WDATA System-Wide Trip Simulation

Testing whether it is possible to tap in and out of all 98 WMATA Metrorail stations in one operating day

The Challenge, Should You Choose to Accept It:

- Log a visit to every Metrorail station in the WMATA system in one open-to-close period. Visits will be verifiable via "trip history" on the WMATA SmarTrip app.

The Details:

- Only Metrorail trains may be used to travel between stations (no coordinated pickups at the end of a line, no running between two adjacent stations, etc.).

Sources and Strategy:

- The challenge will be attempted on a Friday to maximize available time. The system begins operation at 5:00 AM and runs until 1:00 AM the following day, giving challenge participants 20 hours to accomplish the feat. The last train on the Silver Line is rumored to run until 2:15 AM, but this assumption was not made in the simulation.
- Train frequencies were taken from the weekday timetables on the WMATA website. Variation in these frequencies was un-scientifically estimated as a standard deviation of 2 minutes for frequencies under 10 minutes and 3 minutes for frequencies of 10+ minutes.
- At stations where challenge participants will (or may) transfer lines, uniform distributions with a minimum of 2 minutes and maximum of (line frequency + standard deviation) were used to estimate the time before a boardable train arrives.
- Ride times between stations were calculated using the trip planner on the WMATA website. Ride times between stations were assumed to be the same in both directions.

Planned Route:

*Legs of the trip in Green and Red may be switched.

In [1]:
```
# Decide how many simulations you want to run
total_iterations = 1000000

# Enter estimates of frequencies and standard deviations for scheduled frequencies
freq_5_6 = 6
freq_6_8 = 8
freq_10_12 = 12

sd_5_6 = 2
sd_6_8 = 2
sd_10_12 = 3

# Enter estimate of minimum time spent in a station
min_time = 2
```

In [2]:
```
import numpy as np
import pandas as pd
import random

trip_times = []
potomac_ave_splits = []
oranges_at_end = []
current_iterations = 0

times_at_fort_totten = []
```

```
times_at_lenfant = []
times_at_pentagon = []
times_at_lenfant2 = []
times_at_rosslyn = []
times_at_efc = []

for i in range(total_iterations):
    # Red Line Shady Grove to Glenmont and back to Fort Totten
    # 23 stations, 70 then 15 min ride time, frequency 5-6 mins
    # Don't count Shady Grove wait time since it is the starting point
    # Skip Metro Center, Gallery Place, Fort Totten
    rng = np.random.default_rng()
    red_time = sum(rng.normal(size=23)*sd_5_6 + freq_5_6) + 70 + 15
    total_time = red_time
    times_at_fort_totten.append(total_time)

    # Green Line Fort Totten to Greenbelt and back to Shaw
    # 8 stations, 13 then 22 min ride time, frequency 6-8 mins
    wait_time = np.random.uniform(low=min_time, high=freq_6_8+sd_6_8)
    green_time = sum(rng.normal(size=8)*sd_6_8 + freq_6_8) + 13 + 22
    total_time += (wait_time + green_time)


    # Green/Yellow Line Shaw to L'Enfant Plaza
    # 3 stations, 7 min ride time, frequency 6-8 mins
    # One line (whichever we're on) is normal; the other is uniform
    green_time = np.random.uniform(low=min_time, high=freq_6_8+sd_6_8, size=3)
    yellow_time = rng.normal(size=3)*sd_6_8 + freq_6_8
    fast_times = sum(np.minimum(green_time, yellow_time)) + 7
    total_time += fast_times
    times_at_lenfant.append(total_time)

    # Blue/Orange/Silver Line L'Enfant Plaza to Potomac Ave
    # 4 stations, 7 min ride time, frequency 10-12 mins
    # One line (whichever we're on) is normal; the others are uniform
    wait_time = np.random.uniform(low=min_time, high=7) ### High=7 because there are 3
    blue_time = rng.normal(size=4)*sd_10_12 + freq_10_12
    orange_time = np.random.uniform(low=min_time, high=10+sd_10_12, size=4) ### Orange
    silver_time = np.random.uniform(low=min_time, high=freq_10_12+sd_10_12, size=4)
    fast_times = np.minimum(blue_time, orange_time)
    fast_times = sum(np.minimum(fast_times, silver_time)) + 7
    total_time += (wait_time + fast_times)

    # 1) Orange Line Potomac Ave to New Carrollton and back to Stadium-Armory
    # 2) Blue/Silver Line Stadium-Armory to Downtown Largo and back to L'Enfant Plaza
    # 1 and 2 may be switched depending on line at Potomac Ave
    potomac_fork = random.randint(1, 3)
    if potomac_fork == 1 or potomac_fork == 2:
        # Blue/Silver Line Potomac Ave to Downtown Largo and back to Stadium-Armory
        # 5 stations, 18 then 16 min ride time, frequency 10-12 mins
        # One line (whichever we're on) is normal; the other is uniform
        blue_time = np.random.uniform(low=min_time, high=freq_10_12+sd_10_12, size=5)
        silver_time = rng.normal(size=5)*sd_10_12 + freq_10_12
        corridor_time = sum(np.minimum(blue_time, silver_time)) + 18 + 16
        total_time += corridor_time
        # Orange Line Stadium-Armory to New Carrollton and back to L'Enfant Plaza
        # 5 stations, 15 then 24 min ride time, frequency 10 mins
        wait_time = np.random.uniform(low=min_time, high=10+sd_10_12)
        orange_time = sum(rng.normal(size=5)*sd_10_12 + 10) + 15 + 24
        total_time += (wait_time + orange_time)
```

```
        potomac_ave_splits.append(potomac_fork)
    else:
        # Orange line Potomac Ave to New Carrollton and back to Stadium-Armory
        # 5 stations, 17 then 15 min ride time, frequency 10 mins
        orangetime = sum(rng.normal(size=5)*sd_10_12 + 10) + 17 + 15
        total_time += orangetime
        # Blue/Silver Line Stadium-Armory to Downtown Largo and back to L'Enfant Plaza
        # 5 stations, 16 then 24 min ride time, frequency 10-12 mins
        # One line (whichever we're on) is normal; the other is uniform
        blue_wait_time = np.random.uniform(low=min_time, high=freq_10_12+sd_10_12)
        silver_wait_time = np.random.uniform(low=min_time, high=freq_10_12+sd_10_12)
        wait_time = np.minimum(blue_wait_time, silver_wait_time)
        blue_time = np.random.uniform(low=min_time, high=freq_10_12+sd_10_12, size=5)
        silver_time = rng.normal(size=5)*sd_10_12 + freq_10_12
        corridor_time = sum(np.minimum(blue_time, silver_time)) + 16 + 24
        total_time += wait_time + corridor_time
        potomac_ave_splits.append(potomac_fork)

    # Yellow Line L'Enfant Plaza to Pentagon
    # 0 stations, 5 min ride time, frequency 8 mins
    wait_time = np.random.uniform(low=min_time, high=freq_6_8+sd_6_8)
    yellow_time = wait_time + 5
    total_time += yellow_time
    times_at_pentagon.append(total_time)

    # Blue/Yellow Line Pentagon to Arlington Cemetery and back to Braddock Road
    # 4 stations, 3 then 15 min ride time, frequency 8/12 mins
    # One line (whichever we're on) is normal; the other is uniform
    blue_wait_1 = np.random.uniform(low=min_time, high=freq_10_12+sd_10_12)
    blue_wait_2 = np.random.uniform(low=min_time, high=freq_10_12+sd_10_12)
    yellow_time = np.random.uniform(low=min_time, high=freq_6_8+sd_6_8, size=4)
    blue_time = rng.normal(size=4)*sd_10_12 + freq_10_12
    fast_times = sum(np.minimum(yellow_time, blue_time)) + 3 + 15
    total_time += (blue_wait_1 + blue_wait_2 + fast_times)

    # Blue Line Braddock Road to Franconia and back to King Street
    # 2 stations, 12 then 10 min ride time, frequency 10-12 mins
    wait_time =  np.random.uniform(low=min_time, high=freq_10_12+sd_10_12)
    blue_time = sum(rng.normal(size=2)*sd_10_12 + freq_10_12) + 12 + 10
    total_time += (wait_time + blue_time)

    # Yellow Line King Street to Huntington and back to L'Enfant Plaza
    # 2 stations, 3 then 22 min ride time, frequency 6-8 mins
    wait_time = np.random.uniform(low=min_time, high=freq_6_8+sd_6_8)
    yellow_time = sum(rng.normal(size=2)*sd_6_8 + freq_6_8) + 3 + 22
    total_time += (wait_time + yellow_time)
    times_at_lenfant2.append(total_time)

    # Green Line L'Enfant Plaza to Branch Ave and back to L'Enfant Plaza
    # 8 stops, 21 then 21 min ride time, frequency 6-8 mins
    wait_time = np.random.uniform(low=min_time, high=freq_6_8+sd_6_8)
    green_time = sum(rng.normal(size=8)*sd_6_8 + freq_6_8) + 21 + 21
    total_time += (wait_time + green_time)

    # Blue/Orange/Silver Line L'Enfant Plaza to Rosslyn
    # 6 stations, 13 min ride time, frequency 10-12 mins
    # One line (whichever we're on) is normal; the others are uniform
    wait_time = np.random.uniform(low=min_time, high=7) ### High=7 because there are 3
    blue_time = rng.normal(size=6)*sd_10_12 + freq_10_12
    orange_time = np.random.uniform(low=min_time, high=10+sd_10_12, size=6) ### Orange
```

```
    silver_time = np.random.uniform(low=min_time, high=freq_10_12+sd_10_12, size=6)
    fast_times = np.minimum(blue_time, orange_time)
    fast_times = sum(np.minimum(fast_times, silver_time)) + 13
    total_time += (wait_time + fast_times)
    times_at_rosslyn.append(total_time)

    # Orange/Silver Lines Rosslyn to East Falls Church
    # 4 stations, 12 min ride time, frequency 10-12 mins
    # One line (whichever we're on) is normal; the other is uniform
    wait_time = np.random.uniform(low=min_time, high=9) ### High=9 because there are 2
    orange_time = np.random.uniform(low=min_time, high=10+sd_10_12, size=4)
    silver_time = rng.normal(size=4)*sd_10_12 + freq_10_12
    fast_times = sum(np.minimum(orange_time, silver_time)) + 12
    total_time += (wait_time + fast_times)
    times_at_efc.append(total_time)

    # Orange Line East Falls Church to Vienna and back to East Falls Church
    # 3 stations, 12 then 12 min ride time, frequency 15 mins
    # Are challenge participants on Orange Line at East Falls Church?
    orange_at_end = random.randint(0, 1)
    if orange_at_end == 0:
        wait_time = np.random.uniform(low=min_time, high=15+sd_10_12)
        orange_time = sum(rng.normal(size=3)*sd_10_12 + 15) + 12 + 12
        total_time += (wait_time + orange_time)
        oranges_at_end.append(orange_at_end)
    else:
        wait_time = 0
        orange_time = sum(rng.normal(size=3)*sd_10_12 + 15) + 12 + 12
        total_time += (wait_time + orange_time)
        oranges_at_end.append(orange_at_end)

    # Silver Line East Falls Church to Assburn
    # 10 stations, 43 min ride time, frequency 15 mins
    wait_time = np.random.uniform(low=min_time, high=15+sd_10_12)
    silver_time = sum(rng.normal(size=10)*sd_10_12 + freq_10_12) + 43
    total_time += (wait_time + silver_time)

    total_time /= 60
    trip_times.append(total_time)
    current_iterations += 1

    if current_iterations % (total_iterations*0.01) == 0 and current_iterations < (tot
        print('Finished ' + str(current_iterations) + ' iterations, master.')
    elif current_iterations % (total_iterations*0.01) == 0 and current_iterations < (t
        print('Now completed ' + str(current_iterations) + ' iterations, sir. I hope t
    elif current_iterations % (total_iterations*0.01) == 0 and current_iterations < (t
        print('WE HAVE REACHED ' + str(current_iterations) + ' ITERATIONS. SIMULATION
    elif current_iterations % (total_iterations*0.01) == 0 and current_iterations < to
        print('EXCEEDING ' + str(current_iterations) + ' ITERATIONS. SIMULATION ADVANC


prop_under_20 = round((len(list(filter(lambda x: x < 20, trip_times))) / total_iterati
print('IT IS DONE. ALL ' + str(current_iterations) + ' ITERATIONS HAVE BEEN RUN. IN AF
```

```
Finished 10000 iterations, master.
Finished 20000 iterations, master.
Finished 30000 iterations, master.
Finished 40000 iterations, master.
Finished 50000 iterations, master.
Finished 60000 iterations, master.
Finished 70000 iterations, master.
Finished 80000 iterations, master.
Finished 90000 iterations, master.
Finished 100000 iterations, master.
Finished 110000 iterations, master.
Finished 120000 iterations, master.
Finished 130000 iterations, master.
Finished 140000 iterations, master.
Finished 150000 iterations, master.
Finished 160000 iterations, master.
Finished 170000 iterations, master.
Finished 180000 iterations, master.
Finished 190000 iterations, master.
Finished 200000 iterations, master.
Finished 210000 iterations, master.
Finished 220000 iterations, master.
Finished 230000 iterations, master.
Finished 240000 iterations, master.
Now completed 250000 iterations, sir. I hope this pleases you.
Now completed 260000 iterations, sir. I hope this pleases you.
Now completed 270000 iterations, sir. I hope this pleases you.
Now completed 280000 iterations, sir. I hope this pleases you.
Now completed 290000 iterations, sir. I hope this pleases you.
Now completed 300000 iterations, sir. I hope this pleases you.
Now completed 310000 iterations, sir. I hope this pleases you.
Now completed 320000 iterations, sir. I hope this pleases you.
Now completed 330000 iterations, sir. I hope this pleases you.
Now completed 340000 iterations, sir. I hope this pleases you.
Now completed 350000 iterations, sir. I hope this pleases you.
Now completed 360000 iterations, sir. I hope this pleases you.
Now completed 370000 iterations, sir. I hope this pleases you.
Now completed 380000 iterations, sir. I hope this pleases you.
Now completed 390000 iterations, sir. I hope this pleases you.
Now completed 400000 iterations, sir. I hope this pleases you.
Now completed 410000 iterations, sir. I hope this pleases you.
Now completed 420000 iterations, sir. I hope this pleases you.
Now completed 430000 iterations, sir. I hope this pleases you.
Now completed 440000 iterations, sir. I hope this pleases you.
Now completed 450000 iterations, sir. I hope this pleases you.
Now completed 460000 iterations, sir. I hope this pleases you.
Now completed 470000 iterations, sir. I hope this pleases you.
Now completed 480000 iterations, sir. I hope this pleases you.
Now completed 490000 iterations, sir. I hope this pleases you.
WE HAVE REACHED 500000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 510000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 520000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 530000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 540000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 550000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 560000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 570000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 580000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 590000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 600000 ITERATIONS. SIMULATION STILL STABLE.
```

```
WE HAVE REACHED 610000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 620000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 630000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 640000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 650000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 660000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 670000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 680000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 690000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 700000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 710000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 720000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 730000 ITERATIONS. SIMULATION STILL STABLE.
WE HAVE REACHED 740000 ITERATIONS. SIMULATION STILL STABLE.
EXCEEDING 750000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 760000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 770000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 780000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 790000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 800000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 810000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 820000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 830000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 840000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 850000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 860000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 870000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 880000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 890000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 900000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 910000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 920000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 930000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 940000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 950000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 960000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 970000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 980000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
EXCEEDING 990000 ITERATIONS. SIMULATION ADVANCING BEYOND CONTROL.
IT IS DONE. ALL 1000000 ITERATIONS HAVE BEEN RUN. IN APPROXIMATELY 73.29% OF POSSIBLE
SCENARIOS
 YOU LOGGED ALL WMATA STATIONS IN UNDER 20 HOURS. GODSPEED.
```

In [3]:

```
trip_times = pd.Series(trip_times)
potomac_ave_splits = pd.Series(potomac_ave_splits)
oranges_at_end = pd.Series(oranges_at_end)

times_at_fort_totten = pd.Series(times_at_fort_totten)/60
times_at_lenfant = pd.Series(times_at_lenfant)/60
times_at_pentagon = pd.Series(times_at_pentagon)/60
times_at_lenfant2 = pd.Series(times_at_lenfant2)/60
times_at_rosslyn = pd.Series(times_at_rosslyn)/60
times_at_efc = pd.Series(times_at_efc)/60

dataframe = pd.DataFrame({'trip_time': trip_times,
                          'potomac_ave_split': potomac_ave_splits,
                          'orange_at_end': oranges_at_end,
                          'fort_totten': times_at_fort_totten,
                          'lenfant': times_at_lenfant,
                          'pentagon': times_at_pentagon,
```

```
                          'lenfant2': times_at_lenfant2,
                          'rosslyn': times_at_rosslyn,
                          'efc': times_at_efc})

dataframe.head()
```

Out[3]:

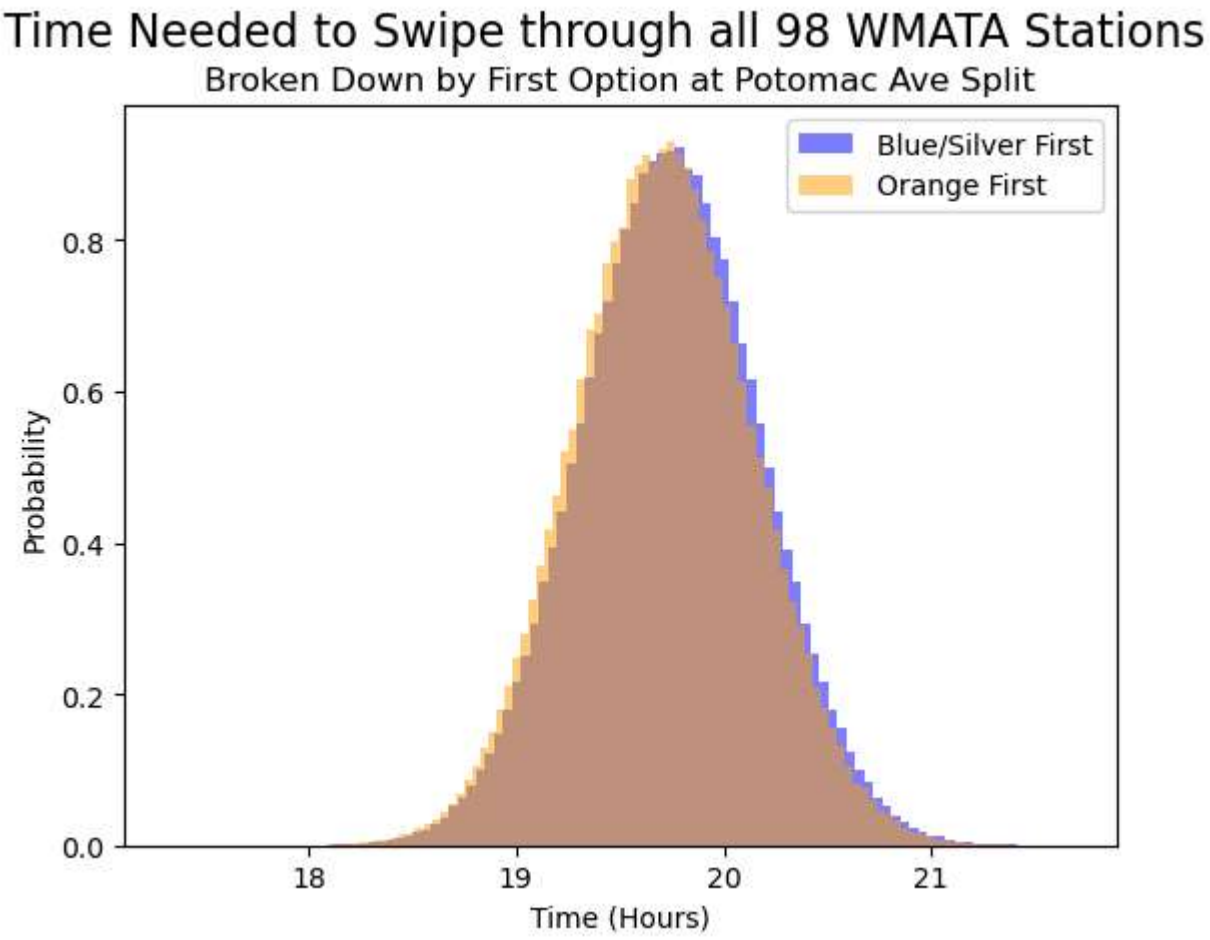
	trip_time	potomac_ave_split	orange_at_end	fort_totten	lenfant	pentagon	lenfant2	rosslyn
0	19.441837	3	1	3.838057	5.941726	9.260939	11.807381	14.38975
1	20.360425	3	1	4.162787	6.301311	9.955040	12.832944	15.56841
2	20.118908	2	1	3.952817	6.176878	9.941541	12.674695	15.58714
3	20.264003	1	1	3.907275	6.040513	9.507244	12.302391	15.24540
4	19.486305	1	1	3.481425	5.698972	9.622741	12.131546	14.96996

In [4]:

```
import matplotlib.pyplot as plt

plt.hist(dataframe[dataframe['potomac_ave_split']<=2]['trip_time'], density=True, bins
         label='Blue/Silver First', color='blue', alpha=0.5)
plt.hist(dataframe[dataframe['potomac_ave_split']>2]['trip_time'], density=True, bins=
         label='Orange First', color='orange', alpha=0.5)
plt.suptitle('Time Needed to Swipe through all 98 WMATA Stations', fontsize=16)
plt.title('Broken Down by First Option at Potomac Ave Split')
plt.legend(loc='upper right')
plt.xlabel('Time (Hours)')
plt.ylabel('Probability')
```

Out[4]:

```
Text(0, 0.5, 'Probability')
```

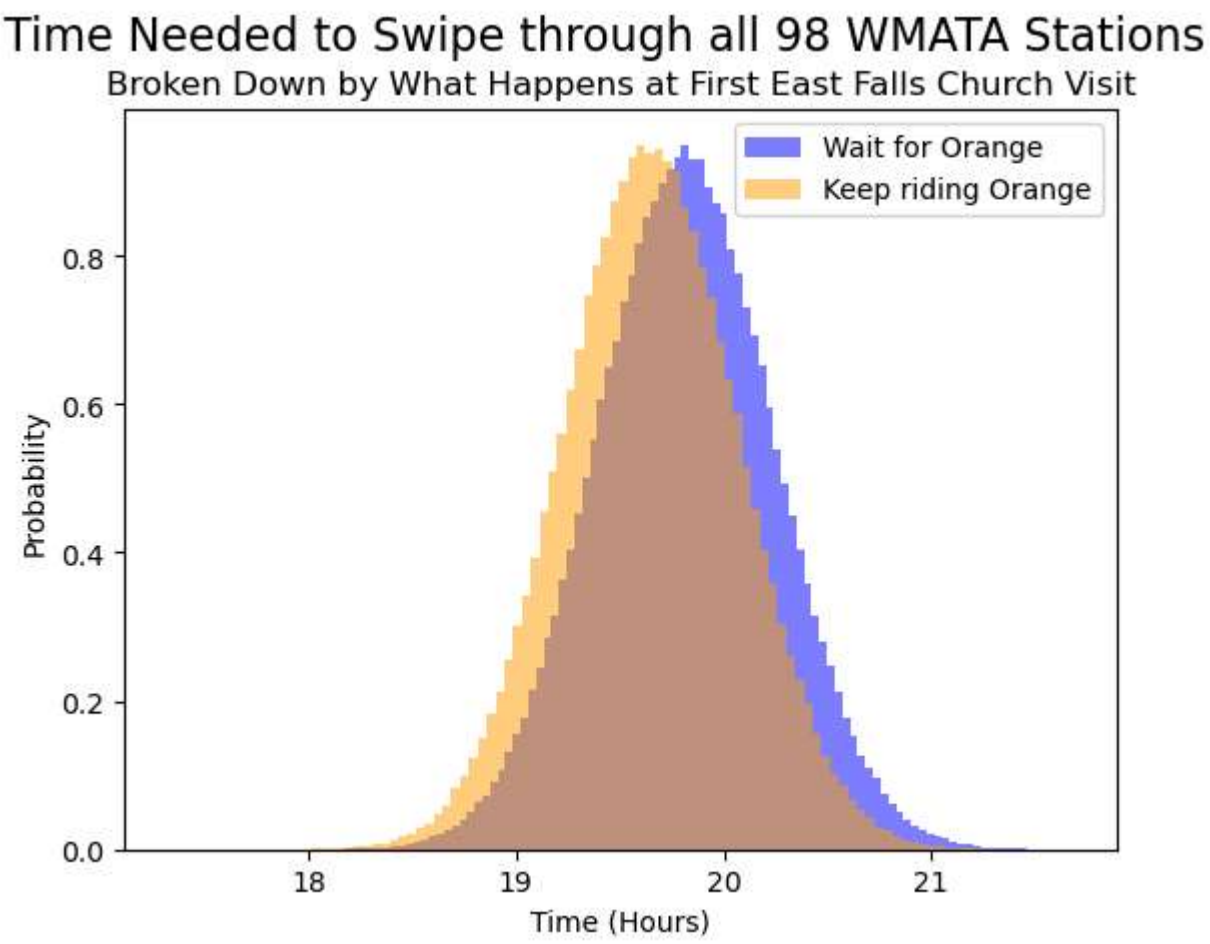


In [5]:
```
blue_mean = dataframe[dataframe['potomac_ave_split']<=2]['trip_time'].mean()
orange_mean = dataframe[dataframe['potomac_ave_split']>2]['trip_time'].mean()
diff = round((blue_mean-orange_mean)*60, 2)

print('Average time when Blue/Silver Line taken first: ' + str(round(blue_mean, 3)) +
print('Average time when Orange Line taken first: ' + str(round(orange_mean, 3)) + ' h
print('You save approximately ' + str(diff) + ' minutes when you take the Orange Line
```

```
Average time when Blue/Silver Line taken first: 19.744 hours
Average time when Orange Line taken first: 19.707 hours
You save approximately 2.21 minutes when you take the Orange Line first at Potomac Av
e.
```

In [6]:
```
plt.hist(dataframe[dataframe['orange_at_end']==0]['trip_time'], density=True, bins=100
         label='Wait for Orange', color='blue', alpha=0.5)
plt.hist(dataframe[dataframe['orange_at_end']==1]['trip_time'], density=True, bins=100
         label='Keep riding Orange', color='orange', alpha=0.5)
plt.suptitle('Time Needed to Swipe through all 98 WMATA Stations', fontsize=16)
plt.title('Broken Down by What Happens at First East Falls Church Visit')
plt.legend(loc='upper right')
plt.xlabel('Time (Hours)')
plt.ylabel('Probability')
```

Out[6]:
```
Text(0, 0.5, 'Probability')
```



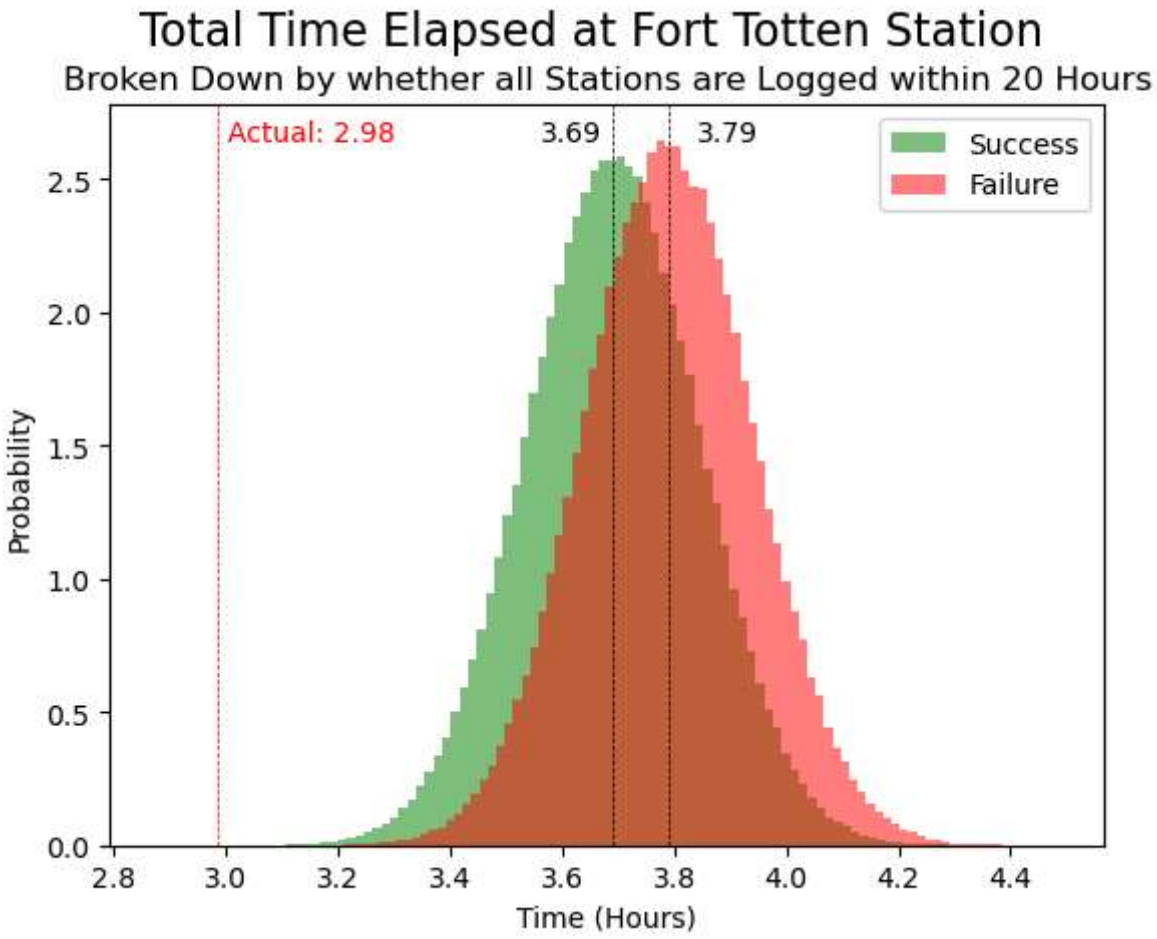
In [7]:
```
wait_mean = dataframe[dataframe['orange_at_end']==0]['trip_time'].mean()
orange_mean = dataframe[dataframe['orange_at_end']==1]['trip_time'].mean()
diff = round((wait_mean-orange_mean)*60, 2)

print('Average time when you have to wait for Orange Line: ' + str(round(wait_mean, 3)
print('Average time when you are already on Orange Line: ' + str(round(orange_mean, 3)
print('You save approximately ' + str(diff) + ' minutes when you are already on Orange
```

```
Average time when you have to wait for Orange Line: 19.815 hours
Average time when you are already on Orange Line: 19.648 hours
You save approximately 10.01 minutes when you are already on Orange Line at East Fall
s Church.
```

In [8]:
```
success = dataframe[dataframe['trip_time']<=20]['fort_totten']
failure = dataframe[dataframe['trip_time']>20]['fort_totten']

plt.hist(success, density=True, bins=100,
         label='Success', color='green', alpha=0.5)
plt.hist(failure, density=True, bins=100,
         label='Failure', color='red', alpha=0.5)

plt.suptitle('Total Time Elapsed at Fort Totten Station', fontsize=16)
plt.title('Broken Down by whether all Stations are Logged within 20 Hours')
plt.legend(loc='upper right')
plt.xlabel('Time (Hours)')
plt.ylabel('Probability')

plt.axvline(success.mean(), color='k', linestyle='dashed', linewidth=0.5)
plt.axvline(failure.mean(), color='k', linestyle='dashed', linewidth=0.5)
```

```
min_ylim, max_ylim = plt.ylim()
plt.text(success.mean()-0.13, max_ylim*0.95, str(round(success.mean(), 2)))
plt.text(failure.mean()+0.05, max_ylim*0.95, str(round(failure.mean(), 2)))

# Post-Attempt Numbers
plt.axvline(2+59/60, color='r', linestyle='dashed', linewidth=0.5)
plt.text(2+59/60+0.02, max_ylim*0.95, 'Actual: ' + str(round(2+59/60, 2)), color='r')
```

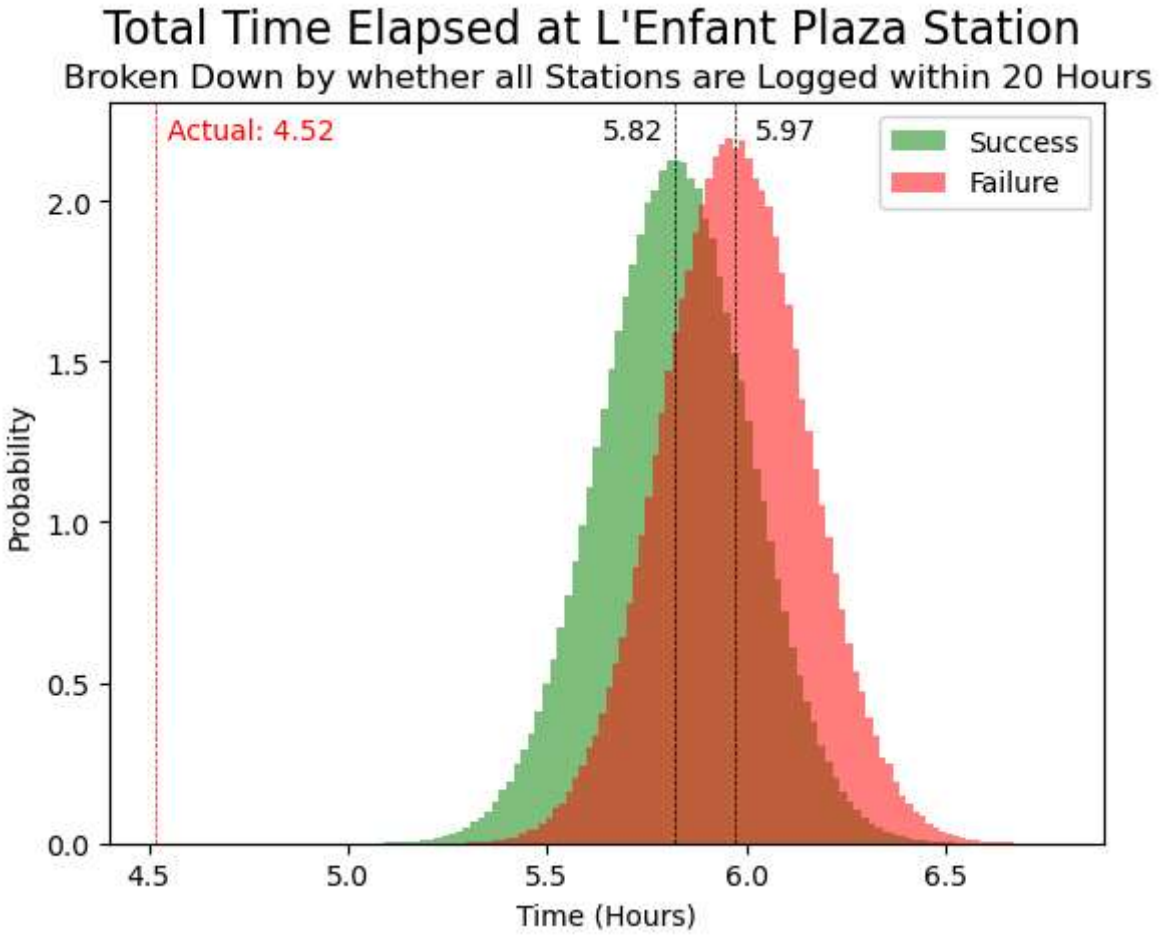
Out[8]:
```
Text(3.0033333333333334, 2.6415025937273513, 'Actual: 2.98')
```



In [17]:
```
success = dataframe[dataframe['trip_time']<=20]['lenfant']
failure = dataframe[dataframe['trip_time']>20]['lenfant']

plt.hist(success, density=True, bins=100,
         label='Success', color='green', alpha=0.5)
plt.hist(failure, density=True, bins=100,
         label='Failure', color='red', alpha=0.5)

plt.suptitle("Total Time Elapsed at L'Enfant Plaza Station", fontsize=16)
plt.title('Broken Down by whether all Stations are Logged within 20 Hours')
plt.legend(loc='upper right')
plt.xlabel('Time (Hours)')
plt.ylabel('Probability')

plt.axvline(success.mean(), color='k', linestyle='dashed', linewidth=0.5)
plt.axvline(failure.mean(), color='k', linestyle='dashed', linewidth=0.5)
min_ylim, max_ylim = plt.ylim()
plt.text(success.mean()-0.18, max_ylim*0.95, str(round(success.mean(), 2)))
plt.text(failure.mean()+0.05, max_ylim*0.95, str(round(failure.mean(), 2)))
```

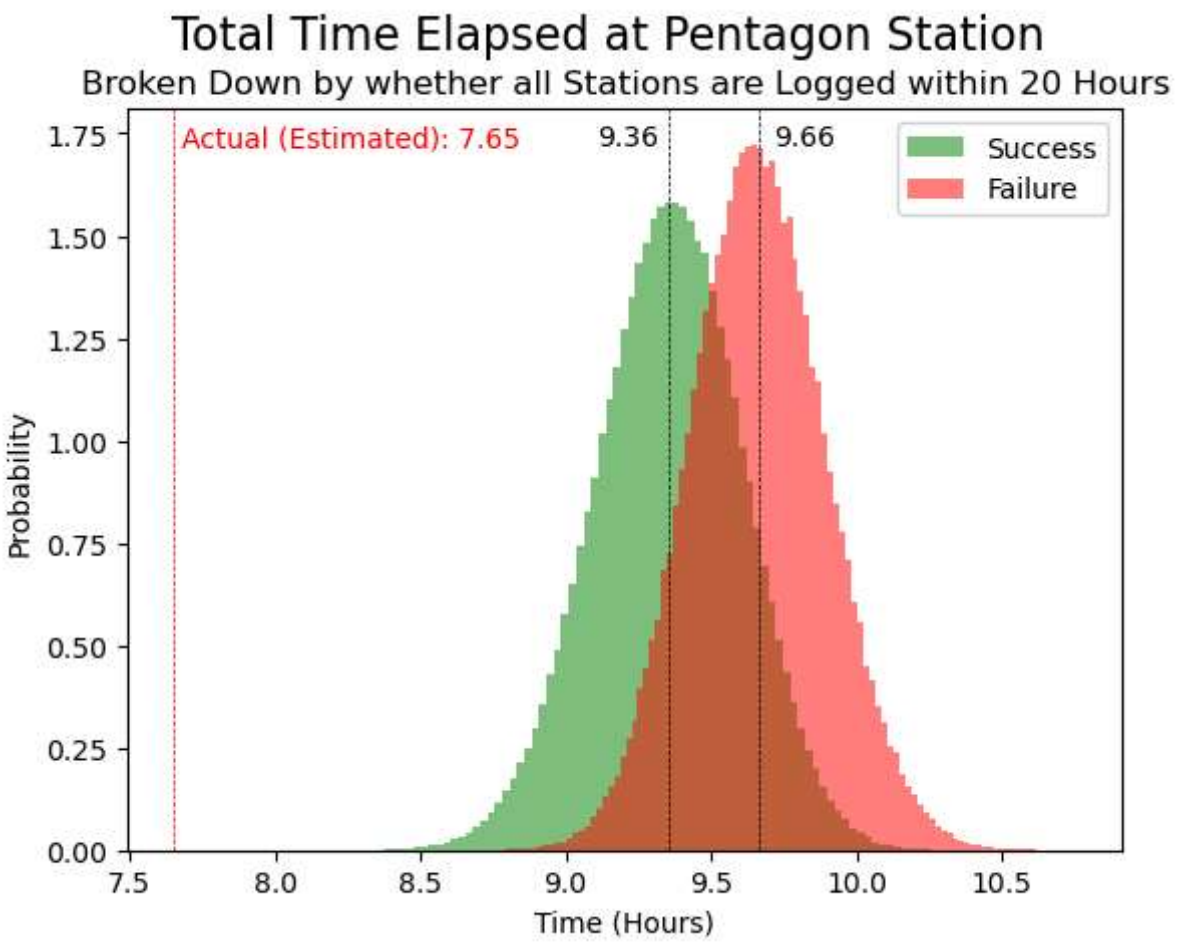
```
# Post-Attempt Numbers
plt.axvline(4+31/60, color='r', linestyle='dashed', linewidth=0.5)
plt.text(4+31/60+0.03, max_ylim*0.95, 'Actual: ' + str(round(4+31/60, 2)), color='r')
```

Out[17]:
```
Text(4.546666666666667, 2.1910667180704064, 'Actual: 4.52')
```



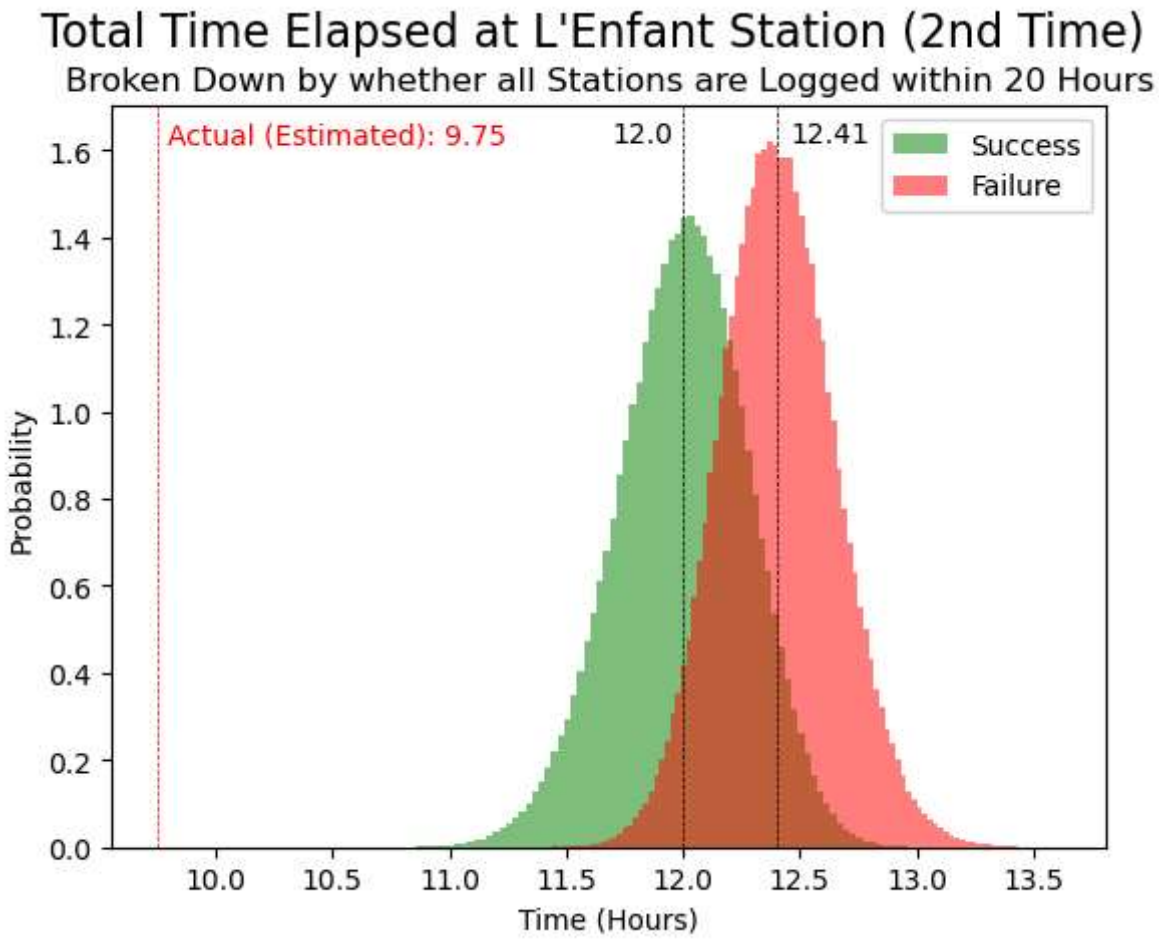
In [18]:
```
success = dataframe[dataframe['trip_time']<=20]['pentagon']
failure = dataframe[dataframe['trip_time']>20]['pentagon']

plt.hist(success, density=True, bins=100,
         label='Success', color='green', alpha=0.5)
plt.hist(failure, density=True, bins=100,
         label='Failure', color='red', alpha=0.5)

plt.suptitle('Total Time Elapsed at Pentagon Station', fontsize=16)
plt.title('Broken Down by whether all Stations are Logged within 20 Hours')
plt.legend(loc='upper right')
plt.xlabel('Time (Hours)')
plt.ylabel('Probability')

plt.axvline(success.mean(), color='k', linestyle='dashed', linewidth=0.5)
plt.axvline(failure.mean(), color='k', linestyle='dashed', linewidth=0.5)
min_ylim, max_ylim = plt.ylim()
plt.text(success.mean()-0.25, max_ylim*0.95, str(round(success.mean(), 2)))
plt.text(failure.mean()+0.05, max_ylim*0.95, str(round(failure.mean(), 2)))

# Post-Attempt Numbers
plt.axvline(7+39/60, color='r', linestyle='dashed', linewidth=0.5)
plt.text(7+39/60+0.03, max_ylim*0.95, 'Actual (Estimated): ' + str(round(7+39/60, 2)),
```

Out[18]: Text(7.680000000000001, 1.7201056225995714, 'Actual (Estimated): 7.65')



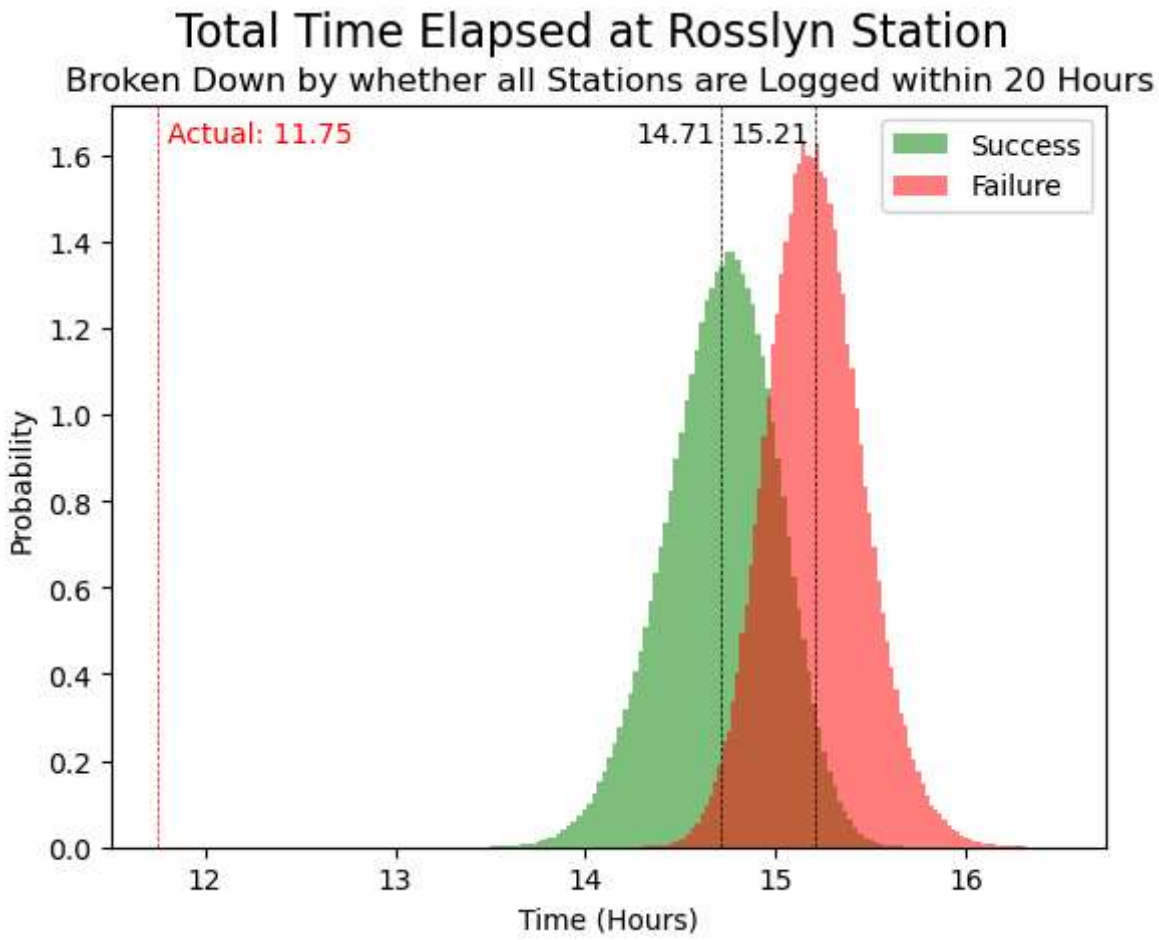
In [20]:
```
success = dataframe[dataframe['trip_time']<=20]['lenfant2']
failure = dataframe[dataframe['trip_time']>20]['lenfant2']

plt.hist(success, density=True, bins=100,
         label='Success', color='green', alpha=0.5)
plt.hist(failure, density=True, bins=100,
         label='Failure', color='red', alpha=0.5)

plt.suptitle("Total Time Elapsed at L'Enfant Station (2nd Time)", fontsize=16)
plt.title('Broken Down by whether all Stations are Logged within 20 Hours')
plt.legend(loc='upper right')
plt.xlabel('Time (Hours)')
plt.ylabel('Probability')

plt.axvline(success.mean(), color='k', linestyle='dashed', linewidth=0.5)
plt.axvline(failure.mean(), color='k', linestyle='dashed', linewidth=0.5)
min_ylim, max_ylim = plt.ylim()
plt.text(success.mean()-0.3, max_ylim*0.95, str(round(success.mean(), 2)))
plt.text(failure.mean()+0.05, max_ylim*0.95, str(round(failure.mean(), 2)))

# Post-Attempt Numbers
plt.axvline(9+45/60, color='r', linestyle='dashed', linewidth=0.5)
plt.text(9+45/60+0.04, max_ylim*0.95, 'Actual (Estimated): ' + str(round(9+45/60, 2)),
```

Out[20]: Text(9.79, 1.6175336300474301, 'Actual (Estimated): 9.75')



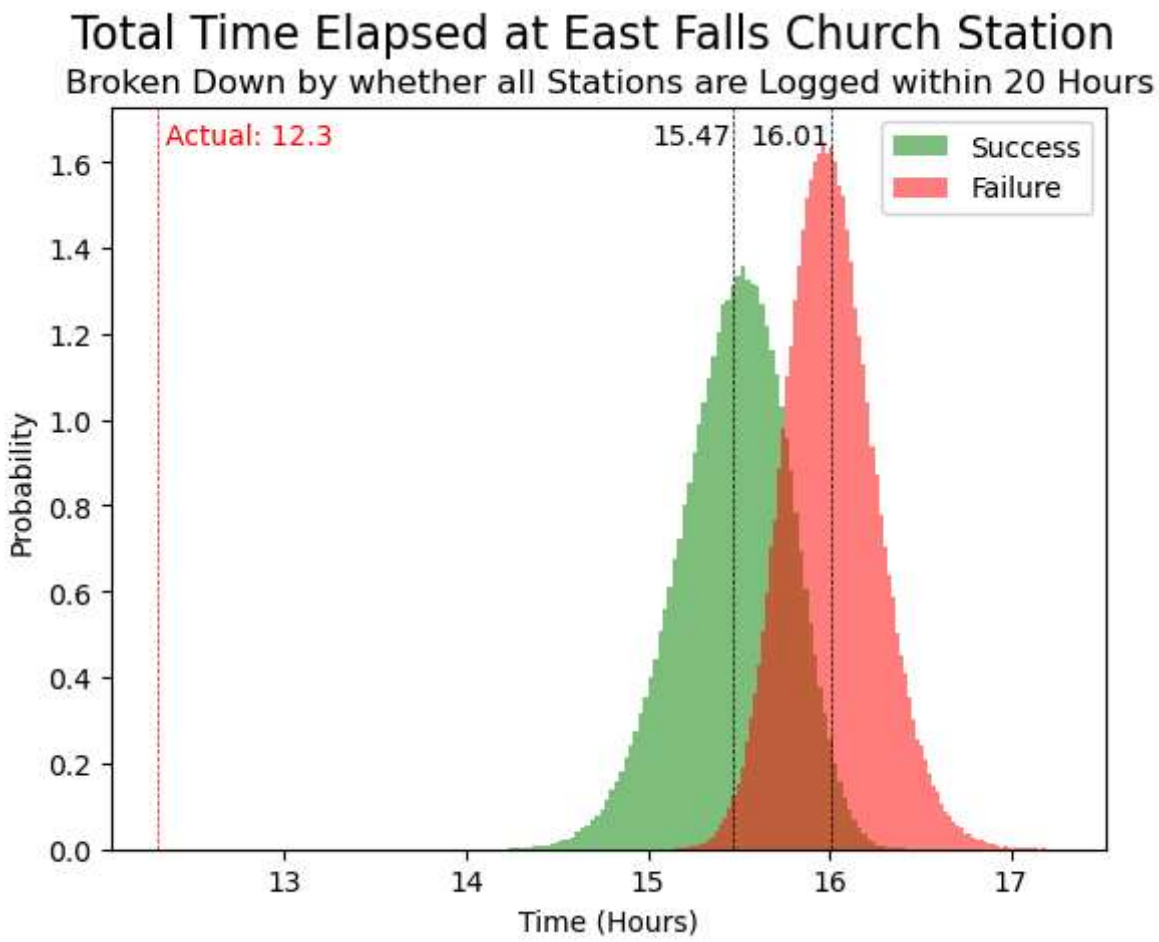
In [24]:
```
success = dataframe[dataframe['trip_time']<=20]['rosslyn']
failure = dataframe[dataframe['trip_time']>20]['rosslyn']

plt.hist(success, density=True, bins=100,
         label='Success', color='green', alpha=0.5)
plt.hist(failure, density=True, bins=100,
         label='Failure', color='red', alpha=0.5)

plt.suptitle('Total Time Elapsed at Rosslyn Station', fontsize=16)
plt.title('Broken Down by whether all Stations are Logged within 20 Hours')
plt.legend(loc='upper right')
plt.xlabel('Time (Hours)')
plt.ylabel('Probability')

plt.axvline(success.mean(), color='k', linestyle='dashed', linewidth=0.5)
plt.axvline(failure.mean(), color='k', linestyle='dashed', linewidth=0.5)
min_ylim, max_ylim = plt.ylim()
plt.text(success.mean()-0.45, max_ylim*0.95, str(round(success.mean(), 2)))
plt.text(failure.mean()-0.45, max_ylim*0.95, str(round(failure.mean(), 2)))

# Post-Attempt Numbers
plt.axvline(11+45/60, color='r', linestyle='dashed', linewidth=0.5)
plt.text(11+45/60+0.05, max_ylim*0.95, 'Actual: ' + str(round(11+45/60, 2)), color='r'
```

Out[24]:
```
Text(11.8, 1.6284737578200152, 'Actual: 11.75')
```



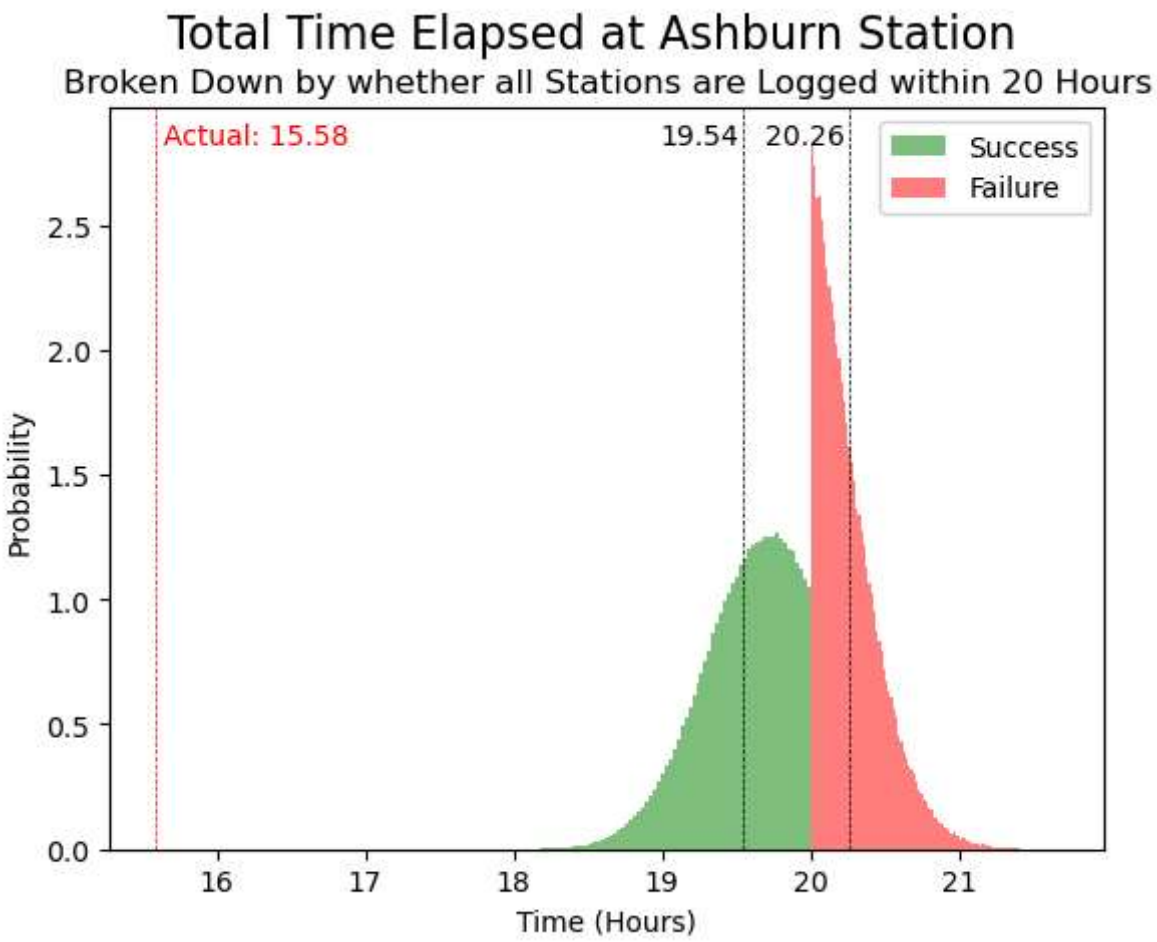
In [27]:
```
success = dataframe[dataframe['trip_time']<=20]['efc']
failure = dataframe[dataframe['trip_time']>20]['efc']

plt.hist(success, density=True, bins=100,
         label='Success', color='green', alpha=0.5)
plt.hist(failure, density=True, bins=100,
         label='Failure', color='red', alpha=0.5)

plt.suptitle('Total Time Elapsed at East Falls Church Station', fontsize=16)
plt.title('Broken Down by whether all Stations are Logged within 20 Hours')
plt.legend(loc='upper right')
plt.xlabel('Time (Hours)')
plt.ylabel('Probability')

plt.axvline(success.mean(), color='k', linestyle='dashed', linewidth=0.5)
plt.axvline(failure.mean(), color='k', linestyle='dashed', linewidth=0.5)
min_ylim, max_ylim = plt.ylim()
plt.text(success.mean()-0.45, max_ylim*0.95, str(round(success.mean(), 2)))
plt.text(failure.mean()-0.45, max_ylim*0.95, str(round(failure.mean(), 2)))

# Post-Attempt Numbers
plt.axvline(12+18/60, color='r', linestyle='dashed', linewidth=0.5)
plt.text(12+18/60+0.05, max_ylim*0.95, 'Actual: ' + str(round(12+18/60, 2)), color='r'
```

Out[27]:
```
Text(12.350000000000001, 1.6400887273276656, 'Actual: 12.3')
```



In [32]:
```
success = dataframe[dataframe['trip_time']<=20]['trip_time']
failure = dataframe[dataframe['trip_time']>20]['trip_time']

plt.hist(success, density=True, bins=100,
         label='Success', color='green', alpha=0.5)
plt.hist(failure, density=True, bins=100,
         label='Failure', color='red', alpha=0.5)

plt.suptitle('Total Time Elapsed at Ashburn Station', fontsize=16)
plt.title('Broken Down by whether all Stations are Logged within 20 Hours')
plt.legend(loc='upper right')
plt.xlabel('Time (Hours)')
plt.ylabel('Probability')

plt.axvline(success.mean(), color='k', linestyle='dashed', linewidth=0.5)
plt.axvline(failure.mean(), color='k', linestyle='dashed', linewidth=0.5)
min_ylim, max_ylim = plt.ylim()
plt.text(success.mean()-0.55, max_ylim*0.95, str(round(success.mean(), 2)))
plt.text(failure.mean()-0.57, max_ylim*0.95, str(round(failure.mean(), 2)))

# Post-Attempt Numbers
plt.axvline(15+35/60, color='r', linestyle='dashed', linewidth=0.5)
plt.text(15+35/60+0.05, max_ylim*0.95, 'Actual: ' + str(round(15+35/60, 2)), color='r'
```

Out[32]:
```
Text(15.633333333333335, 2.82123415730369, 'Actual: 15.58')
```


Total Time Elapsed at Ashburn Station
Broken Down by whether all Stations are Logged within 20 Hours
Actual: 15.58
19.54
20.26
Success
Failure
Probability
0.0
0.5
1.0
1.5
2.0
2.5
16
17
18
19
20
21
Time (Hours)

In [ ]: