CSE 4810 : Algorithm Engineering Lab
Lab - 1

Md Farhan Ishmam (180041120)

Group - CSE 1A

January 12, 2023

Task 1

The function is comprised of a loop where the function is recursively called. The function runs for n iterations where n is the input size and in each recursion call, the function is called for an input size of $n-1$. The function will be recursively called until $l = r$ i.e. $n = 0$. If we construct the recursion tree seen in Figure 1, then after the first recursion call there will be n function calls with parameters $l+1, r$. Each such subsequent function call will again make $n-1$ function calls, and it will continue until the base condition $l = r$ is satisfied. So, each function calls take $O(1)$ time. So, the number of nodes in the tree or the size of the tree will be $O(n * (n-1) * (n-2) * ... * 2 * 1) = O(n!)$.

Hence, the worst case time complexity is $O(n!)$. In the best case, the algorithm will also run for n! iterations. Hence, the best time complexity is $\Omega(n!)$. As, $O(n!) = \Omega(n!)$ the average case time complexity is also $\Theta(n!)$.

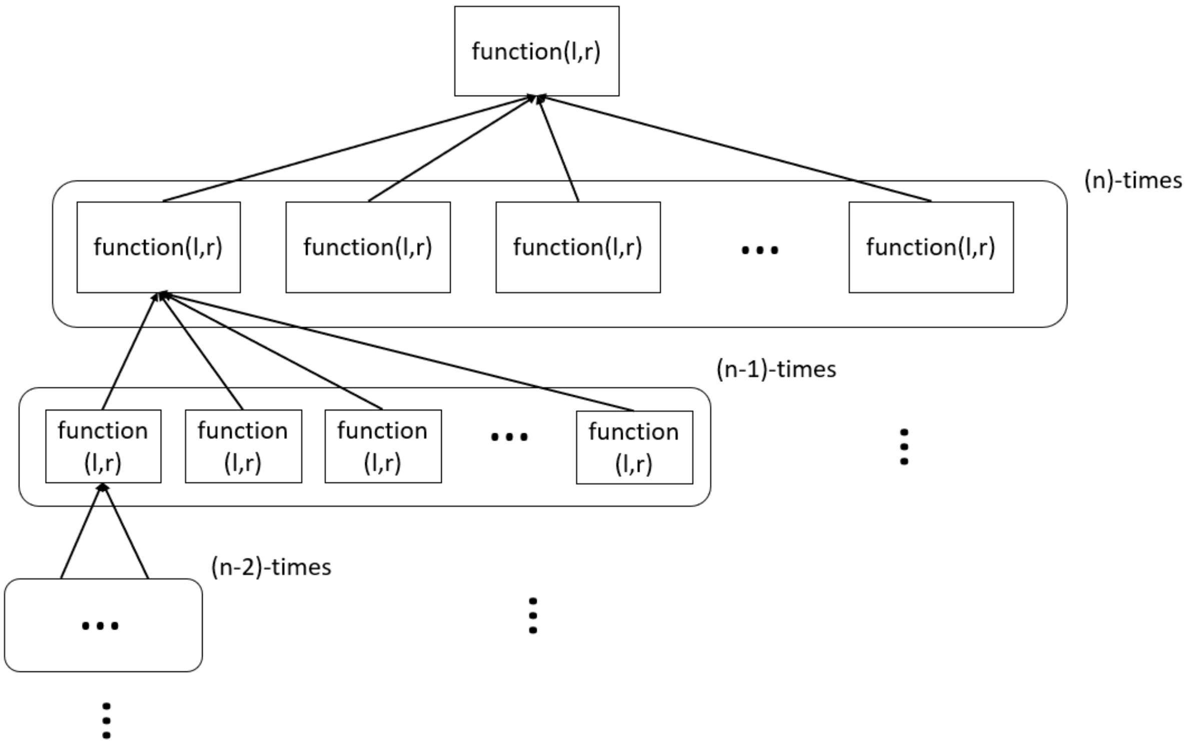

Figure 1: Recursion tree generated by the function of Task-1

Task 2

Idea of Solution

1. Calculate the area occupied by the three triangles formed by joining the given point and the vertices of the triangle. The area of triangle is calculated using $area = \frac{1}{2} * \begin{bmatrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x3 & y3 & 1 \end{bmatrix}$
2. Calculate the area of the triangle with its original vertices.
3. If the summation of the area of the comprising triangles is equal to the area of the triangle then return true i.e. the point falls within the triangle. The case is demonstrated in Figure 2 (a).
4. If the summation of the area of the comprising triangles is greater than the area of the triangle then return false i.e. the point falls outside the triangle. The case is demonstrated in Figure 2 (b).

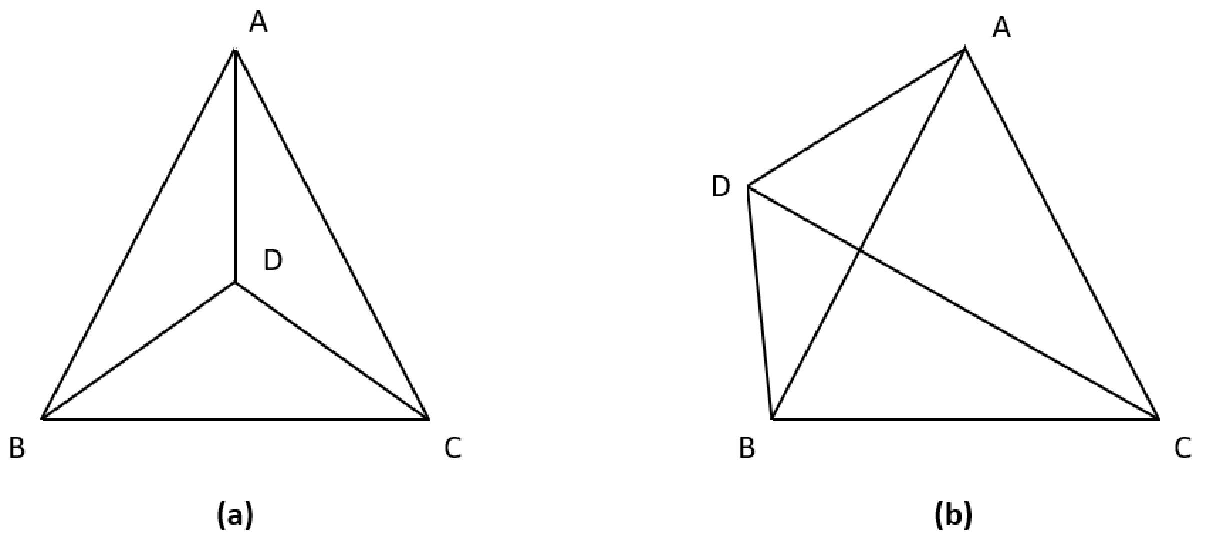

Figure 2: Two cases of Task-2 where the point falls within the triangle (a) and the point falls outside the triangle (b)

Pseudocode

```
# The vertices are 2-tuples accessed by A[x] and A[y]

calc_area(A, B, C)
    return 0.5*(A[x]*(B[y] - C[y]) + B[x]*(C[y] - A[y]) + C[x]*(A[y] - B[y]))

if calc_area(A,B,D) + calc_area(A,C,D) + calc_area(B,C,D) = calc_area(A,B,C):
    return True
else
    return False
```

Complexity of the Algorithm

The operations required to calculate the area of the triangle, summing them up, and checking the *if* condition is input-independent and requires $O(1)$ time complexity. Overall time complexity is $O(1)$.

Task 3

The function is composed of two loops – an outer loop that runs over n inputs and an inner loop that runs for n which decreases by 1 after each iteration of the outer loop. The operations in the inner loop have $O(1)$ time complexity. Hence, the whole function has $n+(n-1)+(n-2)+(n-3)+...+2+1 = \sum_{n=1}^{n} n = \frac{n*(n-1)}{2}$. So, the time complexity is $O(\frac{n*(n-1)}{2}) = O(n^2)$. Similar to Task 1, the best and average case time complexities are also $\Omega(n^2)$ and $\Theta(n^2)$

Task 4

Idea of $O(n^2)$ algorithm

1. Iterate over the numbers in the list.
2. For each number i, if it is greater than the target number, skip it.
3. If the number i is smaller than the target number, then iterate over the list again, and for each number j, if the summation of i and j equals the target number, return the indices of i and j.

Pseudocode of $O(n^2)$ algorithm

```
for i = 0,1 ....(len(nums) - 1):
    if nums[i] > target:
        continue
    for j = 0,1 ....(len(nums) - 1):
        if nums[i] + nums[j] = target:
            return i, j
```

Complexity of $O(n^2)$ algorithm

As there is a nested for loop – each running over n inputs, the time complexity is $O(n^2)$

Idea of the better algorithm

1. We use a hashmap where we store the numbers as keys and their indices as values as seen in Figure 3.
2. Instead of using the inner loop, we access the $target - currentnumber$ using the hashmap.
3. If the index is found in the hashmap, then the indices are of the current number and the index found from the hashmap is returned.

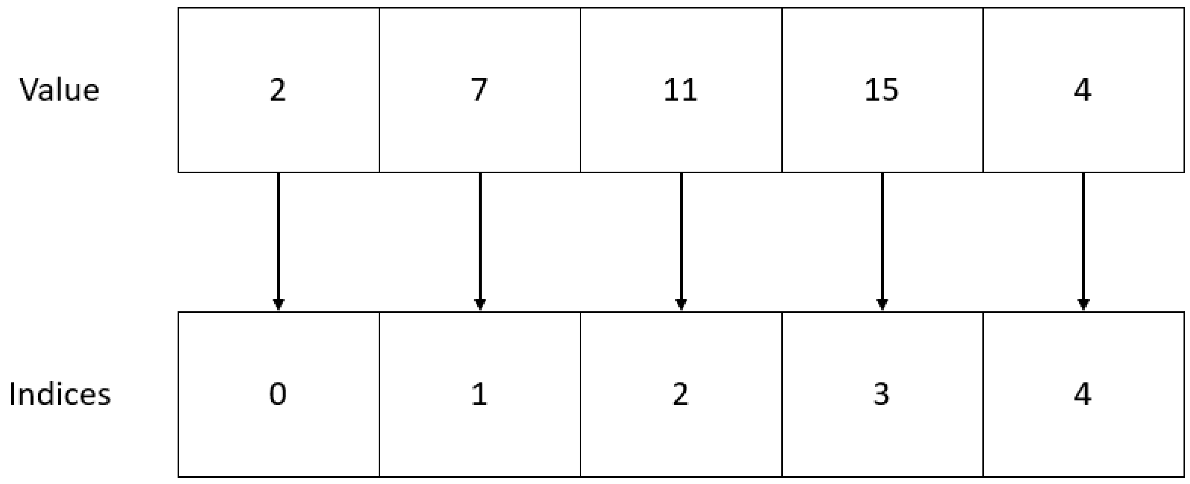

Figure 3: The idea of the hashmap used for Task-4

Pseudocode of the better algorithm

```
for i = 0,1 ....(len(nums) - 1):
    map[nums[i]] <- i
for i = 0,1 ....(len(nums) - 1):
    if nums[i] > target:
        continue
    remaining = target - nums[i]
    if remaining in map:
        return i, map[remaining]     #Returning the indices
```

Complexity of the better algorithm

The algorithm has the outer loop running for n iterations and for each iteration, the hashmap is used to find the number taking $O(1)$ time. Overall time complexity is $O(n)$.

Task 5

Idea of the algorithm

1. We label the leftmost rectangle as A, B, C, and D and label the rightmost rectangle as W, X, Y, and Z.

2. If the corners of the two rectangles overlap which can be seen in the two cases of figure-2, then we return True i.e. the rectangles overlap. Otherwise, we return False.

3. The corner will overlap if $Dx > Wx$ and $Wy > Dy$. Or, if $Bx > Yx$ and $By > Yy$ where x, y represents the co-ordinates of each vertex.

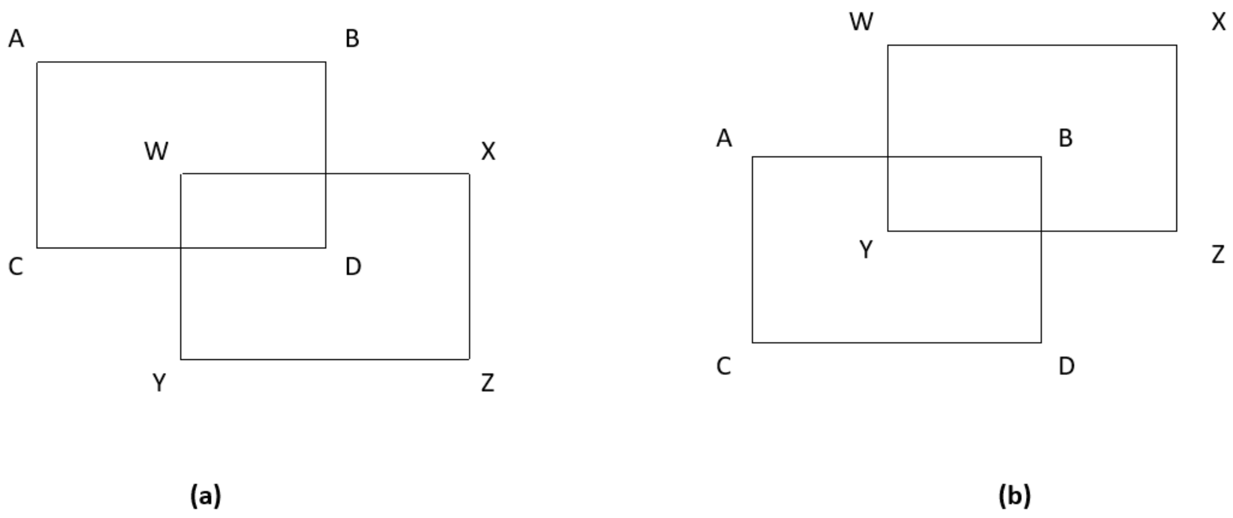

Figure 4: The two cases of rectangle overlapping in Task-5 where the leftmost rectangle is seen above (a) or below (b) the rightmost rectangle

Pseudocode

```
if (D[x] > W[x] and W[y] > D[y]) or (B[x] > Y[x] and B[y] > Y[y])
   return True
else
   return False
```

Complexity of the Algorithm

As a single *if* condition is checked the time complexity is $O(1)$.