Q1. **Non-recursive tower of Hanoi.** We know the recursive Tower of Hanoi algorithm that moves a stack of n disks from peg 0 to peg 2 (we assume that the pegs are numbered 0, 1 and 2 from left to right). Let's call that "Algorithm 0". Now consider these three non-recursive algorithms.

Algorithm 1: If n is even, swap pegs 1 and 2. At the ith step, make the only legal move that avoids peg i mod 3. If there is no legal move, then all disks are on peg i mod 3, and the puzzle is solved.

Algorithm 2: For the first move, move disk 1 to peg 1 if n is even and to peg 2 if n is odd. Then repeatedly make the only legal move that involves a different disk from the previous move. If no such move exists, the puzzle is solved.

Algorithm 3: Pretend that disks $n+1$, $n+2$, and $n+3$ are at the bottom of pegs 0, 1, and 2, respectively. Repeatedly make the only legal move that satisfies the following three constraints, until no such move is possible.
- Do not place an odd disk directly on top of another odd disk.
- Do not place an even disk directly on top of another even disk.
- Do not undo the previous move.

Now answer the following questions.

1. Draw or write down the moves for $n = 1 \ldots 3$ disks for each of the 4 algorithms. **3 × 3**
2. Now compare the number of moves needed by each algorithm for $n = 1...3$ disks by showing them in a table. **4**
3. Which algorithm is the most efficient in terms of number of moves required? **1**

Q2. **Sorting.** Here is yet another sorting algorithm!

```
StoogeSort(A[0..n−1]):
  if n = 2 and A[0] > A[1]
      swap A[0] ↔ A[1]
  else if n > 2
      m = ⌈2n/3⌉
      StoogeSort(A[0..m−1])
      StoogeSort(A[n−m..n−1])
      StoogeSort(A[0..m−1])
```

Figure 1: The StoogeSort algorithm.

1. Would StoogeSort still sort correctly if we replaced $m = \lceil 2n = 3 \rceil$ with $m = \lfloor 2n = 3 \rfloor$? Justify your answer. **3**
2. State a recurrence (including the base case(s)) for the number of comparisons executed by StoogeSort. **3**

3. Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] **4**

4. Prove that the number of swaps executed by StoogeSort is at most $\binom{n}{2}$. **3**

Q3. **Bonus Question.** The QuickSelect algorithm finds the kth smallest element. Consider three scenarios, and for each of them, answer these two questions. 1%

```
QuickSelect(A[1..n], k):
  if n = 1
      return A[1]
  else
      Choose a pivot element A[p]
      r ← Partition(A[1..n], p)
      if k < r
          return QuickSelect(A[1..r−1], k)
      else if k > r
          return QuickSelect(A[r+1..n], k−r)
      else
          return A[r]
```

Figure 2: The QuickSelect algorithm.

1. What will be the recurrence relation for that scenario?

2. Solve the recurrence relation to find the time complexity. [*You can use the recursion tree method we learned in the class.*]

- **Scenario 1.** The pivot is either the maximum or the minimum element.
- **Scenario 2.** The pivot element is the exactly the middle element each time.
- **Scenario 3.** The pivot is always the $\frac{n}{3}$th element.

Q4. **Searching lower and upper bounds.** Your friend Sam is guessing a number between 1 and n (say, $n = 1,000,000$). You can ask questions in the form "is the number x", where x is between 1 and n, to find the number he is thinking. Remember that Sam is a bit of a cheater, and he may change the number in his mind as long as he does not contradict his previous answers.

1. If Sam answers "Yes/No", how many questions you will need in the worst case? **3**

2. If you change the sequence of your inquiries, can you find the number in less number of questions than your previous attempt? **2**

3. Now assume that Sam answers "higher/lower" to your inquiries. How many questions you will need this time? **3**

5. **Convex Hull.** Implement the algorithm for computing the convex hull of a set of points, shown in the class. **[Marks: 7]**

The two zip files contain the java file and python file that you will write your codes in. Do not change the name of the files. The folder 'inputs' will have all the input files used for testing, and the folder 'outputs' should have the generated outputs. Two sample input files are given in the inputs folder, but we will use bigger inputs to test your code.

A file "testingCodeJava.py" ("testingCodePython.py") is provided in the folder to show how we will be testing your code. Your code should read from a .txt file given as a parameter and convert the input to a format that you can use in your code. The output should also be in a text file with the name "output_[inputfile name]" in the 'outputs' folder. For example, if the input file is named "point_sets_20.txt" the output file should be "output_point_sets_20.txt".

Input/output format: The first line contains the number n of points, the next n lines contains the x and y coordinates separated by a space. The output should have the number of points on the convex hull in the first line, followed by the coordinates of the points in clockwise order. See the input and output format below for the point set in Figure 3.

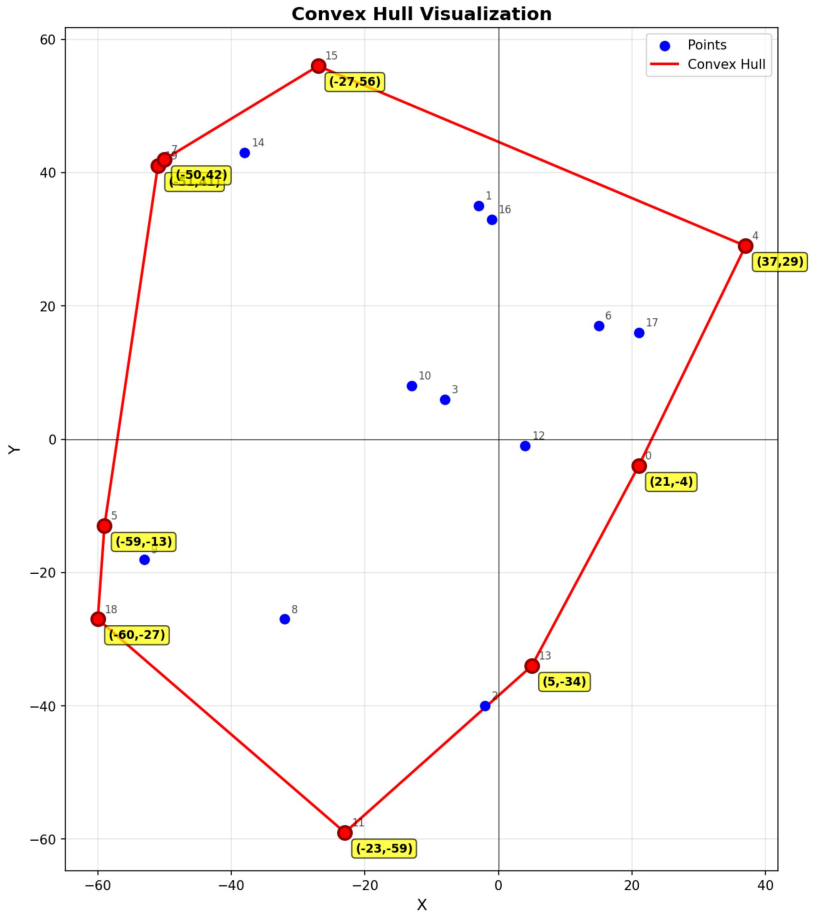

Figure 3: Convex hull of the point set in file "point_sets_20.txt" generated by Claude.

Input 1 (point_sets_20.txt):

```
20
21 -4
-3 35
-2 -40
-8 6
37 29
-59 -13
15 17
-50 42
-32 -27
-53 -18
-13 8
-23 -59
4 -1
5 -34
-38 43
-27 56
-1 33
21 16
-60 -27
-51 41
```

Output 1 (output_point_sets_20.txt):

```
9
-51 41
-50 42
-27 56
37 29
21 -4
5 -34
-23 -59
-60 -27
-59 -13
```

1. Choose either java or python as the language and add your code to the corresponding template provided. Do not change the name of python/java file given to you.
2. For every method/function you declare other than the main method should have [your_name] added to the end of their name.
3. Do not submit the output or input files you use or produce. This will reduce the size of your submission.
4. Add your student number to the folder containing all your files (including the .pdf file with all the other answers), zip it and then submit.