Part A: Algorithm Writing & Analysis

1a) Find Maximum and Minimum Element

Pseudocode:

Algorithm FindMaxMin(arr, n)

max ← arr[0]

min ← arr[0]

for i from 1 to n-1 do

if arr[i] > max then

max ← arr[i]

if arr[i] < min then

min ← arr[i]

return (max, min)

Time Complexity: O(n)  
Space Complexity: O(1)  
Optimal? Yes. Can't do better than O(n) for unsorted array.

1b) Count Odd and Even Elements

Pseudocode:

Algorithm CountOddEven(arr, n)

odd ← 0

even ← 0

for i from 0 to n-1 do

if arr[i] mod 2 = 0 then

even ← even + 1

else

odd ← odd + 1

return (odd, even)

Time Complexity: O(n)  
Space Complexity: O(1)  
Optimal? Yes.

1c) Reverse an Array

Pseudocode:

Algorithm ReverseArray(arr, n)

start ← 0

end ← n - 1

while start < end do

swap arr[start] and arr[end]

start ← start + 1

end ← end - 1

Time Complexity: O(n)  
Space Complexity: O(1)  
Optimal? Yes (in-place reversal).

Part B: Asymptotic Analysis

3a)

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

printf("\*");

}

}

Operations: n \* n = n²

Best / Average / Worst Case: All O(n²)

Time Complexity: O(n²)

Space Complexity: O(1)

3b)

void func(int n) {

if (n == 1) return;

func(n/2);

func(n/2);

}

Recurrence: T(n) = 2T(n/2) + O(1)

Master Theorem: a = 2, b = 2 → T(n) = O(n)

Time Complexity: O(n)

Space Complexity: O(log n) (recursion stack)

3c)

int i = 1;

while (i < n) {

printf("%d ", i);

i = i \* 2;

}

i grows exponentially ⇒ 1, 2, 4, 8, ..., up to < n

Number of operations: log₂(n)

Time Complexity: O(log n)

Space Complexity: O(1)

3d)

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

printf("\*");

}

}

}

Operations: n × n × n = n³

Time Complexity: O(n³)

Space Complexity: O(1)

3e)

int fib(int n) {

if (n <= 1) return n;

return fib(n - 1) + fib(n - 2);

}

Time Complexity: O(2^n) (Exponential)

Space Complexity: O(n) (recursive call stack)

Worst-case scenario. (Can be improved with DP to O(n))

3f)

printf("Hello World");

Time Complexity: O(1)

Space Complexity: O(1)

3g)

for (int i = 0; i < n; i++) {

printf("\*");

}

for (int j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

printf("#");

}

}

First loop: O(n)

Second loop: O(n²)

Total: O(n²) (dominant term)