Part B Checking for Convexity

According to the definition of convex ,we can conclude that if there is a **segment between 2 points**, an all the other points are **on the same side** of segment, then these 2 points are the **points of convex polygon**.

The isConvex Algorithm check whether **all points** p_i on polygon satisfied the condition that in the **counter-clockwise order**, p_i is on the **left hand side** of $\overrightarrow{p_{i-2}p_{i-1}}$. So that for **any vector** $\overrightarrow{p_ip_{i+1}}$ vector, **all the other points is on its left**.

The convex polygon confirmed by this algorithm is satisfied the definition of convex.So the algorithm will work.

Part D Deque Analysis

I used a double linked list to implement deque, and there are two struct Node and Deque.
The **Node structure** contain:

1) A **Point variable** to store the Point's x and y coordinate
2) A **Node type pointer 'next'** point to the next Node it connected
3) A **Node type pointer 'prev'** point to the previous Node that point to it

The **Deque structure** contain:

1) A **Node type pointer 'head'** point to the top of deque
2) A **Node type pointer 'tail'** point to the bottom of deque
3) A **Int variable 'size'** to record the number of Points in deque

1. I choose double linked list as the underlying data structure

1) I choose linked list because, in this task, **It only need to insert and delete first and last Node of list.** It is **very fast for linked list** to implement these operation. The size of the input is **not fixed**, So need to use linked list instead of array. At the same time, this task do not need to search the ith element, so that it perfectly **avoid the downside of the linked list**.

2) I choose double linked list because, in this task, I need to **read the second last Node in list (Have to read the data in reverse direction),** Use double linked list will **reduce retrieval time**.

2. I add an int variable variable 'size' to record the number of element that **store in the deque**, increase or decrease the **value of 'size'** when add or delete Points, so that It only takes **constant time** to get the size of deque.

PUSH(deque, point)	**INSERT(deque, point)**	**POP(deque)**	**REMOVE(deque)**
Create new node ∈O(1)	Create new Node ∈O(1)	Get Point information from deque head node∈O(1)	Get Point information from deque tail node∈O(1)
Store Point in node ∈O(1)	Store Point in Node ∈O(1)	Check last Node in deque? ∈O(1)	Check last Node in deque ?∈O(1)
Check whether deque is empty ∈O(1)	Check whether deque is empty ∈O(1)	If this is last element	If this is last element
If deque is empty	If deque is empty	Deque head and tail equal to NULL ∈O(1)	Deque head and tail equal to NULL ∈O(1)
Deque head and tail point to it ∈O(1)	Deque head and tail point to it ∈O(1)	If this is not last element	If this is not last element
If deque is not empty	If deque is not empty	Deque head point to its next node ∈O(1)	Deque tail point to its prev node ∈O(1)
Previous head node point to it ∈O(1)	Previous tail node point to it ∈O(1)	Make new head point not point to it ∈O(1)	Make new tail node not point to it ∈O(1)
Deque head point to it ∈O(1)	Deque tail point to it ∈O(1)	Free this node ∈O(1)	Free this node ∈O(1)
Deque size +1 ∈O(1)	Deque size +1∈O(1)	Deque size - 1 ∈O(1)	Deque size - 1 ∈O(1)
Time Complexity =O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1) ∈O(1)	Time Complexity =O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1) ∈O(1)	Time Complexity =O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1) ∈O(1)	Time Complexity =O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1) ∈O(1)

Part E Inside Hull Tracing

Polygon[A, B, C, D, E, F, G, H] , n=8
C = $_{\text{bottom}}$ <C, A, B, C> $_{\text{top}}$

i=3:
C = $_{\text{bottom}}$ <D, A, B, D> $_{\text{top}}$
i=4:
C = $_{\text{bottom}}$ <E, A, B, D, E> $_{\text{top}}$

i=5:
C = $_{\text{bottom}}$ <F, A, B, D, F> $_{\text{top}}$
i=6:
C = $_{\text{bottom}}$ <G, A, B, D, G> $_{\text{top}}$
i=7:
C = $_{\text{bottom}}$ <H, B, D, G, H> $_{\text{top}}$
Output: B, D, G, H

Part G Inside Hull Analysis

Function used in InsideHull

```
Orientation(p1, p2, p3)  ∈O(1)
Calculate the check value(position relationship)  ∈O(1)
Check it's positive, negative or 0  ∈O(1)
Return result  ∈O(1)
Time Complexity =O(1)+O(1)+O(1)  ∈O(1)

InsideHull(Polygon, n, hull)
Check the validation of the input  ∈O(1)
Create new deque  ∈O(1)
Create counter i <--- 3  ∈O(1)
(Initialize the deque)
Check the position of first three point  ∈O(1)
Insert them in the deque in certain order  ∈O(1)
(check whether the points in polygon is on convex hull)
While i<n:
    currentPoint<---Find get next point in polygon  ∈O(1)
    Find ct, ct-1, cb, cb+1  ∈O(1)
    if(Orientation(ct-1, ct, currentpoint)==Left
            and Orientation(cb, cb+1, currentpoint) )∈O(1)
            i = i+1  ∈O(1)
            continue
    While(Orientation(ct-1, ct, currentpoint)==Right)
          Remove(ct)∈O(1)                                   1.O(n)
O(1)~O(n)
    Insert(currentPoint)∈O(1)
    While (Orientation(cb, cb+1, currentpoint)==Right)
          Pop(cb)∈O(1)
O(1)~O(n)
    Push (currentPoint)∈O(1)
    i = i+1  ∈O(1)
(the convex hull has been found, insert to hull array)
remove the duplicate point  ∈O(1)
Pop out all the point in the points in the deque  ∈O(n)
Insert into hull  ∈O(n)                                     2.O(n)
```

Total Time Complexity
=O(1)+O(1)+O(1)+O(1)+O(1)+O(n)+O(n)

Basic Operation:
1. Insert or remove point from deque

Explanation for two O(n) time complexity
1. It may seem that the time complexity in the loop is $O(n^2)$, because for every point it should goes back to check if it is on the right hand side of any previous vector(which consist of the previous two point).
While it is actually O(n), because for each point is considered at most twice.
   1) When point will being check as ct or ct-1 or cb or cb+1, result is left, it remain in the deque, and will not being check after that
   2) When point will being check as ct or ct-1 or cb or cb+1, result is right, it delete from the deque, and will never check again

So the time complexity is O(n)
2. The number of points in the final boundary of convex is smaller or equal to the number of points in the polygon
So it should take time O(n)

Conclusion
No, they are not contradict. O(nlogn) time is to sort the input set first and only need to check on the right part, while our algorithm check both right and left part of the graph.actually they did the same work.