**Batch : T7**

**Practical No . 3**

**Title of Assignment : Divide and Conquer Strategy**

**Student Name: Om Kulkarni**

**Student PRN : 22510034**

**Problem Statement:**

**Q1)** Implement algorithm to Find the maximum element in an array which is first increasing and then decreasing, with Time Complexity *O(Logn).*

1. Algorithm/Pseudocode

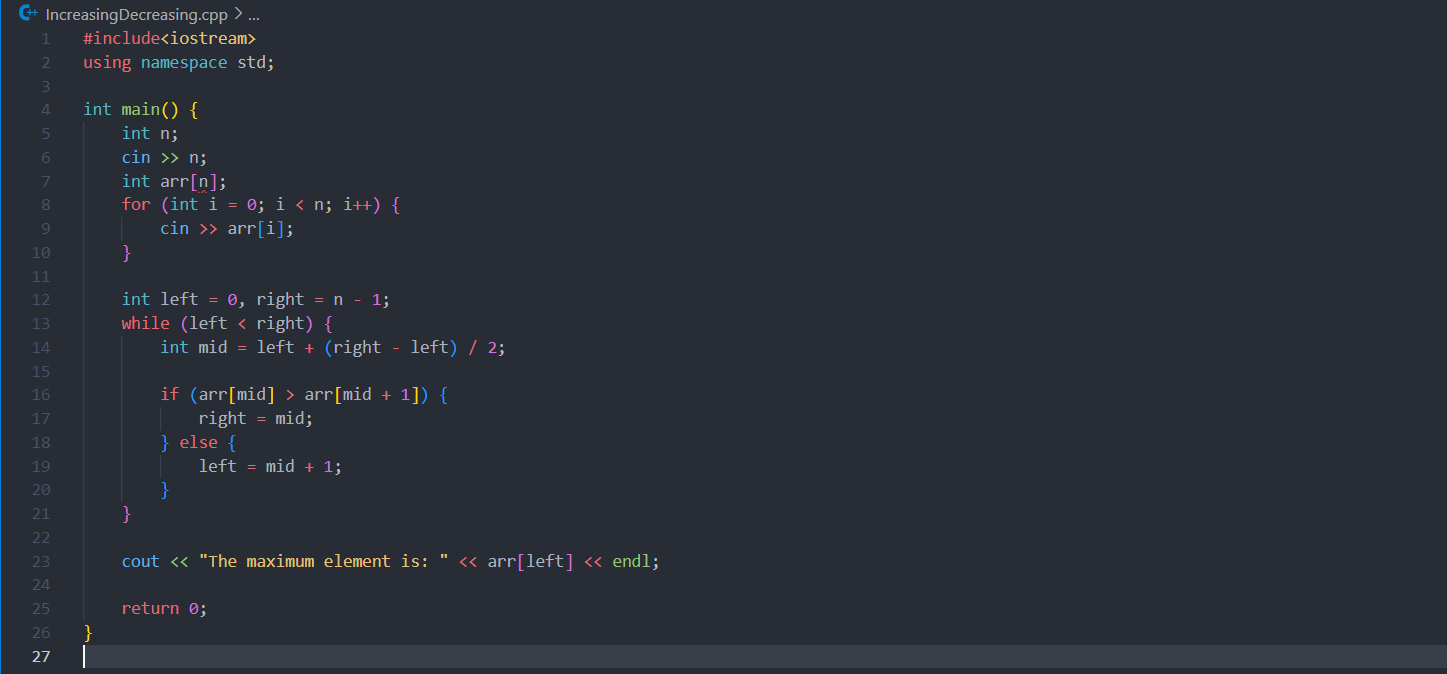
**Initialize Pointers**:

* Set low = 0 and high = n-1 where n is the length of the array.

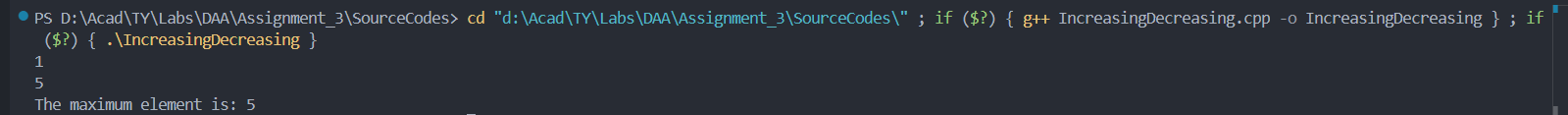
**Binary Search**:

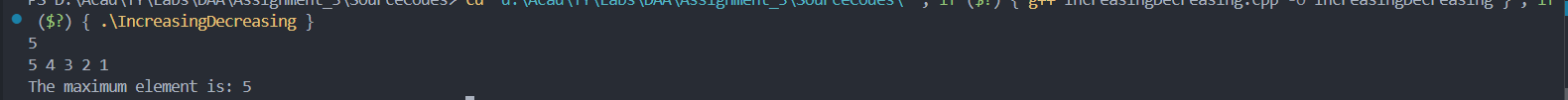
* While low <= high, calculate mid as mid = low + (high - low) / 2.
* Compare arr[mid] with its neighbors:
  + If arr[mid] > arr[mid - 1] and arr[mid] > arr[mid + 1], then arr[mid] is the maximum element.
  + If arr[mid] > arr[mid - 1] but arr[mid] < arr[mid + 1], it means the peak is in the right half, so set low = mid + 1.
  + If arr[mid] < arr[mid - 1] and arr[mid] > arr[mid + 1], it means the peak is in the left half, so set high = mid - 1.

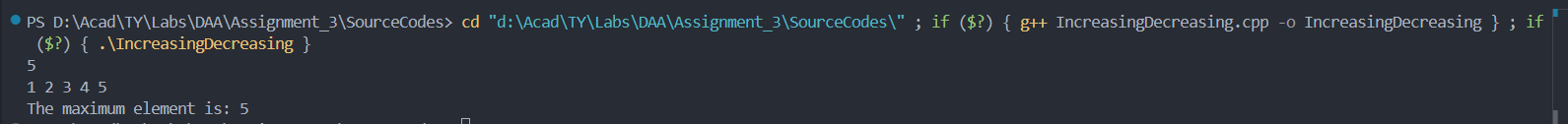
1. Program Code



1. Output with verity of test cases



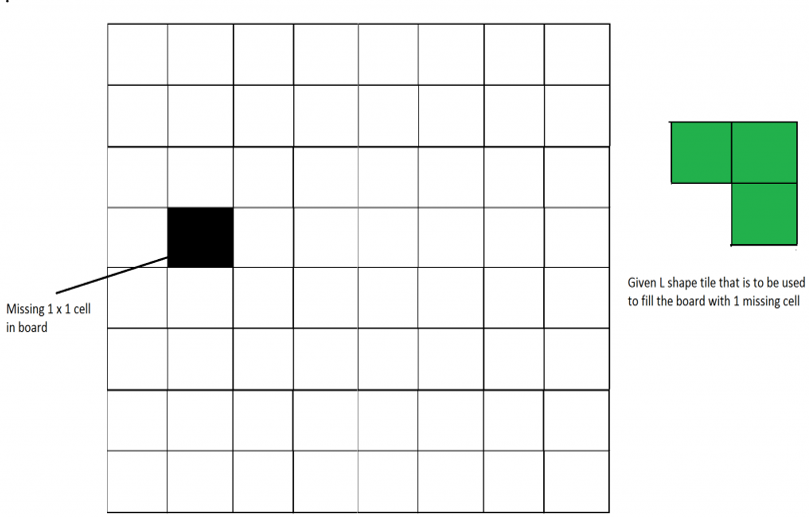




1. Analysis in terms of complexity wherever applicable.
2. Time Complexity**:** O(log n) in the average and worst case, O(1) in the best case.
3. Space Complexity: O(log n) for the recursive approach, O(1) for the iterative approach.

**Problem Statement:**

**Q2)** Implement algorithm for Tiling problem: Given an *n by n* board where n is of form *2k* where *k >= 1* (Basically n is a power of *2* with minimum value as *2*). The board has one missing cell (of size *1 x 1*). Fill the board using L shaped tiles. An *L* shaped tile is a *2 x 2* square with one cell of size *1×1*

****

1. Algorithm/Pseudocode

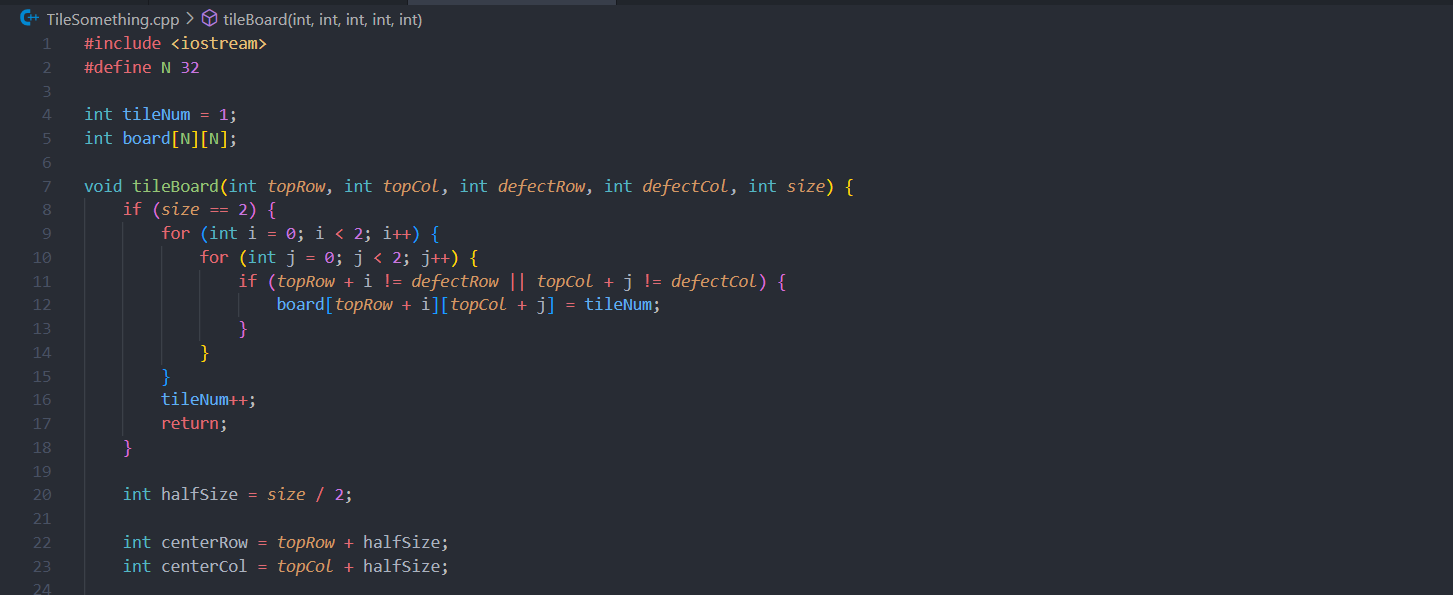
Base Case:

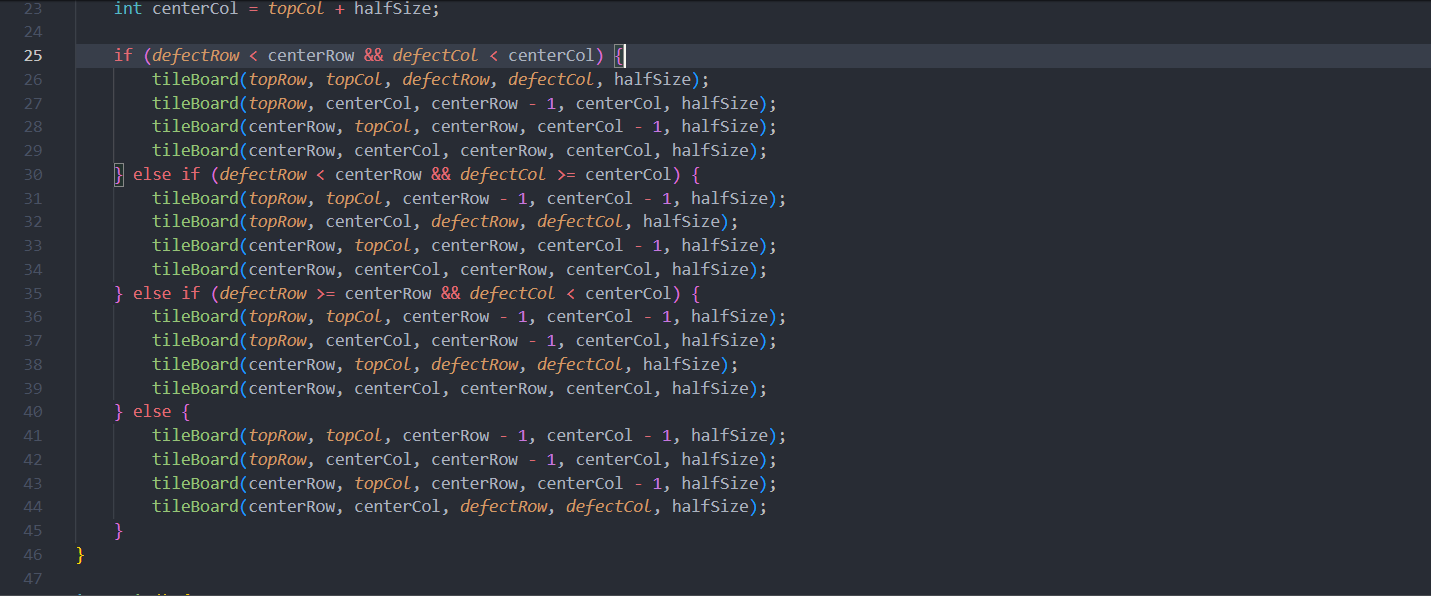
* When n=2, directly place the L-shaped tile to cover the 2×2 board with one cell missing.

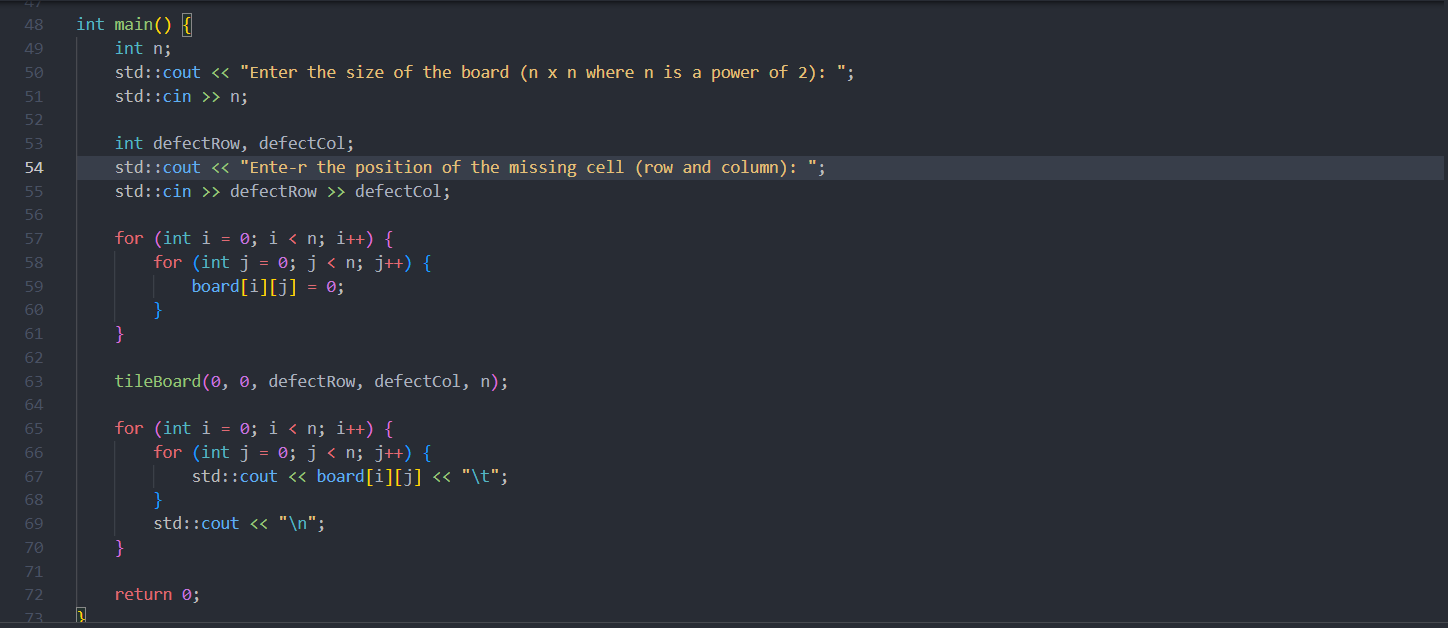
Recursive Case:

* Divide the board into four n/2×n/2sub-boards.
* Identify which of the four sub-boards contains the missing cell.
* Place an L-shaped tile in the centre of the board, covering the three cells that are not in the same sub-board as the missing cell.
* Recursively apply the same procedure to the four n/2×n/2 sub-boards.

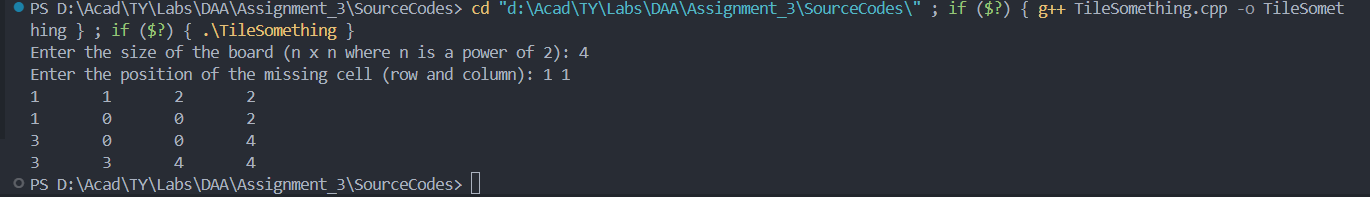
1. Program Code







1. Output with verity of test cases



1. Analysis in terms of complexity wherever applicable.

**Time Complexity:**

1. **Base Case (2x2 Board)**:
   * For a 2x2 board, the function directly places an L-shaped tile. This operation involves checking each cell in the 2x2 board, which is a constant time operation O(1).
2. **Recursive Case (n x n Board)**:
   * For an n×n board, where n is a power of 2, the function divides the board into four n/2×n/2 and recursively solves the problem for each quadrant.
   * Therefore, the recursive function is called 4 times on subproblems of size n/2 x n/2
   * This gives the following recurrence relation:

T(n)=4T(n/2)+O(1)

* + Using the Master Theorem for divide-and-conquer recurrences:
    - a=4, b=2, d=0 (since the additional work outside the recursion is constant, i.e., O(1)).
    - We have nlogba=nlog24=n^2
  + Comparing n^d with nlogab, we find that the dominant term is n^2, so: T(n) = O(n^2)
  + Therefore, the time complexity of the algorithm is O(n^2)

**Space Complexity:**

1. **Auxiliary Space**:
   * The algorithm uses recursion, so it consumes stack space proportional to the depth of the recursive calls.
   * The depth of the recursion islog2​n, as each call halves the size of the board.
   * The auxiliary space complexity due to recursion is O(logn).
2. **Board Storage**:
   * The board itself is stored in an n× n array, requiring O(n^2)space.

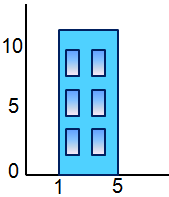
**Problem Statement:**

Q3) Implement algorithm for The Skyline Problem: Given n rectangular buildings in a 2-dimensional city, computes the skyline of these buildings, eliminating hidden lines. The main task is to view buildings from a side and remove all sections that are not visible.

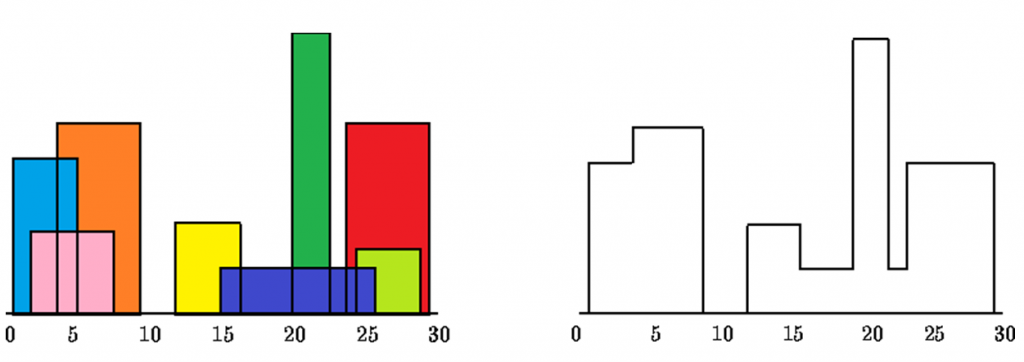
All buildings share common bottom and every **building**is represented by triplet (left, ht, right)

‘left’: is x coordinate of left side (or wall).  
‘right': is x coordinate of right side  
‘ht': is height of building.

For example, the building on right side is represented as *(1, 11, 5)*

[](http://www.geeksforgeeks.org/divide-and-conquer-set-7-the-skyline-problem/building/)

A **skyline**is a collection of rectangular strips. A rectangular **strip**is represented as a pair (left, ht) where left is x coordinate of left side of strip and ht is height of strip.



With Time Complexity *O(nLogn)*

1. Algorithm/Pseudocode

Divide:

* Divide the list of buildings into two halves.
* Recursively find the skyline for each half.

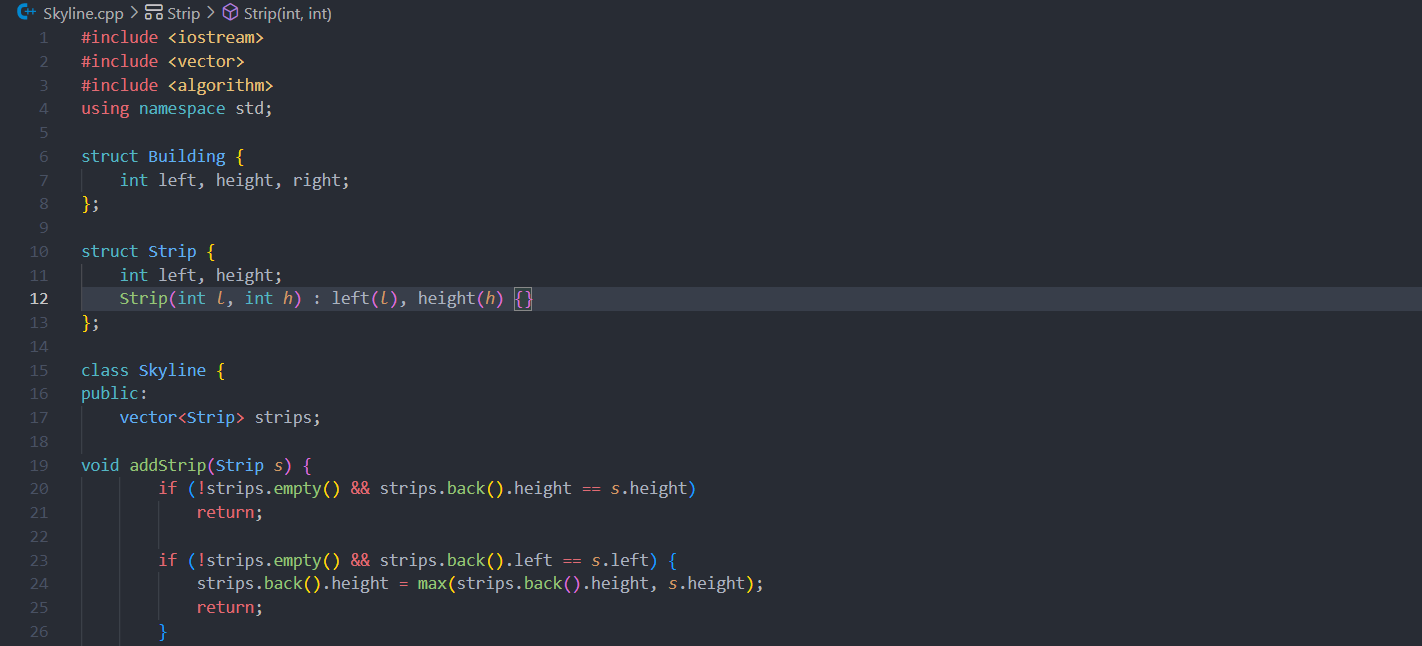
Conquer:

* Once the skyline for both halves is found, merge the two skylines to form the final skyline.

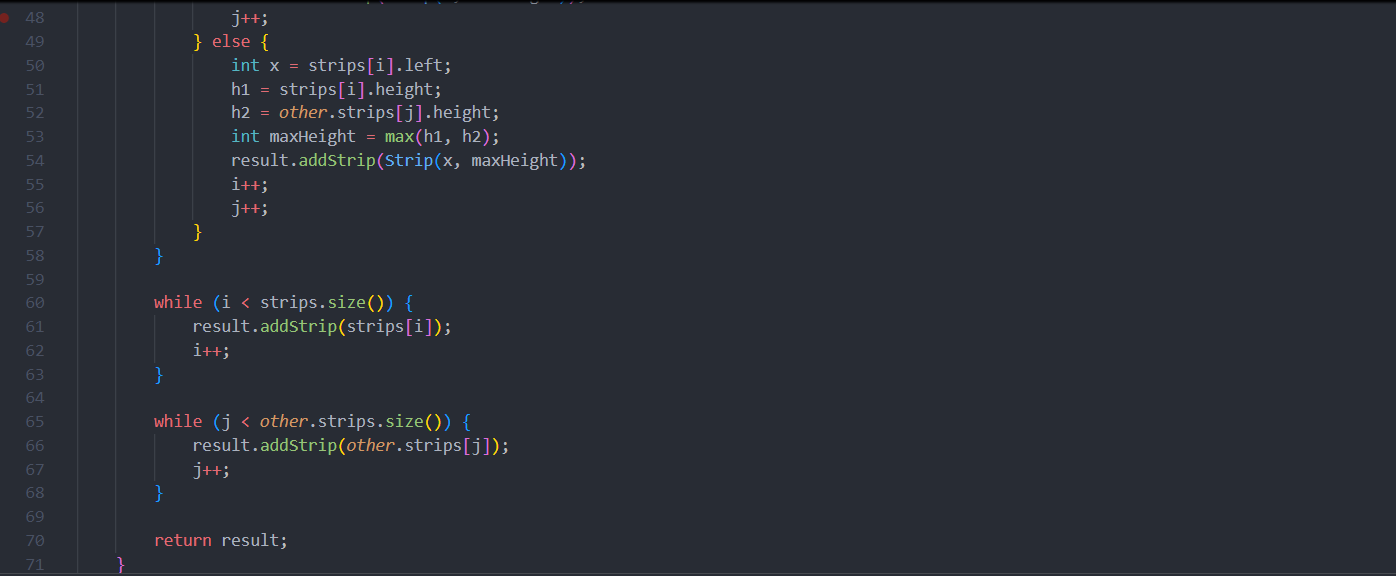
Merge**:**

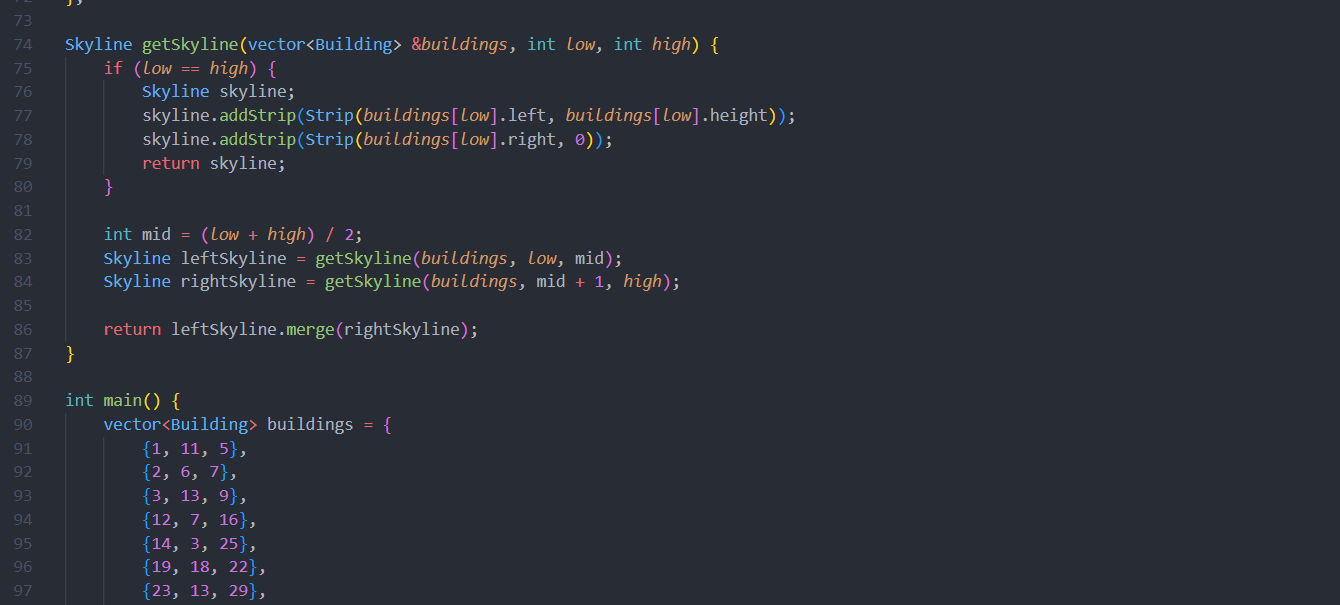
* Merge two skylines by comparing the heights of the buildings at various x-coordinates.
* The merge process involves iterating through both skylines and taking the maximum height at each x-coordinate.

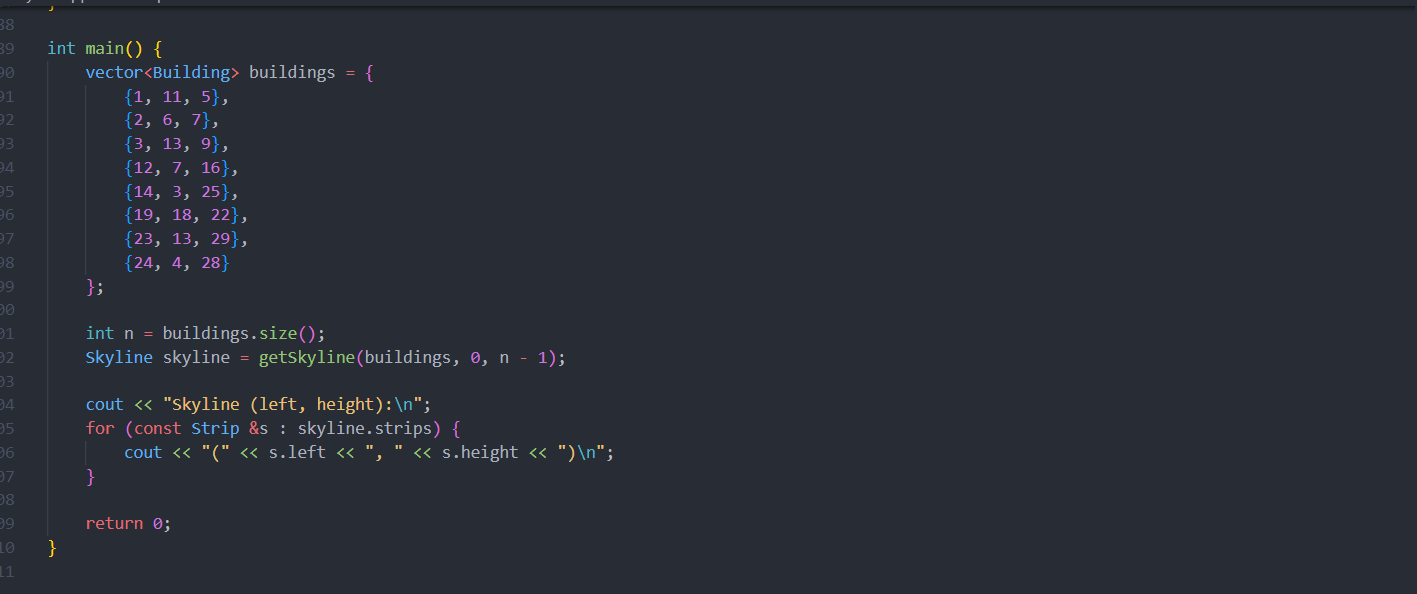
1. Program Code



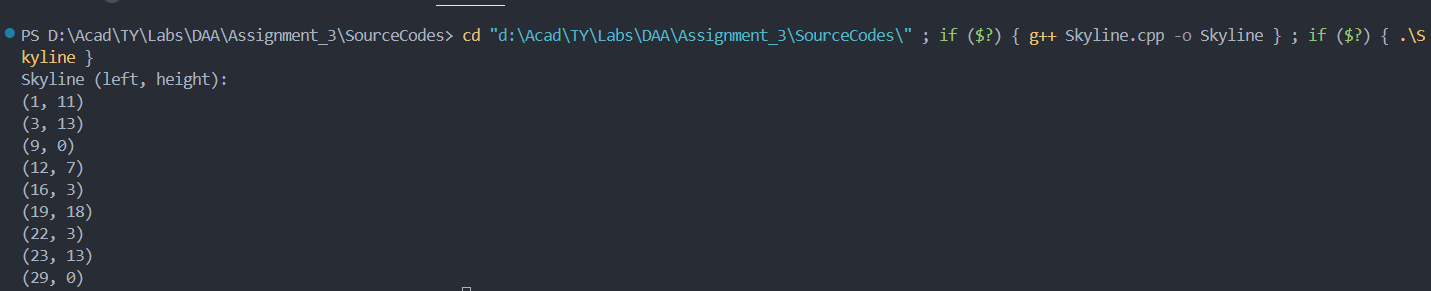








1. Output with verity of test cases



1. Analysis in terms of complexity wherever applicable.

**Time Complexity:**

* **Divide and Conquer:** The algorithm splits the problem into halves and merges the results, leading to a recurrence relation of T(n)=2T(n/2)+O(n)
* **Overall Time Complexity:**O(nlog n), which is efficient for this type of problem.

**Space Complexity:**

* **Auxiliary Space:** O(n) for storing the skyline strips and O(log n) for the recursion stack.
* **Overall Space Complexity:** O(n).

Note:

1. Scan the document and **create a pdf** **file** with **“ExamSeatNum\_P#PS#” as its name**.
2. Create zip folder of your project/assignment which contains source code
3. Add the scanned document into the zip folder and name it as **ExamSeatNum\_P#PS#” as its name**
4. Upload the folder on the **WCE** **Moodle/ ERP** before the given deadline.

(P#: practical number

PS#: Problem statement Number)