A. Câu về "Master Theorem"

(0) Thuật toán	(1) Công thức đệ quy	(2) Giải thích cho công thức đệ quy	(3) Áp dụng Master Theorem và So sánh	(4) Kết luận
Prefix Sum	$\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + O(n)}$	Chia mảng thành 2 nửa, gọi đệ quy tính prefix từng phần, sau đó cộng dồn phần tử phải với tổng cuối trái → Tốn O(n) gộp	Dạng chuẩn: $\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n)}$; Ta có: a=2; b=2; f(n)=O(n); Vậy: $\boldsymbol{n^{\log_b a} = n^{\log_2 2} = n}$ So sánh: $\boldsymbol{f(n) = \theta(n) = \theta(n^{\log_b a})}$ → Là **Case 2** trong Master Theorem	$\boldsymbol{T(n) = \theta(n \log n)}$
Flatten		Mảng 2 chiều được chia thành 2 phần (hàng/khối), gọi đệ quy xử lý từng phần song song. Sau đó dùng prefix sum để tính vị trí và gộp lại kết quả thành mảng 1 chiều → Tốn O(n) gộp		
Filter		Mảng được chia làm 2 phần, mỗi phần lọc các phần tử thỏa điều kiện (ví dụ: a[i] > 0) một cách song song. Sau đó, dùng prefix sum để xác định vị trí và copy phần tử sang mảng kết quả → Tốn O(n) gộp		
Merge Sort		Chia mảng thành 2 nửa, đệ quy sắp xếp từng nửa, sau đó merge hai nửa lại → Tốn O(n) gộp		
⌦ *Quick Sort*		Chia mảng theo pivot, giả định chia đều → 2 phần ~ n/2, phân hoạch (partition) → Tốn O(n) gộp		
⌦ *Fibonacci*	$\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + O(1)}$	Tính song song 2 nhánh (fib(n-1), fib(n-2)) với độ sâu giả định chia đều. Bước gộp kết quả chỉ cần cộng 2 số → Tốn O(1) gộp	Dạng chuẩn: $\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n)}$; Ta có: a=2; b=2; f(n)=O(1); Vậy: $\boldsymbol{n^{\log_b a} = n^{\log_2 2} = n}$ So sánh: $\boldsymbol{f(n) = \theta(1) = \theta(n^{1-\varepsilon})}$ với $\varepsilon = 1$ → Là **Case 1** trong Master Theorem	$\boldsymbol{T(n) = \theta(n)}$
Matrix Multiply	$\boldsymbol{T(n) = 8T\left(\frac{n}{2}\right) + O(n^2)}$	Ma trận n×n chia thành 4 khối → nhân 2 ma trận sẽ tạo ra 8 phép nhân ma trận con, đồng thời gộp kết quả qua phép cộng các khối → Tốn O(n^2) gộp	Dạng chuẩn: $\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n)}$; Ta có: a=8; b=2; f(n)=O(n^2); Vậy: $\boldsymbol{n^{\log_b a} = n^{\log_2 8} = n^3}$ So sánh: $\boldsymbol{f(n) = \theta(n^2) = \theta(n^{3-\varepsilon})}$ với $\varepsilon = 1$ → Là **Case 1** trong Master Theorem	$\boldsymbol{T(n) = \theta(n^3)}$

B. Các câu về "Viết Code"

(0) Thuật toán	(1) Ý tưởng	(2) Mã giả + Giải thích	(3) Ví dụ	(4) Độ phức tạp

Prefix Sum	Thuật toán Prefix Sum tính dãy mới với mỗi phần tử là tổng của tất cả phần tử trước đó (bao gồm chính nó).
	Trong song song: Sử dụng divide-and-conquer: chia mảng thành 2 nửa, tính prefix sum song song; Sau đó cộng tổng cuối của nửa trái vào tất cả phần tử nửa phải
		function prefix_sum(A, B, s, t, offset):
		  if s == t - 1:
		  B[s] = A[s] + offset
		  return
		  mid = (s + t) // 2
		  in parallel:
		  prefix_sum(A, B, s, mid, offset)
		  prefix_sum(A, B, mid, t, offset + sum(A[s:mid]))
		
		A: mảng đầu vào, **B:** mảng kết quả, **offset:** giá trị cần cộng thêm
		Dòng 1–2: nếu chỉ còn 1 phần tử, gán tổng vào B[s]
		Dòng 4: chia đôi mảng
		Dòng 5–6: xử lý hai nửa song song; phần bên phải cộng thêm tổng bên trái
			Input: A = [1, 2, 3, 4]:
			• **B1:** chia thành [1, 2] và [3, 4]
			• **B2:** tính prefix [1, 3] và [3, 7] → [3, 7] do mỗi phần tử cộng thêm 3 (tổng cuối bên trái)
			Output: B = [1, 3, 6, 10]
				Đệ quy: T(n) = 2T (n/2) + O(n)
				Master Theorem: a=2, b=2, f(n) = O(n) → Case 2
				Kết quả: T(n) = Θ(n log n), Depth = O(log n)

Fibo-nacci	Tính số Fibonacci thứ n bằng công thức đệ quy: F(n) = F(n-1) + F(n-2).
	Trong song song: Gọi hai nhánh đệ quy cho F(n-1) và F(n-2) song song sau đó tính tổng
		function fib(n):
		  if n <= 1:
		  return n
		  in parallel:
		  a = fib(n-1)
		  b = fib(n-2)
		  return a + b
		
		Dòng 1–2: điều kiện dừng (cơ sở đệ quy).
		Dòng 4–5: hai lời gọi thực hiện song song.
		Dòng 7: trả về tổng của hai nhánh.
			Input: fib(4)
			• fib(4) = fib(3) + fib(2)
			• fib(3) = fib(2) + fib(1), fib(2) = fib(1) + fib(0)
			• Cây đệ quy tạo ra 7 lời gọi
			Song song hóa: các lời gọi ở cùng tầng có thể chạy đồng thời.
				Không thể dùng Master Theorem vì không có dạng T(n) = aT(n/b) + f(n).
				Tuy nhiên, dạng chia nhánh song song có thể biểu diễn: T(n) = T(n-1) + T(n-2) + O(1)
				• Work: T(n) = Θ(2n) (không memoization)
				• Memoization và song song hóa: Work = Θ(n), Depth = Θ(n)

Flatten	Chuyển các mảng con thành mảng 1 chiều duy nhất bằng cách nối các mảng lại liên tiếp.
	Trong song song:
	• Tính kích thước mỗi mảng con → mảng S
		function flatten(A[0..n-1]):
		  parallel_for i = 0 to n-1:
		  S[i] = size(A[i])
		  offset = prefix_sum(S)
		  parallel_for i = 0 to n-1:
		  off = offset[i]
		  parallel_for j = 0 to S[i] - 1:
		  B[off + j] = A[i][j]
		  return B
			Input: A = [[5,1,2], [3,1], [9,3,5]]
			• S = [3,2,3]
			• Offset = [0,3,5] (tính bằng prefix sum)
			Output: B = [5,1,2,3,1,9,3,5]
				Work: O(N + n log n) ≈ O(N) (N = tổng phần tử, n = số mảng con)
				Depth: O(log n) (do Prefix Sum)
				Master Theorem: Case

	• Dùng Prefix Sum trên S để tìm offset của từng mảng • Sao chép song song các phần tử vào đúng vị trí theo offset	**Dòng 2–3:** Tính độ dài mỗi mảng con **Dòng 5:** Dùng prefix sum để biết vị trí bắt đầu của mỗi mảng con **Dòng 7–9:** Copy từng phần tử vào vị trí chính xác (song song)		2 → T(n) = Θ (n log n)

Filter Packing	Dùng để lọc các phần tử từ một mảng đầu vào dựa trên điều kiện cho trước (predicate), và lưu lại kết quả vào một mảng mới.
	Trong song song:
	• Tạo mảng đánh dấu F: 1 nếu thỏa điều kiện, 0 nếu không
	• Prefix Sum trên F để tìm vị trí đích
	• Copy song song phần tử thỏa điều kiện vào đúng vị trí
		function filter(A[0..n-1]):
		  parallel_for i = 0 to n-1:
		  F[i] = 1 if predicate(A[i]) else 0
		  offset = prefix_sum(F) // scan exclusive
		  parallel_for i = 0 to n-1:
		  if F[i] == 1:
		  B[offset[i]] = A[i]
		  return B
		
		Dòng 2–3: Đánh dấu phần tử nào sẽ được giữ lại
		Dòng 5: Dùng prefix sum để tính vị trí copy trong mảng mới
		Dòng 7–8: Copy đúng phần tử vào đúng chỗ nếu F[i] = 1
			Input: A = [3, 5, -1, 7, 0], predicate: x > 0
			• F = [1, 1, 0, 1, 0]
			• offset = [0, 1, 2, 2, 3]
			Output: B = [3, 5, 7]
				Work:
				• Đánh dấu: O(n)
				• Prefix sum: O(n log n)
				Depth: O(log n) (từ Prefix Sum)
				Master Theorem: a = 2, b = 2, f(n) = O(n) → Case 2 → T(n) = Θ (n log n)

Matrix Multiply	Nhân hai ma trận vuông A và B, kết quả là ma trận C với kích thước n×n, trong đó:
	$C[i][j] = \sum_{k=0}^{n-1} A[i][k] \cdot B[k][j]$
	Trong song song: Chia ma trận thành 4 khối con mỗi chiều → tổng 8 phép nhân ma trận con. Dùng chia để trị: nhân từng khối song song và cộng kết quả
		function matmul(A, B):
		  if size(A) == 1:
		  return A[0][0] * B[0][0]
		  divide A and B into 4 submatrices: A11, A12, A21, A22; B11, B12, B21, B22
		  in parallel:
		  M1 = matmul(A11, B11)
		  M2 = matmul(A12, B21)
		  M3 = matmul(A11, B12)
		  M4 = matmul(A12, B22)
		  M5 = matmul(A21, B11)
		  M6 = matmul(A22, B21)
		  M7 = matmul(A21, B12)
		  M8 = matmul(A22, B22)
		  in parallel:
		  C11 = M1 + M2
		  C12 = M3 + M4
		  C21 = M5 + M6
		  C22 = M7 + M8
		  return C = combine(C11, C12, C21, C22)
		
		Dòng 2–3: Điều kiện dừng, phần tử 1x1
		Dòng 5: Chia ma trận thành 4 phần
		Dòng 7–14: Gọi đệ quy 8 phép nhân ma trận con
			Input: Hai ma trận 4x4
			• Chia mỗi ma trận ra 4 khối 2x2
			• Thực hiện 8 phép nhân khối → cộng lại thành 4 khối kết quả
			• Ghép lại thành C
				Đệ quy: T(n) = 8T(n/2) + O(n^2)
				MasterTheorem: a=8, b=2, f(n)=O(n^2) → Case 1
				Kết quả: T(n) = Θ(n^3), Depth = O(log n)

		Dòng 16–19: Cộng kết quả thành các khối của ma trận C **Dòng 21:** Ghép các khối thành ma trận hoàn chỉnh		

Merge Sort	Chia mảng thành 2 nửa, sắp xếp từng nửa bằng đệ quy, sau đó trộn (merge) hai mảng con đã sắp.
	Trong song song: Gọi đệ quy song song để sắp xếp từng nửa.
	Hoặc: Chia đôi mảng đích → Tìm pivot position trong mảng kia → Gộp từng phần song song
		function mergesort(A, s, t):
		if t - s <= 1:
		return
		m = (s + t) // 2
		in parallel:
		mergesort(A, s, m)
		mergesort(A, m, t)
		parallel_merge(A, s, m, t)
		
		Dòng 2: Dừng đệ quy nếu chỉ còn 1 phần tử
		Dòng 4: Tìm điểm chia
		Dòng 6–7: Sắp xếp hai nửa song song
		Dòng 9: Gộp hai mảng đã sắp bằng merge song song
			Input: A = [4, 2, 7, 1]
			• Chia thành [4, 2] và [7, 1]
			• Gọi đệ quy song song sắp thành [2, 4] và [1, 7]
			• Gộp lại thành [1, 2, 4, 7]
			Merge: Chia mảng giữa pivot, ghép từng phần song song
				Chép nội dung cột (1), (3), (4) ở Bảng A

Quick Sort	Chọn một phần tử làm pivot, chia mảng thành: • Các phần tử nhỏ hơn pivot • Các phần tử lớn hơn hoặc bằng pivot **Trong song song:** • Dùng filter song song để chia mảng theo pivot • Sau đó đệ quy sắp xếp 2 phần kết quả song song • Cuối cùng nối kết quả theo thứ tự: [left, pivot, right]	function quicksort(A): if size(A) <= 1: return A pivot = A[0] in parallel: L = filter(A[1:], x < pivot) R = filter(A[1:], x >= pivot) in parallel: L_sorted = quicksort(L) R_sorted = quicksort(R) return concat(L_sorted, [pivot], R_sorted) **Dòng 2:** Cơ sở đệ quy – mảng rỗng hoặc 1 phần tử **Dòng 4:** Chọn pivot (phần tử đầu) Dòng 6–7: Lọc mảng thành hai phần song song **Dòng 9–10:** Sắp xếp hai phần đó song song **Dòng 12:** Ghép kết quả lại	**Input:** A = [7, 3, 8, 2] • Chọn pivot = 7 • Lọc song song: L = [3, 2], R = [8] • Đệ quy sắp [3, 2] → [2, 3], [8] → [8] • Ghép lại: [2, 3, 7, 8]	