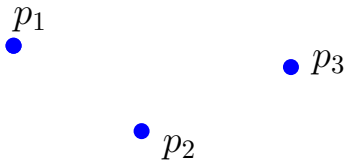
2D CH: In Class Exercises

Task 1: Implementing the orientation test

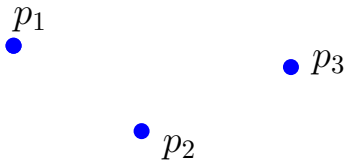
Basic formula:
$x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$



2D CH: In Class Exercises

Task 1: Implementing the orientation test

Basic formula:
$x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$



Write a simple program where point $p_i = (x_i, y_i)$ has the following coordinates:

$(x_0, y_0) = (0, 0)$

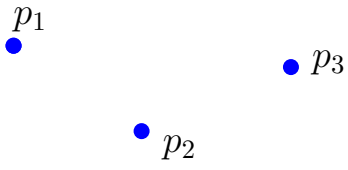
$y_i = x_i^2$

and for $i > 0$, $x_i = x_{i-1}+$ a random floating number between 0 and 1.

2D CH: In Class Exercises

Task 1: Implementing the orientation test

Basic formula:
$x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$



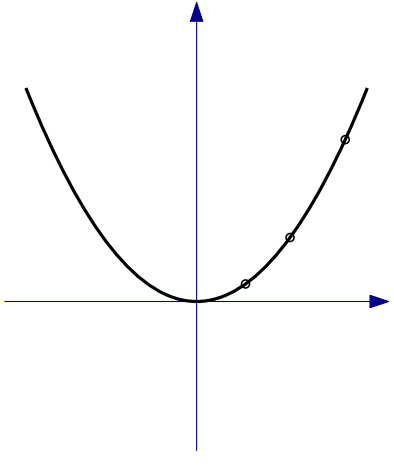
Write a simple program where point $p_i = (x_i, y_i)$ has the following coordinates:

$(x_0, y_0) = (0, 0)$

$y_i = x_i^2$

and for $i > 0$, $x_i = x_{i-1}+$ a random floating number between 0 and 1.

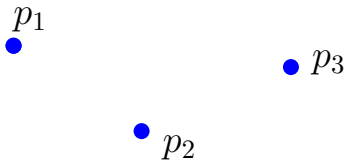
Now, p_i, p_{i+1}, and p_{i+2} always make a left turn.



2D CH: In Class Exercises

Task 1: Implementing the orientation test

Basic formula:
$x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$



Write a simple program where point $p_i = (x_i, y_i)$ has the following coordinates:

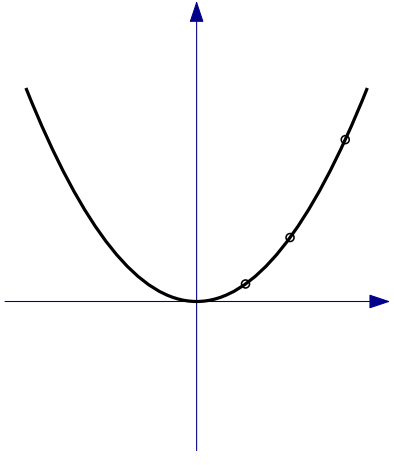
$(x_0, y_0) = (0, 0)$

$y_i = x_i^2$

and for $i > 0$, $x_i = x_{i-1}+$ a random floating number between 0 and 1.

Now, p_i, p_{i+1}, and p_{i+2} always make a left turn.

Check this for $i = 1, \ldots n-2$ and count how many times this holds, for different values of n.
Use 32bit floating point numbers for this test!



2D CH: In Class Exercises

Task 1: Implementing the orientation test

Basic formula:

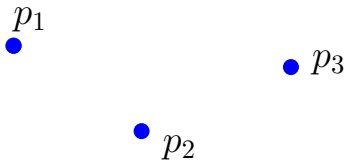
$x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$

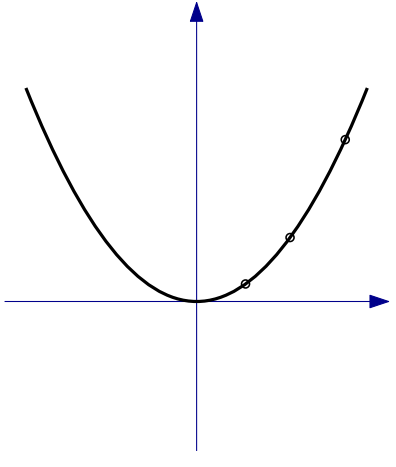
Task 2: Try with different ways of writing the same formula, for example:

(V2) $x_1y_2 - x_1y_3 + x_2y_3 - x_2y_1 + x_3y_1 - x_3y_2$

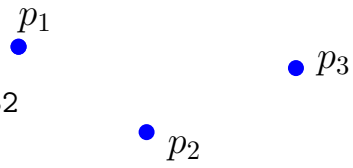
or

(V3) $x_1y_2 + x_2y_3 + x_3y_1 - x_1y_3 - x_2y_1 - x_3y_2$





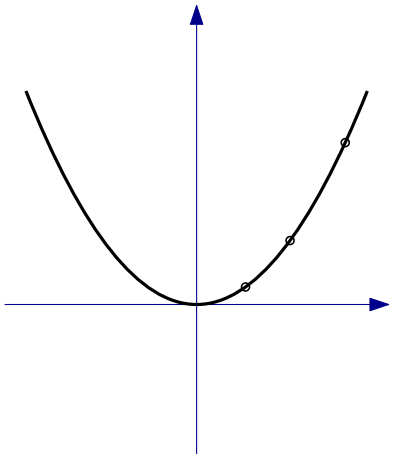
2D CH: In Class Exercises



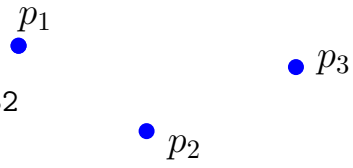
Task 3: The typical `float` data type is based on IEEE 754, e.g., the `binary32` format. Here, a floating point number f is represented as:
$f = m \cdot 2^x$
The standard allocates 24 bits to represent number m and 8 bits for x.

Question 1: If we have two floating points numbers $f_1 = m_1 \cdot 2^{x_1}$ and $f_2 = m_2 \cdot 2^{x_2}$, how many bits do we need to represent $f_1 \cdot f_2$ without error?

Question 2: How many bits do we need to represent $f_1 + f_2$ with no error?



2D CH: In Class Exercises



Task 3: The typical `float` data type is based on IEEE 754, e.g., the `binary32` format. Here, a floating point number f is represented as:
$f = m \cdot 2^x$
The standard allocates 24 bits to represent number m and 8 bits for x.

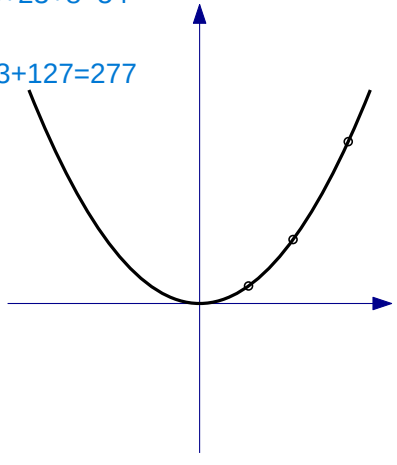
Question 1: If we have two floating points numbers $f_1 = m_1 \cdot 2^{x_1}$ and $f_2 = m_2 \cdot 2^{x_2}$, how many bits do we need to represent $f_1 \cdot f_2$ without error?

23+23+8=54

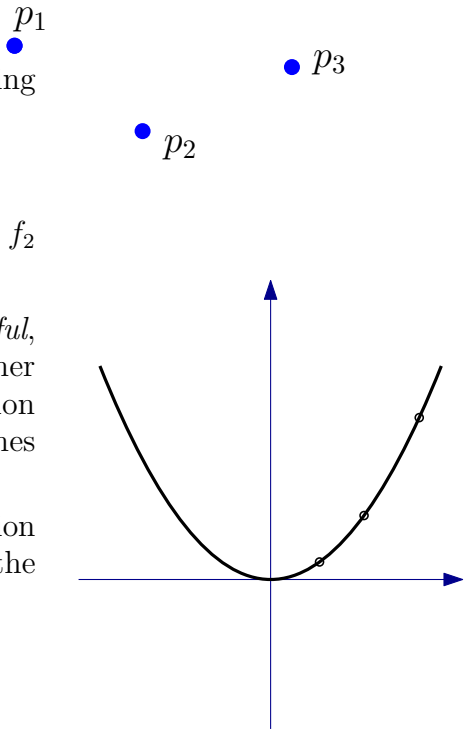
Question 2: How many bits do we need to represent $f_1 + f_2$ with no error?

127+23+127=277

Take-home Question: How difficut do you think is it to write a code that evaluates the basic formula without any errors?



2D CH: In Class Exercises

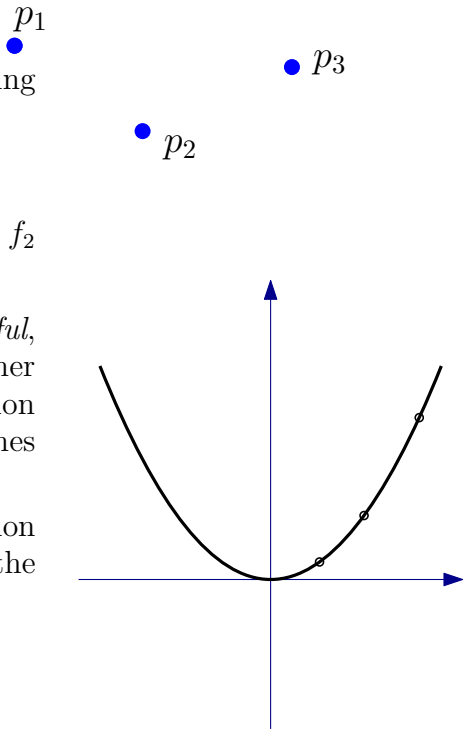


Question 3: Assume a floating point format, $f = m \times 2^x$, has k bits for representing m and x. Answer the following questions:

- What's the smallest and largest positive numbers we can represent?
- How many bits, asymptotically, do we need to represent $f_1 \times f_2$ or $f_1 + f_2$ exactly for two floating numbers f_1 and f_2?
- For $f = f_1 + f_2$, one can argue that at most $O(k)$ bits will be *meaningful*, meaning, the rest will be either a long sequence of all 0s or all 1s. In other words, one can show that the total number of changes in the representation of f is $O(k)$, where the number of changes is defined as the number of times the two consecutive bits of f are different.

  Come up with k floating point numbers $f_1, \ldots f_k$, and an arithmetic expression involving them that uses $O(k)$ additions and multiplications such that the result contains $\Omega(2^k)$ changes.

2D CH: In Class Exercises



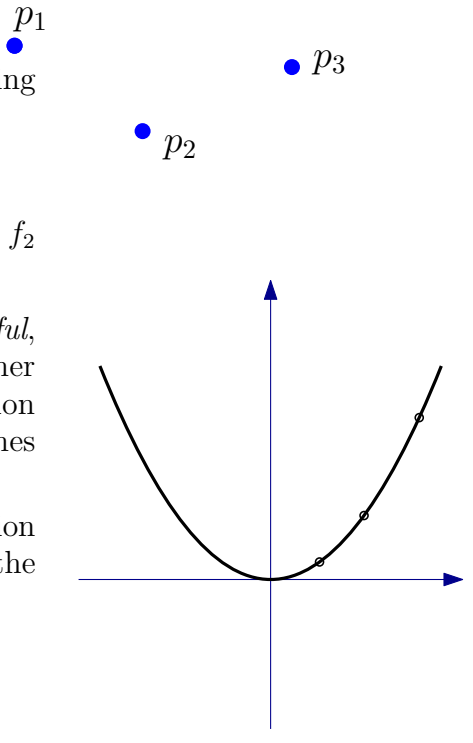
Question 3: Assume a floating point format, $f = m \times 2^x$, has k bits for representing m and x. Answer the following questions:

- What's the smallest and largest positive numbers we can represent?
- How many bits, asymptotically, do we need to represent $f_1 \times f_2$ or $f_1 + f_2$ exactly for two floating numbers f_1 and f_2?
- For $f = f_1 + f_2$, one can argue that at most $O(k)$ bits will be *meaningful*, meaning, the rest will be either a long sequence of all 0s or all 1s. In other words, one can show that the total number of changes in the representation of f is $O(k)$, where the number of changes is defined as the number of times the two consecutive bits of f are different.

  Come up with k floating point numbers $f_1, \ldots f_k$, and an arithmetic expression involving them that uses $O(k)$ additions and multiplications such that the result contains $\Omega(2^k)$ changes.

Hint 1 (white on white text; select to see):

2D CH: In Class Exercises



Question 3: Assume a floating point format, $f = m \times 2^x$, has k bits for representing m and x. Answer the following questions:

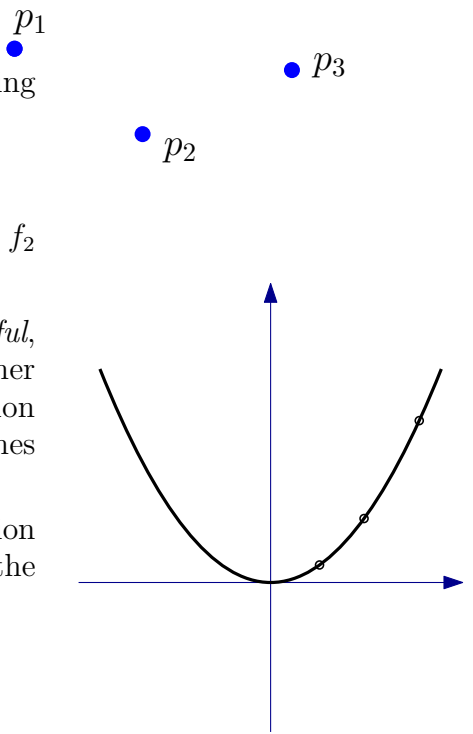
- What's the smallest and largest positive numbers we can represent?
- How many bits, asymptotically, do we need to represent $f_1 \times f_2$ or $f_1 + f_2$ exactly for two floating numbers f_1 and f_2?
- For $f = f_1 + f_2$, one can argue that at most $O(k)$ bits will be *meaningful*, meaning, the rest will be either a long sequence of all 0s or all 1s. In other words, one can show that the total number of changes in the representation of f is $O(k)$, where the number of changes is defined as the number of times the two consecutive bits of f are different.

  Come up with k floating point numbers $f_1, \ldots f_k$, and an arithmetic expression involving them that uses $O(k)$ additions and multiplications such that the result contains $\Omega(2^k)$ changes.

Hint 1 (white on white text; select to see):

Hint 2 (white on white text; select to see):

2D CH: In Class Exercises



Question 3: Assume a floating point format, $f = m \times 2^x$, has k bits for representing m and x. Answer the following questions:

- What's the smallest and largest positive numbers we can represent?
- How many bits, asymptotically, do we need to represent $f_1 \times f_2$ or $f_1 + f_2$ exactly for two floating numbers f_1 and f_2?
- For $f = f_1 + f_2$, one can argue that at most $O(k)$ bits will be *meaningful*, meaning, the rest will be either a long sequence of all 0s or all 1s. In other words, one can show that the total number of changes in the representation of f is $O(k)$, where the number of changes is defined as the number of times the two consecutive bits of f are different.

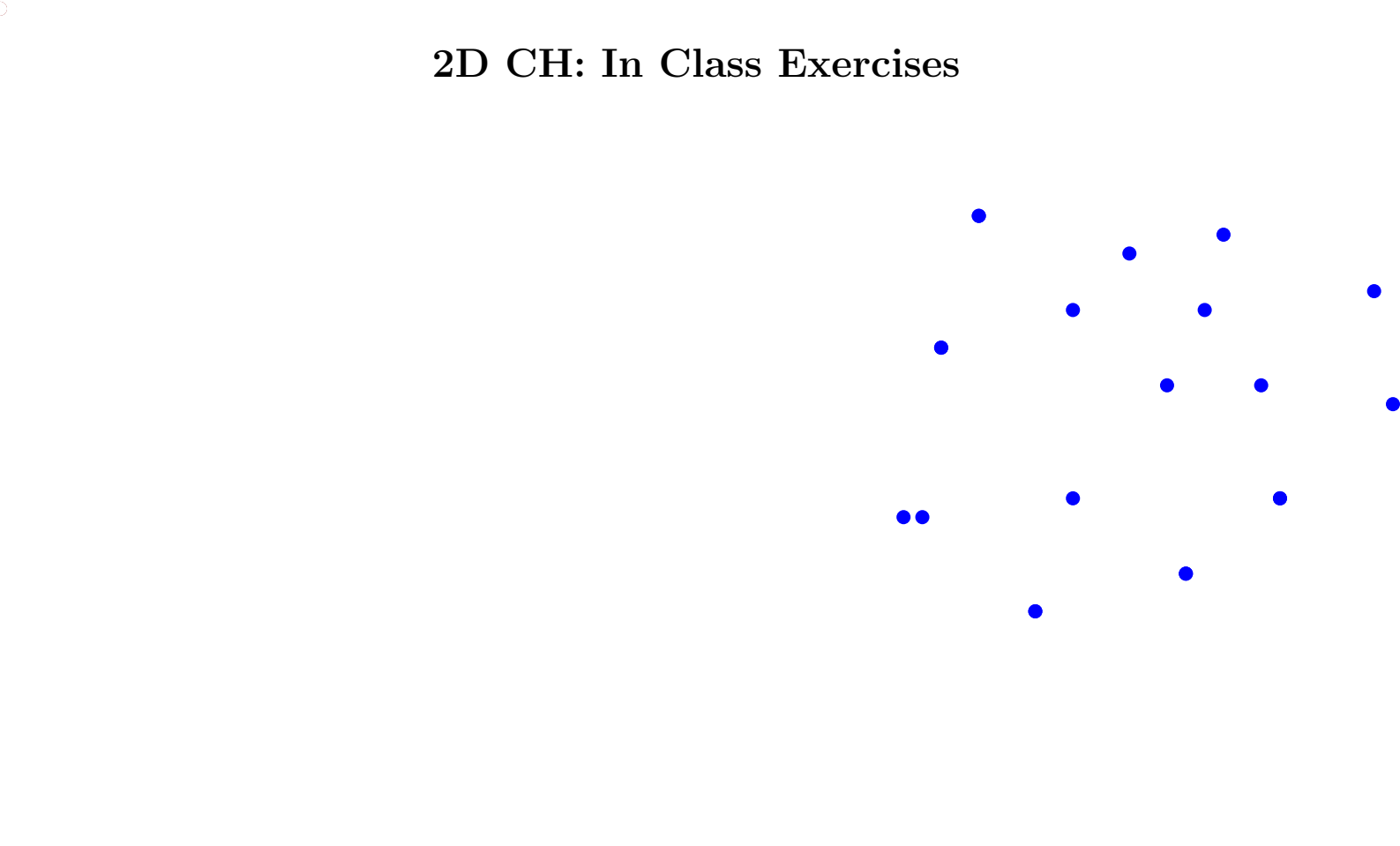
  Come up with k floating point numbers $f_1, \ldots f_k$, and an arithmetic expression involving them that uses $O(k)$ additions and multiplications such that the result contains $\Omega(2^k)$ changes.

Hint 1 (white on white text; select to see):

Hint 2 (white on white text; select to see):

Challenge question: Come up with a program that uses $O(k)$ additions and multiplications such that it computes a number whose exact representation has at least 2^{2^k} changes.

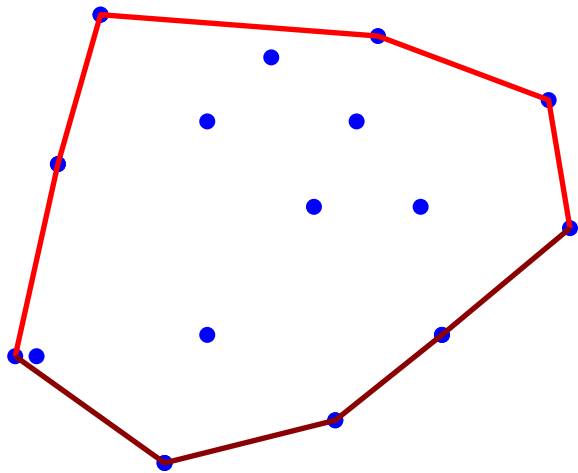
2D CH: In Class Exercises



2D CH: In Class Exercises

Implement Graham's Scan

1. Sort by x-coordinate
2. Initialize **UH** to empty; **UH**.add(p_1, p_2); s=2
3. For $i = 3 \rightarrow n$ do
   - 3.1. While s$\geq$2 & **UH**(s-1), **UH**(s), p_i make a Left Turn
     - 3.1.1. Remove **UH**(s); s--;
   - 3.2. **UH**.add(p_i); s++

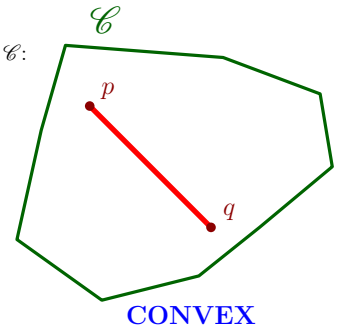


2D CH: In Class Exercises

Theory Question 1:

Prove that if $\mathscr{C}_1, \ldots, \mathscr{C}_n$ are convex, then $\mathscr{C} = \mathscr{C}_1 \cap \mathscr{C}_2 \ldots \cap \mathscr{C}_n$ is also convex.

$\mathscr{C}$ is convex if: for any two points $p, q \in \mathscr{C}$:
$\overline{pq}$ is inside $\mathscr{C}$

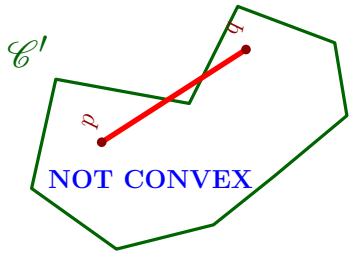


Hints:

We are going to practice with proofs, however, as a starting point, whenever you encounter a theoretical questions that asks for a proof try the following steps.

1. Understand the assumptions. You can use the definitions to fully understand what assumptions you are given.
2. Understand what you need to prove. Once again, you can use the available definitions to fully understand what you need to prove.

In this basic exercise, the two above steps are sufficient to solve the problem. In more complicated problems, more steps will be needed.

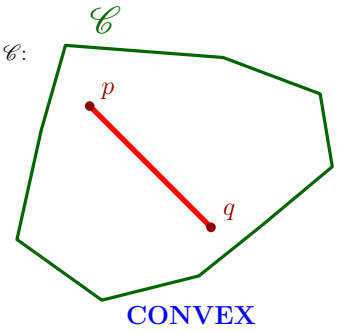


2D CH: In Class Exercises

Theory Question 1:

Prove that if $\mathscr{C}_1, \ldots, \mathscr{C}_n$ are convex, then $\mathscr{C} = \mathscr{C}_1 \cap \mathscr{C}_2 \ldots \cap \mathscr{C}_n$ is also convex.

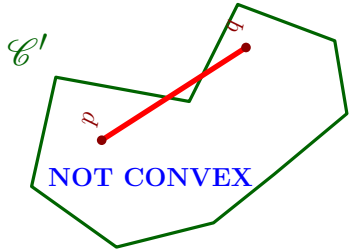
$\mathscr{C}$ is convex if: for any two points $p, q \in \mathscr{C}$:
$\overline{pq}$ is inside $\mathscr{C}$



Hints:
We are going to practice with proofs, however, as a starting point, whenever you encounter a theoretical questions that asks for a proof try the following steps.

1. Understand the assumptions. You can use the definitions to fully understand what assumptions you are given.
2. Understand what you need to prove. Once again, you can use the available definitions to fully understand what you need to prove.

In this basic exercise, the two above steps are sufficient to solve the problem. In more complicated problems, more steps will be needed.



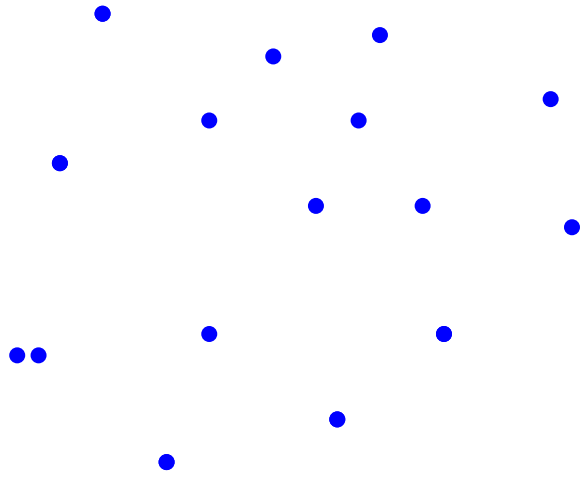
Theory Question 2:

Prove that if $\mathscr{C}_1$ and $\mathscr{C}_2$ are convex, then $\mathscr{C} = \mathscr{C}_1 \cup \mathscr{C}_2$ may not be convex.

Hint:
This type of claim can be proven using only a counterexample.

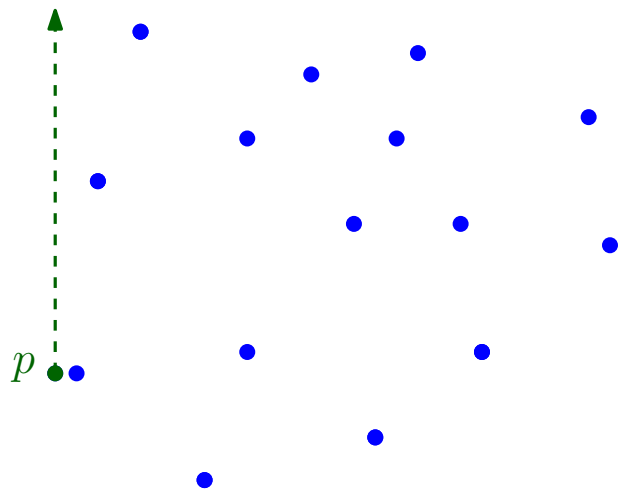
Exercise: Implement Gift Wrapping (Jarvis March)

1. Initialize:
   $p \leftarrow$ the left most point
   $\vec{r}$ an upward ray from p
   CH.add(p)
2. Repeat{
   2.1. Rotate $\vec{r}$ clockwise around p until the first point q is hit
   2.2. **CH**.add(q)
   2.3. $p \leftarrow q$
   } Until p equals the left most point



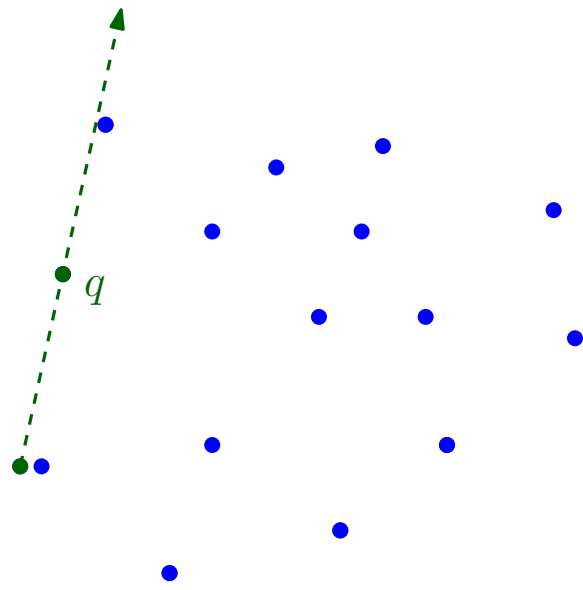
Exercise: Implement Gift Wrapping (Jarvis March)

1. Initialize:
   $p \leftarrow$ the left most point
   $\vec{r}$ an upward ray from p
   CH.add(p)
2. Repeat{
   2.1. Rotate $\vec{r}$ clockwise around p until the first point q is hit
   2.2. **CH**.add(q)
   2.3. $p \leftarrow q$
   } Until p equals the left most point



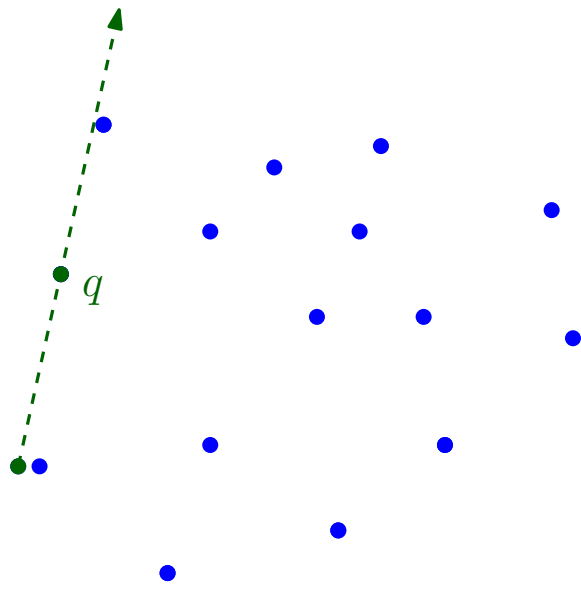
Exercise: Implement Gift Wrapping (Jarvis March)

1. Initialize:
   $p \leftarrow$ the left most point
   $\vec{r}$ an upward ray from p
   CH.add(p)
2. Repeat{
   2.1. Rotate $\vec{r}$ clockwise around p until the first point q is hit
   2.2. **CH**.add(q)
   2.3. $p \leftarrow q$
   } Until p equals the left most point



Exercise: Implement Gift Wrapping (Jarvis March)

1. Initialize:
   $p \leftarrow$ the left most point
   $\vec{r}$ an upward ray from p
   CH.add(p)
2. Repeat{
   2.1. Rotate $\vec{r}$ clockwise around p until the first point q is hit
   2.2. **CH**.add(q)
   2.3. $p \leftarrow q$
   } Until p equals the left most point



Step 2.1 is equivalent to finding the point that makes the smallest rotation angle and thus it is equivalent to calculating the rotation angle for every point and taking their minimum.
Before you implement, show that using the sidedness predicate, this can be done **without** explicitly calculating the angle.