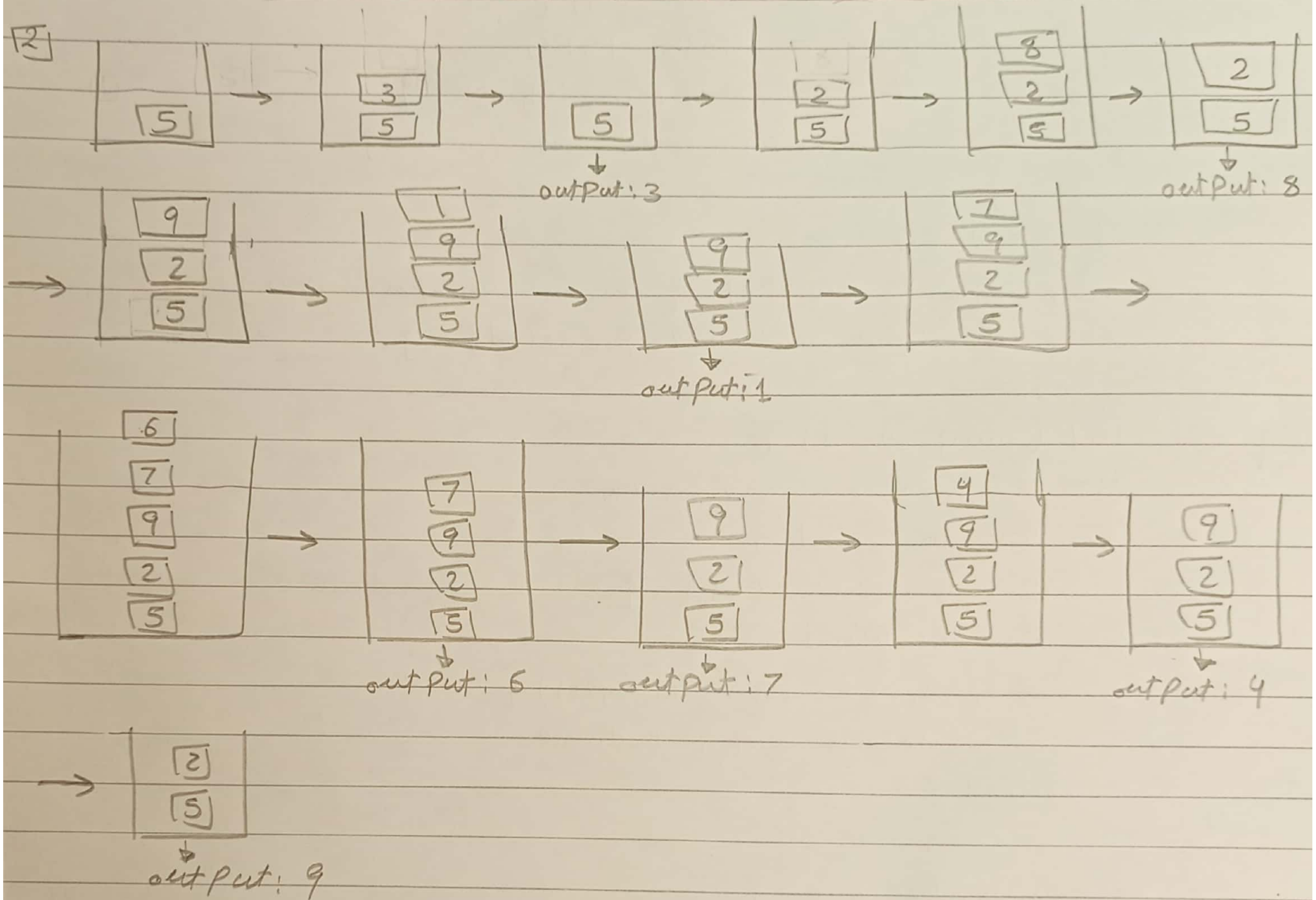
علي مصطفى النكلاوي
1916024

[1] a) 25 elements and 10 PoP operations (3 of them were invalid) → 25 - (10 - 3) = 25 - 7 = 18

S contains (18) elements

b) The toP element is at index (17)

[2]



[3]

```
Size(stack S) {        // S is the desired stack
    len ← 0
    stack temp ← new stack
    While ( S is not empty) {
        temP. Push(S. PoP())
        len ++ }
    ~~While (temP is not empty) {~~
        ~~S. Push (temP. PoP()) }~~

    return len }
```

```
[4]  Check Valid (String doC) {
         Stack tags ← new Stack
         for Ch in doC {
             if (Ch is open tag) {
                 tags . Push (Ch) }
             elif (Ch is Close tag) {
                 curr ← tags . PoP()
                 if (curr Not match Ch) {
                     return Not-Well-formated }
             } //end elif
         } // end for
         if (tags is empty) { return well-formatted }
         else { return Not-well-formatted }  }
```

```
[5]  Check Pal (String Word) {
         Stack Chars ← new Stack
         for Ch in Word {
             Chars . Push (Ch) }
         for Ch in Word {
             if (Ch != Chars . PoP()) {
                 return Not Palindrome
             } // end if
         } // end for
         return Palindrome }
```

```
[6]
(a)  CoPy (stack S) {                          // S = 1 2 3 4
         Stack final ← new Stack
         Stack temp ← new Stack
         stArr ← []
         While (not isEmpty (S)) {
             stArr. append( S. PoP() ) }       // [4 3 2 1]
for elmnt in stArr {
    temp.push(elmnt) }        // temp = 4 3 2 1
while (Not isEmpty(temp)) {
      top ← temp.Peak()
      final.push(top)
      S.push(top)              // S = 1 2 3 4
      temp.pop() }
} return final }
```

```
[b] reverseCpy(Stack S) {
        Stack temp ← new Stack
        Stack final ← new Stack
        While (Not isEmpty(S)) {
              top ← S.pop()
              final.push(top)
              temp.push(top)
        }
        While (Not isEmpty(temp)) {
              S.push(temp.pop()) }
        return final }
```

```
[c] sort (Stack S) {
        cpy ← copy(S)        // copy(S) returns a copied version of S
        max ← -Inf
        arr ← []
        While (Not is Empty(cpy)) {
              arr.append(cpy.pop()) }
        While (arr is not empty) {
              for elmnt in arr {
                    if elmnt > max {
                          max ← elmnt } }
              final.push(max)
              arr.remove(max)
              max ← -Inf }        return final }
```