**Part-B**

**Question - a**

For this solution, 1-based indexing is used

The recursive formulation for the above problem can be represented as:

dp[i] = max(dp[i-1], dp[i-w-2]+V[i]) if i-w-2 > 0 & ceil(sqrt(V[i])) <= W

dp[i-1] if ceil(sqrt(V[i])) > W & i > 1

max(V[i], dp[i-1]) if i-w-2 <= 0 & ceil(sqrt(V[i])) <= W & i > 1

V[i] if i == 1 & ceil(sqrt(V[i])) <= W &

0 otherwise

The pseudocode of the algorithm can be written as :

function attackRec(V, n, W, i, dp, prev)

if dp[i] != -1

return dp[i]

end if

dp[i] = prev[i] = 0

w = ceil(sqrt(V[i]))

if w > W then

if i > 1 then

dp[i], prev[i] = attackRec(i-1)

end if

return dp[i], prev[i]

end if

if i == 1 then

dp[i] = V[i]

return dp[i], 1

end if

attackDp, attackPrev = attackRec(i-1)

currAttck = attackPrev

if i > (w+2) then

attackDp2, attackPrev2 = attackRec(i-w-2)

if attackDp > V[i] + attackDp2

dp[i] = attackDp

prev[i] = attackPrev

else

dp[i] = V[i]+attackDp2

prev[i] = attackPrev2

currAttck = i

endif

else

if attackDp > V[i]

dp[i] = attackDp

prev[i] = attackPrev

else

dp[i] = V[i]

prev[i] = 0

currAttck = i

endif

end if

return dp[i], currAttck

end function

ans, lastAttck = attackRec(n)

index = lastAttck

while index > 0 do

tis.add(index)

ws.add(ceil(sqrt(V[index])))

index = prev[index]

end while

As we are iterating through the vehicles array only once, the time complexity of the solution is O(n). “previous” array is used to store the index of attack hence it takes a space complexity of O(n).

When the input is given as n = 13, v = [2; 3; 0; 1; 4; 4; 1; 0; 4; 0; 1; 6; 4], W = 3, the dp table is shown below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| dp[1] | dp[2] | dp[3] | dp[4] | dp[5] | dp[6] | dp[7] | dp[8] | dp[9] | dp[10] | dp[11] | dp[12] | dp[13] |
| 2 | 3 | 3 | 3 | 6 | 7 | 7 | 7 | 10 | 10 | 10 | 13 | 14 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| prev[1] | prev[2] | prev[3] | prev[4] | prev[5] | prev[6] | prev[7] | prev[8] | prev[9] | prev[10] | prev[11] | prev[12] | prev[13] |
| 0 | 0 | 2 | 2 | 1 | 2 | 6 | 6 | 5 | 9 | 10 | 6 | 9 |

After running the function, it will return dp[n] = 14 which is the maximum damage, and the last attack as 13 following which

tis array will contain {13, 9, 5, 1} which are the time of attacks and

ws will contain {2, 2, 2, 2} the amount of ammo the drone needs to carry

**Question - b**

The time complexity of this recursive formulation will be O(n\*M)

1-based indexing

dp[i][k] = max(dp[i-1][k], dp[i-w-2][k-w]+V[i]) if i-w-2 > 0 & w <= W & k >= w

dp[i-1][k] if w > W & i > 1

max(V[i], dp[i-1][k]) if i-w-2 <= 0 & w <= W & i > 1 & k >= w

V[i] if i == 1 & w <= W & k >= w

0 otherwise

where w = ceil(sqrt(V[i]))