hw1

May 31, 2025

DS5012: Foundations of Computer Science

Module 1 Homework: Analysis of Algorithms

Robert Clay Harris: jbm2rt

May 31, 2025

Q1

(a) $O(n^2)$

(b) $O(n \log n)$

(c) $O(2^n)$

Q2

False. Big-O describes a worst-case upper bound on growth, ignoring constants and lower-order terms, so it does not give an exact rate.

Q3

```python
for i in range(n):          # Repeat n times
    for j in range(i):      #   Repeat i times each outer iteration (n)
        print(i, j)         #       O(1)

# O(n) * O(n) * O(1)
# O(n^2)
```

$O(n^2)$

Q4

Solution for $T(n) = 2\,T(\frac{n}{2}) + n$

Identify Components:

Match to the Master Theorem form $T(n) = a\,T(\frac{n}{b}) + f(n)$.

Here:

$$a = 2, \quad b = 2, \quad f(n) = n.$$

Compute the Critical Exponent:

$$\log_b(a) \;=\; \log_2(2) \;=\; 1,$$

so the critical function is

$$n^{\log_b(a)} \;=\; n^1 \;=\; n.$$

Compare $f(n)$ to the Critical Function:

We have

$$f(n) = n, \quad n^{\log_b(a)} = n.$$

Therefore $f(n)$ and $n^{\log_b(a)}$ grow at the same rate.

Check Master Theorem Case:

Since $f(n) = \Theta(n^{\log_b(a)})$, we are in Case 2 of the Master Theorem.

Case 2 implies:

$$T(n) \;=\; \Theta(n^{\log_b(a)}\,\log n) \;=\; \Theta(n\,\log n).$$

So the complexity is

$$\boxed{T(n) = \Theta(n\,\log n).}$$

Q5

[2]:
```
import math

max_fail = 0
limit = 1000000

for n in range(1, limit):
    if 100 * n * math.log2(n) >= n**2:
        max_fail = n

n0 = max_fail + 1
print("The smallest n  is", n0)
```

```
The smallest n  is 997
```

Q6

Time-complexity analysis

1. **Transpose X**
    - Loops: `for i in range(n)` and `for j in range(d)`
    - Cost: $O(n \cdot d)$
2. **Compute XT_X = XT * X**
    - Loops: `for i in range(d)`, `for j in range(d)`, `for k in range(n)`
    - Cost: $O(d \cdot d \cdot n) = O(n \cdot d^2)$

3. **Matrix inversion (d×d)**
   - Cost: $O(d^3)$
4. **Compute XT__y**
   - Loops: `for i in range(d)`, `for j in range(n)`
   - Cost: $O(d \cdot n)$
5. **Final multiplication inv__XT__X * XT__y**
   - Loops: `for i in range(d)`, `for j in range(d)`
   - Cost: $O(d^2)$

Combine all terms:
$O(n \cdot d) + O(n \cdot d^2) + O(d^3) + O(d \cdot n) + O(d^2) = O(n \cdot d^2 + d^3)$

Therefore the overall time complexity is $O(n \cdot d^2 + d^3)$.

If $n \gg d$, the $O(n \cdot d^2)$ term dominates.
If $d \gg n$, the $O(d^3)$ term dominates.

Q7

Recurrence setup

Each call does three recursive calls on lists of size $\approx n/3$, and the slicing plus the final loop together cost $O(n)$. So:

$$T(n) \;=\; 3\,T(n/3) \;+\; O(n).$$

Apply Master Theorem

- $a = 3,\ b = 3 \;\Rightarrow\; n^{\log_b a} = n^{\log_3 3} = n.$
- $f(n) = n = \Theta(n^{\log_b a})$, so this is Case 2.

Hence

$$T(n) \;=\; \Theta(n \log n),$$

and in Big-O notation:

$$T(n) = O(n \log n).$$

Q8

Worst-case cost:
- When the array is full, `append` calls `_resize`, which copies all n elements in $O(n)$ time.
- So a single `append` can take $O(n)$ in the worst case.

Amortized cost over m appends:
- Most `append` calls do a single assignment in $O(1)$.
- Whenever capacity is reached, it doubles ($1 \to 2 \to 4 \to \dots$), and copies existing elements.
- Total number of copies across all resizes is

$$1 + 2 + 4 + 8 + \cdots + 2^{\lfloor \log_2 m \rfloor} \;\le\; 2m \;=\; O(m).$$

- Adding m constant-time appends gives total work $O(m) + m \cdot O(1) = O(m)$.
- Amortized cost per append $= O(m)/m = O(1)$.

Therefore, while a single `append` operation can take $O(n)$ time in the worst case, the amortized cost per `append` over a sequence of m operations is $O(1)$.