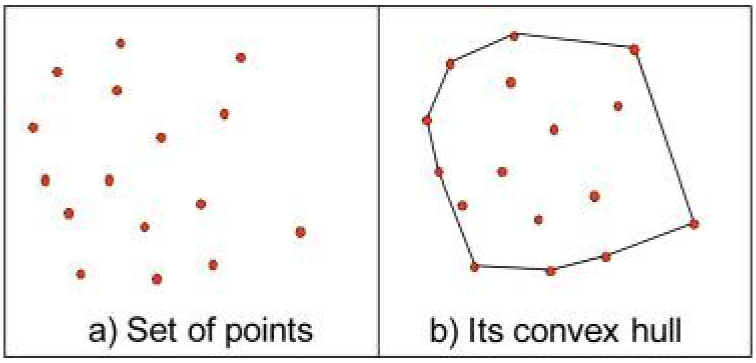
Convex Hull Algorithms Comparison

Student info: Biraaj Rout (UTA ID:1002071886)

Motivation:

This project will tackle various convex hull problems using multiple algorithms such as Grahams scan, and Jarvis March.

Algorithms and Explanation:

1. Definitions:
    a. Convex Polygon:
        i. A bounded polygon where all its interior angles are less than 180 degrees.
            1. 

    b. Convex Hull:
        i. The convex hull of a set Q of points, denoted by CH(Q), is the smallest convex polygon p for which each point in Q is either on the boundary of p or in its interior. [1]
2. Grahams Scan:
    a. Definitions:
        i. Graham's scan solves the convex-hull problem by maintaining a stack S of candidate points. It pushes each point of the input set Q onto the stack one time, and it eventually pops from the stack each point that is not a vertex of CH(Q). When the algorithm terminates, stack S contains exactly the vertices of CH(Q), in counterclockwise order of their appearance on the boundary. [1]

    b. Pseudocode [1]:
        i. GRAHAMSCAN(D)

```
1. let point0 be the point in D with the minimum y-coordinate or
   the leftmost such point in case of a tie. #O(N)
```

```
2. let (p1, p2, …, pm〉 be the remaining points in D, sorted by the
   polar angle in counterclockwise order around p0 (if more than
   one point has the same angle, remove all but the one that is
   farthest from p0) #(NlogN)
3. if modified_array < 2
4.        return []
5. else let _stack be an empty stack
6.    push(point0, _stack)
7.    push(point1, _stack)
8.    push(point2, _stack)
9.     for k = 3 to modified_array_size   #O(N)
10.        while the angle formed by points NEXT-TO-TOP(S),
   TOP(S), and pointk doesn't create convex angle
11.            POP(_stack)
12.      PUSH(pointk, _stack)
13.    return _stack
```

c. Time Complexity:
   i. Looking at each iteration of Psedocode the time complexity is **O(N log N)** as sorting has the highest computation.

3. Jarvis March:

a. Definitions:
   i. Jarvis's march is an algorithm that solves the convex hull problem using package wrapping. The algorithms runs in O(NH) where h is the number of vertices of a set D.

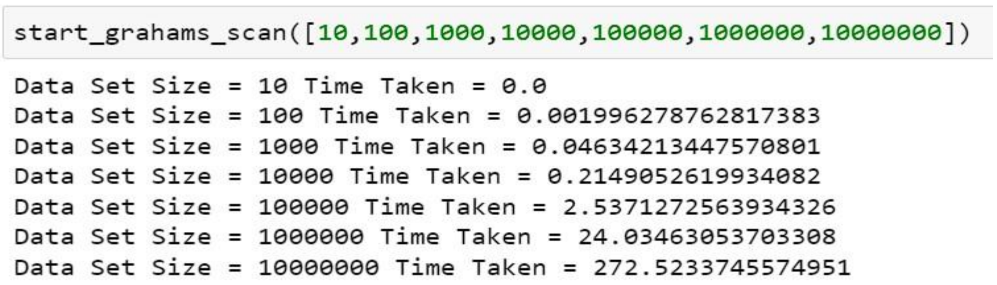
b. Pseudocode [3]:

```
i. Initialize g as the leftmost point
ii. While(Till the starting point is reached) O(h)
    1. The next point m is the point, such that the orientation of (g, m,
       i) is counterclockwise for any other point i.  O(n)
        a. If i is more counterclockwise then we update m to i.
    2.  The final value m is going to be the most counterclockwise
       point.
    3. g = m for the next iteration
```

c. Time Complexity: **O(NH)**
   i. As we check vertices connecting the current point in each iteration of m the time complexity comes to **O(NH).**

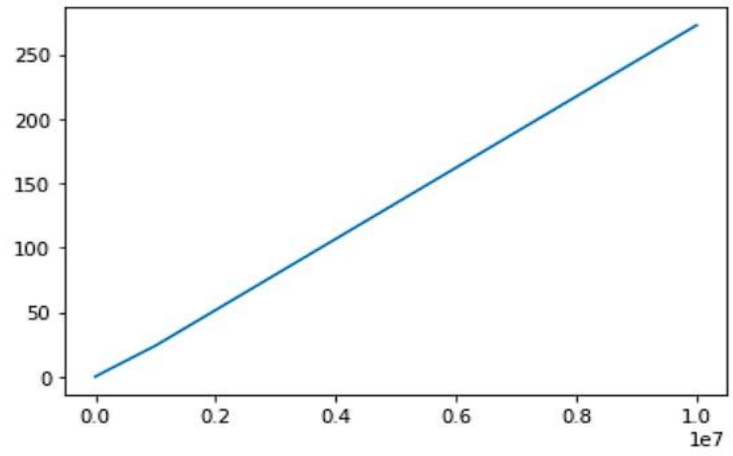
4. Experiment:

a. Grahams Scan:



```
start_grahams_scan([10,100,1000,10000,100000,1000000,10000000])
```

```
Data Set Size = 10 Time Taken = 0.0
Data Set Size = 100 Time Taken = 0.001996278762817383
Data Set Size = 1000 Time Taken = 0.04634213447570801
Data Set Size = 10000 Time Taken = 0.2149052619934082
Data Set Size = 100000 Time Taken = 2.5371272563934326
Data Set Size = 1000000 Time Taken = 24.03463053703308
Data Set Size = 10000000 Time Taken = 272.5233745574951
```

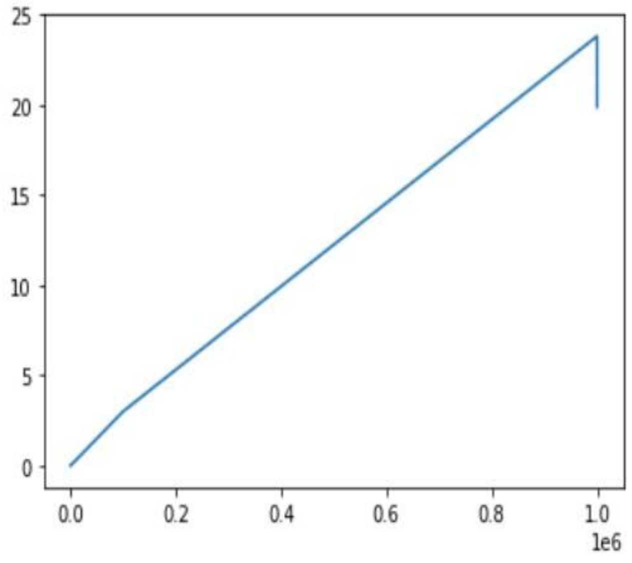


i.
ii. Here we took a dataset containing 10,100,1000,100000,1000000,10000000 points that were randomly generated using the NumPy library of python.
iii. In the above figure, the y-axis shows us time in seconds and the x-axis shows the input size.
iv. We can see the time increasing with the given input.
v. I performed multiple experiments with various random points the average time for 10000000 points came around 270 seconds and for 1000000 it came around 23 seconds.

b. Jarvis March

```
start_jarvis_march([10,100,1000,10000,100000,1000000,1000000])
```

```
Data Set Size = 10 Time Taken = 0.000133514404296875
Data Set Size = 100 Time Taken = 0.0016934871673583984
Data Set Size = 1000 Time Taken = 0.025521278381347656
Data Set Size = 10000 Time Taken = 0.29172372817993164
Data Set Size = 100000 Time Taken = 3.003873586654663
Data Set Size = 1000000 Time Taken = 23.810930252075195
Data Set Size = 1000000 Time Taken = 19.89277219772339
```



i.

ii. Time(in seconds) is shown in the y-axis and the x-axis shows the number of points given as input and the x-axis is in formal x*1e6.

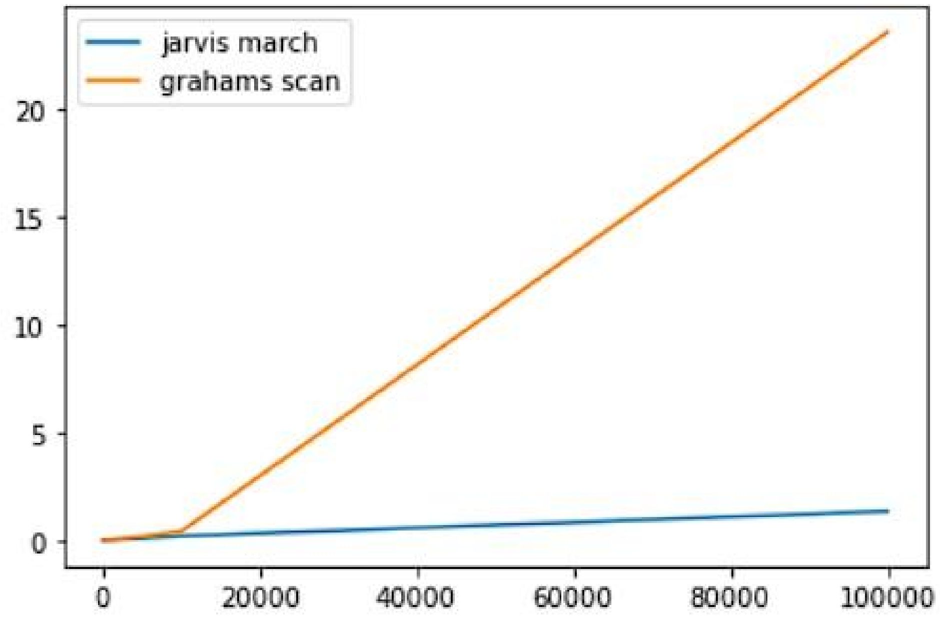
iii. Here we took multiple inputs of size [10,100,1000…1000000] we can see their time is quite less than Graham's scan the reason is due to the vertices being quite less which might be in the order of log(n) where n is the input size.

iv. As we do not perform any type of sorting here the time taken is very less.

5. Comparison:

```
comapare_algorithms([100,1000, 10000, 100000])
```

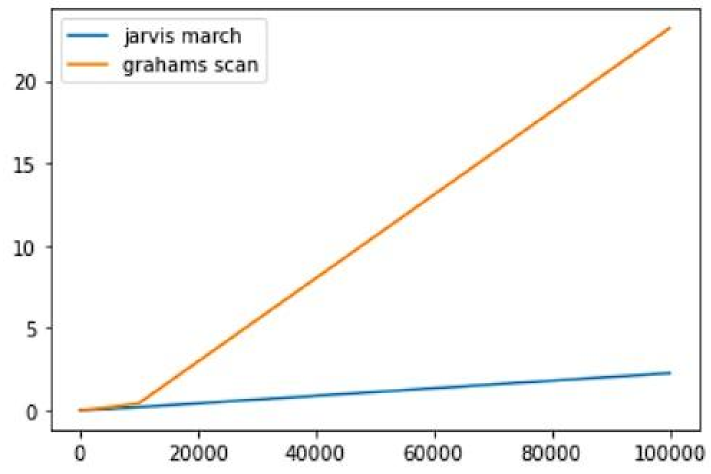
```
Jarvis March
Data Set Size = 100 Time Taken = 0.0009953975677490234
Grahams Scan
Data Set Size = 100 Time Taken = 0.0
Jarvis March
Data Set Size = 1000 Time Taken = 0.022259235382080078
Grahams Scan
Data Set Size = 1000 Time Taken = 0.015474557876586914
Jarvis March
Data Set Size = 10000 Time Taken = 0.19605278968811035
Grahams Scan
Data Set Size = 10000 Time Taken = 0.4085557460784912
Jarvis March
Data Set Size = 100000 Time Taken = 1.3477058410644531
Grahams Scan
Data Set Size = 100000 Time Taken = 23.604417085647583
```



a.

```
comapare_algorithms([100,1000, 10000, 100000])
```

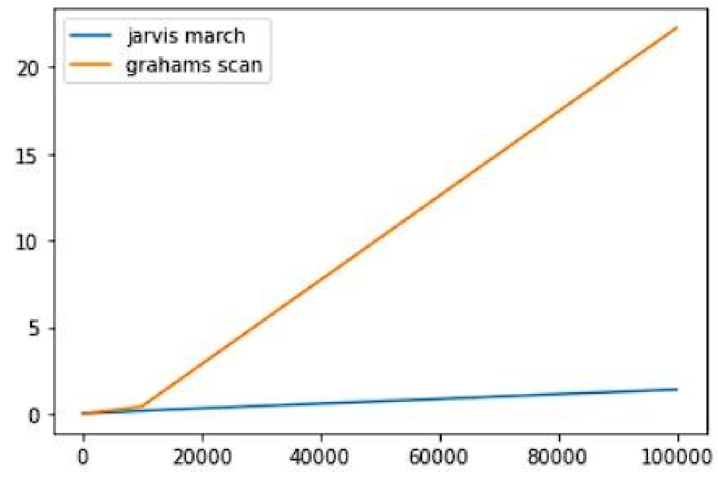
```
Jarvis March
Data Set Size = 100 Time Taken = 0.0010030269622802734
Grahams Scan
Data Set Size = 100 Time Taken = 0.0010035037994384766
Jarvis March
Data Set Size = 1000 Time Taken = 0.016545772552490234
Grahams Scan
Data Set Size = 1000 Time Taken = 0.017023086547851562
Jarvis March
Data Set Size = 10000 Time Taken = 0.18813443183898926
Grahams Scan
Data Set Size = 10000 Time Taken = 0.4152247905731201
Jarvis March
Data Set Size = 100000 Time Taken = 2.2523584365844727
Grahams Scan
Data Set Size = 100000 Time Taken = 23.20590853691101
```



b.

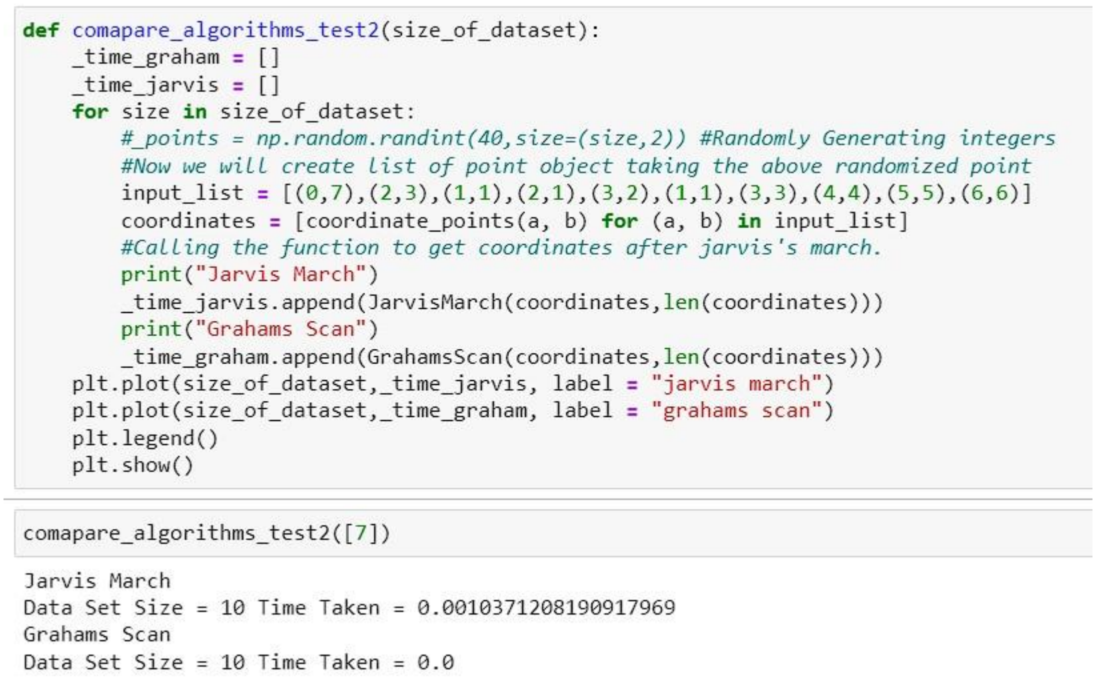
```
comapare_algorithms([100,1000, 10000, 100000])
```

```
Jarvis March
Data Set Size = 100 Time Taken = 0.0019881725311279297
Grahams Scan
Data Set Size = 100 Time Taken = 0.0010111331939697266
Jarvis March
Data Set Size = 1000 Time Taken = 0.014007091522216797
Grahams Scan
Data Set Size = 1000 Time Taken = 0.0160064697265625
Jarvis March
Data Set Size = 10000 Time Taken = 0.1550755500793457
Grahams Scan
Data Set Size = 10000 Time Taken = 0.4038684368133545
Jarvis March
Data Set Size = 100000 Time Taken = 1.3801705837249756
Grahams Scan
Data Set Size = 100000 Time Taken = 22.2816059589386
```



c.

d. In the above figures, we can see Jarvis march performs better than Graham's Scan as the input has vertices that are less than o(logN).



```
def comapare_algorithms_test2(size_of_dataset):
    _time_graham = []
    _time_jarvis = []
    for size in size_of_dataset:
        #_points = np.random.randint(40,size=(size,2)) #Randomly Generating integers
        #Now we will create list of point object taking the above randomized point
        input_list = [(0,7),(2,3),(1,1),(2,1),(3,2),(1,1),(3,3),(4,4),(5,5),(6,6)]
        coordinates = [coordinate_points(a, b) for (a, b) in input_list]
        #Calling the function to get coordinates after jarvis's march.
        print("Jarvis March")
        _time_jarvis.append(JarvisMarch(coordinates,len(coordinates)))
        print("Grahams Scan")
        _time_graham.append(GrahamsScan(coordinates,len(coordinates)))
    plt.plot(size_of_dataset,_time_jarvis, label = "jarvis march")
    plt.plot(size_of_dataset,_time_graham, label = "grahams scan")
    plt.legend()
    plt.show()
```

```
comapare_algorithms_test2([7])
```

```
Jarvis March
Data Set Size = 10 Time Taken = 0.0010371208190917969
Grahams Scan
Data Set Size = 10 Time Taken = 0.0
```

e.
f. In the above figure, we can see graham's algorithm performs better than Jarvis's because the points are quite close to each other, and the number of vertices on the outer edges is more. A similar dataset can be created to test the Grahams scan performance along with Jarvis's march.
g. In the future, the experiment can be done on a dense plane where vertices will be higher in number

6. Challenges:

a. Due to memory and time limitations, the experiment was performed on a maximum of 1000000 points which had their outer points spread out.

7. Learnings:

a. Even though the graham scan has a constant time complexity O(nlogn) Jarvis is faster as the number of vertices is less in most planes that are randomly generated.
b. Practically implementing Jarvis is simple compared to Graham scan as there are no sorting steps involved.

8. Future Work:

a. The Experiment should be done on a properly defined dataset where the vertices are high which will result in a graham scan performing well.

References:

[1] Introduction to Algorithms, Third Edition - Books24x7(LINK)
[2] https://blog.devgenius.io/grahams-scan-visually-explained-be54b712e2ba
[3] https://www.geeksforgeeks.org/convex-hull-using-jarvis-algorithm-or-wrapping/