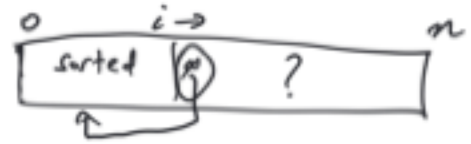
Lecture 17: Analyzing sorting algorithms

- Time complexity of insertion & selection sort
- Divide & conquer sorting: Mergesort & Quicksort

Announcements:

- Solutions posted

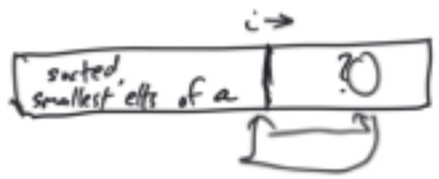
insertion sort



(loop inv.)

selection sort

a



---

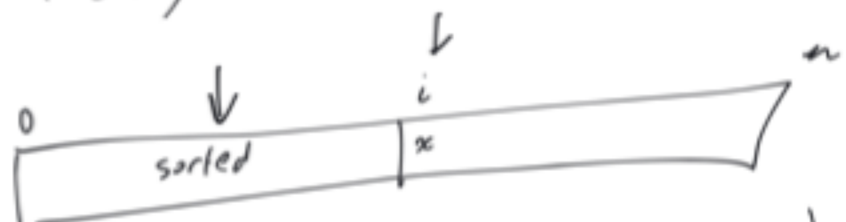
insertion sort:

n iterations of the loop.

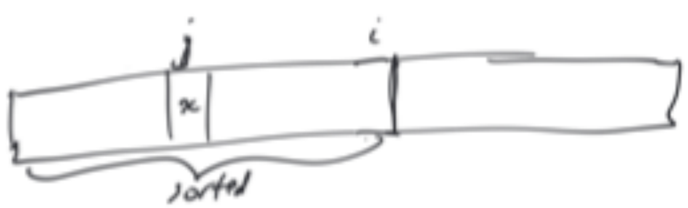
time proportional to size of sorted portion.
- find position for x in sorted portion
- move everything after to the right

(in worst case)

state at begin. of loop



worst case: x is smallest
takes i steps.



happens if array is reverse-sorted to start.

1st iteration: 1 step

2nd iter: 2 steps

3rd iter: 3

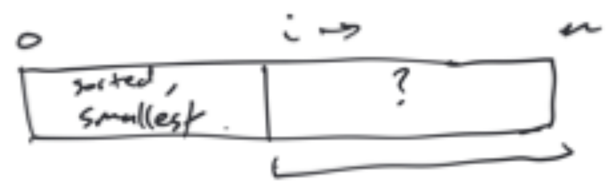
⋮

nth iter: n steps
(in worst case)

$$1+2+3+4+\cdots+n = \frac{n(n+1)}{2} = \frac{n^2 + n}{2}$$

is $O(n^2)$

Selection sort:



n iterations of loop,

in each iteration, find min. of unsorted portion. $\}$ have to examine every elt. of unsorted portion.

always takes $n-i$ steps.

swap $\}$ constant time

1st iteration: $n-1$ steps

2nd iteration: $n-2$ steps

$\vdots$

nth iter: $n-n=0$ steps

$$(n-1)+(n-2)+\cdots+2+1+0 = \frac{n(n-1)}{2} = \frac{n^2+\cdots}{2}$$

is $O(n^2)$ steps.

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    for (k=0; k<n; k++)
      x=x+1;
```

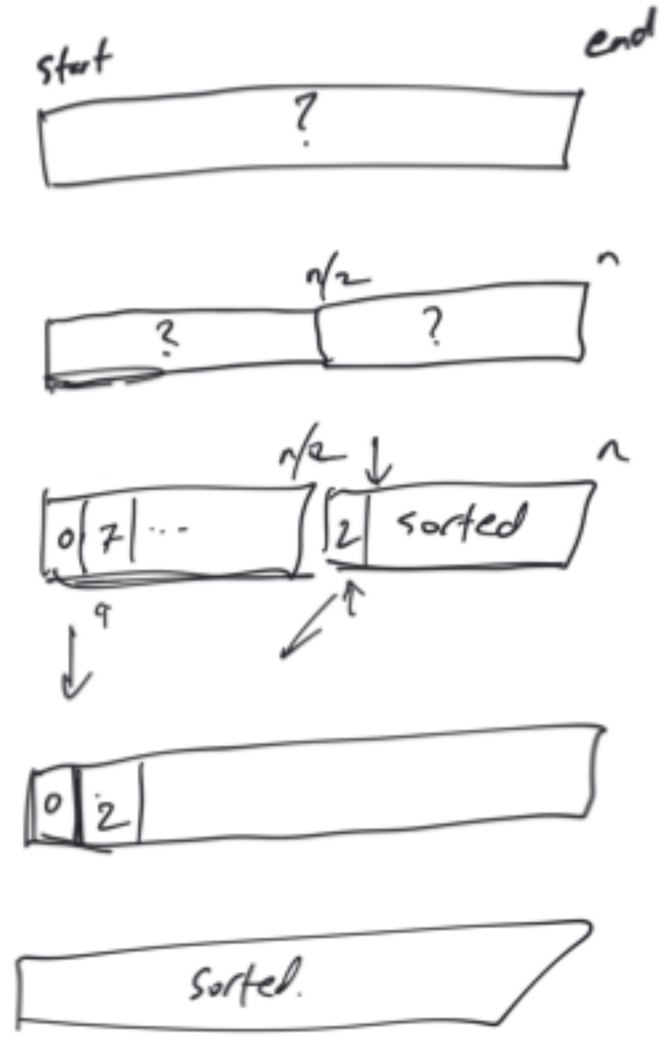
$\}$ n steps $\}$ n^2 steps $\}$ n^3 steps

Divide & conquer:

take a big problem, split into subproblems (multiple, much smaller)

solve subproblems, put solutions together.

Sort:



loop invariants!

recursion!

walk through both arrays, copying smallest elts into big array.

```
/* sorts elts of a */
mergeSort(a, start, end)
  mid = (start + end) / 2
  mergesort(a, start, mid) !
  mergesort(a, mid, end) !
  merge two halves of a
```

base case: sort array of size 0 or 1 by doing nothing.

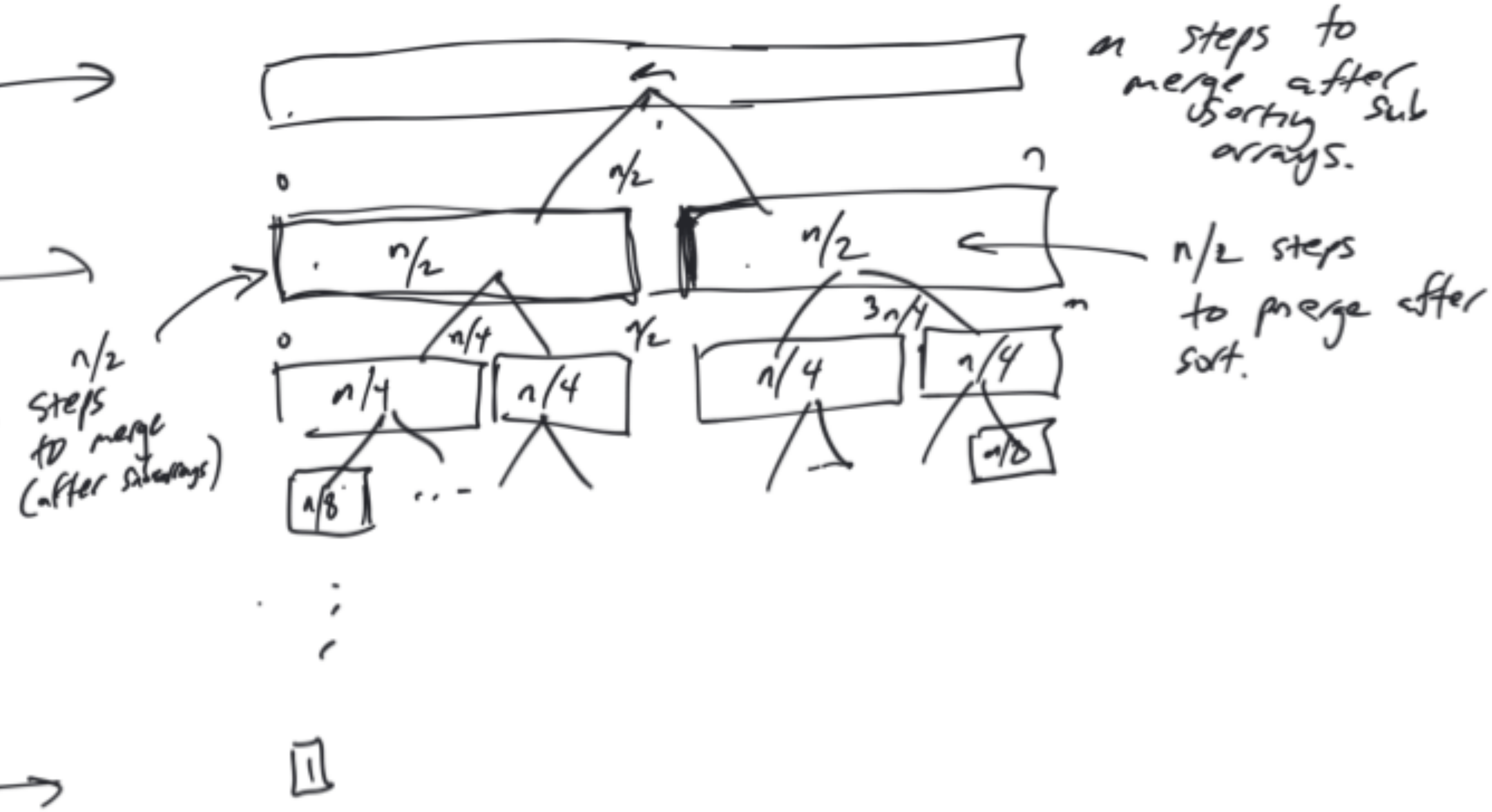
total: n

$n/2 + n/2 = n$

n

n

n



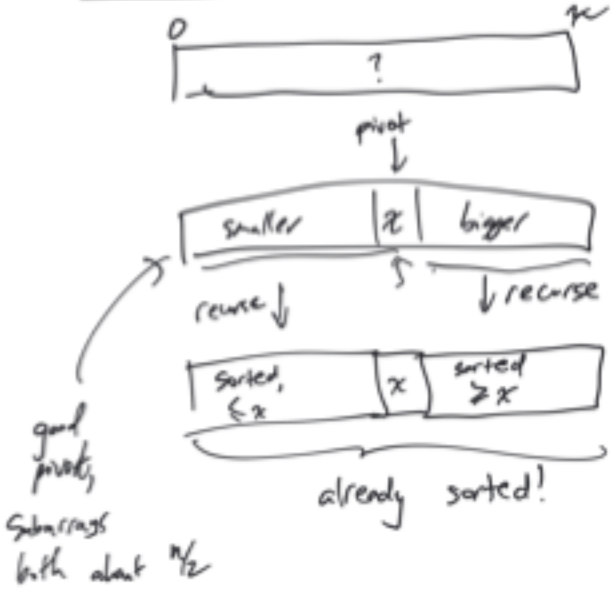
log n layers, each layer is half size of previous layer.

---

total: $O(n \log n)$ steps

better than $O(n^2)$

Quicksort



① partition (linear time) (need to rearrange to partition)

② recursively sort

③ Profit!

each layer has roughly n steps

n log n time.

worst case for quick sort:

pivot is smallest or largest.

$O(n^2) = n + (n-1) + (n-2) + \cdots + 0$

randomly select a pivot, on average you get close to $n \log n$ performance.