CS3000
5/13 - Tue.

Admin
- HW1 due thurs 9pm
- Rec 2 today!

Agenda
1. Divide + Conquer: sorting!
2. the merge
3. mergesort run-time (APP2)

0. Proof by Induction

↳ start with thing you're trying to prove (logic statement)

$S(n)$: my algo works on array of length n

$\forall n \in \mathbb{Z}^+ \; S(n)$

Structure of proof:
- base case $S(1)$ (or whatever smallest case)

Inductive step
- (IH) simple $S(k)$ for arbitrary k
- (IH) strong $S(1), S(2), \ldots, S(k)$ for arbitrary k

If (IH), then $S(k+1)$

- label base case
- state IH
- label when IH is applied!

1. Divide + Conquer Sorting

Algos so far...
- linear search
- insertion sort

} no strategy

strategies to apply:
"can we do better?"
1. Divide + conquer
2.
3.

Divide + conquer...
- search $\Theta(n) \rightarrow \Theta(\lg n)$
  linear → binary: works on sorted array
  is it worth it? It depends
  - search for one key, once, don't bother!
  - but ex: google
    1. crawl
    2. index — sort (once/hour)
- sort $\Theta(n^2) \rightarrow$?

Sorting: many algos
research area

otherwise

Searching: done!

online

3. query search 570 mil/hour

4. rank sort

GOOG(A, n)

INSSORT(A, n) $\Theta(n^2)$

BINSEARCH(A, n, key) $\Theta(\lg n)$

overall run time: $\Theta(n^2) + \Theta(\lg n)$

$= \Theta(n^2)$

Divide + Conquer

- break into smaller subproblems
- solve smaller subproblems recursively
- combine smaller solutions to solve original problem

D+C sorting: mergesort

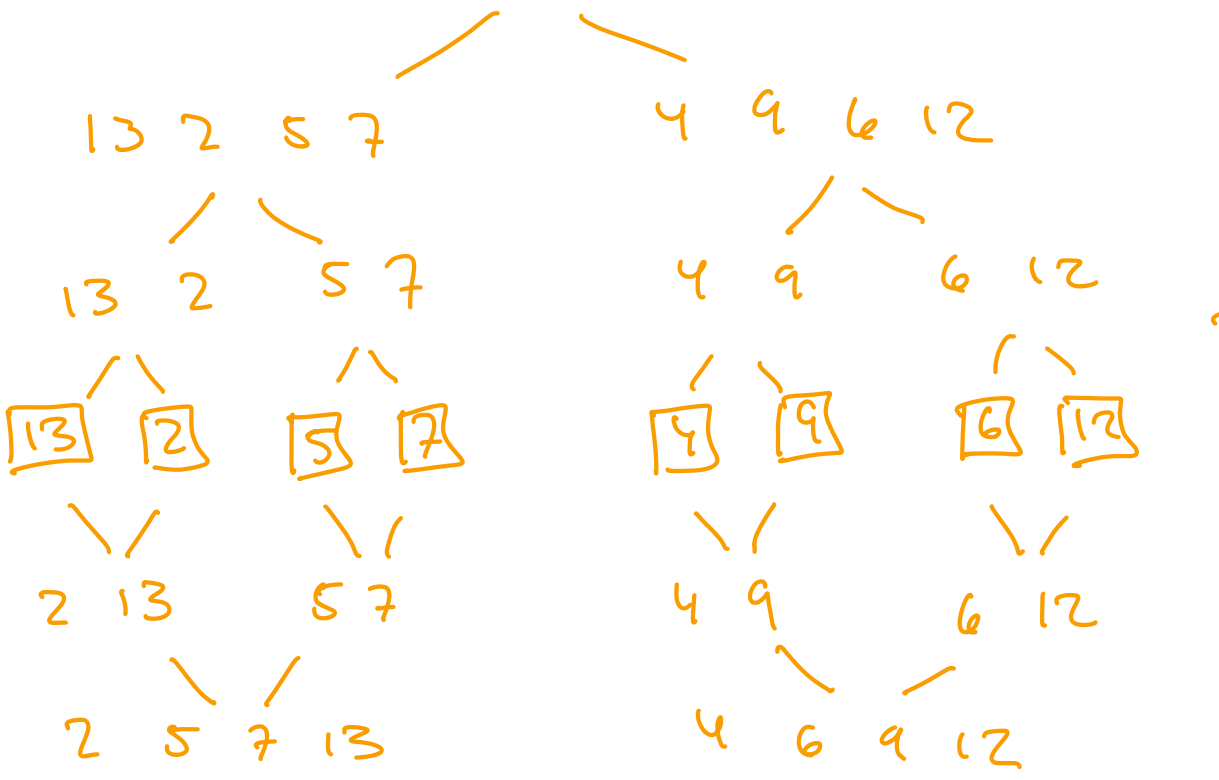
```
MERGESORT(A, p, r)
1  if p < r                          ⟶ check for base case
2      q = ⌊(p + r)/2⌋               ⟶ q is midpoint
3      MERGESORT(A, p, q)            ⟶ recurse
4      MERGESORT(A, q + 1, r)
5      MERGE(A, p, q, r)             ⟶ merge two sorted halves
```

Input: array A, left p, right r

- pass in p and r instead of modified array

Base case: if p < r // recurse else // base case

ex 13 2 5 7 4 9 6 12



n arrays of length 1 sorted!

Sorted arrays of length 2

Sorted, length 4

2 4 5 7 9 12 13 sorted, length 7

2. The merge

↳ merging 2 already-sorted arrays

Investigating merge (on its own)

A = <1, 7, 4, 9, 6, 8, 2, 5>

MERGE(A, 1, 2, 4)	MERGE(A, 5, 6, 8)

After merge runs:

- What is in A?
- How much space was allocated?
- Roughly how many steps in first while loop?
- What do second / third while loops do?

```
MERGESORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGESORT(A, p, q)
4      MERGESORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

$T(n) = c_1 + c_2 + T(n/2) + T(n/2) + ?$

Line 5 = how long does merge take?

```
MERGE(A, p, q, r)
1   n_L = q − p + 1
2   n_R = r − q
3   let L[1 : n_L] and R[1 : n_R] be new arrays
4   for i = 1 to n_L
5       L[i] = A[p + i − 1]
6   for j = 1 to n_R
7       R[j] = A[q + j]
8   i = 1
9   j = 1
10  k = p
11  while i ≤ n_L and j ≤ n_R
12      if L[i] ≤ R[j]
13          A[k] = L[i]
14          i = i + 1
15      else
16          A[k] = R[j]
17          j = j + 1
18      k = k + 1
19  while i ≤ n_L
20      A[k] = L[i]
21      i = i + 1
22      k = k + 1
23  while j ≤ n_R
24      A[k] = R[j]
25      j = j + 1
26      k = k + 1
```

A = <1, 7, 4, 9, 6, 8, 2, 5>

MERGE(A, 1, 2, 4)	MERGE(A, 5, 6, 8)
A = <1, 4, 7, 9, 6, 8, 2, 5>	A = <1, 7, 4, 9, 2, 5, 6, 8>
space ≈ current n (length of subarray)	while loop #1
while loop #1 twice	

12:47

3. Mergesort Run-Time

$T(n) = c + 2 \cdot T(n/2) +$ however long merge takes

```
MERGE(A, p, q, r)
1  n_L = q − p + 1
2  n_R = r − q
3  let L[1 : n_L] and R[1 : n_R] be new arrays
4  for i = 1 to n_L
5      L[i] = A[p + i − 1]
6  for j = 1 to n_R
7      R[j] = A[q + j]
8  i = 1
9  j = 1
10 k = p
11 while i ≤ n_L and j ≤ n_R
12     if L[i] ≤ R[j]
13         A[k] = L[i]
14         i = i + 1
15     else
16         A[k] = R[j]
17         j = j + 1
18     k = k + 1
19 while i ≤ n_L
20     A[k] = L[i]
21     i = i + 1
22     k = k + 1
23 while j ≤ n_R
24     A[k] = R[j]
25     j = j + 1
26     k = k + 1
```

- How many steps for while loop #1?
- What do second/third while loops do?

$A = \langle 1, 7, 4, 9, 6, 8, 2, 5 \rangle$

MERGE(A, 5, 6, 8)

p 6 r

6 8 | 2 5

L , R

$n_L = 6 - 5 + 1 = 2$

$n_R = 8 - 6 = 2$

$i=1$ $j=1$ while $i \le 2$ and $j \le 2$

$L[1] \le R[1]$

Counting:

1 | #1

1 | 1 | #2

1 | #3

$i=1$ $j=2$

$L[1] \le R[2]$

$i=1$ $j=3$

$A = \langle 1, 7, 4, 9, 2, 5, ?, ? \rangle$

$i=1$ while $i \le 2$ (19)

$A = \langle 1, 7, 4, 9, 2, 5, 6, 8 \rangle$

merge on array of length n, $\Theta(n)$ run-time

$T(n) = 2 \cdot T(n/2) + n + c$ (1)

↳ $2 \cdot T(n/4) + n/2 + c$

$T(n) = 2[2 \cdot T(n/4) + n/2 + c] + n + c$

$= 4 \cdot T(n/4) + n + 2c + n + c$

$= 4 \cdot T(n/4) + 2n + 3c$ (2)

$\hookrightarrow 2 \cdot T(n/8) + n/4 + c$

$T(n) = 4\left[2 \cdot T(n/8) + n/4 + c\right] + 2n + 3c$

$= 8 \cdot T(n/8) + n + 4c + 2n + 3c$

$= 8 \cdot T(n/8) + 3n + 7c$ (3)

Pattern on k^{th} iteration:

$T(n) = 2^k \cdot T(n/2^k) + kn + (2^k - 1) \cdot c$

choose value for k to get the base case $T(2) = 1$

want: $\frac{n}{2^k} = 2$ solve for k

$n/2^k = 2$

$n = 2 \cdot 2^k$

$n = 2^{k+1}$

$\lg n = k + 1$

$k = \lg n - 1$

$\rightsquigarrow \Theta(n \lg n)$!

insertion → mergesort

$\Theta(n^2)$ $\Theta(n \lg n)$

Another approach to solving recurrence: Master method

↳ if recurrence has a certain form, we can look up the bound

$T(n) = a \cdot T(n/b) + f(n)$

$a, b > 0$

$f(n)$ driving function

$n^{\log_b a}$ watershed function

Cases:

1. watershed grows faster than driving – $T(n) = \Theta(n^{\log_b a})$
2. watershed / driving grow the same – $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$
3. watershed grows slower than driver – $T(n) = \Theta(f(n))$

(ex) $T(n) = 9 \cdot T(n/3) + n$

$a = 9$

$b = 3$

driving function $f(n) = n$

watershed $n^{\log_3 9} = n^2$

case 1! $T(n) = \Theta(n^{\log_3 9}) = \Theta(n^2)$