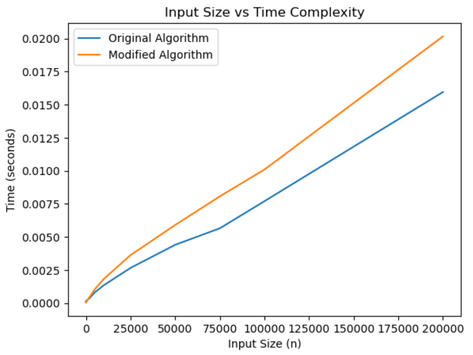
Data Analysis and Algorithms Assignment 2 - Surya , Ramnaresh and ARAVINDH

(1) Suppose that we were to rewrite the for loop header in line 10 of the Counting sort as 10 for j = 1 to A.length. Write a program, so that the algorithm still works properly.

In [ ]:
```
import time
import matplotlib.pyplot as plt
import random
random.seed(10)

def Original_Counting_sort(A,B,k):
    C = [0]*(k+1)
    # print(C)
    for j in range(0,len(A)):
        C[A[j]] += 1
    # print(C)

    for i in range(1,k+1):
        C[i]= C[i]+C[i-1]
    # print(C)

    for j in range(len(A)-1,-1,-1):
        C[A[j]] = C[A[j]]-1
        B[C[A[j]]] = A[j]

def modified_counting_sort(A,B,k):
    C = [0] * (k + 1)

    for i in range(0, len(A)):
        C[A[i]] += 1
    # print(C)

    CU = 0
    for i in range(0, k + 1):
        temp = C[i]
        C[i] = CU
        CU += temp

    # print(C)
    for j in range(1, len(A) + 1):
        B[C[A[j - 1]]] = A[j - 1]
        C[A[j - 1]] += 1
    # print(C)


input_sizes = [5,10,25,50,75,100,250,500,750,1000,2000, 5000, 10000,25000, 50000,75000, 100000,200000]
time_original = []
time_modified = []

for  n in input_sizes:

    A = [random.randint(0,1000) for i in range(n)]
    B = [0]*len(A)
    k = max(A)
    start_time = time.time()
    Original_Counting_sort(A,B,k)
    time_original.append(time.time() - start_time)

    start_time = time.time()
    modified_counting_sort(A,B,k)
    time_modified.append(time.time()- start_time)



plt.plot(input_sizes, time_original, label='Original Algorithm')
plt.plot(input_sizes, time_modified, label='Modified Algorithm')
plt.xlabel('Input Size (n)')
plt.ylabel('Time (seconds)')
plt.title('Input Size vs Time Complexity')
plt.legend()
plt.show()
```

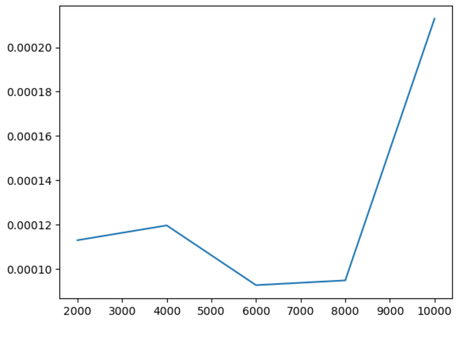


(2) John has to attend some conferences. There are N cities numbered from 1 to N and conferences can be held in any city. John lives in city1 and he will attend the conference as per schedule.Design and implement an algorithm with minimum time complexity that will find the shortest path from John's location to any conference's location

In [ ]:
```
import networkx as nx

N,M = map(int,input().split())
print("Number of vertices : ",N," Number of routes : ",M)
X = [0]* M # this is for the first vertex
Y = [0]* M # this is for the second vertex
W = [0]* M # this is for the weight
for i in range(M):
    X[i],Y[i],W[i] = map(int,input().split())

X_set = set(X)
Y_set = set(Y)
T = set.union(X_set,Y_set) # this is for the total number of vertices
T = list(T)

graph = nx.Graph()
for i in range(M):
    graph.add_edge(X[i],Y[i],w=W[i])

for i in range(N-1):
    shortest_path = nx.shortest_path(graph,source = T[0],target = T[i+1],weight = 'w')
    path_weight = sum(graph[shortest_path[j]][shortest_path[j + 1]]['w']
    for j in range(len(shortest_path) - 1))
    print(T[0]," ",T[i+1]," ",path_weight)
```

```
Number of vertices :  6  Number of routes :  8
1   2   1
1   3   3
1   4   3
1   5   2
1   6   4
```

In [ ]:
```
import networkx as nx
import time
import random
random.seed = 42
Num_cities = [2000,4000,6000,8000,10000]
Num_paths = [4000,8000,12000,16000,20000]
time_counts = []

for m in Num_paths:
    X = [random.randint(0,2000) for i in range(m)]
    Y = [random.randint(0,2000) for i in range(m)]
    W = [random.randint(0,2000) for i in range(m)]



    X_set = set(X)
    Y_set = set(Y)
    T = set.union(X_set,Y_set) # this is for the total number of vertices
    T = list(T)
    # print(T)
    # print(X)
    # print(Y)
    start_time = time.time()
    graph = nx.Graph()
    for i in range(M):
        graph.add_edge(X[i],Y[i],w=W[i])
        shortest_path = nx.shortest_path(graph,source = X[0],target = Y[0],weight = 'w')
        path_weight = sum(graph[shortest_path[j]][shortest_path[j + 1]]['w']
        for j in range(len(shortest_path) - 1))
        # print(T[0]," ",T[i+1]," ",path_weight)
    end_time = time.time()
    time_counts.append(end_time - start_time)

plt.plot(Num_cities,time_counts)
plt.show()
```



Question 2 - Drunken-Donuts

Problem Statement

Drunken Donuts, a new wine-and-donuts restaurant chain, wants to build restaurants on many street corners with the goal of maximizing their total profit. The street network is described as an undirected graph G = (V, E), where the potential restaurant sites are the vertices of the graph. Each vertex u has a non-negative integer value pu, which describes the potential profit of site u. Two restaurants cannot be built on adjacent vertices (to avoid self competition). You are supposed to design an algorithm that outputs the chosen set U ⊂ V of sites that maximizes the total profit $\sum_{u \in U} p_u$. First, for parts (a)–(c), suppose that the street network G is acyclic, i.e., a tree.

(a) Consider the following "greedy" restaurant-placement algorithm: Choose the highest-profit vertex u0 in the tree (breaking ties according to some order on vertex names) and put it into U. Remove u0 from further consideration, along with all of its neighbors in G. Repeat until no further vertices remain. Give a counterexample to show that this algorithm does not always give a restaurant placement with the maximum profit.

(b) Suppose that, in the absence of good market research, DD decides that all sites are equally good, so the goal is simply to design a restaurant placement with the largest number of locations. Give a simple greedy algorithm for this case, and prove its correctness.

Implementation

In [ ]:
```
def sorting_key(e):
    return e[1]  # e -> ('vertex', profit), function returns profit, used for sorting
```

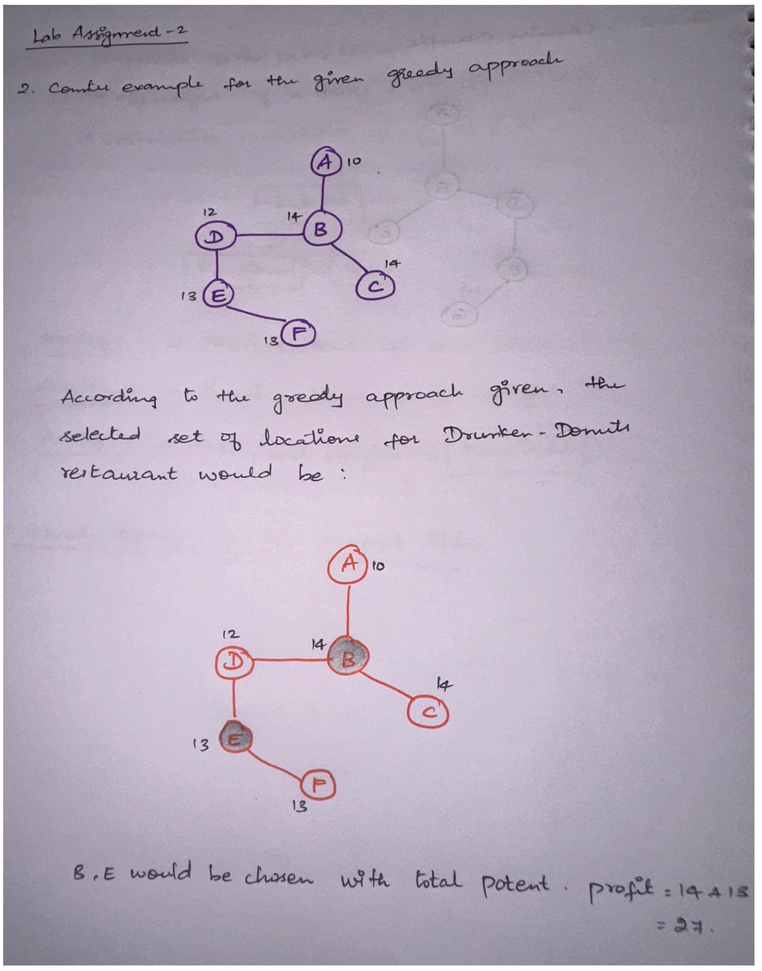
In [ ]:
```
class UndirectedGraph:
    def __init__(self):
        self.graph = {}
        self.vertex_profit = {}     # stores the potential profit of each vertex

    def add_vertices(self, vertex, potent_profit):
        self.graph[vertex] = []
        self.vertex_profit[vertex] = potent_profit

    def add_edge(self, vertex1, vertex2):
        # undirected graph implementation - 2 way edges
        self.graph[vertex1].append(vertex2)
        self.graph[vertex2].append(vertex1)

    def find_locations(self):  # solution for (a)
        u = []
        total_profit = 0
        # sorting in descending with key set to the potential profit unit
        list_vertices = list(self.vertex_profit.items())
        list_vertices.sort(key=sorting_key, reverse=True)

        while list_vertices:
            vertex, profit = list_vertices.pop(0)
            u.append((vertex, profit))
            total_profit += profit

            for neighbours in self.graph[vertex]:
                for i in list_vertices:
                    if i[0] == neighbours:
                        list_vertices.remove(i)
        print(u)
        print(total_profit)

    def find_locations_equal(self):  # solution for (b)
        u = []
        total_locations = 0
        list_vertices = []
        for _ in self.graph:
            list_vertices.append((_, len(self.graph[_])))
        # sorting in ascending with key set to no of neighbours
        list_vertices.sort(key=sorting_key)

        while list_vertices:
            vertex, no_neighbours = list_vertices.pop(0)
            u.append((vertex, self.vertex_profit[vertex]))
            total_locations += 1
            for neighbours in self.graph[vertex]:
                for i in list_vertices:
                    if i[0] == neighbours:
                        list_vertices.remove(i)
        print(u)
        print(total_locations)
```

In [ ]:
```
def main():
    street_net = UndirectedGraph()
    vertices = [('A', 10), ('B', 10), ('C', 10), ('D', 12), ('E', 10)]
    for i in vertices:
        street_net.add_vertices(i[0], i[1])

    edges = [('A', 'B'), ('A', 'C'), ('C', 'E'), ('C', 'D')]
    for _ in edges:
        street_net.add_edge(_[0], _[1])

    street_net.find_locations()
    print()
    street_net.find_locations_equal()
```

In [ ]:
```
main()
```

```
[('D', 12), ('A', 10), ('E', 10)]
32

[('B', 10), ('D', 12), ('E', 10)]
3
```

Counter Example



Optimal Solution