Case Study

1) Write an algorithm that suits to solve the problem?

Sol Step 1 :- Divide the pile of socks into two roughly equal parts

Step 2 :- Each group of campers independently sorts their respective piles of socks using any efficient sorting algorithm

step3 :- Merge the 2 sorted piles into single sorted pile, Combine the sorted piles in a way that maintains the sorted order, Use a merge algorithm, similar to the one used in Mergesort.

step4 :- If there are more than 1 pile, continue step1,2,3 again

step 5 :- The final result is a completely sorted list of socks.

2) Determine the no:of phases needed for PLP to arrange the socks for faster wash.

Sol The no:of phases needed for PLP is determined by the no:of recursive calls in the algorithm. Each recursive call represents a phase. In this case, since the algorithm divides the piles into 2 halves at each level, the no:of phases is $\log_2(n)$, where n is the no:of socks.

3) Write a pseudocode algorithm outlining the sorting process, considering individual sorting & merging phase.

Sol

```
function PLP_Sort(socks):
    if length(socks) <=1 :
        return socks

mid = length(socks)/2
left_pile = socks [0 to mid-1]
right_pile = socks [mid to end]
```

```
    left_pile = PLP_Sort(left_pile)
    right_pile = PLP_Sort(right_pile)
    result = merge(left_pile , right_pile)
    return result

function merge(left, right):
    result = [ ]
    left_index =0
    right_index=0
    while left_inder <length(left) & right_index <length(right):
        if left[left_index] < right[right_index]:
            result.append(left[left_index])
            left_index++
        else:
            result.append(right[right_index])
            right_index++
    result.extend(left[left_index:])
    result.extend(right[right_index:])
return result.
```

4) Derive the recurrence relation of above algorithm?

Sol) The recurrence relation for the given algorithm is:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)$$

This recurrence relation describes the time complexity of the PLP sorting algorithm. The $2 \cdot T\left(\frac{n}{2}\right)$ term represents the time to sort & merge 2 sub-piles, & $O(n)$ represents the time to merge the 2 sorted sub-piles.