Yan Zhu Yangbo Chen

3. What is the work and span of `get_positions`? (assume our more efficient version of `scan` from class)
Work: $O(k)$ because every element in the counts array must be processed once

•Span: $O(\log k)$ because the parallel prefix sum algorithm reduces the critical path to a logarithmic number of steps

5. What is the work and span of `construct_output`?

. work: This takes $O(n)$, where n is the length of the input list a, because we create a list of size n.

. Span: these steps are done sequentially for each element in the list. As a result, span: $O(n)$

6. What is the work and span of `supersort`?

. Work: The total work is the sum of the work for each part:

work = $O(n) + O(k) + O(n) = O(n + k)$

This means the work depends on both the size of the input list n and the maximum value k
• Span: The total span is the maximum span of the components:

Span = $\max(O(n), O(\log k), O(n)) = O(n)$

This means the span is primarily affected by the size of the input list n, because some steps must be done sequentially.

Work: $O(n + k)$ depends on the size of the list and the maximum value

Span: $O(n)$ mainly depends on the size of the list

8. What is work and span of `count_values_mr`?

> **put in answers. md**

. Work (Map) = $O(n)$

. Span(Map) = $O(1)$

. Work (Reduce) = $O(n)$

. Span(Reduce) = $O(\log n)$

. Work (Group) = $O(n \log n)$

. Sapn(Group) = $O(\log n)$

. Total Work = $O(n) + O(n \log n) + O(n) = O(n \log n)$

. Total Span = $\max(O(1), O(\log n), O(\log n)) = O(\log n)$

a

(a): $S_n = \sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$, for $a \neq 1$

Inductive hypothesis: $S_k = \frac{a^{k+1}-1}{a-1}$

Base case:

For n=0. $\frac{a-1}{a-1} = 1 = S_0$

Inductive step:

For n=k+1

$$\frac{a^{(k+1)+1}-1}{a-1} = \frac{a^{k+2}-1}{a-1}$$

$$S_{k+1} = \frac{a^{k+1}-1}{a-1} + a^{k+1} = S_k + a^{k+1}$$

$$S_{k+1} = \frac{a^{k+1}-1}{a-1} + \frac{a^{k+1}\cdot(a-1)}{a-1}$$

$$S_{k+1} = \frac{a^{k+1}-1+a^{k+1}\cdot(a-1)}{a-1}$$

$$S_{k+1} = \frac{a^{k+1}-1+a^{k+2}-a^{k+1}}{a-1}$$

$$S_{k+1} = \frac{-1+a^{k+2}}{a-1}$$

$$S_{k+1} = \frac{a^{k+2}-1}{a-1}$$

thus. $S_k = \frac{a^{k+1}-1}{a-1}$

b.

Inductive hypothesis: scan(a) = [a₁, a₁+a₂, a₁+a₂+a₃, a₁+a₂ ... aₖ]

Base case:

For n=0.

when n=0, it return id

Inductive step:

For n=k:

reduce (f, id, a) = f(a_1, f(a_2, f... (a_(k-1), a_k) ...)))

For n=k+1

First part: [a_1, a_2, ..., a_m]

Thecond part [a_(m+1), ... a_(k+1)]

Reduce First part: R left

Reduce Right part: R right.

Combine(R left, R right) = f(R left, R right)

The Reduce function are correctly reduce the parts, and function f correctly combines the results, the entire list a is correctly reduced by reduce.

C.

Base case: n=1

span(1) = O(1)

Inductive Hypothesis: n=k

span(k) = O(log k)

Inductive step: n=k+1 $m = \frac{k+1}{2}$

array a = [a₁, a₂, a..., a_(k+1)]

a left = [a₁...aₘ] a right = [a_(m+1), ... a_(k+1)]

span(a left) = O(log m) span(a right) = O(log k+1−m)

Combine:

Combine = O(1).

Thus:

span(k+1) = O(log m) + O(log k+1−m) + O(1)

span(k+1) = Max(O(log m), O(log k+1−m)) + O(1)

= O(log m)

so this is same as our hypothesis O(log n).

d:

base case: input 1 ← contraction-based

span(1) = O(1)

inductive hypothese: n=k

span(a) = O(log k)

inductive steps:

1. Contraction:
   the new size is k/2
2. recursive span:
   O(log k/2)
3. expansion
   The input expend to orignal size
4. overall size
   span(k+1) = O(log k/2) + O(1) = O(log(k+1))

difference:

1. contraction-based.
   Contracts the problem by pariting the elements then put it back. It reduces it very fast. But need adjustments of the result.
2. Divide-and-Conquer:
   Divided the size into two equal parts. Computed them then combine them. The combination happens after both computing complete. It has no extra adjustments or contraction.

Conclusion:

Divide-and-Conquer is easier to understand and prove. Becase it has staightforward structure.