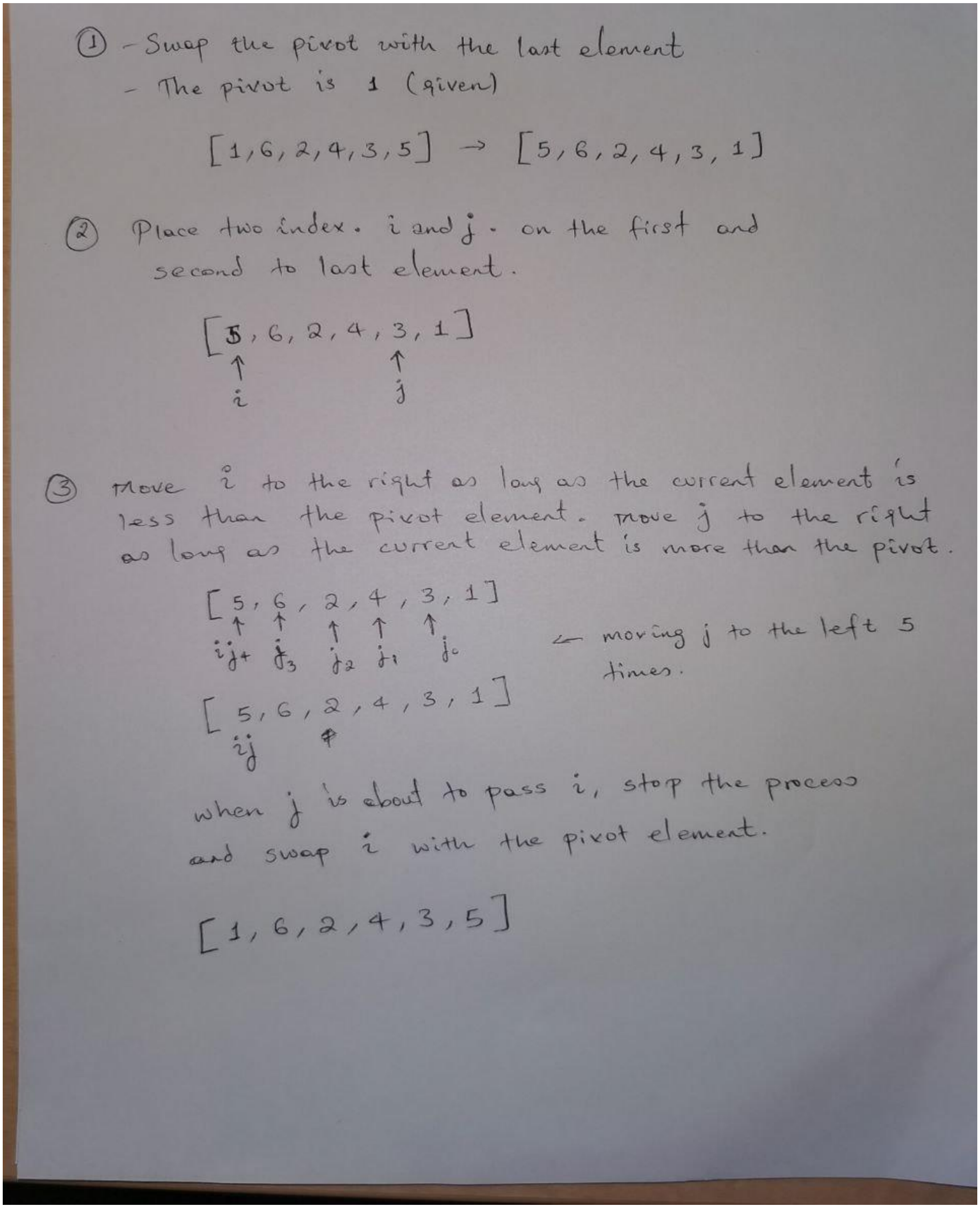
Lab assignment solution

Problem 1:



① - Swap the pivot with the last element
- The pivot is 1 (given)

[1, 6, 2, 4, 3, 5] → [5, 6, 2, 4, 3, 1]

② Place two index. i and j. on the first and second to last element.

```
[5, 6, 2, 4, 3, 1]
 ↑           ↑
 i           j
```

③ Move i to the right as long as the current element is less than the pivot element. Move j to the right as long as the current element is more than the pivot.

```
[5,  6,  2,  4,  3, 1]
 ↑   ↑   ↑   ↑   ↑
 ij4 j3  j2  j1  j0        ← moving j to the left 5
                             times.
[5,  6,  2,  4,  3, 1]
 ij      ₱
```

when j is about to pass i, stop the process and swap i with the pivot element.

[1, 6, 2, 4, 3, 5]

Problem 2:

3n/4 = 3*9/4 = 6.75

For x = 5, L = {1, 1, 2, 3, 3, 4 } , R = {6, 7} and in each case |L| < 6.75, |R|<6.75, therefore 5 is a good pivot

For x = 6 and 7, |R| > 6.75, and therefore is not a good pivot

For x = 4, R = {5,6,7}, L = {1,1,2,3,3}, |R| < 6.75 and |L|<6.75 therefore is a good pivot

For x = 3, R = {4,5,6,7}, L = {1,1,2} again is a good pivot

For x = 2, R = {3,3,4,5,6,7} and L = {1} and is a good pivot point

For x = 1, R = {2,2,3,3,4,5,6,7} and |R|>6.75 and is not a good pivot

a. So to recap: x in $Bad = [1,6,7]$ are not a good pivot but x in $Good\ = \{2,2,3,4,5\}$ are good pivot, there are 3 bad pivot, and 5 good pivot.
b. 5 out of 9 are good pivots, 5 is more than half of 9 so, yes, the number of pivot is more than half.

Problem 3:

1. **Valid recursion**: Base case is provided, i.e., when n <= 1. And each recursion leads to base case because on each stage we are subtracting 1.
2. **Correct base case:** when $n \leq 0$, it is correctly returned because $0! = 1! = 1$
3. **Recursion steps correct:** Assuming the recursion gives the correct result for $1 \leq n \leq k$, for some k, its then its true that it gives the correct result for $n = k + 1$ because,

$$recursiveFactorial(k + 1) = (k + 1) * recursiveFactorial(k)$$

Problem 4:

```
Algorithm SubsetSum(S, k)
    Input: a set S of n elements, and a number k which is the sum
    Output: a subset of S whose elements sum up to k.
    elem <- S.getLast()
    if(k = 0)
        return emptySet()
    if(S.isEmpty() && k != 0)
        return null
    S.removeLast()
    if(elem>k)
        return SubsetSum(S, k)
    include <- SubsetSum(S, k)
    exclude <- SubsetSum(S, k-elem)
    if(include != null)
        include.addLast(elem)
        return include
    return exclude
```