Süleyman KORAMAZ
1901042615

CSE321 HOMEWORK-4

① procedure findFlawedFuse(fuses):

```
    n = fuses.length
    if n == 1:
        return fuses[0]
    end if

    step = determineStepSize(n)
    start = 0

    while start < n:
        end = min(start+step, n)
        if isCircuitOperational(fuses[start:end]):
            return findFlawedFuse(fuses[end:])
        end if
    end while
    return fuses[n-1]
end
```

② procedure findBrightestPixel(image):

```
    row, column = 0,0
    n, m = image.rows.length, image.columns.length
    while true:
        max = image[row][column]
        brightestNeighbor = None
        if top neighbor brighter:
            brightestNeighbor = (row-1, column)
        else if bottom neighbor brighter:
            brightestNeighbor = (row+1, column)
        else if left neighbor brighter:
            brightestNeighbor = (row, column-1)
        else if right neighbor brighter:
            brightestNeighbor = (row, column+1)
        end if
        if brightestNeighbor is None:
            return (row, column)
        else:
            row, col = brightestNeighbor
        end if
    end while
end
```

```
procedure determineStepSize(n):
    # Specific constant based on specifics
    return 1
end

procedure isCircuitOperational(subset):
    if Electricity Reaches End:
        return True
    else:
        return False
    end if
end
```

① Time Complexity:

determineStepSize function called once and its time complexity is constant O(1)

while loop runs n/stepSize O(n/step)

Overall => O(1) + O(n/step) = O(n)

② Time Complexity:

Time complexity is O(n.m)

n = # of rows

m = # of columns

③ procedure findMaxArea(points):

```
    n = points.length
    max = 0
    current = 0
    start = 0
    for i in range(n):
        current = current + points[i].y
        if current > max:
            max = current
            optimal (points[start].x, points[i].y)
        end if
        if current < 0:
            current = 0
            start = end + 1
        end if
    end for
    return optimal
end
```

③ Time Complexity:

Algorithm iterates the points once.

Time complexity = $O(n)$

n = # of points.

④ procedure findMinLatency(graph, source, destination):

```
    procedure dfs(node, path, latency):
        if node == destination:
            if latency < min_latency:
                min_latency = latency
                best_path = path
            end if
            return
        end if
        for neighbor, edge_latency in graph[node]:
            if neighbor not in path:
                dfs(neighbor, path + [neighbor], latency, edge_latency)
            end if
        end for
    end
min_latency = ∞  # global for functions
best_path = []  # global for functions
dfs(source, [source], 0)
return best_path, min_latency
end
```

④ Time Complexity:

V = # of nodes (vertices)

E = # of edges

Time complexity of dfs is $O(V+E)$

Algorithm explores all possible paths, the overall time complexity = $O((V+E).P)$

P = # of paths

P is constant

Time complexity = $O(V+E)$

⑤ procedure resourceAllocation(tasks):

```
    if tasks.length == 1:
        return tasks[0], tasks[0]
    end if
    mid = tasks.length / 2
    leftHalf = tasks[:mid]
    rightHalf = tasks[mid:]
    leftMax, leftMin = resourceAllocation(leftHalf)
    rightMax, rightMin = resourceAllocation(rightHalf)
    max = max(leftMax, rightMax)
    min = min(leftMin, rightMin)
    return max, min
end
```

⑤ Time Complexity:

Recurrence = $T(n) = T(\frac{n}{2}) + \underbrace{f(n)}_{1}$

Master theorem:

$a = 2$ $\quad a > b^d$

$b = 2$ $\quad 2 > 1$

$d = 0$

$f(n) = n^0$ $\quad$ Time Complexity = $O(n^{\log_b a})$

$= O(n)$