lab01

September 4, 2019

1 Lab 01: Algorithm Design and Analysis

ISC 4221 Due September 11th, 2019 Connor Poetzinger

1.1 1. Selection Sort

Implementation of Selection Sort algorithm. This algorithm takes in an array of n real numbers to be sorted and outputs the the sorted array in ascending order as well as an index vector.

In [1]:
```
#Import modules
import numpy as np
"""
Input: Numpy array of random real numbers from 0 to 100
Output: Sorted array and index vector

Goal: Take current element and swap it with the smallest element on its right
"""

def selectionSort(A):
    #use numpy argsort to return indicies that would sort the array
    indx = np.argsort(A)
    #Traverse through numpy array
    for i in range(len(A)):
        #initial minimum location
        min_loc = i
        #find location of smallest element on right
        for j in range(i + 1, len(A)):
            #Check if number to the right is smaller then minimum location
            if A[j] < A[min_loc]:
                min_loc = j
        #Within the first for loop swap the minimum location with first element
        A[min_loc], A[i] = A[i], A[min_loc]

    return A, indx
```

In [2]:
```
A = np.random.rand(25) * 100
print("Unsorted list or random floats from 0-100\n\n", A)
```

```
Unsorted list or random floats from 0-100

 [84.51072645 75.41110101 29.89915731 69.86006121 88.50108385 93.57665455
 16.27665239 97.26734289 33.4728781   6.77392892 77.19066636 63.13768262
 14.43479252 52.36400697 55.67741168 47.0662365  25.8872446  81.14967202
 20.80098299 18.86218278 46.80263541 79.34956518 53.94922409 60.26605507
 22.26789957]
```

```
In [3]: B, indxd = selectionSort(A)
        print("Sorted list of random floats from 0-100 and its position indicies\n\n", B, "\n\n
```

```
Sorted list of random floats from 0-100 and its position indicies

 [ 6.77392892 14.43479252 16.27665239 18.86218278 20.80098299 22.26789957
 25.8872446  29.89915731 33.4728781  46.80263541 47.0662365  52.36400697
 53.94922409 55.67741168 60.26605507 63.13768262 69.86006121 75.41110101
 77.19066636 79.34956518 81.14967202 84.51072645 88.50108385 93.57665455
 97.26734289]

 [ 9 12  6 19 18 24 16  2  8 20 15 13 22 14 23 11  3  1 10 21 17  0  4  5
  7]
```

1.2 2. Bubble Sort

Implementation of Bubble Sort algorithm. This algorithm takes in an array of n real numbers to be sorted and outputs the the sorted array in ascending order as well as an index vector.

```
In [4]: #Import modules
        import numpy as np
        """
        Input: Numpy array of random real numbers from 0 to 100
        Output: Sorted array and index vector

        Goal: Move left to right, compare consedcutive elements and switch them if
        they are out of order. Continue until no swaps are made through an entire
        sweep.
        """

        def bubbleSort(A):
            #use numpy argsort to return indicies that would sort the array
            indx = np.argsort(A)
            #Traverse the numpy array
            for i in range(len(A)):
                #At each sweep compare the current j with the next value
                #Use length minus 1 since we are comparing the current value
                #with the next
                for j in range((len(A) - 1) - i):
```

```
                    #Swap positions if element found is greater than the next
                    #element. Largest nums bubble to the back
                    if A[j] > A[j + 1]:
                        #Current element moves to the back
                        A[j], A[j + 1] = A[j + 1], A[j]

            return A, indx
```

```
In [5]: A = np.random.rand(25) * 100
        print("Unsorted list or random floats from 0-100\n\n", A)
```

```
Unsorted list or random floats from 0-100

 [33.40713158 43.88588858 35.91324338 92.57478295  1.83854881 68.94500976
 14.97275798 93.35713468 31.20660497  9.46817711 45.76029049 64.69395618
 47.18885866 73.34510341 17.87854299 85.53554301 89.22555663 21.45257256
 46.64392853 50.22647229 65.91640387  2.77992469 89.28895248 81.66166671
 70.42408651]
```

```
In [6]: B, indxd = bubbleSort(A)
        print("Sorted list of random floats from 0-100 and its position indicies\n\n", B, "\n\n
```

```
Sorted list of random floats from 0-100 and its position indicies

 [ 1.83854881  2.77992469  9.46817711 14.97275798 17.87854299 21.45257256
 31.20660497 33.40713158 35.91324338 43.88588858 45.76029049 46.64392853
 47.18885866 50.22647229 64.69395618 65.91640387 68.94500976 70.42408651
 73.34510341 81.66166671 85.53554301 89.22555663 89.28895248 92.57478295
 93.35713468]

 [ 4 21  9  6 14 17  8  0  2  1 10 18 12 19 11 20  5 24 13 23 15 16 22  3
  7]
```

1.3 3. A Basic Application of Sorting

Using the haversine algorithm, find the distance between the store distance and the user inputed longitude and latitude.

```
In [7]: import numpy as np
        import pandas as pd

        def haversin(data):

            """
            This function reads in user latitude and logitude to simulate logging customer
            call locations. The user's long and lat are then converted to radians along
            with other pre-defined longs and lats imported from a text file and save to a
```

```
    pandas dataframe. The function then computes the distance from the caller to
    the the stores in the dataframe using the haversine formula. The distances are
    then sorted in ascending order and printed out to provide the customer with
    a list of closest stores, and how many miles to the store.
    """

    #Read in user long and lat
    #must transform string values to float
    lon1 = float(input("Enter longitude: "))
    lat1 = float(input("Enter latitude: "))

    #Radius of earth from the equator in miles (found on google)
    R = 3963.0

    #assign user lat and lon and datatable lat and long to variables
    #I use the in-built function map to assign the numpy function np.radians to the deg
    #lats and long to transform degree into radians
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, data.latitude, data.longitude

    #calculate the distance between the lats and longs
    lon_dist = lon2 - lon1
    lat_dist = lat2 - lat1

    #apply haversine formula
    #np cos and sin provide for faster calculations
    c = 2 * R * np.arcsin(np.sqrt((np.sin(lat_dist)/2)**2 + np.cos(lat1) * np.cos(lat2)

    #prompt user for the info provided
    print("\nBelow are the closest stores from your location in ascending order\n")

    #apply selection sort
    sorted_result, indx = selectionSort(c)

    return sorted_result, indx

In [8]: #import data table
        #use pandas to assign columns using strings
        data = pd.read_table('stores_location.dat', delim_whitespace=True, names = ('store', '

        #call function
        sorted_dist , indx = haversin(data)

        #Sort the cities and stores based on the index from selectionSort(c)
        sorted_cities = [data['city'][indx[i]] for i in range(len(data))]
        sorted_stores = [data['store'][indx[i]] for i in range(len(data))]

        #create a tuple to package store num, city, and dist together
        sorted_zip = list(zip(sorted_stores, sorted_cities, sorted_dist))
```

```
        #Create output dataframe
        final_sort = pd.DataFrame(sorted_zip, columns=['Store','City','Distance(m)'])
        print(final_sort.to_string(index=False))

Enter longitude: 82
Enter latitude: 29

Below are the closest stores from your location in ascending order

Store          City  Distance(m)
store#2   Gainesville    49.170962
store#6       Orlando    58.666064
store#5         Tampa    77.023368
store#4  Jacksonville    94.676263
store#1   Tallahassee   168.646953
store#7       Hialeah   241.000734
store#3         Miami   248.602665
```