HeapSort, CountingSort, and RadixSort

Tufts University

Warm-up Question

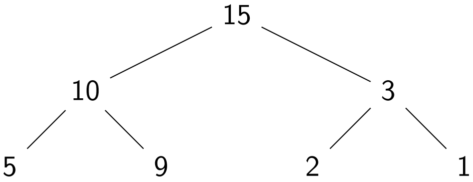
How would you sort this array?

A =	1	0	1	1	0	0	1	1	0	1	1	0	0	1	0

Cheat of the day

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$$

Heaps



15	10	3	5	9	2	1

Fairly simple data structure

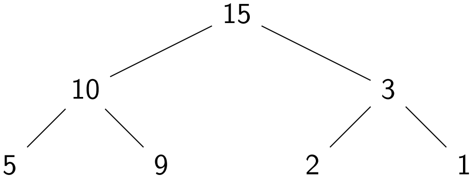
Used to dynamically maintain smallest/largest number

Two main invariants:

Shape We insert numbers from left to right and top to bottom

Size Each node is smaller/larger than its children

Heap Operations

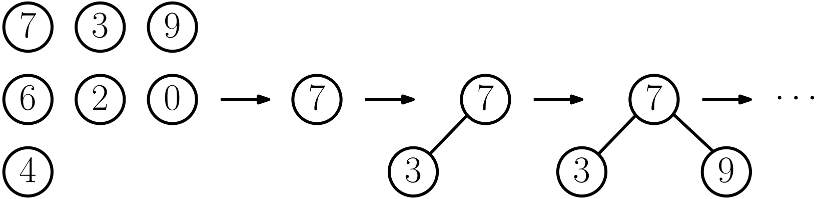


Initialization Create an empty array. Remember that size = 0

Insert Create node as bottom-rightmost leaf. Possibly **float**

Extract root Replace root with last leaf. Possibly **sink**

Top-down heap construction



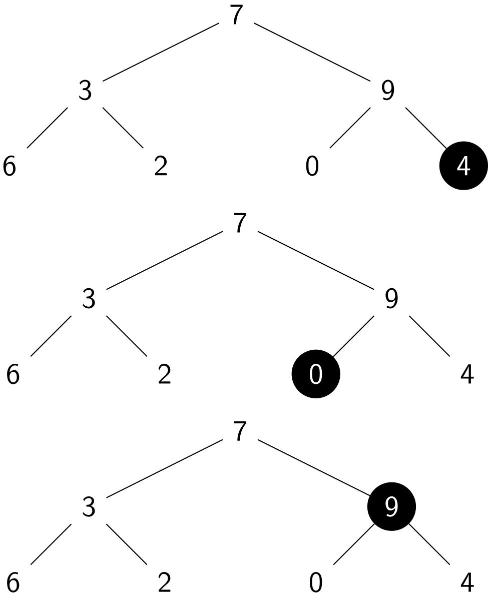
Given *n* numbers, how to insert them in a heap?

Simple version: create empty heap

Insert elements one by one

Single insertion needs $\Theta(\log n)$ time ⇒ total time = $\Theta(n \log n)$

Bottom-up heap construction



Bottom-up heap construction

Fixing the root of a heap with n elements needs $\log n$ time

Many subproblems are small, some are big

Total time is $\sum_{i=0}^{n/2} \frac{n}{2^i} \log 2^i = \sum_{i=0}^{n/2} \frac{n}{2^i} i = n \sum_{i=0}^{n/2} \frac{i}{2^i} < n \frac{1/2}{1-1/2} = n$

Lemma

We can create a heap containing n given numbers in $\Theta(n)$ time.

HeapSort

1. Insert all elements in a Min-heap (bottom-up or top-down)
2. Report top of the heap as smallest remaining element
3. Remove top of the heap
4. Return to step 2 until heap is empty

inplace HeapSort

An inplace algorithm only uses $O(1)$ space (other than the input)

9	5	8	3	2	7	6

9	5	8	3	2	6	7	$\Rightarrow$	8	5	7	3	2	6	

1. Insert all elements in a ~~Min-heap~~ Max-heap
2. ~~Report top of the heap as smallest remaining element~~
3. Remove top of the heap and place at the end of the array
4. Return to step 2 until heap is empty
5. Return input array

Theorem

HeapSort *is an inplace algorithm that sorts n numbers in* $\Theta(n \log n)$ *time.*

Additional Practice questions

Run HeapSort for an array of 10 numbers

- Build heap bottom-up
- Draw heap and tree after each extraction
- What portion of the array corresponds to the heap?
- What portion of the array corresponds to the solution?

Warm-up Question

How would you sort this array?

$$A = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ \hline\end{array}$$

CountingSort

A =	4	2	3	2	1	4	4	4	2	1	3	2	2	1	3

- Outside-the-box sorting strategy

  Numbers to sort are in range $\{0, \ldots, k\}$

  In the example, $n = 15$ and $k = 4$

  Rather than compare, we *count*

Two step algorithm:

Count occurrences of each key

Create solution from frequency array

Step 1: counting occurrences

$A =$

4	2	3	2	1	4	4	4	2	1	3	2	2	1	3

Counting array

$C =$

$C =$

0	3	5	3	4

```
Create an array $C$ of length $k$. All entries zero.
For $i$ from 1 to $n$
    $C[A[i]] \leftarrow C[A[i]] + 1$
```

Step 2: produce solution

A =

4	2	3	2	1	4	4	4	2	1	3	2	2	1	3

Counting array

C =

0	3	5	3	4

Output array

B =


```
$\mathrm{pos} \leftarrow 0$
For $i$ from 1 to $k$
    For $j$ from 1 to $C[i]$
    $B[\mathrm{pos}] \leftarrow i$
    $\mathrm{pos} \leftarrow \mathrm{pos} + 1$
```

Discussion

Theorem

Given an array A with n numbers whose values range from 0 to k, CountingSort *will sort A in $\Theta(n + k)$ time*

What if A contains *more* than just numbers?

Example: office hours for COMP 160
TA in charge, time slot, location, etc
key = number of students attending

Must link each entry in B with an entry in A. How?

Easy solution

$A =$

4	2	3	$\underline{2}$	1	$\underline{4}$	$\overline{4}$	4	2	$\underline{1}$	$\underline{3}$	2	2	$\overline{1}$	$\overline{3}$

Counting array $C =$

Output array

$B =$

Counting phase C now stores linked list. Insert at back

Production phase Unload from LL onto C

Discussion

Counting Sort is **very** fast ($O(n+k)$)

is also **stable** (in ties, returns same order as input)

...but has heavy requirements

Can you modify so that it works for wider ranges of A?

- $[s, s+k]$ (for any $s, k > 0$)?
- $[-k, k]$ (that is, negative range)?
- $[a, z]$ (i.e., lowercase letters)?

Can we push beyond?

RadixSort

How can we sort these numbers?

$A =$

4234	2331	23	9982	1887	677	4456	4432
4338	0	561	17	2555	7567	233	110
6785	9374	5624	4402	5656	3992	1345	309
331	2348	434	9994	3456	177	32	4589

CountingSort does not help ($k > 9300$ but $n = 32$)

How about we sort one digit at a time?

Introducing RadixSort

Apply CountingSort one digit at a time

Key point: sort digits from right to left!

4234				
2331				
6785				
0331				
5654				
2234				
7134				
0034				
4230				

Discussion

Runtime?

Depends on two new parameters

r or radix (number of different characters).

- 2 (if binary)
- 10 (for decimal)
- 26 (alphabet)

ℓ or length (number of digits in each item)

Run ℓ instances of $\textsc{CountingSort} \Rightarrow O(\ell(n+r))$

Correctness? By induction on ℓ

Base Case $\ell = 1$ We are just running $\textsc{CountingSort}$

Induction in recitation!

Additional practice questions

- Give pseudocode of RadixSort
- Make QuickSort, InsertionSort, etc stable
- How many different numbers with radix d and length l?
- Can you use RadixSort on words with different lengths?