I & C SCI 46 Winter 2023
Lecture 22: Fundamentals of Sorting II

InsertionSort

Idea:

85	24	63	45	17	31	96	50
24	85						
24	63	85					

InsertionSort

for $j \leftarrow 2$ to n **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$key **do**
    $A[i+1] \leftarrow A[i]$
    $i = i - 1$
  $A[i+1] \leftarrow$ key

each iteration fixes one inverted pair from the input

- What is the running time of InsertionSort?

$\Theta(n + d)$

fewest inv pairs input: zero

most inverted pairs: $\binom{n}{2}$

InsertionSort

for $j \leftarrow 2$ to n **do**
  key $\leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] >$key **do**
    $A[i+1] \leftarrow A[i]$
    $i = i - 1$
  $A[i+1] \leftarrow$ key

- Why is InsertionSort correct?
- What is true *every time* we check the **for** loop? (including the time we find $j > n$ and stop)

$A[1 \ldots j-1]$ is sorted AND is the elements that began there

About that running time ...

- Why are we so concerned with worst case?
- Why not examine average case?

all permutations are equally likely.

NO $\cancel{\frac{n^2+n}{2}}$

Expected number of inverted pairs?

probability any given pair is inverted

$\frac{1}{2}$

$\binom{n}{2}$ potentially inverted

$\frac{1}{2} \cdot \frac{n(n-1)}{2}$

HeapSort

Idea: Use a max heap.

- Find max, put max at end
- Then second-max, etc.
- Use the yet-to-be-sorted array as max heap

Heapify: make array into max heap

not "in place"

- Idea 1: insert each into growing heap

$O(n)$ add'tl space

```
H = empty max heap
for i=1..n
    H.insert(A[i])
```

$O(n \log n)$

```
for j=n...1
    A[j] = H.extractMax()
```

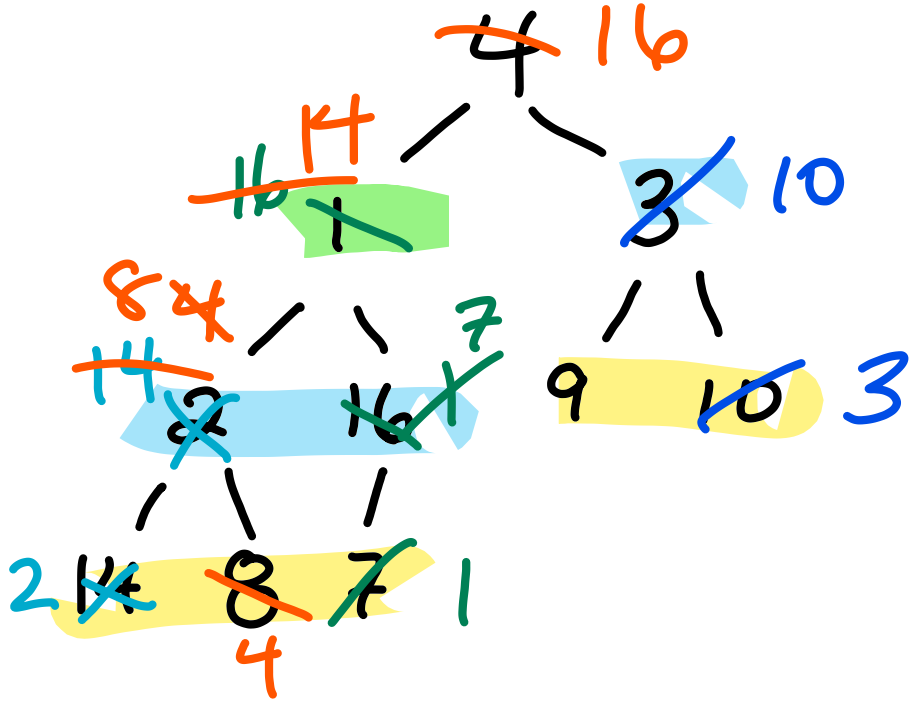
$O(n \log n)$

Heapify: Better way

n/4 of vector: one layer of children

4	1	3	2	16	9	10	14	8	7

- Treat array as ~~heap~~ complete binary tree. Where are leaf nodes? last half
- What should we do with non-leaf nodes?
- In which order?



How much work to heapify?

how many nodes?	work each?
$\frac{n}{2}$	0
$n/4$	1
$n/8$	2
$n/16$	3
$n/32$	4

How long to heapify?

- The cost to insert varies by height.
- Node at height h costs $\mathcal{O}(h)$.
  How many nodes at height h? $\approx \frac{n}{2^{h+1}}$
- How many different height values are there?

  $0 \ldots \log n$
- Cost for total is:

$$\sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \mathcal{O}(h) = \mathcal{O}\left(n \underbrace{\sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h}}_{\leq 2}\right)$$

Using formula for infinite geometric series:

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2}$$

And how, HeapSort

16	14	10	8	7	9	3	2	4	1