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					
24	45	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

is this an inverted pair?

while loop, one iteration: fix one inverted pair

- What is the running time of InsertionSort?

$\Theta(n + d)$

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)

$A[1..j-1]$ is sorted and original elements

About that running time ...

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

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

$\frac{1}{2}\binom{n}{2}$ potential inverted pairs

$E[\#\text{inv pairs}] = \frac{1}{2} \cdot \frac{n(n-1)}{2} = \frac{n(n-1)}{4}$

probability any is inverted

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

- Idea 1: insert each into growing heap

$O(n)$ space

```
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

	16	10	14	7		3	2		1
4	~~1~~	~~3~~	~~2~~	~~16~~	9	~~10~~	~~14~~	8	~~7~~

- Treat array as ~~heap~~. Where are leaf nodes?

  complete binary tree

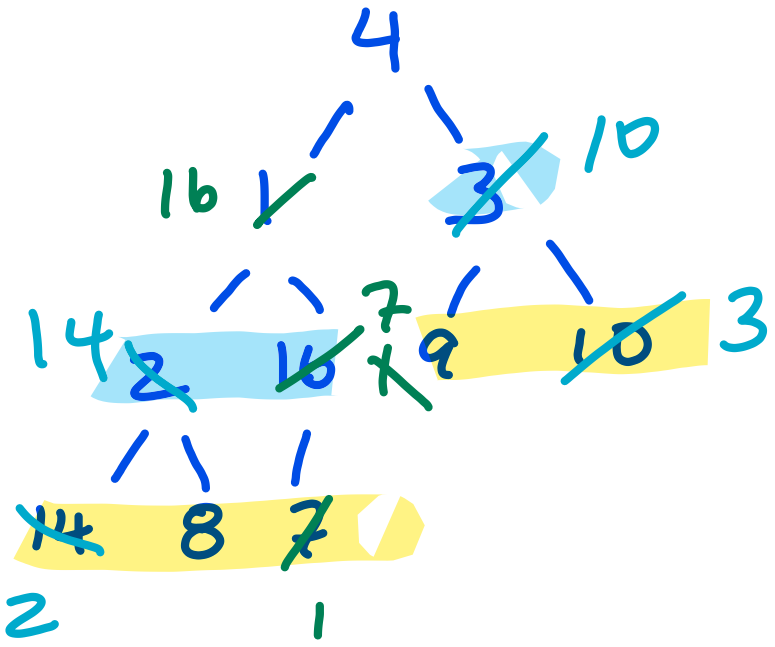
- What should we do with non-leaf nodes?

- In which order?

$\frac{n}{2}$: no comparisons/swaps

$\frac{n}{4}$: one comparison/swap

$\frac{n}{8}$: two



How much work to heapify?

How many nodes?	How many steps? (each)
$\frac{n}{2}$	0
$\frac{n}{4}$	1
$\frac{n}{8}$	2
$\frac{n}{16}$	3
$\frac{n}{64}$	5

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? $\frac{n}{2^{h+1}}$
- How many different height values are there? $\log n$
- Cost for total is:

$$\sum_{h=0}^{\lfloor \log n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil \mathcal{O}(h) = \mathcal{O}\Big(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h}\Big)$$

$O(1)$

Using formula for infinite geometric series:

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

And how, HeapSort

16	14	10	8	7	9	3	2	4	1