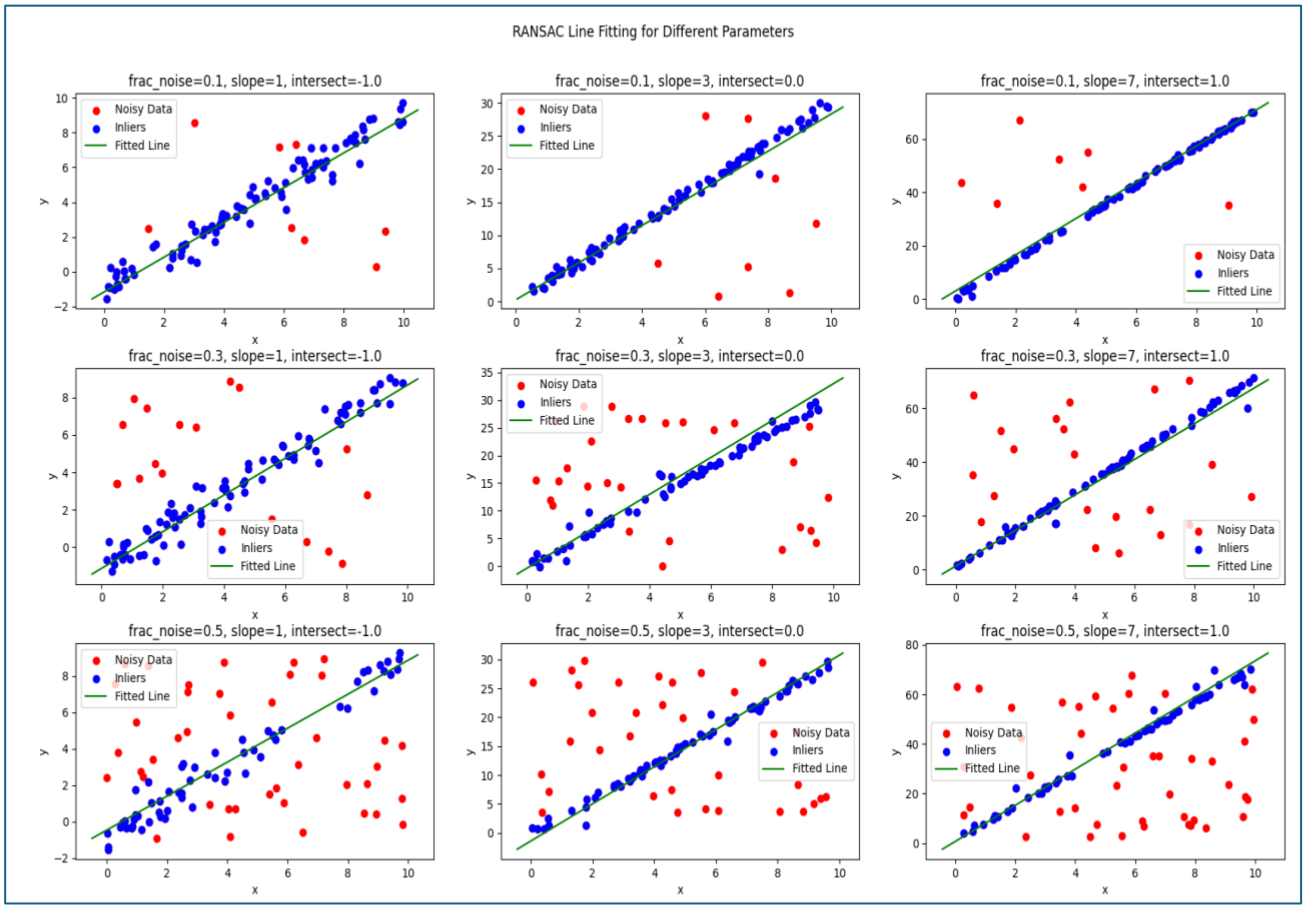


Fig 1: RANSAC Implementation for combination of parameters (frac_noise, slopes, and intercepts).

- The scatter plot shows the noisy data points (in red) and the inliers (in blue).
- The green line represents the fitted line obtained by RANSAC.
- Each subplot corresponds to a specific combination of parameters.

These plots illustrate how RANSAC can be a powerful tool for fitting models in the presence of noisy data and outliers. Let’s explore how RANSAC behaves under varying noise levels and different true line characteristics. Despite noisy data and outliers, RANSAC aims to robustly estimate the true line.

1. **Noise Robustness:** RANSAC is specifically designed to handle outliers effectively. In the provided plots, even as the noise fraction increases, the green fitted line (RANSAC's result) manages to approximate the true relationship (depicted by the blue line) better than a standard linear regression would. This demonstrates RANSAC's proficiency in dealing with noisy data. As the noise fraction rises from 0.1 to 0.5, the scatter of the red points (noisy data) becomes

more pronounced, making it increasingly challenging for the green fitted line to accurately represent the true relationship.

2. **Parameter Sensitivity:** The plots exhibit various slopes and noise levels. Despite these variations, RANSAC's performance remains relatively consistent, indicating its robustness to different underlying data distributions and noise levels.
3. **Outlier Detection:** The red points signify noisy data, including outliers. RANSAC works by iteratively selecting random subsets of the data to fit the model and identifying inliers that conform to the model. The green line represents the best fit after considering these inliers, effectively ignoring the outliers.
4. **Model Accuracy:** At lower noise levels (e.g., 0.1), the green fitted line closely follows the blue true line, signifying high accuracy. As the noise level increases, the green line still approximates the true line reasonably well, showcasing RANSAC's ability to maintain accuracy despite increasing noise.
5. **Slope Variations:** The varying slopes across the plots demonstrate that RANSAC can adapt to different underlying relationships in the data. Even with different slopes, the algorithm consistently finds a good fit, highlighting its flexibility. The slope of the true relationship varies from 1 to 7 across the plots. Despite the changes in slope, the effect of noise remains consistent: higher noise levels lead to greater deviations of the fitted line from the true line.

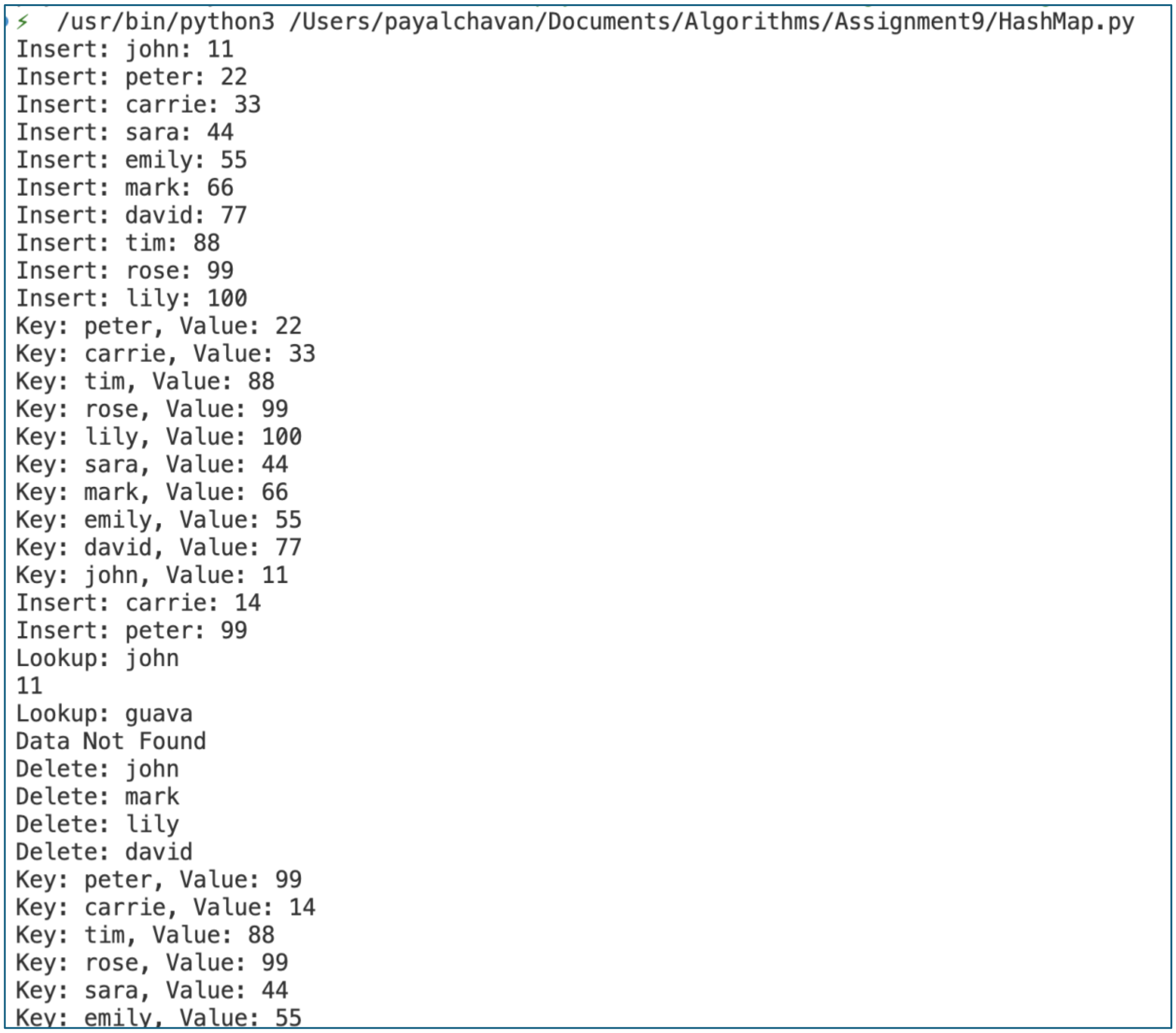
# Question 2: HashMap Implementation.

A HashMap stores data as key-value pairs. Each key maps to a specific value. The essential operations performed by HashMap are:

1. **Insertion:** When you insert a key-value pair into a HashMap, it computes the hash code for the key and assigns it to an index within an internal array. The associated value is then stored at that index. In case of a collision (when two keys hash to the same index), the HashMap employs methods like chaining or open addressing to manage it. Some keys are updated with new values using insert().

2. **Deletion:** When removing a key-value pair from a HashMap, the process includes identifying the key, computing its hash code, and determining the corresponding index. Subsequently, the value associated with that key is removed from the specified index. The delete() method deletes the key-value pair based on its given associated key. For ex; the key-value entry is deleted by its key “mark”.

3. **Lookup (Search):** When you provide a key, the HashMap efficiently retrieves the corresponding value by calculating the key's hash code, mapping it to an index, and accessing the value stored at that index. For ex; the lookup() method is used to find the value associated with the key "john". The lookup() method can also be called with a non-existent key "guava".
4. **Traversal:** The traverse() method prints all key-value pairs.



```
⚡  /usr/bin/python3 /Users/payalchavan/Documents/Algorithms/Assignment9/HashMap.py
Insert: john: 11
Insert: peter: 22
Insert: carrie: 33
Insert: sara: 44
Insert: emily: 55
Insert: mark: 66
Insert: david: 77
Insert: tim: 88
Insert: rose: 99
Insert: lily: 100
Key: peter, Value: 22
Key: carrie, Value: 33
Key: tim, Value: 88
Key: rose, Value: 99
Key: lily, Value: 100
Key: sara, Value: 44
Key: mark, Value: 66
Key: emily, Value: 55
Key: david, Value: 77
Key: john, Value: 11
Insert: carrie: 14
Insert: peter: 99
Lookup: john
11
Lookup: guava
Data Not Found
Delete: john
Delete: mark
Delete: lily
Delete: david
Key: peter, Value: 99
Key: carrie, Value: 14
Key: tim, Value: 88
Key: rose, Value: 99
Key: sara, Value: 44
Key: emily, Value: 55
```

Fig 2: Output of testing Insertions, Deletions, Lookup, and Traversal Operations on HashMap.

Now, let's see analyze the Performance for all the operations- Insertions, Lookup, and Delete.

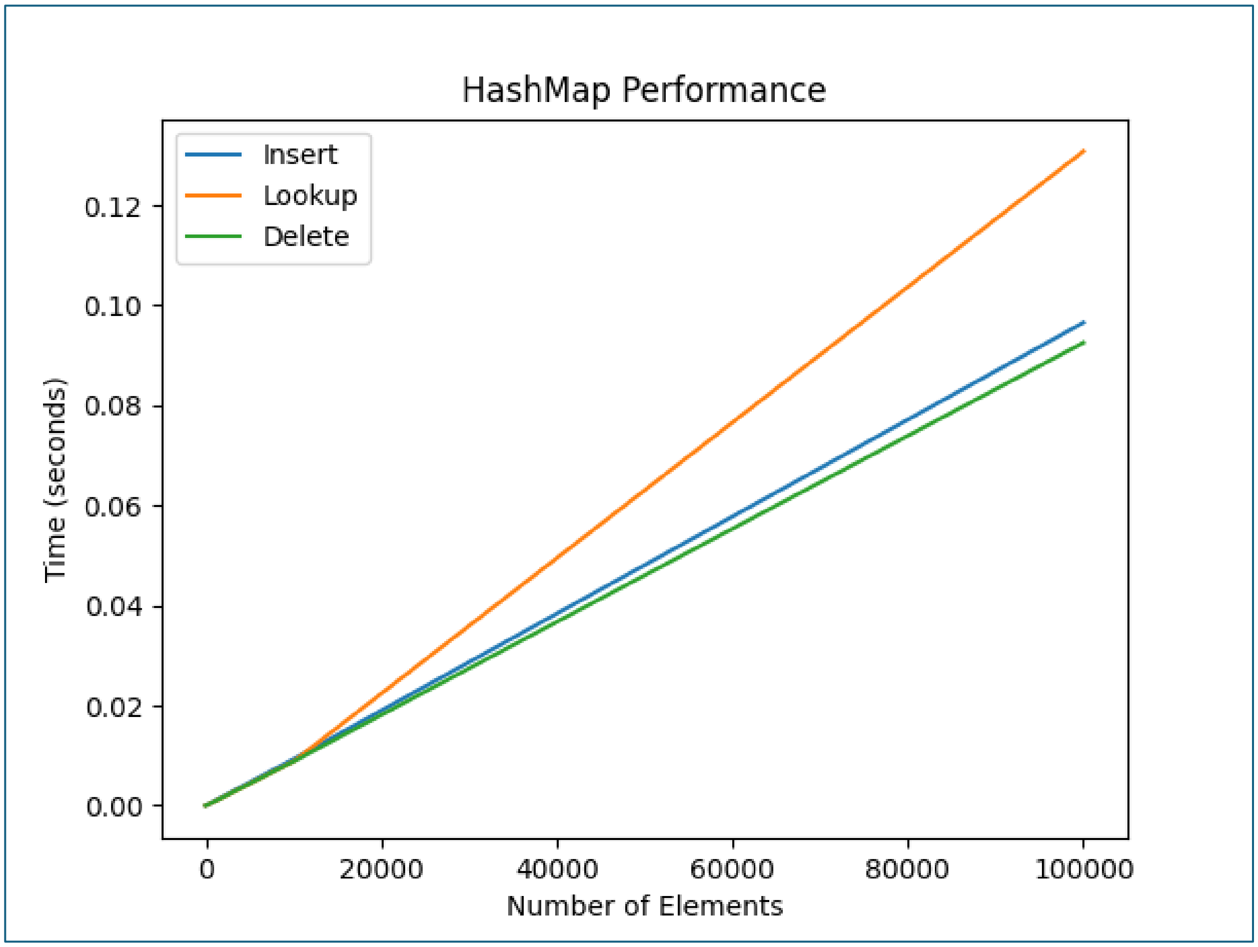


Fig 3: HashMap Performance Analysis

Let’s see the insights of HashMap performance on various operations.

1. **Insert Operation:**

- The blue line shows the time required to insert elements into the HashMap.
- As the number of elements increases (from 0 to 100,000), the insertion time also rises.
- The relationship appears linear, indicating that insertion time grows proportionally with the number of elements.

2. **Lookup Operation:**

- The orange line represents the time taken to look up elements in the HashMap.
- Similar to insertion, the lookup time increases linearly as the number of elements grows.

- This behavior is consistent with the expected performance of a well-implemented HashMap.

3. **Delete Operation:**

- The green line corresponds to the time taken to delete elements from the HashMap.
- Again, a linear relationship is observed with the number of elements.
- Deleting elements becomes slower as the HashMap size increases.

4. **Overall Insights:**

- The graph indicates that the HashMap's performance is consistent across insertion, lookup, and deletion operations.
- The constant rate of increase suggests efficient scaling, highlighting the HashMap's ability to handle growing data sizes effectively.

Reflection:

In this assignment, I gained valuable experience implementing the RANSAC algorithm across various parameters. By visualizing the results, I deepened my understanding of how RANSAC operates in fitting lines, especially under different noise levels, slopes, and intercepts. Additionally, I learned to implement a HashMap and perform essential operations such as insertion, deletion, lookup, and traversal, which enhanced my grasp of data structures and their practical applications.

Acknowledgements:

I would like to express my sincere gratitude to the following individuals and resources for their invaluable contributions to this assignment:

1. Prof. Bruce Maxwell: Thank you, Professor Bruce Maxwell for your guidance in this assignment. Your expertise in algorithms greatly influenced my work.
2. Classmates and TA's: I appreciate the discussions of my classmates, and TA's who clarified my doubts.
3. DPV Algorithms Textbook: This textbook was a useful resource for me to understand the basics of algorithms.
4. https://www.geeksforgeeks.org/hash-map-in-python/#: This website was a useful resource to understand the HashMap.
5. https://www.baeldung.com/cs/ransac: This website was a useful resource to understand RANSAC.