Assignment No. 1

Course Title: Advanced Algorithm

Course No: CSE 511

Submitted to: Mohammad Mahmudul Alam (MLD)

Submitted by: Sumaiya Tarannum Noor

Section: 02

ID: 2425410650

Submission date: 20th October, 2025

CSE-511
Assignment_01

Answer to the Question No. 1

(a) Loop Invarient:

At the start of every iteration of the loop with index i, the algorithm has already checked all elements in position 0, 1, ..., i-1 and none of them are equal to x.

(b) Correctness Proof using Invarient Method -

We prove the invarient holds through Initialization, Maintenance, and Termination.

(1) Initialization

Before the first iteration, i=0.
No elements have been checked yet - so it is trivially true that none of A[0 ... -1] contain x.
Thus, the invarient holds before the loop starts.

(2) Maintenance

Assume that at the start of iteration i, the invarient holds, i.e., A[0 ..... i-1] has been checked and contains no x.

Now, in this iteration, the algorithm checks A[i].

- If A[i] == x, the algorithm returns i, which is correct because we found x at index i.
- If A[i] != x, then now indices 0....i have been checked and none contains x, so the invariants holds for the next iteration (i+1).

Thus the invariant is maintained.

③ Iteration.

The loop terminates in two ways:

Case 1 - Early return:

If for some i, we find A[i] == x, the algorithm return i.

Since at that moment this is exactly the index where x appears, this is correct.

Case 2 - Loop ends normally (i = n):

The loop finishes without returning.

By the invariant, all elements in A[0.... n-1] have been checked and none is equal to x.

Thus, x is not in the array - so returning -1 is correct.

Answer to the Question NO:2

We analyze each function by identifying the dominating term for large n, then giving formal Big-O, Big-Ω and Big-Θ bounds with short proofs (constants and n_0).

1) $f_1(n) = 3n^2 + 10n + 5$

Dominant term : n^2

(a) Big-O : $f_1(n) = O(n^2)$

Proof: For $n \geq 1$ we have $10n \leq 10n^2$ and $5 \leq 5n^2$. So,

$$f_1(n) = 3n^2 + 10n + 5 \leq 3n^2 + 10n^2 + 5n^2 = 18n^2$$

Take $c = 18$ and $n_0 = 1$. Then, $f_1(n) \leq cn^2$ for all $n \geq n_0$.

(b) Big-Ω : $f_1(n) = \Omega(n^2)$

Proof: For all $n \geq 0$,

$$f_1(n) = 3n^2 + 10n + 5 \geq 3n^2$$

Take $c = 3$ and $n_0 = 0$. Then $f_1(n) \geq cn^2$ for all $n \geq n_0$.

(c) Big-Θ : Since $f_1(n)$ is both $O(n^2)$ and $\Omega(n^2)$, we have $f_1(n) = \Theta(n^2)$

2) $f_2(n) = 2n + n^3$

Dominant term: n^3

(a) Big-O: $f_2(n) = O(n^3)$

Proof: for $n \geq 1$, $2n \leq 2n^3$ because $n^3 \geq n$. Thus,

$$f_2(n) = 2n + n^3 \leq 2n^3 + n^3 = 3n^3$$

Take $c = 3$, $n_0 = 1$

(b) Big-Ω: $f_2(n) = \Omega(n^3)$

Proof: for all $n \geq 0$

$$f_2(n) = 2n + n^3 \geq n^3$$

Take $c = 1$, $n_0 = 0$

(c) Big-Θ: Because both upper and lower bounds hold, $f_2(n) = \Theta(n^3)$

3) $f_3(n) = n \log n + 20n$

Here $\log n$ denotes logarithm in any fixed base - asymptotic classes are base-independent up to constant factors.)

Dominant term: $n \log n$ ~~$+ 20n$~~ (since $\log n \to \infty$ as $n \to \infty$, it eventually dominates the linear term).

(a) Big-O: $f_3(n) = O(n \log n)$

Proof: For $n \geq 2$, $\log n \geq 1$. So $20n \leq 20 n \log n$.

Therefore,

$$f_3(n) = n \log n + 20n \leq n \log n + 20 n \log n = 21 n \log n.$$

Take $c = 21$, $n_0 = 2$

(b) Big-Ω: $f_3(n) = \Omega(n \log n)$

Proof: Obviously $20n \geq 0$, so for all $n \geq 1$,

$$f_3(n) = n \log n + 20n \geq n \log n$$

Take $c = 1$, $n_0 = 1$

(c) Big-Θ: Since $f_3(n)$ is both $O(n \log n)$ and $\Omega(n \log n)$, we conclude $f_3(n) = \Theta(n \log n)$.

Answer to the Question No. 3

(a) Recurrence relation

Each pass (finding the minimum and swapping) takes linear time $\Theta(n)$ and then we recurse on $n-1$ elements. A natural recurrence is

$$T(n) = T(n-1) + c \cdot n \text{ for } n \geq 2, T(1) = d$$

where $c > 0$ and $d > 0$ are constans representing cost per element for the scan and the base-case cost.

(b) Solve by recursion-tree method

Draw the recursion tree: root cost $= c \cdot n$. It's child cost $= c \cdot (n-1)$. Next level $c \cdot (n-2)$ until $c \cdot 1$

Summing costs accross all levels:

$$T(n) = c(n + (n-1) + (n-2) + \cdots + 2) + T(1).$$

The sum of the first n positive integers is $n(n+1)/2$. Our sum $2 + 3 + \cdots + n = \frac{n(n+1)}{2} - 1$.

So,

$$T(n) = T(1) + c\left(\frac{n(n+1)}{2} - 1\right) = \frac{c}{2}n^2 + \frac{c}{2}n + (T(1) - c).$$

Therefore,

$$T(n) = \Theta(n^2)$$

So, selection sort runs in quadratic time.

(c) Verify by substitution (induction)

We show both an upper bound $T(n) = O(n^2)$ and a lower bound $T(n) = \Omega(n^2)$ to conclude $\Theta(n^2)$.

Upper bound (O)

There exists $c > 0$ and ~~n_0~~ n_0 such that for all $n \geq n_0$, $T(n) \leq cn^2$.

Use the recurrence $T(n) = T(n-1) + cn$ and assume induction hypothesis $T(k) \leq ck^2$ for all $k < n$.

Then

$$T(n) = T(n-1) + cn \leq c(n-1)^2 + cn = c(n^2 - 2n + 1) + cn$$
$$= cn^2 + (-2c + c)n + c$$

choose c large enough so the linear term is non-positive for all $n \geq 1$. For example, choose $c \geq c$. Then, ~~$-2c+c$~~ $-2c + c \leq -c \leq 0$, so

$$T(n) \leq cn^2 + c \leq c'n^2$$

for some c' (e.g. $c' = 2c$) and all $n \geq 1$. Base case $T(1) \leq c'$ can be satisfied by choosing c' large enough. Thus, $T(n) = O(n^2)$

Lower bound (Ω)

There exists $c_1 > 0$ and n_1 such that for all $n \geq n_1$, $T(n) \geq c_1 n^2$.

From recurrence and $T(1) = d > 0$, expand:

$$T(n) = d + c\sum_{i=2}^{n} i = d + c\left(\frac{n(n+1)}{2} - 1\right)$$

$$= \frac{c}{2}n^2 + \frac{c}{2}n + (d - c)$$

So for $n \geq 1$, $T(n) \geq \frac{c}{2}n^2 + \frac{c}{2}n + (d - c)$

This is at least $(c/2)n^2$ minus some lower-order terms.

Now, choose a constant $c_1 = c/4$

For sufficiently large n, the quadratic term $\frac{c}{2}n^2$ will domain the linear and constant parts.

Therefore, for those large n, we have,

$$T(n) \geq \frac{c}{4}n^2$$

which matches the definition of an Ω bound.

So, we conclude,

$$T(n) = \Omega(n^2)$$

since we have $T(n) = O(n^2)$ and $T(n) = \Omega(n^2)$

Therefor, $T(n) = \Theta(n^2)$

Answer to the Question No. 5

a. Kth Largest Element in an Array using Divide & Conquer Approach:

```
def kth_largest(nums, k):
    k_index = len(nums) - k

    def quickselect(left, right):
        pivot = nums[right]
        p = left
        for i in range(left, right):
            if nums[i] <= pivot:
                nums[i], nums[p] = nums[p], nums[i]
                p += 1

        nums[p], nums[right] = nums[right], nums[p]

        if p == k_index:
            return nums[p]
        elif p < k_index:
            return quickselect(p + 1, right)
        else:
            return quickselect(left, p-1)

    return quickselect(0, len(nums) - 1)


print(kth_largest([18, 8, 3, 2, 1, 5, 6, 12, 4, 7, 10, 14, 16, 17], 6))
```

Solution:

10

Answer to the Question no. 5

(b) Time complexity

Average case - Random distribution of pivots - balanced on expectation.

$O(n)$

Best case - Pivot splits array into two equal halves every time (balanced partition)

$O(n)$

Worst case - Pivot always smallest / largest - partitions are highly unbalanced.

$O(n^2)$

Space complexity

Best case - Balanced recursion → depth = $\log n$

$O(\log n)$ stack

Average case - Similar to best - typically balanced

$O(\log n)$ stack.

Worst case - fully skewed recursion → depth = n

$O(n)$ stack.

[The End]