CS2100

Sorting
(part 1)

Recap

- HW - due Friday, 3/8 (by midnight)
- Lab this Thursday, due Sunday
- Reading (due Wed. by 2pm)
- HW on linked lists
  ↳ due Fri. 3/22
- <u>No class</u> on Friday

Last time:

Searching:

1. Linear Search

   Run time: $O(n)$ (either vectors or lists)

2. Binary Search

   Run time: depends!

   need: sorted list

   Vector/array: $O(\log n)$

   Linked: $O(n)$

Sorting:

1) Selection sort:

In loop: lowest ← 1 to n

find index of smallest value (between lowest & n)

move it to lowest spot

But → implementation?

for loop, i ← 1 to n (n times)

[linear search, (n−i) time

$$\sum_{i=1}^{n}(n-i) = (n-1)+(n-2)+\cdots+(1)$$

$$= \sum_{i=1}^{n-1} i = O(n^2)$$

Selection sort run time:

Vector: search in $n-i$ time.

swap lowest with min

Linked List:

Same: (use iterators)

② Insertion Sort:

Sorted part of list (initially size 1)

and unsorted part

For $i \leftarrow 2$ to n

swap i^{th} element to correct location.

search

How?

Array: binary search & then insert

$O(\log n)$

$O(n)$ worst case

Linked:

linear search, $O(n)$, & then insert, $O(1)$

Runtime:

$$\sum_{i=2}^{n} i = O(n^2)$$

Caveat: What if "nearly" sorted?

(New idea: output sensitive)

gets considerably faster.

3 Quick sort

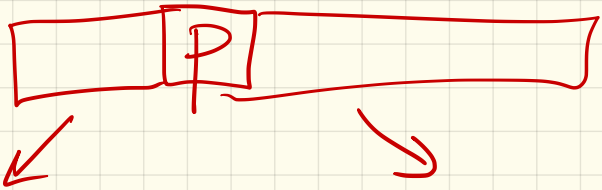
Recursive:

If size of list is > 1:

Pick a "pivot" p (usually 1st element)

Divide List into elements =< p and > p.

Recursively quicksort the 2 parts



qs(list, min, max)

Implementation:

Vector: Not too bad

```
max = n, i = 2
while (i < max)
    if (p < A[i])
        swap A[i] ↔ A[max]
        max--
    else
        swap A[i] ↔ p
        i++
```

Linked:

Call insertAfter
insertBefore O(1)

Runtime:

Worst case:

$$Q(n) \le 3n + Q(n-1)$$

$$\le 3n + 3(n-1) + Q(n-2)$$

$$\cdots = 3\sum_{i=1}^{n} i = O(n^2)$$

Randomized:

$$Q(n) = 2Q\left(\frac{n}{2}\right) + O(n)$$

$$= O(n \log n)$$

Another (not in zybook):

Bubble sort:

Similar to insertion.

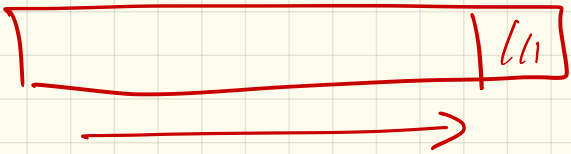
Idea: "bubble" largest value to be last in the list.

→ for ($i \to n$ down to 2)

→ for ($j \to 1$ to i)

compare $A[j]$ to $A[j+1]$

Swap if out of order



("backwards selection sort)

Implementation / Runtime:

$O(n^2)$

$$\sum_{i=1}^{n} \sum_{j=1}^{i} 1$$

$$= \sum_{i=1}^{n} i = O(n^2)$$

Trade-offs:

- Data structure ←
- Type of data : sorted
  : random
- Size of list : $O(n^2)$ vs $O(n \log n)$
  is huge, on large lists
- Ease of programming

Next time:

- Merge sort
- Radix sort
- Maybe ~~radix~~ Shell sort