HW2

Zhengmao Zhang

Problem 1: Jarvis March (Gift Wrapping Algorithm)

(a) [10 points] Give pseudocode describing the Jarvis March algorithm, a brief description of how it works, and explain its best and worst case efficiency.

Answer:

```
jarvis(S)
   // S is the set of points
   pointOnHull = leftmost point in S //which is guaranteed to be part of
the Hull(S)
   i = 0
   repeat
      P[i] = pointOnHull
      endpoint = S[0]      // initial endpoint for a candidate edge on the
hull
      for j from 1 to |S|
         if (endpoint == pointOnHull) or (S[j] is on left of line from
P[i] to endpoint)
            endpoint = S[j]   // found greater left turn, update endpoint
      i = i+1
      pointOnHull = endpoint
   until endpoint == P[0]      // wrapped around to first hull point
```

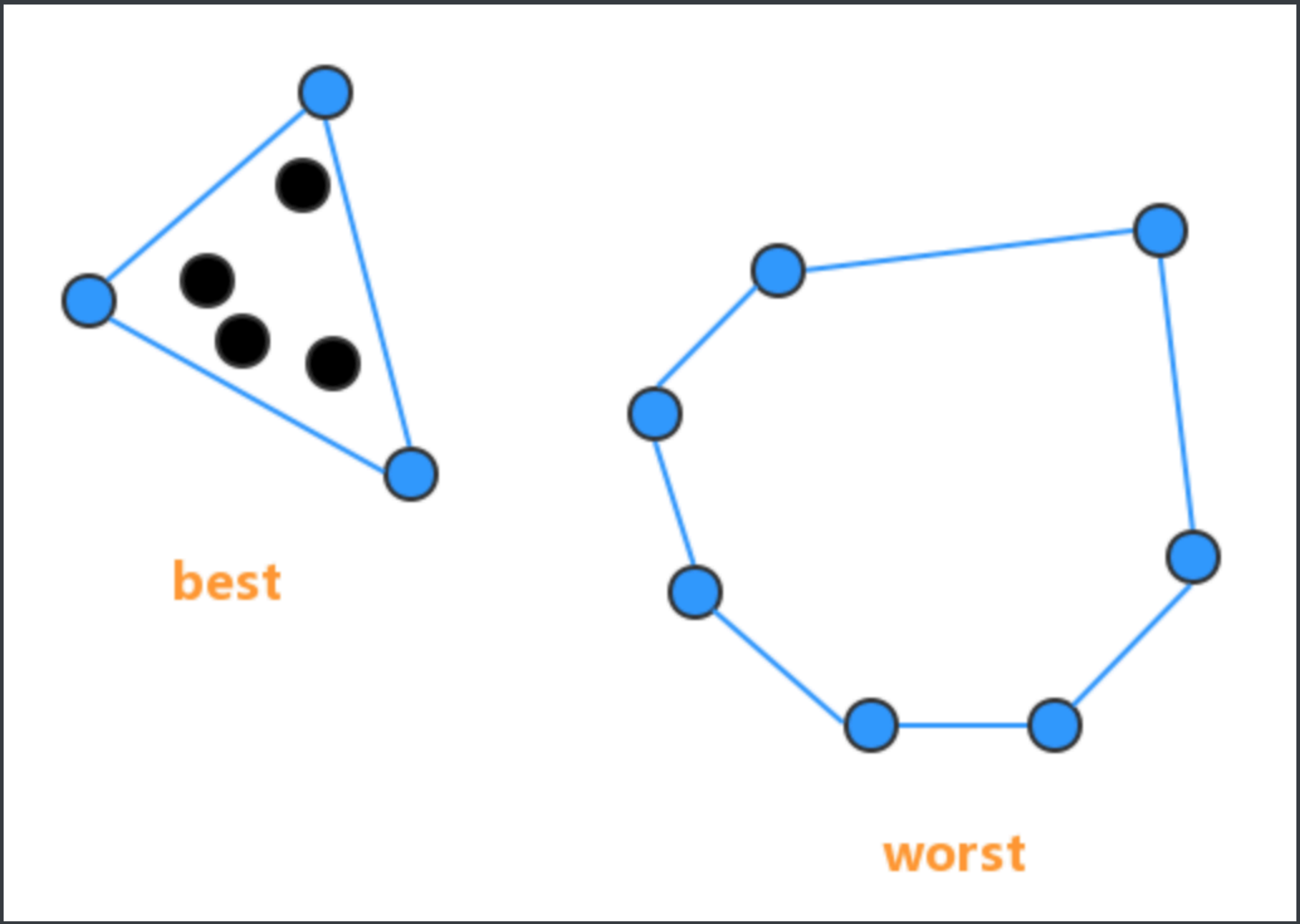
TIME COMPEXITY O(NlogN)

Worst Case **Θ(n^2)**

Best case **O(NlogN)**

for the **best case**, its all points are in the result hall. and all next point is the next point need append.

for the **worest case**, its all point that need join in, is the last element of input set.



(b) [5] points Give an example input on which Jarvis March will perform significantly better than Graham's scan and explain why it will perform better.

```
algorithm Grahamscan(S) -> res is
  // S is the set of points S is S[x][y] set
  // res is the opint
  res = []
```

```
  n = S.size() // n is number of all points

  // in this part we need find the lowest point, OR
  // highest point
  lowestPoint = S[0] //its start point
  for s in all S do
    lowestPoint = lowerPoint(lowestPoint, s)

  // this part will triversal all point set
  firstPoint = lowestPoint
  secondPoint = NULL
  repeat:
    res.push(firstPoint)
    secondPoint = S[firstPoint.Next]
    for s in S do
      if counterClockWise(firstPoint, s, secondPoint) do
        secondPoint = s
    firstPoint = secondPoint
 until firstPoint == lowestPoint
// this function is to test if the point s is in the line (that point1 and
point2 connect) left
function counterClockWise(point1, point s, point2) -> bool do
  return ((point2.x - point1.x) * (s.y - point1.y) - (point2.y - point1.y)
* (c.x - point1.x)) < 0
```

(c) [5] points Give an example input on which Graham's Scan will perform significantly better than Jarvis March and explain why it will perform better.

Problem 2: Find the Missing Number

(a) [5 points] Give an efficient algorithm for finding the missing number, show its complexity, and argue its correctness. (You should try for O(n)-time, less efficient solutions will still get partial credit)

Answer:

if its a sorted list, we can just do a compare function to make a[i] compare with i.

```
dataSet = []
algorithm getMissingNumber() is:
  // n is the size of arr DataSet
  n = dataSet.size
  for i from 0 to n - 1 do
    if dataSet[i] is not i then
      // attention: should return i, its not dataSet[i], cuz dataSet i is
exist
      return i
```

In this algorithm, its ofc **O(n)** cuz its only one loop from 0 to n, so its **O(n)**

if its a **unsorted** list, we need get the sum of range n - 1

```
dataSet = []
algorithm getMissingNumber() is:
  // n is the size of arr dataSet
  n = dataSet.size
  sum = 0
  for i in range(0, n - 1) do
    sum += i
  for each dates in dataSet do
    sum -= dates
  return sum
```

Its **O(n)**, cuz the get sum part is **O(n)**, and the search part is **O(n)**, so its **O(n)**

(b) [10 points] For this question you are not allowed to access an entire integer with a single operation. The elements of the list are represented in binary, and the only operation you can use to access them is GetBinaryDigit(A[i],j) which returns the jth bit of element A[i] which runs in constant time. Give an efficient algorithm for finding the missing number under these constraints, show its complexity, and argue its correctness. (You should try for O(n)-time and O(log n)-space, less efficient solutions will still get partial credit)

Example: If we run GetBinaryDigit(A[i],j) with A[i] = 29 and j = 2, it would return a 0 since 29 = 11101.

Answer: Use XOR operator.

```
dataSet = []
algorithm getMissingNumber() is:
  // n is the size of arr dataSet
  n = dataSet.size

  x1 = dataSet[0]
  x2 = 1
  for i in range(0, n - 2) do
    x1 = x1 XOR dataSet[i]
  for i in range(1, n - 1) do
    x2 = x2 XOR dataSet[i]
  return x1 XOR x2
```

cuz XOR can delete repeat number.

> like x1 = x2 XOR x3
>
> if we wanna get the x2, we can just x1 XOR x3

for complexity, TIME is O(n), NSPACE is O(1). Cuz for the 1st XOR loop, its O(n), the 2nd same. So its O(n)