# Project Report

**Name**: Raj Vora

**UFID**: 3555-1411

**E-mail**: rajvora@ufl.edu

## Function prototypes

### ride.py

class Ride:

void init(Ride, int, int, int)

boolean lt([int, int, int], [int, int, int])

string repr(Ride):

### min_heap.py

class MinHeap:

void init(MinHeap)

int parent(MinHeap , int)

int left_child(MinHeap , int)

int right_child(MinHeap, int)

void swap(MinHeap, int, int)

void insert(MinHeap, Ride)

Ride extract_min(MinHeap)

void decrease_key(MinHeap, int, int)

void delete(MinHeap, int)

void min_heapify(MinHeap, int)

void update(MinHeap, min, min)

## red_black_tree.py

class Node:

void init(Node, Ride, string)

class RedBlackTree:

void init(RedBlackTree)

void left_rotate(RedBlackTree , Node)

void right_rotate(RedBlackTree , Node)

void insert(RedBlackTree , Ride)

void fix_insert(RedBlackTree, Node)

void delete(RedBlackTree, int)

void fix_delete(RedBlackTree, Node)

void transplant(RedBlackTree, Node, Node)

void minimum(RedBlackTree, Node)

Node search(RedBlackTree, int)

void update(RedBlackTree, int, int)

## gatorTaxi.py

class RideManager:

void init(RideManager)

void get_ride_by_number(RideManager, int)

void get_rides_by_range(RideManager, int, int)

void _inorder_traversal(RideManager, Node, Ride[], int, int)

void add_ride(RideManager, int, int, int)

void get_next_ride(RideManager)

void delete_ride(RideManager, int)

void update_ride(RideManager, int, int)

# Time complexity

## Min Heap:

- insert(): O(log n)
  This is because the worst case scenario is when the element being inserted is larger than all other elements in the heap, and it needs to be swapped with its parent all the way up to the root node until it finds its correct position.
- extract_min(): O(log n)
  After removing the root node, the function needs to reorder the heap by moving the last element to the root position and then calling min_heapify to maintain the min-heap property. min_heapify has a time complexity of O(log n) as well.
- delete(): O(n)
  This is because the function needs to search for the element to delete, which can take up to n steps in the worst case scenario where the element is at the end of the heap. Once the element is found, the function calls decrease_key and extract_min, which both have a time complexity of O(log n).
- update(): O(n log n)
  This is because the function calls delete() and insert(), which are both O(n) in the worst case, and also because it iterates over the entire heap to find the ride with the given rideNumber.

## Red Black Tree:

- insert(): O(log n)
  Since the function uses a while loop to traverse the tree and find the appropriate position for the new node.
- delete(): O(log n)
  The function first searches for the node to be deleted, which takes O(log n) time. If the node to be deleted has two children, the function finds the minimum value in the right subtree, which also takes O(log n) time. The function then calls the transplant function, which takes constant time. Finally, the fix_delete function is called to restore the Red-Black tree properties, which takes O(log n) time in the worst case.
- search(): O(log n)
  The function uses a while loop to traverse the tree until it finds the node with the given value or reaches a leaf node.
- update(): O(n)
  This is because the function iterates through the entire heap to find the ride with the given ride number. If the ride is found, the time complexity for updating the ride duration is O(1). However, if the new duration is greater than the current duration, then the ride needs to be removed and reinserted into the heap. Removing an element from the heap

takes O(log n) time and inserting a new element into the heap takes O(log n) time as well, giving a total time complexity of O(log n) for this operation.

## Space Complexity

### Min Heap

The space complexity of the Min Heap is O(n), where n is the maximum number of elements that can be stored in the heap. This is because we need to allocate space for each element in the heap.

### Red Black Tree

The maximum height of the Red-Black Tree is logarithmic with respect to the number of elements inserted, and hence the space required for a balanced tree is O(log n). However, the worst-case height of a Red-Black Tree is 2log(n+1), which is still logarithmic but with a higher constant factor. The overall space complexity of the implementation is O(n), where n is the number of elements in the tree.