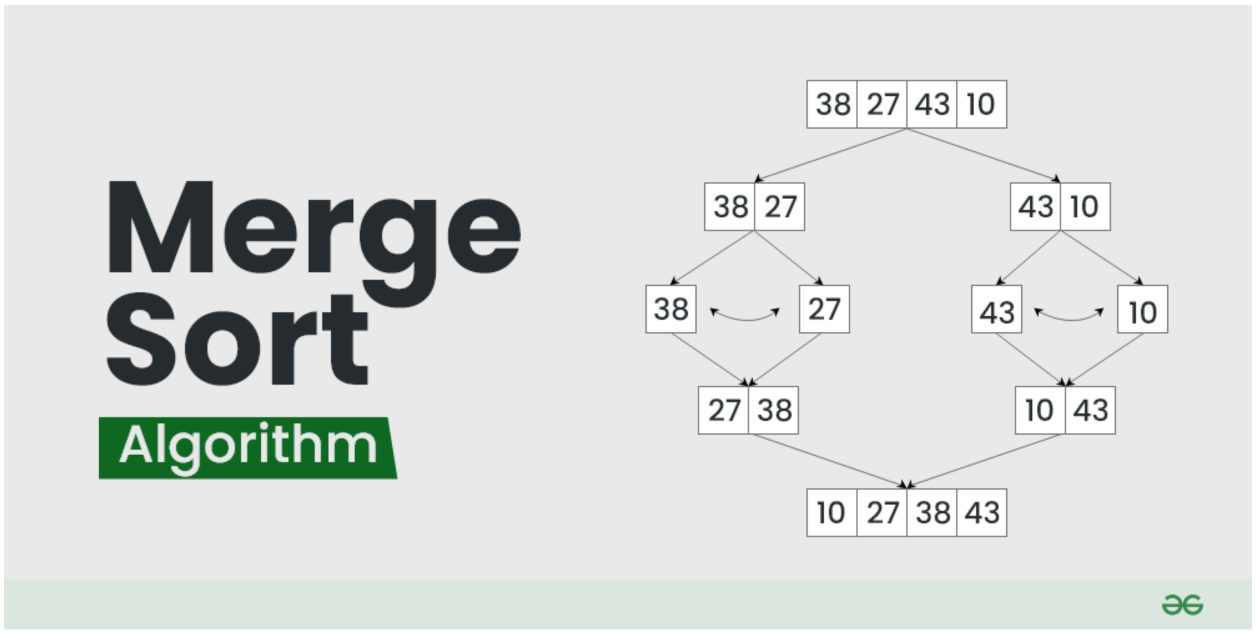
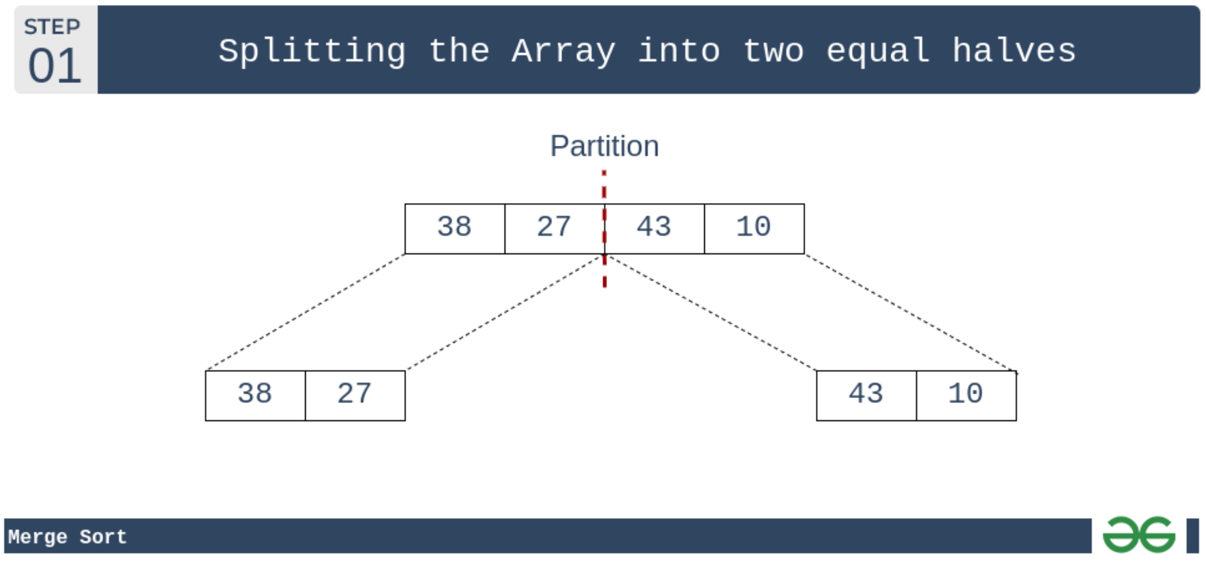
Merge Sort Algorithm (Quick Recap)

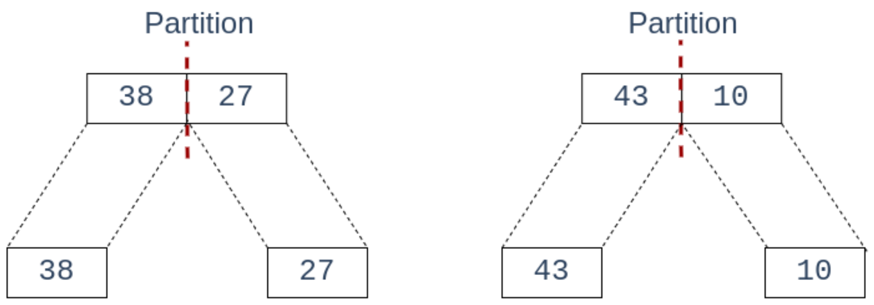


Merge sort is a recursive algorithm that continuously splits the array in half until it cannot be further divided i.e., the array has only one element left (an array with one element is always sorted). Then the sorted subarrays are merged into one sorted array.



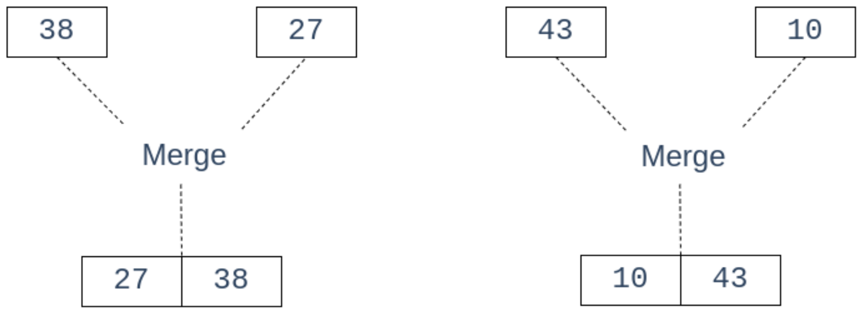
STEP 02

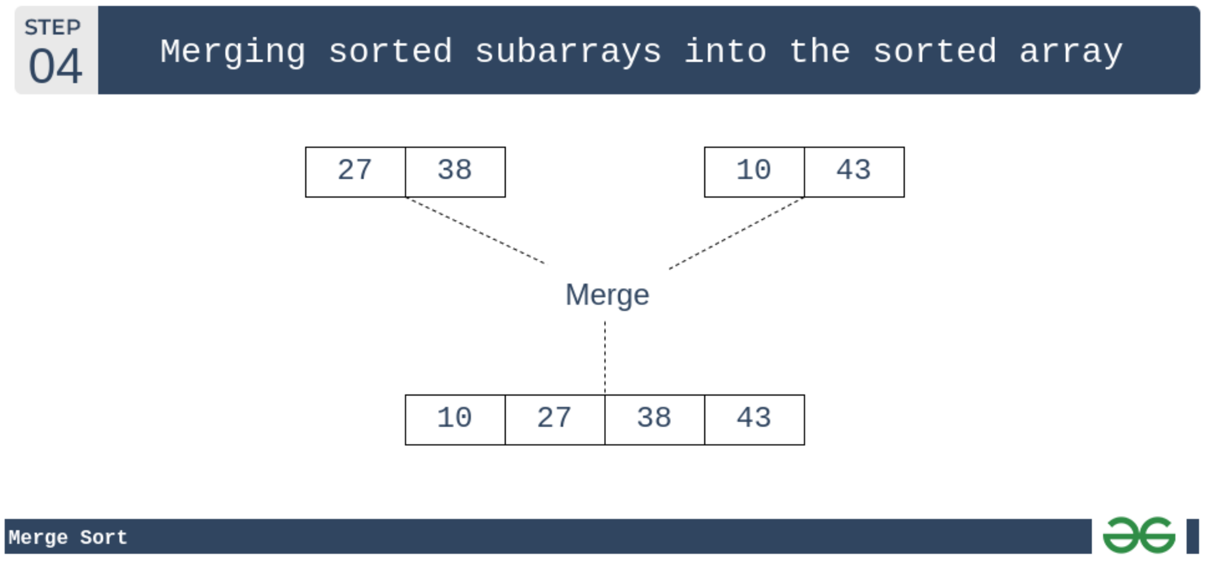
Splitting the subarrays into two halves



STEP 03

Merging unit length cells into sorted subarrays





Algorithm:

- Declare left and right var which will mark the extreme indices of the array.
- Left will be assigned to 0 and right will be assigned to n-1.
- Find mid = (left+right)/2.
- Call mergeSort on (left,mid) and (mid+1,rear)
- Above will continue till left<right.
- Then we will call merge on the 2 subproblems.

```
MERGE($A, p, q, r$)
1.  $n_1 = q - p + 1$
2.  $n_2 = r - q$
3.  let $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$ be new arrays
4.  for $i = 1$ to $n_1$
5.      $L[i] = A[p + i - 1]$
6.  for $j = 1$ to $n_2$
7.      $R[j] = A[q + j]$
8.  $L[n_1 + 1] = \infty$
9.  $R[n_2 + 1] = \infty$
10. $i = 1$
11. $j = 1$
12. for $k = p$ to $r$
13.     if $L[i] \leq R[j]$
14.         $A[k] = L[i]$
15.         $i = i + 1$
16.     else $A[k] = R[j]$
17.         $j = j + 1$
```

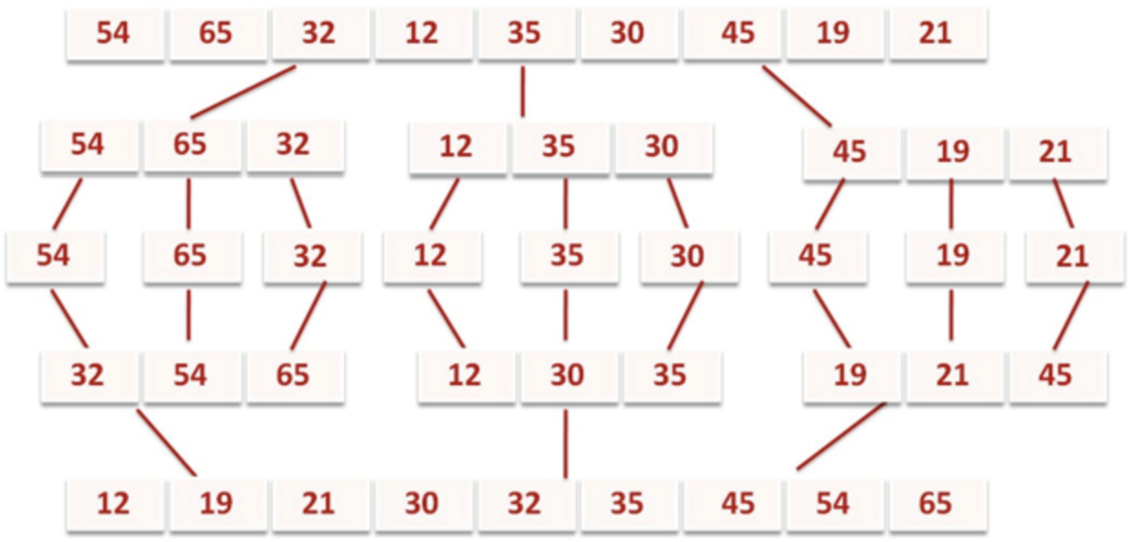
```
MERGE-SORT($A, p, r$)
1  if $p < r$
2      $q = \lfloor (p + r)/2 \rfloor$
3      MERGE-SORT($A, p, q$)
4      MERGE-SORT($A, q + 1, r$)
5      MERGE($A, p, q, r$)
```

Merge-Sort = T(n) = 2T(n/2)+O(n) = 2T(n/2) + cn
Tree below:

level= 1 cn = cn
level = 2 cn/2 + cn/2 = cn
level = 3 cn/4 + cn/4+ cn/4 + cn/4 = cn
….
level = log2(n)+ 1 = c + c + c … + c = cn
= cn(log2(n)+1) = cnlog2(n) + cn -> O(nlogn)

3-way Merge Sort (split into 3)

The recursion tree for the 3 way merge sort is:



1. **Class Activity:** txt file: Merge-Sort 3-way. Explain in the comment lines. Any programming language. It should be done by the end of the class hours

Random fun fact: ChatGPT Generated a problem inspired by Jules Verne narrative:

In the fog-shrouded coastal town of Sortington, which lay in the shadow of the great Sorting Mountain, a renowned adventurer named Captain Phineas Merger set sail on a quest to bring order to the tumultuous seas of unsorted data. He was a man known throughout the kingdom for his daring expeditions and his love for all things organized.

The Dividing Expedition:

With his trusty compass and a curious device known as the "Data Divider," Captain Merger set off on his ship, the "Sorter's Voyager." As he ventured into the uncharted waters of unsorted chaos, he discovered vast data islands. These islands were riddled with disarray, and his mission was to divide the untamed data into two, smaller and more manageable piles.

His Data Divider worked like a sextant, allowing him to pinpoint the middle of the data islands. With each divide, Captain Merger ensured that only the most crucial data fragments remained.

The Sorting Odyssey:

Once the data islands were divided, the Captain's journey continued. His Sorting Odyssey was a quest to sort these smaller data fragments into order. He navigated the seas, comparing data points and carefully rearranging them to establish the correct order. It was as if he were deciphering ancient scrolls of forgotten wisdom.

With unwavering determination, Captain Merger conquered every fragment, ensuring that the knowledge contained within was ordered correctly, and that no data artifact was left adrift.

The Merging Expedition:

As the Sorting Odyssey drew to a close, Captain Merger undertook his final challenge—the Merging Expedition. He combined the sorted data fragments with an ingenious device known as the "Data Combinator." This device allowed him to merge the fragments harmoniously, creating data archives of great order and clarity.

The data fragments came together, uniting like different factions of a grand empire, sharing their wisdom and organizing it into a single, monumental archive.

At last, Captain Phineas Merger's ship, the "Sorter's Voyager," sailed back to the harbor of Sortington, proudly bearing the fruits of his labor—a kingdom with organized data, where the people marveled at the harmony brought by Captain Merger's incredible sorting prowess.

And so, Captain Phineas Merger, through his daring sorting expeditions, ensured that the kingdom's knowledge was organized, reminding everyone that even in the vast sea of unsorted data, there exists a method to bring order, and it takes an adventurer with a heart for sorting to embark on such journeys.

—-----

Challenging Problems on Divide and Conquer

Divide and conquer algorithms are used to solve various types of problems. Some of the most challenging divide and conquer problems include:

1. **Closest Pair of Points:** Given a set of points in a two-dimensional space, the task is to find the pair of points that are closest to each other. This problem involves both divide-and-conquer techniques and computational geometry.
2. **Matrix Multiplication:** Matrix multiplication can be challenging when it involves large matrices. Algorithms like Strassen's algorithm aim to reduce the number of multiplications and optimize the process, making it a challenging divide-and-conquer problem.
3. **Integer Multiplication:** Multiplying large integers efficiently is a challenging divide-and-conquer problem. Algorithms like Karatsuba multiplication and Toom-Cook multiplication address this problem.
4. **Maximum Subarray Sum:** Given an array of numbers, the problem is to find the contiguous subarray with the largest sum. Algorithms like Kadane's algorithm use divide and conquer principles to find the solution efficiently.
5. **Convex Hull:** Finding the convex hull of a set of points is a classic computational geometry problem. Algorithms like Graham's scan and QuickHull are based on divide-and-conquer techniques to find the convex hull.
6. **Fibonacci Numbers:** Calculating Fibonacci numbers efficiently is a common example of divide and conquer problems. Algorithms like matrix exponentiation are used to calculate Fibonacci numbers in logarithmic time.
7. **Inverse Square Root:** Finding the inverse square root of a number efficiently is a problem that has applications in computer graphics and physics simulations. The famous

"Fast Inverse Square Root" algorithm uses divide and conquer in combination with bitwise operations.

8. **Strassen's Matrix Multiplication:** Strassen's algorithm for matrix multiplication is a classic example of a divide and conquer approach. It's challenging due to its recursive structure and requires clever optimization techniques.
9. **Longest Common Subsequence (LCS):** Finding the longest common subsequence between two sequences is a classic divide and conquer problem in computer science and bioinformatics.

The Skyline Problem:

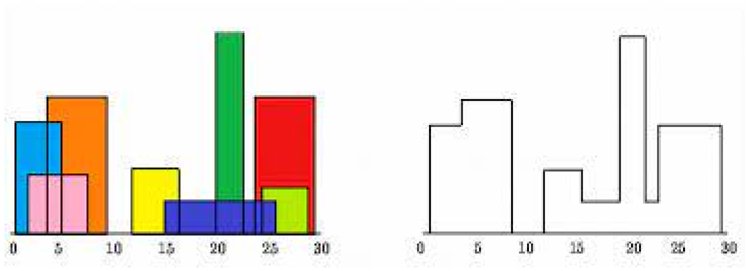
This is a classic computational problem in computer science, and it's often encountered in urban planning, architecture, and computer graphics. It involves finding the outline, or "skyline," of buildings or structures in a two-dimensional space. The skyline represents the visible silhouette of these structures when viewed from a particular perspective.

Here's a more detailed explanation of the problem:
Input:
You are given a set of building structures, each represented by a triplet (L, H, R), where:
L is the left (x-coordinate) of the building.
H is the height of the building.
R is the right (x-coordinate) of the building.
Output:
The goal is to determine the skyline, which is a sequence of key points in the form (x, y). Each point represents a horizontal line segment of the skyline.

Problem Statement:
Given a set of building structures, you need to determine the points at which the skyline changes height, i.e., where buildings start or end, or where one building's height differs from the previous one. These points represent the outline of the buildings when viewed from a particular perspective.

The key properties of the skyline are as follows:

1. The left endpoint of a horizontal line segment is a critical point. It represents the beginning of a building.
2. The right endpoint of a horizontal line segment is a critical point. It represents the end of a building.
3. A point where two building structures intersect is also a critical point. It represents the point where one building starts, and another ends or overlaps.
4. The height associated with a critical point represents the height of the skyline at that position.

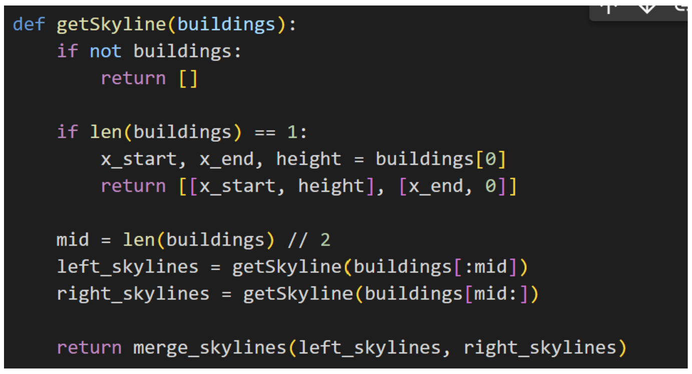


1. Divide: Divide the set of buildings into two halves.
2. Conquer: Recursively find the skyline of each half.
3. Merge: Merge the two skylines to find the overall skyline.

4. Merge the two skylines by comparing their critical points. Handle the following cases:
    a. If the next critical point comes from the left skyline, update the current height in the merged skyline.
    b. If the next critical point comes from the right skyline, update the current height in the merged skyline.
    c. If the heights of the two skylines differ, update the merged skyline with the higher height.
5. Base Case: When there is only one building, compute the skyline for that building.

This approach ensures that the skyline is computed for smaller subsets of buildings, and then the results are combined to form the overall skyline.
Modifying the merge step is essential to handle the varying heights of the buildings correctly. The overall algorithm involves both divide and conquer principles and a modified merging operation that considers the heights of the skylines.

While this approach is inspired by divide-and-conquer techniques, it's important to note that the overall algorithm may involve more than just a traditional merge sort. It often requires careful handling of the skyline's critical points and efficient data structures to achieve optimal performance.

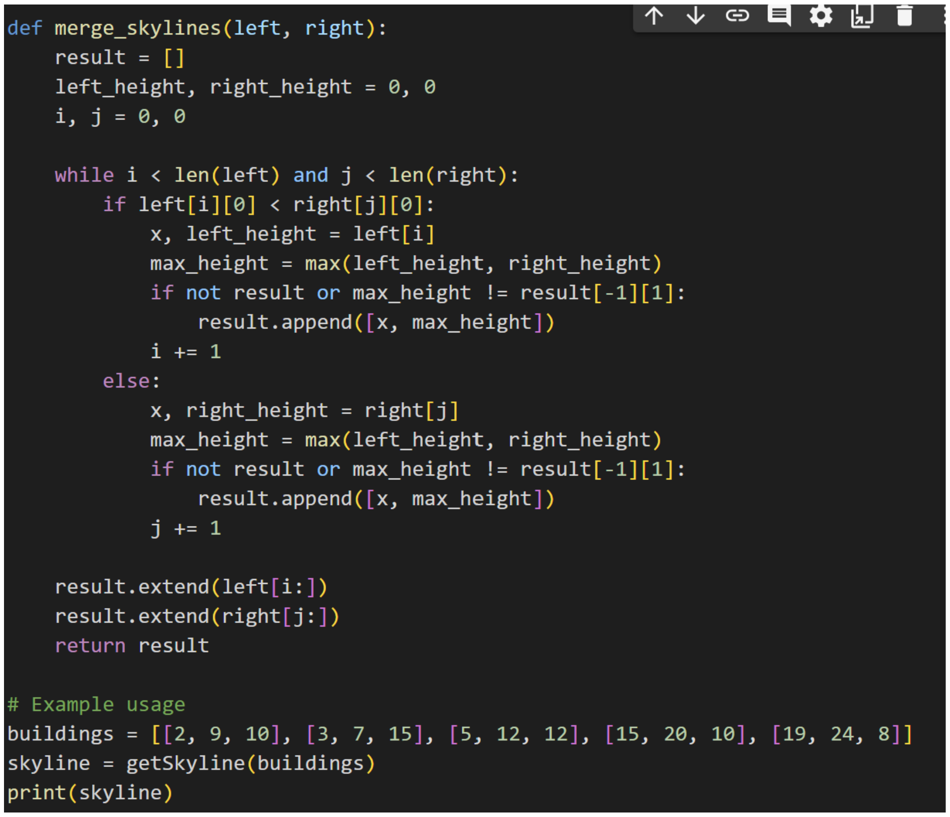


```python
def getSkyline(buildings):
    if not buildings:
        return []

    if len(buildings) == 1:
        x_start, x_end, height = buildings[0]
        return [[x_start, height], [x_end, 0]]

    mid = len(buildings) // 2
    left_skylines = getSkyline(buildings[:mid])
    right_skylines = getSkyline(buildings[mid:])

    return merge_skylines(left_skylines, right_skylines)
```



```python
def merge_skylines(left, right):
    result = []
    left_height, right_height = 0, 0
    i, j = 0, 0

    while i < len(left) and j < len(right):
        if left[i][0] < right[j][0]:
            x, left_height = left[i]
            max_height = max(left_height, right_height)
            if not result or max_height != result[-1][1]:
                result.append([x, max_height])
            i += 1
        else:
            x, right_height = right[j]
            max_height = max(left_height, right_height)
            if not result or max_height != result[-1][1]:
                result.append([x, max_height])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])
    return result

# Example usage
buildings = [[2, 9, 10], [3, 7, 15], [5, 12, 12], [15, 20, 10], [19, 24, 8]]
skyline = getSkyline(buildings)
print(skyline)
```

2. **Class Activity:** Your student ID number's last 5 digits in modulo 9 is the algorithm you'll be picking from the list above explain in your own words how you will implement it. Give the problem statement, inputs, and outputs.

   Explain your steps with your simple words. Explain the details of algorithm time and space costs in a few sentences. Time cost: approximate the number of steps for n inputs. Space cost: approximate the space in memory used for n integers. Hint: When you use recursion and pointers, you do not need to allocate new space in the memory for occasions. But each new instance created allocates a space.

   e.g. your id number 20235153141=> 53141 mod 9 = 5 => Convex Hull algorithm.

   The submission will be in a single-page PDF. You can't use handwriting. Use 1 page wisely. Code submission is not accepted. Your algorithm must perform in a divide-and-conquer/recursive fashion.

Deadline: 17/10/2023 23:59