CSE 321
HW 4

1)

Python code is submitted as "findFlawed.py"

Time Complexity Analysis

- I create a class named "fuse" and add one instance variable named "isFlawed" to test my code.
- I use binary search algorithm for decrease and conquer mechanism. The binary search reduces the search space on each iteration. Thats why time complexity of this code is O(log n).

2-)

Python code is submitted as "identifyPixel.py"

Time Complexity Analysis

- I create a class named "Pixel" and add one instance variable named "brightness" to test my code.
- The function reduces the size of problem instance by a constant value (one pixel) at each step until it finds the pixel the breaks monotonic Pattern.
- The function uses a while loop to traverse the image from center and compare the brightness of the current pixel with its four neighbors. The function stops when it reaches a pixel that is brighter then all its neighbors.
- In the worst case the pixel we want to find is in the corners of the grid.
- Thats why the time complexity of this code is $O(m+n)$ where m is the number of rows and n is the number of columns.
- After all $O(m+n)$ is linear we can show this as $O(n)$.

3-)

Python code is submitted as "largestArea.py"

Time Complexity Analysis

- The function uses decrease and conquer algorithm to reduces the size of the input at each step.
- "area" function takes linear time.
- "largestArea" function repetedly decreases the size of the interval by either incrementing "start" or decrementing "end".
- In worst case "largestArea" calls "area" function at most n times and each call takes $O(n)$ time.
- So the time complexity of "largestArea" function is $O(n^2)$.

4-)

Python code is submitted as "findMinLatency.py"

Time Complexity Analysis

- I create a class named "Graph" and add one function to that class for adding edges.
- The function uses DFS algorithm to find path and the min latency between two points we wanted.
- DFS uses a stack to explore usable paths and the algorithm keeps track of the minimum latency.
- Because of DFS time complexity of this code is $O(E+V)$. E is the number of edges and V is number of nodes

5-)

Python code is submitted as "allocateResources.py"

Time Complexity Analysis

- The algorithm first checks for base cases for length of tasks list and returns appropriate values. Subsequentally, the list is recursively divided into two halves and the max and min resources is calculated. The left and right results are combined by taking max and min values.
- So the time complexity of this code is O(logn) because of only recursive calls.