Ambedkar College of Arts and Science, Wandoor

Class Test 2 — Solutions

Data Structures and Algorithms — MAT5EJ302(1)

B.Sc. Mathematics Honours (CUFYUGP 2024) — Semester V

Maximum Marks: 30 **Time: 30 Minutes**

8 questions of 5 marks each; students answer any 6 (ceiling 30 marks).

Question 1. Fibonacci Algorithms. **[5 Marks]**

(a) The Fibonacci sequence is defined by $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$. Therefore

$$\boxed{F_0, F_1, \ldots, F_8 = 0, 1, 1, 2, 3, 5, 8, 13, 21}.$$

(b) For $n > 1$, `fib1(n)` makes two recursive calls plus a constant amount of additional work (the comparisons and the final addition). Hence

$$\boxed{T(n) = T(n-1) + T(n-2) + \mathcal{O}(1)}$$

(any correctly-justified constant, e.g. "+3" or "$+c$", is an acceptable instantiation of this).

(c) *Claim: $T(n) \geq F_n$ for all $n \geq 0$.* We prove this by strong induction. Base cases: $T(0) \geq 1 \geq 0 = F_0$ and $T(1) \geq 1 = F_1$ (both algorithms do at least one step even in the base case). Inductive step: assume $T(k) \geq F_k$ for all $k < n$. Then, using the recurrence from (b) together with the inductive hypothesis on $T(n-1)$ and $T(n-2)$,

$$T(n) \geq T(n-1) + T(n-2) \geq F_{n-1} + F_{n-2} = F_n,$$

which completes the induction. Since the Fibonacci numbers grow exponentially ($F_n \approx 2^{0.694n}$, or more crudely $F_n \geq 2^{n/2}$ for $n \geq 2$), and $T(n) \geq F_n$, the running time of `fib1` is

$$\boxed{\text{exponential in } n.}$$

Question 2. Improving Fibonacci. **[5 Marks]**

(a) In `fib1`, values such as F_{n-2} are recomputed independently through different branches of the recursion tree. In `fib2`, each F_i is computed once, stored in the array, and simply looked up thereafter — so

$$\boxed{\texttt{fib2} \text{ avoids repeated subproblems by storing previously computed values.}}$$

(b) The loop runs for $i = 2, \ldots, n$ ($n - 1$ iterations), doing one addition per iteration. Hence

$$\boxed{T(n) = \mathcal{O}(n)}.$$

(c) The algorithm stores the array $f[0], f[1], \ldots, f[n]$, so the storage used is proportional to n:

$$\boxed{S(n) = \mathcal{O}(n)}.$$

(d) Since F_n needs $\mathcal{O}(n)$ bits to represent once n is large, the two numbers being added at step i can themselves have $\mathcal{O}(i)$ bits. Adding two k-bit numbers costs $\mathcal{O}(k)$ time (not $\mathcal{O}(1)$), so the "constant time per addition" assumption silently breaks down: the true bit-cost of `fib2` is $\mathcal{O}(n^2)$, not $\mathcal{O}(n)$, once this is accounted for. So:

$$\boxed{\text{The cost of each addition grows with the number of bits in the operands, so it is not truly constant.}}$$

Question 3. Asymptotic Analysis. **[5 Marks]**

(a) The highest-order term of $7n^3 + 4n^2 + 100n + 50$ is n^3; constant factors and lower-order terms are dropped in asymptotic notation. So

$$\boxed{\mathcal{O}(n^3)}.$$

(b)

$$\boxed{\log n \prec n \prec n \log n \prec n^2 \prec 2^n.}$$

(c) We need $c > 0, n_0 > 0$ such that $3n^2 + 5n + 7 \leq c\,n^2$ for all $n \geq n_0$. For $n \geq 1$, $n \leq n^2$ and $1 \leq n^2$, so

$$3n^2 + 5n + 7 \leq 3n^2 + 5n^2 + 7n^2 = 15n^2 \qquad \text{for all } n \geq 1.$$

Taking $\boxed{c = 15,\ n_0 = 1}$ verifies the definition, so $3n^2 + 5n + 7 = \mathcal{O}(n^2)$. (Any other valid pair is equally acceptable.)

(d) True. $\mathcal{O}(n^2)$ means $T(n) \leq c \cdot n^2$ for all $n \geq n_0$. Since $n^2 \leq n^3$ for every $n \geq 1$, the same bound gives $T(n) \leq c \cdot n^3$ for all $n \geq \max(n_0, 1)$ — so $T(n) = \mathcal{O}(n^2)$ automatically implies $T(n) = \mathcal{O}(n^3)$ (Big-O is an *upper* bound, not a tight one; it says nothing about how loose the bound is).

Question 4. Algorithms with Numbers. **[5 Marks]**

(a) Standard addition processes the n digit positions one at a time, from right to left, doing $\mathcal{O}(1)$ work (add two digits plus an incoming carry) at each position. With n positions and constant work per position,

$$\boxed{T(n) = \mathcal{O}(n)}.$$

(b) Every digit of the first n-digit number is multiplied against every digit of the second, giving $n \times n = n^2$ digit-pair products (each an $\mathcal{O}(1)$ single-digit multiplication), which then have to be combined with appropriate place-value shifts and carries. So

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

(c) n^2 grows faster. Indeed,

$$\frac{n^2}{n \log n} = \frac{n}{\log n} \longrightarrow \infty \quad \text{as } n \to \infty,$$

so n^2 eventually exceeds any constant multiple of $n \log n$: $\boxed{n^2 \text{ grows faster than } n \log n}$.

Question 5. Modular Arithmetic. **[5 Marks]**

(a) $347 = 28 \times 12 + 11 \Rightarrow \boxed{347 \bmod 12 = 11}$.

(b) $-23 = (-5) \times 5 + 2 \Rightarrow \boxed{-23 \bmod 5 = 2}$. (A common error is to write -3, forgetting that the remainder must satisfy $0 \le r < N$.)

(c) $37 \bmod 12 = 1$ and $58 \bmod 12 = 10$, so

$$(37 \times 58) \bmod 12 = (1 \times 10) \bmod 12 = \boxed{10}.$$

(d) x has a multiplicative inverse mod N if and only if $\gcd(x, N) = 1$. Here $\gcd(8, 12) = 4 \neq 1$, so

$$\boxed{8 \text{ has } \textit{no} \text{ inverse modulo } 12.}$$

Question 6. Modular Exponentiation — by Hand. **[5 Marks]**

(a) $13 = 8 + 4 + 1 \Rightarrow \boxed{13 = (1101)_2}$.

(b) By repeated squaring:

$$7^1 \equiv 7, \quad 7^2 \equiv 49 \equiv 9, \quad 7^4 \equiv 9^2 = 81 \equiv 1, \quad 7^8 \equiv 1^2 \equiv 1 \pmod{20}.$$

So $\boxed{7^1 \equiv 7,\ 7^4 \equiv 1,\ 7^8 \equiv 1 \pmod{20}}$.

(c) Since $13 = 8 + 4 + 1$, $7^{13} = 7^8 \cdot 7^4 \cdot 7^1 \equiv 1 \times 1 \times 7 \equiv \boxed{7 \pmod{20}}$.

Question 7. Modular Exponentiation — the Algorithm. **[5 Marks]**

(a) Trace of `ModExp`$(3, 13, 17)$:

Call	y (parity)	Returns
`ModExp`(3, 0, 17)	0	1
`ModExp`(3, 1, 17)	1 (odd)	$3 \times 1^2 \bmod 17 = 3$
`ModExp`(3, 3, 17)	3 (odd)	$3 \times 3^2 \bmod 17 = 27 \bmod 17 = 10$
`ModExp`(3, 6, 17)	6 (even)	$10^2 \bmod 17 = 100 \bmod 17 = 15$
`ModExp`(3, 13, 17)	13 (odd)	$3 \times 15^2 \bmod 17 = 3 \times 4 = 12$

So $\boxed{3^{13} \bmod 17 = 12}$ (check: $3^1{=}3, 3^2{=}9, 3^4{\equiv}13, 3^8{\equiv}16 \pmod{17}$, and $3^{13} = 3^8 \cdot 3^4 \cdot 3^1 \equiv 16 \times 13 \times 3 \equiv 12 \pmod{17}$ — consistent).

(b) Let n be the number of bits of N. The chain of calls halves y at every step, and $y < N$ has at most n bits, so there are $\mathcal{O}(n)$ recursive calls. Each call does exactly one modular multiplication ($z^2 \bmod N$, or that followed by one more multiplication by x), which costs $\mathcal{O}(n^2)$ by hypothesis. Combining:

$$(\text{number of calls}) \times (\text{cost per call}) = \mathcal{O}(n) \times \mathcal{O}(n^2) = \boxed{\mathcal{O}(n^3)}.$$

Question 8. Growth Rates — Look Before You Rank. **[5 Marks]**

(a) Using the log power rule, $\log(n^n) = n \log n$ — so $\log(n^n)$ and $n \log n$ are *exactly the same function*, not two different growth rates:

$$\boxed{\log(n^n) = n \log n \text{ for all } n.}$$

(b) That leaves four distinct growth rates: $n \log n$ (equivalently $\log(n^n)$), $n^{1.5}$, 2^n, $n!$.

- $n \log n$ vs. $n^{1.5}$: dividing both by n gives $\log n$ vs. $n^{0.5}$; any polynomial eventually dominates any logarithm, so $n \log n = o(n^{1.5})$.
- $n^{1.5}$ vs. 2^n: any exponential dominates any polynomial, so $n^{1.5} = o(2^n)$.
- 2^n vs. $n!$: $n!$ eventually dominates every exponential with a fixed base, so $2^n = o(n!)$.

Increasing order of growth:

$$\boxed{n \log n \ (= \log(n^n)) \ \prec \ n^{1.5} \ \prec \ 2^n \ \prec \ n!.}$$

— End of Solutions —