Chapter 1 & 2 → Homework

Exercise 2. 2-1

$\frac{n^3}{1000} - 100n^2 - 100n + 3$ in terms of big (O)

a) Find leading term → $\frac{n^3}{1000}$

b) n^3 → simplify

① ~~Retu~~ Return (c) {

```
x = c
y = 0
while (x > 0) {
    x = x-1
    y = y+1
}
return y
}
```

Prove this returns c

Return (2)

```
x = 2
y = 0
while (x > 0) {
    x = 2-1 => 1
    *
    y = y+1 => 1
}
return y
}
```

↓

x = 1

x = 1-1 => 0

y = 1+1 => 2

exit the loop

returns 2 ✓

1) Continued.. loop invariant $\boxed{x+y=C}$

Initialization

- Before the loop starts, $x=C$ and $y=0$
- Substitute these values in the invariant

$$x+y = C+0 = C$$

- loop invariant holds true after this iteration

Maintenance

- Assume the loop invariant $x+y=C$ holds true before any arbitrary iteration of the loop.
- During the iteration
  the loop ~~drives~~ decreases 'x' by 1 ($x=x-1$)
  the loop increments 'y' by 1 ($y=y+1$)

After the update — Calculate $x+y$ after the update

$x=x-1$
$y=y+1$

$(x-1)+(y+1) = x+y+C$

loop invariant holds true after the iteration.

Termination

the loop terminates when $x=0$. The loop condition $x>0$ is no longer ~~affected~~ satisfied.

Substituting the invariant : $x+y=0+y=C$

$\boxed{y=C}$

① Continued ..

Since $y = c$ when the loop terminates, the function ~~code~~ correctly ~~ret~~ returns 'c'. Thus, the function is correct and returns the initial value of 'c'.

---

②

```
prod (a,b) {
    x = a
    y = b
    z = 0
    while (x > 0) {
        z = z + y
        x = x - 1
    }
    return z
}
```

Prove this returns a * b

```
prod (2,3) {
    x = 2
    y = 3
    z = 0
    while (x > 0) {
        z = 0 + 3 => 3
        x = 2 - 1 => 1
    }
    return z
}
```

returns a + b ✓

Loop Invariant

~~z~~ $x \quad z + x * y = a * y$

Initialization

$x = a$
$y = b$
$z = 0$

$z + x * y = a * b$
loop invariant holds true before iteration.

(2) Continued Maintenance

Assume the invariant is true at the start of the iteration.

$$z + x * y = a * y$$

During the loop Invariant : ~~$(z * y) + (x - 1)$~~

$z = z + y$

$x = x - 1$

~~$(z * y) + x$~~

$$(z * y) + (x - 1) * y = a * y$$

Simplify

$$z + y + x * y = a * y$$

$$z + x * y = a * y$$

The loop invariant holds true.

Termination

The loop terminates when $x = 0$. At this point:

~~$z + 0$~~ $\quad z + 0 * y = a * y$

$$z = a * y$$

x will be zero at termination.

(2) Continued..

Conclusion

The loop invariant $z + x * y = a * y$ holds true throughout the execution of the loop, ensuring that the function performs the multiplication through the repeated additions.

~~that proving~~

(3) Bubble sort (A)

```
for i=1 to A.length-1 {
    swapped = False
    for j= A.length down to i+1 {
        if A[j] < A[j+1] {
            swap (A[j], A[j+1])
            swapped = true
        }
    if (!swapped) {
        break
    }
}
```

(A) Demonstrate how the alg works [5, 1, 4, 2, 8]

First Pass:

Compare 5 & 1 : swap → [1, 5, 4, 2, 8]
Compare 5 and 4 : swap: [1, 4, 5, 2, 8]
Cmp 5 & 2 : swap: [1, 4, 2, 5, 8]
Cmp 5 & 5 : no swap
break

Second pass

Compare 1 & 4: No swap → [1, 4, 2, 5, 8]
Compare 4 & 2: swap → [1, 2, 4, 5, 8]

the remaining comparisons will not have any swaps. Then loop breaks

Third pass

Compare 1 & 2 : No swap → [1, 2, 4, 5, 8]

There are no more swaps in the pass.

The array is now sorted

4th pass → no swaps, program terminates

There are no more passes, The array is sorted in ascending order

(b) Given $t(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n}$ for the above algorithm, find a formula for this by reducing the ~~down to~~ well known Summation down to more familiar formulas.

Simplify

$$t(n) = \sum_{i=1}^{n-1} (n-i)$$

Simplify outer summation

$$t(n) = \sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i$$

The first term is a constant. 'n' repeated 'n' times.

~~The~~

$$\sum_{i=1}^{n-1} (n) = n(n-1)$$

The 2nd term can be simplified ~~as~~ as follows -

$$\sum_{i=1}^{n-1} (i) = \frac{(n-1)(n)}{2}$$

Final Simplification

$$t(n) = n(n-1) - \frac{(n-1)(n)}{2}$$

$$= n(n-1)\left(1 - \frac{1}{2}\right)$$

$$\boxed{t(n) = \frac{n(n-1)}{2}}$$

(c) Given $T(n) = \sum_{j=2}^{n} 1$

- Add number 1 for each integer value of j from 2 to n
- The number of terms in this summation is n-1 because you are summing from $j = 2$ to n.

result: $\sum_{j=2}^{n} 1 \Rightarrow (n-1) * 1 = n-1$

$$\boxed{T(n) = n-1}$$

(4)

(A) Demonstrate how it sorts ~~[64, 25,~~ [64, 25, 12, 22, 11]

Iteration i = 1

initial min index, min = 0 (A[0] = 64)

inner loop (j = 1 to 4):

compare A[1] (25) with A[min] (64)
⇒ 25 < 64, min = 1

- compare A[2] (12) with A[min] (25)
⇒ 12 < 25, min = 2

- compare A[3] (22) with A[min] (10)
⇒ 22 > 10, min = 3

- compare A[4] (11) with A[min] (12)
⇒ 11 < 12, min = 4

- ~~compare A~~
- swap A[0] and A[4] (swap 64 and 11)

Iteration 2

initial min index = 1 (A[1] = 25)

inner loop (j = 2 to 4):

- compare A[2] (12) with A[min] (25)
⇒ 12 < 25, min = 2

- Compare A[3] (22) with A[min] (12)
  ⇒ 22 > 12, min remains as 2
- Compare A[4] (64) with A[min] (12)
  ⇒ 64 > 12, min remains as 2
- Swap A[1] and A[2] (swap 25 and 12)

Iteration 3

initial min index = 2 (A[2] = 25)

inner loop (j = 3 to 4)

- Compare A[3] (22) with A[min] (25)
  ⇒ 22 < 25, min = 3
- Compare A[4] (64) with A[min] (22)
  ⇒ 64 > 22, min remain 3
- Swap A[2] and A[3] (Swap 25 & 22)

Array at present: [11, 12, 22, 25, 64] = sorted

Iteration 4

no change to be made in any comparison since array is already sorted.

Array after 4th iteration = [11, 12, 22, 25, 64]

Array is now fully sorted using selection sort.

4 (B) Tim Analysis

Step 1 (for i = 1 to A.length - 1)

- cost : c_1
- Time : n ~~the to~~

Step 2 (min = i)

- cost : c_2
- time : n-1 (n-1 times loop iterations, which is approximately n)

Step 3 (for j = i+1 to n)

- cost: c_3
- time : the inner loop runs n-1 ~~times~~ times. Over the ~~the~~ entire algorithm, this is the sum of all(n-i) from i=1 to n-1.

Total time: $\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} i$

Simplified time : $\frac{(n-1)(n)}{2}$

Step 4 (if {A[j] < A[min])

cost: c_4

time: The comparison is due $\sum_{i=1}^{n-1}(n-1)$ times.

Simplified time = $\frac{n(n-1)}{2}$

4(C) Find a formula for T(n)

Sum up time complexities of all steps

$$T(n) = c_1(n) + c_2(n-1) + c_3 \sum_{i=1}^{n-1} (n-i) + c_4 \sum_{i=1}^{n-1} (n-i) + \cdots$$

Simplify this summation

$$\sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2}$$

$$T(n) = c_1(n) + c_2(n-1) + c_3 \cdot \frac{n(n-1)}{2} + \cdots$$

$$T(n) \approx (c_3 + c_4) \frac{n(n-1)}{2} \approx O(n^2)$$

$$\boxed{T(n) = O(n^2)}$$

Therefore, the time complexity of selection sort is $O(n^2)$.