Name: Le Thanh Cong

Student ID: 20245998

Programming – TP1

Ex1:

```
# Idea: at index i, we can jump maximum T[i] steps,
# -> So we find the index in range [i+1, i + T[i]] that i + T[index] is max
# -> Do that until we reach the end
# Be careful with T[i] = 0, never come here

def find_mininum_steps(arr):
    l_arr = len(arr)
    # Start at index 0
    idx = 0

    step_count = 0
    while(idx < l_arr):
        # If we can't pass through the 0 case, print error and return
        # This case will not appear in the test cases because the
        # exercise assume that the final index is always reachable
        if arr[idx] == 0:
            print("This array can't reach to the end")
            return -1
        # Start from the idx + 1 and stop at idx + T[i]
        start = idx + 1
        stop = idx + arr[idx]
        # stop >= l_arr-1 means we find the way to the end,
        # stop here and return step_count + 1
        if stop >= l_arr-1:
            return step_count + 1

        # Find the maximum of idx + T[idx]
        max = (idx + 1) + arr[idx+1]
        idx = idx + 1
        for i in range(start+1, stop+1):
            # Step over the 0 case
            if arr[i] == 0:
                continue

            if max < i + arr[i]:
                max = i + arr[i]
                idx = i
        # Increase counter
        step_count += 1
    return step_count
# Example
arr = [2,3,0,1,4,1,1,1,0,1]
print(find_mininum_steps(arr))
```

Ex2:

```
# ideas: Calculate all possible distances, store them in sorted list (ascending order) which has at most k elements
# -If new distance is smaller than the last elements, insert it the list (follow the sorted order)
# and pop the last element (if the length of the list is greater than k)
# -At the end, return the last element of the list


# Import this library to create a sorted list
import bisect


def calculate_distance(x, y):
    if (x > y):
        return x - y
    return y - x


def find_kth_smallest_distance(arr, k):
    sorted_list = []

    # 2 loop for calculating the distance and insert them in the sorted_list
    for i in range(len(arr)-1):
        for j in range(i+1, len(arr)):
            # always insert the first distance
            if len(sorted_list) == 0:
                sorted_list.append(calculate_distance(arr[i], arr[j]))
                continue

            # If the distance is smaller than the last element, insert it to the sorted_list
            if calculate_distance(arr[i], arr[j]) < sorted_list[-1]:
                bisect.insort(sorted_list, calculate_distance(arr[i], arr[j]))
                # If length of the sorted_list > k, pop the last element (we only care about the first k-th
smallest distances)
                if len(sorted_list) > k:
                    sorted_list.pop()


    return sorted_list[-1]


arr = [10, 4, 2, 9, 1, 4, 6]
k = 6
print(find_kth_smallest_distance(arr, k))
```

Ex3:

```
# Idea: The monster comes for the knight
# We will use BFS algorithm
# Go from bottom right to top left, and we can only go up and go left
# The monster start from 0 health and it can be greater 0 (if it does, set it be 0)
# The result will be 1 - visited[0][0] because we need the health of the knight is greater than 0


def save_the_pricess(board):
    m = len(board)
    n = len(board[0])


    # matrix that store the cost to go from bottom to all locations
    visited_matrix = [[-1e9 for _ in range(n)] for _ in range(m)]
    # Implement BFS using queue
    queue = []
    queue.append([m-1, n-1])
    visited_matrix[-1][-1] = board[-1][-1]
    while(len(queue) > 0):
        x, y = queue.pop(0)

        # Go up if possible
        if (x-1 >= 0):
             # If health of the monster is greater than 0, set it to 0
            tmp = visited_matrix[x][y] + board[x-1][y] if visited_matrix[x][y] + board[x-1][y] < 0 else 0
            if tmp > visited_matrix[x-1][y]:
                visited_matrix[x-1][y] = tmp
                queue.append([x-1, y])

        # Go left if possible
        if (y-1 >= 0):
            tmp = visited_matrix[x][y] + board[x][y-1] if visited_matrix[x][y] + board[x][y-1] < 0 else 0
            if tmp > visited_matrix[x][y-1]:
                visited_matrix[x][y-1] = tmp
                queue.append([x, y-1])
    print(visited_matrix)

    return 1 - visited_matrix[0][0]

board = [[-2, -3, 3],
         [-5, -10, 1],
         [10, 30, -5]]
print(save_the_pricess(board1))
```