CS3000
5/14 - Week

Admin

- HW1 due 5/15 9pm
- APP3 due 11:30 AM 5/15

Agenda

1. Quicksort overview
2. Partition step
3. Quicksort correctness

1. Quicksort Overview

Can we do better?

1. Divide + conquer
2.
3.

Search: linear $\Theta(n)$ → binary $\Theta(\lg n)$

Sort: insertion $\Theta(n^2)$ worst case → mergesort $\Theta(n \lg n)$ best + worst case

- predictable :)
- no bonus :(

quicksort
? run-time

Is quicksort 'better' than mergesort?

mergesort
- recurse left
- recurse right
- merge

quicksort
- partition
- recurse left
- recurse right

QUICKSORT(A, p, r)

```
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

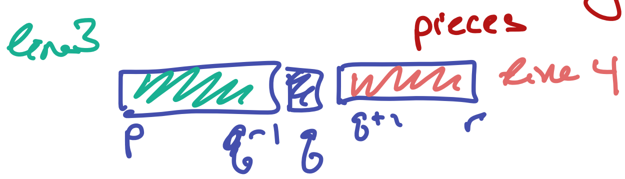
array A
left index p
right index r

$\boxed{\text{first call: QSORT}(A, 1, n)}$

Base case: nothing

line 2 → q gives us position to break array into pieces

After line 2, element at position q is in its correct, final position



In partition:
- choose pivot
- everything smaller on left
- everything larger on right

After partition:
- q is position of pivot
- recursively sort left (p to q-1)
- recursively sort right (q+1 to r)

Best case run-time:

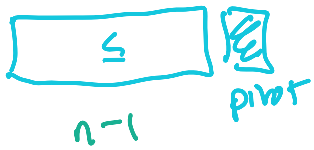
$$T(n) = n + c + 2T(n/2)$$
$$= \Theta(n \lg n)$$

Worst case run time:

$$T(n) = n + c + T(n-1) + T(0)$$
$$= \Theta(n^2)$$

lucky

unlucky :(



Run-Time

- partition: n
- other: c
- recursive:
  - best case: $T(n/2) + T(n/2)$
  - worst case: $T(n-1) + T(0)$

Why use quicksort?

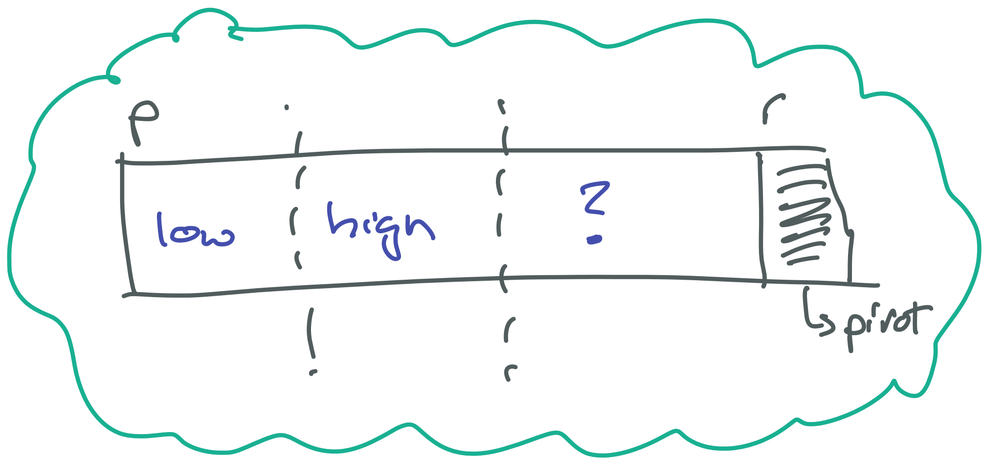
- very commonly used
- space efficiency
- lower start-up cost
- choice of pivot
- constant factor differences
- inertia

↳ usually close to best case!

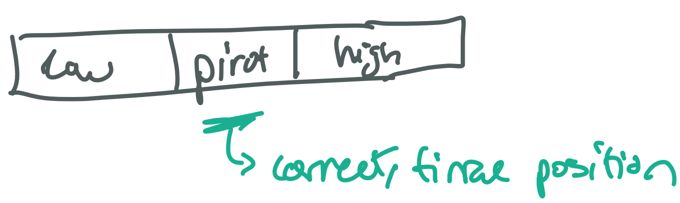
2. Partition Step

- Input: A, p, r
- output: q, position of pivot

Divides array into regions
1. smaller than pivot
2. larger than pivot
3. Haven't seen
4. pivot



last ⇝



ex 5 12 2 3 7

pivot

p = 1
r = 5
pivot = A[r] = 7

next element: pos p, 5

5 ≤ 7? ✓

Swap!

5 | 12 2 3 7

low pivot

next element: pos p+1, 12

12 ≤ 7? × (nothing)

next element: p+2, 2

2 ≤ 7?

Swap!

5 2 | 12 3 7

low pivot

next element: p+3, 3

3 ≤ 7?

Swap!

5 2 3 | 12 7

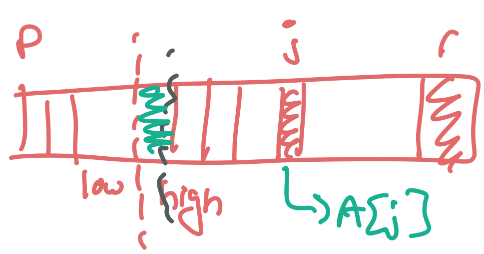
low

swap A[r], A[i]

need in pseudocode:

- loop from p to r-1
- index end of low region
- small, then swap!
- put pivot in final position (last!)

```
PARTITION(A, p, r)
  pivot = A[r]
  low = p-1
  for j = p to r-1
      if A[j] ≤ pivot
          low = low+1
          swap A[j], A[low]
  swap A[low+1], A[r]
  return low+1
```



12:52

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q - 1)
4      QUICKSORT(A, q + 1, r)
```

But, does it work?

→ loop invariant
to show that partition works

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p - 1
3  for j = p to r - 1
4      if A[j] ≤ x
5          i = i + 1
6          swap A[i], A[j]
7  swap A[i + 1], A[r]
8  return i + 1
```

For any elements we've seen so far...

Proving the low region:
for any index k, if $p \le k \le i$ then
$A[k] \le x$ (pivot)

- $\boxed{\text{initialization}}$
  $i = p-1$
  $j = p$ no values between p and i, so trivially true

- $\boxed{\text{maintenance}}$:
  - assume true through $j-1$ and now we're in j
  - case 1: condition at line 4 is false. nothing happened. j increments, preserves invariant
  - case 2: condition at line 4 is true.
    i increments
    $A[j]$, which is $\le x$, moves to new position i
    so it's within the low region

- $\boxed{\text{termination}}$
  - at termination, $j = r$ so all elements from p to i are $\le x$

2 4 3 8 9 1 . . . $\boxed{7}$

i j

$A[j] \le x?$ ✓

2 4 3 1 9 8 10 $\boxed{7}$

i $i+1$ j

$A[j] \ge 7?$ ✓

$i = p-1$

$j = p$

$i+1$ to $j-1$

p to $p-1$