- **Team Name :** AirBnb Experts
- **Team Members:** Jesse Watson, Miguel Reyes, Braden Lord
- **Project Title:** Operation AirBnb
- **Link to GitHub Repo:** https://github.com/cashjmoney/Creative-Works
- **Link to Video Demo:** https://www.youtube.com/watch?v=UffYuG1hxGk

Extended and refined project proposal:

1. **Problem:** AirBnb listings lack clarity on where they are specifically in the US which can lead to customer excessive time consumption. Airbnb also does not let you filter for reviews or reviews per month which could help to gauge how nice the Airbnb is compared to the pictures that are posted with it.

2. **Motivation:** (1) AirBnb tends to be expensive and finding locations on airbnb website can be tedious and time consuming.Our application will serve to make pinpointing the location AirBnbs time efficient (2) Wanting to look for listings in specific areas you may not know the name to.

3. **Features**: Based on the user input surrounding filters: pricings,reviews per month,total reviews and radius from which the user clicks on the map(searches around that point),markers will be placed on the map according to those filters.The program allows for a range of values for each filter accounting for any choice the user may want.These functionalities together allows the user to choose from a range of AirBnb based on their preference. While the GUI does show all points in this area, the program also creates an excel spreadsheet containing all information on the filtered Airbnb's for the user to make their final decision.

4. **Data**: We will be using a public dataset from Kaggle: https://www.kaggle.com/datasets/kritikseth/us-airbnb-open-data?select=AB_US_2023.csv This data set stores information such as the Airbnb host's ID, their name and neighborhood as well as more specific information like the price of the air bnb and its longitude and latitude. Additionally it also stores the number of reviews and the reviews per month which will be used for filtering.

5. **Tools:** Programming languages or any tools/frameworks we will be using: Python, NiceGUI, Pandas

6. **Algorithms Implemented:** The main type of algorithm that we wanted to focus on were sorting algorithms. We decided to focus specifically on quick sort and merge sort as they are some of the faster sorting algorithms and both work well with large data sets. These two algorithms will be used for sorting the price as this will be the largest dataset and the merge sort will be used on this

filtered data set to then sort for the total number of reviews and the reviews per month. After each sorting algorithm is used the dataset is then filtered based on the inputs given by the user.

7. **Additional Data structures/ Algorithms:** The data from the data set is stored as an ordered map that has the name of the data as the key, such as host ID or airbnb price and then stores the rest of the data for each airbnb in the dataset as a list behind each key. After filtering or sorting is done, the data is then written to a csv file to be stored on the hard drive for the user to view. The data structures that will hold the data from the csv files will be DataFrames. We will convert these DataFrames into dictionaries for easy filtering methods, then convert back to DataFrames which we will then convert into csv files.

8. **Distribution of Responsibility and Roles:**

   Braden: Back end, developed class for organization and worked on a final function that allows for easy use as well as assisting with the filtering functions.

   Jesse: Front end, developed all visuals as well as the GUI that the user can interact with.

   Miguel: Back end, developed merge sort algorithm and quick sort algorithm as well as the first filtering function that was used as a template for the others.

Analysis:

General Analysis:

After the proposal, the group decided to include multiple parameters for the filter built into our project; which we initially only included a filter for price ranges. For instance, we included an option to filter AirBnb listings based on total reviews and/or reviews per month. The reason we added these new filter parameters is simply to assist the customer looking for an AirBnb with the specific amount of reviews or reviews per month they'd be interested in. Moreover, we decided to use two different sorting algorithms, Merge Sort and Quicksort, to filter out the data based on user input. The reason for using these two algorithms is to showcase which is faster when utilizing data structures such as DataFrames and Dictionaries within python.

Big O Worst Time Complexity Analysis:

Function: quicksort_price(self, arr=None):
Time complexity: The time complexity is mainly based on partitioning the main array into three different arrays (left, middle, right) (O(n) complexity based on n elements) and recursion sorting (calling quickosrt_price until arrays have only one or less elements) (O(n) complexity based on n elements). Therefore, the overall worst-case time complexity for this function is O(n^2).

Function: _merge(self, left, right, key):

Time complexity: initializing an empty list O(1) and the while loop depends on the amount of elements in the left and right dictionaries O(n + m). Once the while loop ends, all the elements remaining in either array are added into the empty list O(n + M). The overall worst time complexity for this function is O(n + m), n and m being the number of elements in the left and right array, respectively.

Function: mergesort_number_of_reviews(self, arr=None):
Time complexity: splitting the given dictionary into a left and right half takes O(n) time, n being the amount of elements in the dictionary. The recursion of sorting the arrays with each half of the original array leads to a time complexity of n/2. Therefore the worst overall time complexity is O(nlogn).

Function: mergesort_reviews_per_month(self, arr=None):
Time complexity: splitting the given dictionary into a left and right half takes O(n) time, n being the amount of elements in the dictionary. The recursion of sorting the arrays with each half of the original array leads to a time complexity of n/2. Therefore the worst overall time complexity is O(nlogn). (Same thing as previous merge sort function, just for a different key=reviews_per_month)

Function: price_range(self, start_price, end_price):
Time complexity: the price_range function utilizes the quicksort_price function which has a worst-case time complexity of O(n^2). Filtering through the data based on user input is linear so a O(n) complexity. Creating a DataFrame off a dictionary and writing to a CSV file all have O(n) time complexities. Overall, the worst time complexity is O(n^2) as it dominates O(n).

Function: number_of_reviews_range(self, start_num, end_num):
Time complexity: this function does the exact same thing as the price_range function except it utilizes the MergeSort function which has a time complexity of O(nlogn) and is based on the key=number_of_reviews. Therefore, this function has an overall worst time complexity of O(nlogn).

Function: reviews_per_month_range(self, start_num, end_num):
Time complexity: this function does the exact same thing as the price_range function except it utilizes the MergeSort function which has a time complexity of O(nlogn) and is based on the key=number of reviews. Therefore, this function has an overall worst time complexity of O(nlogn). (This function is the same as number_of_reviews_range function with a different key=reviews_per_month)

Function: find_distance(lat1, long1, lat2, long2):

Time complexity: This function takes in 4 numerical values from user input. The function converts attitude and longitude from degrees to radians which are O(1) operations. The function utilizes trigonometric functions which all are done in constant time O(1). Therefore, the overall worst-time case complexity of this function is O(1) regardless of the values inputted into the function.

Function: sort_all(start_price, end_price, start_reviews, end_reviews, start_rpm, end_rpm,clicked_lat,clicked_long,distance):
Time complexity: This function is a combination of all previous functions with their respective parameters. Therefore, this function is simply dominated with the worst-case time complexity given by the quicksort algorithm: O(n^2).

Reflection: While this project was challenging we found the final product to be rewarding and being able to complete something that visualizes the outcome of the algorithms we are applying in class was also beneficial to us. One challenge for us was getting working separately but still ensuring our code worked together, and we worked around this by having frequent meetings where we debugged our collaborative code to ensure no errors when combining code. If we were to start over again using gitup to collaborate could have been beneficial as our main method of collaboration was to send back and forth copies of our individual code.

References:

Seth, K. (2023b). U.S. Airbnb Open Data. Retrieved from https://www.kaggle.com/datasets/kritikseth/us-airbnb-open-data?select=AB_US_2023.csv