# Parallel (merge) sorting

9/26/24

# Administrivia

- Reading for tomorrow: Section 14.1
- HW 3 (multithreading) due Tuesday 10/1
- Exam 1 out Wednesday
  - Multi-day takehome
  - Open notes and book, closed internet and friends
  - No class next Thursday (10/3)
  - Due early the next week (probably Monday night)
  - Everything thru multithreaded (induction, asymptotic ordering, AVL trees, D&C, multithreaded)

# Recall: Merge sort

```
merge_sort(A, p, r) {
        if p < r {
                q = (p+r) / 2
                merge_sort(A, p, q)
                merge_sort(A, q+1, r)
                merge(A, p, q, r)
        }
}
```

# Parallelizing merge sort

```
merge_sort(A, p, r) {
        if p < r {
                q = (p+r) / 2
                spawn merge_sort(A, p, q)
                merge_sort(A, q+1, r)
                sync
                merge(A, p, q, r)
        }
}
```

# Parallelizing merge sort

```
merge_sort(A, p, r) {
        if p < r {
                q = (p+r) / 2
                spawn merge_sort(A, p, q)
                merge_sort(A, q+1, r)
                sync
                merge(A, p, q, r)
        }
}
```

What is the recurrence for the work of this function?

A. $T_1(n) = T_1(n/2) + \log n$
B. $T_1(n) = 2T_1(n/2) + \log n$
C. $T_1(n) = T_1(n/2) + n$
D. $T_1(n) = 2T_1(n/2) + n$
E. None of the above

# Parallelizing merge sort

```
merge_sort(A, p, r) {
        if p < r {
                q = (p+r) / 2
                spawn merge_sort(A, p, q)
                merge_sort(A, q+1, r)
                sync
                merge(A, p, q, r)
        }
}
```

What is the recurrence for the work of this function?

A. $T_1(n) = T_1(n/2) + \log n$

B. $T_1(n) = 2T_1(n/2) + \log n$

C. $T_1(n) = T_1(n/2) + n$

D. $T_1(n) = 2T_1(n/2) + n$

E. None of the above

# Parallelizing merge sort

```
merge_sort(A, p, r) {
        if p < r {
                q = (p+r) / 2
                spawn merge_sort(A, p, q)
                merge_sort(A, q+1, r)
                sync
                merge(A, p, q, r)
        }
}
```

What is the recurrence for the span of this function?

A. $T_\infty(n) = T_\infty(n/2) + \log n$
B. $T_\infty(n) = 2T_\infty(n/2) + \log n$
C. $T_\infty(n) = T_\infty(n/2) + n$
D. $T_\infty(n) = 2T_\infty(n/2) + n$
E. None of the above

# Parallelizing merge sort

```
merge_sort(A, p, r) {
        if p < r {
                q = (p+r) / 2
                spawn merge_sort(A, p, q)
                merge_sort(A, q+1, r)
                sync
                merge(A, p, q, r)
        }
}
```

What is the recurrence for the span of this function?

A. $T_\infty(n) = T_\infty(n/2) + \log n$

B. $T_\infty(n) = 2T_\infty(n/2) + \log n$

C. $T_\infty(n) = T_\infty(n/2) + n$

D. $T_\infty(n) = 2T_\infty(n/2) + n$

E. None of the above

# Parallelizing merge sort

```
merge_sort(A, p, r) {
        if p < r {
                q = (p+r) / 2
                spawn merge_sort(A, p, q)
                merge_sort(A, q+1, r)
                sync
                merge(A, p, q, r)
        }
}
```

The span is O(n)

Work is O(n log n), so this doesn't give much parallelism

Need to look at parallelizing merge

# Parallel merge (idea)

```
p_merge(A, B, C) {          //merge A and B into C
        handle base cases
        swap A and B if necessary so |A| ≥ |B|
        use binary search to find median of A in B
        spawn p_merge(left(A), left(B), left(C))
        p_merge(right(A), right(B), right(C))
        sync
}
```