Algorithms and Data Structures – Exercises 2

# Task 1: Algorithm A (T(n) = n²)

Given:  
T(n) = n²  
T(10) = 1 second

* a) T(20) = (20² / 10²) \* 1 = 4 seconds
* b) Max n for 100 seconds: n² = 100 ⇒ n = √100 = 10 \* √10 = 100
* c) T(50) = (50² / 10²) = 25 seconds ⇒ 25 / 5 = 5 seconds

# Task 2: Algorithm B (T(n) = n³)

Given:  
T(3) = 81 ⇒ c = 81 / 27 = 3 ⇒ T(n) = 3n³

* a) T(9) = 3 \* 9³ = 2187 seconds
* b) Max n for 375 sec: 3n³ = 375 ⇒ n³ = 125 ⇒ n = 5
* c) T(9) on 27× faster computer = 2187 / 27 = 81 seconds

# Task 3: Euclidean Algorithm (Subtraction-based)

a) GCD(48, 180) steps:

180 - 48 = 132  
132 - 48 = 84  
84 - 48 = 36  
48 - 36 = 12  
36 - 12 = 24  
24 - 12 = 12  
⇒ GCD = 12

* b) GCD(1424, 3084) – steps performed manually (recommended in notebook)

# Task 4: Euclidean Algorithm (Division-based)

* a) GCD(48, 180): GCD(180,48) → GCD(48,36) → GCD(36,12) → GCD(12,0) = 12
* b) GCD(3084,1424) → GCD(1424,236) → GCD(236,4) → GCD(4,0) = 4
* c) GCD(56, 72): 4 recursive calls
* d) GCD(72, 56): same as (c), 4 recursive calls

# Task 5: Big-O, Theta, and Omega Notation

Given expression: 5n³ + 2n² + 100  
Determine if the following statements are TRUE or FALSE.

* a) 5n³ + 2n² + 100 = O(n³)  
  ✅ TRUE – Dominant term is n³, so it grows no faster than O(n³).
* b) 5n³ + 2n² + 100 = Θ(n³)  
  ✅ TRUE – It grows exactly like n³ (upper and lower bounded).
* c) 5n³ + 2n² + 100 = O(n⁴)  
  ✅ TRUE – n⁴ is a looser upper bound.
* d) 5n³ + 2n² + 100 = O(n¹⁰)  
  ✅ TRUE – Any polynomial of higher degree still bounds it from above.
* e) 5n³ + 2n² + 100 = Θ(n⁴)  
  ❌ FALSE – It grows slower than n⁴, so not tightly bounded.
* f) 5n³ + 2n² + 100 = Ω(n²)  
  ✅ TRUE – It grows faster than n², so this is a valid lower bound.
* g) 5n³ + 2n² + 100 = Ω(n⁴)  
  ❌ FALSE – It doesn’t grow as fast as n⁴.
* h) 200818 = O(1)  
  ✅ TRUE – A constant is always O(1).
* i) lg(n²) = O(log n)  
  ✅ TRUE – log(n²) = 2 log(n), so it is proportional to log n.
* j) (lg n)² = O(log n)  
  ❌ FALSE – (log n)² grows faster than log n.

# Task 6: Θ Notation Classification

* a) 6n + 1 → Θ(n)
* b) 2n² + 1 → Θ(n²)
* c) 6n³ + 12n² + 1 → Θ(n³)
* d) 2n log n + 3n² → Θ(n²)
* e) 2 log n + 10000n + n log n → Θ(n log n)
* f) 2n⁶ + 22222n + 4 → Θ(n⁶)
* g) (6n + 4)(1 + log n) → Θ(n log n)
* h) ((n + 1)(n + 3)) / (n + 2) → Θ(n)
* i) ((n² + log n)(n + 1)) / (n + n²) → Θ(n)
* j) 25 log(2n + 10) → Θ(log n)
* k) 2 + 4 + 8 + ... + 2ⁿ → Θ(2ⁿ)

# Task 7: Operation Count – x ← x + 1

* a) One loop (1 to 2n): → Θ(n)
* b) Nested loop (2n × n): → Θ(n²)
* c) Triple nested (i → j → k): → Θ(n³)
* d) Triple nested (dependent ranges): → Θ(n³)
* e) Triple nested (j and k depend on i): → Θ(n³)
* f) While loop with i // 2: → Θ(log n)
* g) While loop + inner loop (1 to n): → Θ(n log n)

# Task 8: Horner’s Scheme (Manual in Notebook)

* a) p(x) = 2x⁶ - 3x³ + 5x - 1 at x = -2
* b) Same polynomial at x = 3
* c) p(x) = 2x³ - 4x² + 3x + 1 at x = 2
* d) p(x) = -x⁵ + 2x³ + 3x + 5 at x = -2

# Task 9: Selection Sort

* a) [3, 41, 52, 26, 38, 57, 9] → 21 comparisons
* b) [13, 19, 9, ..., 21] → 66 comparisons

# Task 10: Insertion Sort

* a) [3, 41, 52, 26, 38, 57, 9] → approx. 12 comparisons
* b) [13, 19, 9, ..., 21] → approx. 45–66 comparisons

# Task 11: Bubble Sort

* a) Python code implementation:

def bubble\_sort(arr):  
 n = len(arr)  
 for i in range(1, n):  
 for j in range(n - 1, i - 1, -1):  
 if arr[j] < arr[j - 1]:  
 arr[j], arr[j - 1] = arr[j - 1], arr[j]

* b) Time Complexity: Θ(n²)
* c) Comparisons:
* • 7-element list → 21 comparisons
* • 12-element list → 66 comparisons