Heapsort
Announcements

Moodle had issues last night, homework due tonight at 11:55pm
It is possible to represent binary trees as an array.
Binary tree as array

index 'i' is the parent of '2i' and '2i+1'

A = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]
Binary tree as array

Is it possible to represent any tree with a constant branching factor as an array?
Binary tree as array

Is it possible to represent any tree with a constant branching factor as an array?

Yes, but the notation is awkward
A max heap is a tree where the parent is larger than its children (A min heap is the opposite)
Heapsort

The idea behind heapsort is to:

1. Build a heap
2. Pull out the largest (root) and re-compile the heap
3. (repeat)
Heapsort

To do this, we will define subroutines:

1. Max-Heapify = maintains heap property

2. Build-Max-Heap = make sequence into a max-heap
Max-Heapify

Input: a root of two max-heaps
Output: a max-heap
Max-Heapify

Pseudo-code Max-Heapify(A,i):
left = left(i)  // 2*i
right = right(i)  // 2*i+1
L = arg_max( A[left], A[right], A[i] )
if (L not i)
    exchange A[i] with A[L]
Max-Heapify(A, L)
// now make me do it!
Max-Heapify

Runtime?
Max-Heapify

Runtime?

Obviously (is it?): \( \lg n \)

\[ T(n) = T(2/3 \ n) + O(1) \] // why?

Or...

\[ T(n) = T(1/2 \ n) + O(1) \]
Master's theorem: (proof 4.6)
For \( a \geq 1, b \geq 1, T(n) = a \ T(n/b) + f(n) \)

If \( f(n) \) is... (3 cases)
\( O(n^c) \) for \( c < \log_b a \), \( T(n) \) is \( \Theta(n^{\log_b a}) \)
\( \Theta(n^{\log_b a}) \), then \( T(n) \) is \( \Theta(n^{\log_b a \ \lg n}) \)
\( \Omega(n^c) \) for \( c > \log_b a \), \( T(n) \) is \( \Theta(f(n)) \)
Max-Heapify

Runtime?

Obviously (is it?): \( \lg n \)

\[ T(n) = T(2/3 \ n) + O(1) \quad \text{// why?} \]

Or...

\[ T(n) = T(1/2 \ n) + O(1) = O(\lg n) \]
Next we build a full heap from an unsorted sequence

Build-Max-Heap(A)
for i = floor( A.length/2 ) to 1
  Heapify(A, i)
Build-Max-Heap

Red part is already Heapified
Build-Max-Heap

Correctness:
Base: Each alone leaf is a max-heap.
Step: if A[i] to A[n] are in a heap, then Heapify(A, i-1) will make i-1 a heap as well.
Termination: loop ends at i=1, which is the root (so all heap)
Build-Max-Heap

Runtime?
Build-Max-Heap

Runtime?

O(n lg n) is obvious, but we can get a better bound...

Show ceiling(n/2^{h+1}) nodes at any level 'h', with h=0 as bottom
Build-Max-Heap

Heapify from height 'h' takes $O(h)$

$$\sum_{h=0}^{\lg n} \text{ceiling}(n/2^{h+1}) \in O(h)$$

$$= O(n \sum_{h=0}^{\lg n} \text{ceiling}(h/2^{h+1}))$$

$$= O(n \sum_{x=0}^{\infty} k \ x^k = x/(1-x)^2, \ x=1/2)$$

$$= O(n \ 4/2) = O(n)$$
Heapsort:
Build-Max-Heap(A)
for i = A.length to 2
    Swap A[ 1 ], A[ i ]
A.heapsize = A.heapsize – 1
Max-Heapify(A, 1)
Heapsort

You try it!

Sort: $A = [1, 6, 8, 4, 7, 3, 4]$
Heapsort

First, build the heap starting here

\[ A = [1, 6, 8, 4, 7, 3, 4] \]
\[ A = [1, 6, 8, 4, 7, 3, 4] \]
\[ A = [1, 7, 8, 4, 6, 3, 4] \] - recursive
\[ A = [8, 7, 4, 4, 6, 3, 1] \] - done
Heapsort

Move first to end, then re-heapify

A = [8, 7, 4, 4, 6, 3, 1], move end
A = [1, 7, 4, 4, 6, 3, 8], heapify
A = [7, 1, 4, 4, 6, 3, 8], rec. heap
A = [7, 6, 4, 4, 1, 3, 8], move end
A = [3, 6, 4, 4, 1, 7, 8], heapify
A = [6, 3, 4, 4, 1, 7, 8], rec. heap
A = [6, 4, 4, 3, 1, 7, 8], next slide..
Heapsort

A = [6, 4, 4, 3, 1, 7, 8], move end
A = [1, 4, 4, 3, 6, 7, 8], heapify
A = [4, 4, 1, 3, 6, 7, 8], move end
A = [3, 4, 1, 4, 6, 7, 8], heapify
A = [4, 3, 1, 4, 6, 7, 8], move end
A = [1, 3, 4, 4, 6, 7, 8], heapify
A = [3, 1, 4, 4, 6, 7, 8], move end
A = [1, 3, 4, 4, 6, 7, 8], done
Heapsort

Runtime?
Heapsort

Runtime?

Run Max-Heapify $O(n)$ times
So... $O(n \lg n)$
Sorting comparisons:

<table>
<thead>
<tr>
<th>Name</th>
<th>Average</th>
<th>Worst-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion[i]</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Merge[p]</td>
<td>$O(n \lg n)$</td>
<td>$O(n \lg n)$</td>
</tr>
<tr>
<td>Heap[i]</td>
<td>$O(n \lg n)$</td>
<td>$O(n \lg n)$</td>
</tr>
<tr>
<td>Quick[p]</td>
<td>$O(n \lg n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Counting[s]</td>
<td>$O(n + k)$</td>
<td>$O(n + k)$</td>
</tr>
<tr>
<td>Radix[s]</td>
<td>$O(d(n+k))$</td>
<td>$O(d(n+k))$</td>
</tr>
<tr>
<td>Bucket [s,p]</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
Sorting comparisons:

https://www.youtube.com/watch?v=kPRA0W1kECg
Selection

Graphs for Quantitative Variables

Boxplot

**NOTE:**
Min=16
is greater than
Q1-1.5(Q3-Q1)
=18.5-1.5(2)
=15.5
SO...stop at Min

Max=23
is less than
Q3+1.5(Q3-Q1)
= 20.5+1.5(2)
= 23.5
So...stop at Max.
Heaps can also be used to implement priority queues (i.e. airplane boarding lines)

Operations supported are: Insert, Maximum, Extract-Max and Increase-key
Priority queues

Maximum(A):
return A[1]

Extract-Max(A):
max = A[1]
A.heapsize = A.heapsize – 1
Max-Heapify(A, 1), return max
Priority queues

Increase-key(A, i, key):
A[i] = key
while (i > 1 and A[floor(i/2)] < A[i])
    swap A[i], A[floor(i/2)]
i = floor(i/2)

Opposite of Max-Heapify... move high keys up instead of low down.
Priority queues

Insert(A, key):
A.heapsize = A.heapsize + 1
A[A.heapsize] = -\infty
Increase-key(A, A.heapsize, key)
Priority queues

Runtime?

Maximum =
Extract-Max =
Increase-Key =
Insert =
Priority queues

Runtime?

Maximum = $O(1)$
Extract-Max = $O(lg\ n)$
Increase-Key = $O(lg\ n)$
Insert = $O(lg\ n)$
Selection

Selection given a set of (distinct) elements, finding the element larger than $i - 1$ other elements

Selection with...

$i=n$ is finding maximum

$i=1$ is finding minimum

$i=n/2$ is finding median
Maximum

Selection for any i is $O(n)$ runtime

Find max in $O(n)$?
Maximum

Selection for any i is $O(n)$ runtime

Find max in $O(n)$?

$$\text{max} = A[1]$$

for $i = 2$ to $A.length$

  if $(A[i] > \text{max})$
  $$\text{max} = A[i]$$
Max and min

It takes about $n$ comparisons to find max

How many would it take to find both max and min at same time?
Max and min

It takes about $n$ comparisons to find max

How many would it take to find both max and min at same time?

Naïve = $2n$
Smarter = $3/2 \ n$
Max and min

\[
\begin{align*}
\text{smin} &= \min(A[1], A[2]) \\
\text{smax} &= \max(A[1], A[2]) \\
\text{for } i &= 3 \text{ to } A.\text{length} \text{ step } 2 \\
\text{if } (A[i] > A[i+1]) \\
&\quad \text{smax} = \max(A[i], \text{smax}) \\
&\quad \text{smin} = \min(A[i+1], \text{smin}) \\
\text{else} \\
&\quad \text{smax} = \max(A[i+1], \text{smax}) \\
&\quad \text{smin} = \min(A[i], \text{smin})
\end{align*}
\]
Randomized selection

Remember quicksort?

Partition step

Choose 31 as the pivot

Recursively sort subsequence on each side of pivot
Randomized selection

To select i:

1. Partition on random element

2. If partitioned element i, end otherwise recursively partition on side with i
Randomized selection

\{2, 6, 4, 7, 8, 4, 7, 2\} find \(i = 5\)

Pick pivot = 4

\{2, 6, 4, 7, 8, 2, 7, 4\}

\{2, 6, 4, 7, 8, 2, 7, 4\}

\{2, 6, 4, 7, 8, 2, 7, 4\}

\{2, 4, 6, 7, 8, 2, 7, 4\}

\{2, 4, 6, 7, 8, 2, 7, 4\}

\{2, 4, 6, 7, 8, 2, 7, 4\}

\{2, 4, 6, 7, 8, 2, 7, 4\}

\{2, 4, 6, 7, 8, 2, 7, 4\}
Randomized selection

\{2, 4, 6, 7, 8, 2, 7, 4\}
\{2, 4, 2, 7, 8, 6, 7, 4\}
\{2, 4, 2, 7, 8, 6, 7, 4\}
\{2, 4, 2, 7, 8, 6, 7, 4\}
\{2, 4, 2, 4, 7, 8, 6, 7\}
\{2, 4, 2, 4, 5, 6, 7, 8\}

i=5 on green side, recurse
Randomized selection

\[
\{7, 8, 6, 7\} \text{ pick pivot } = 6 \\
\{7, 8, 7, 6\} \\
\{7, 8, 7, 6\} \\
\{7, 8, 7, 6\} \\
\{7, 8, 7, 6\} \\
\{6, 7, 8, 7\} \\
\{5, 6, 7, 8\}
\]

found \(i=5\), value = 6
Randomized selection

Quicksort runs in $O(n \log n)$, but we only have sort one side and sometimes stop early.

This gives randomized selection $O(n)$ running time (proof in book, I punt)
Randomized selection

Just like quicksort, the worst case running time is $O(n^2)$

This happens when you want to find the min, but always partition on the max
A worst case $O(n)$ selection is given by Select: (see code)

1. Make $n/5$ groups of 5 and find their medians (via sorting)
2. Recursively find the median of the $n/5$ medians (using Select)
3. Partition on median of medians
4. Recursively Select correct side
Proof of the general case:
\[ T(n) = \sum_i T(k_i n + q_i) + O(n) \]

// assume \( T(n) \) is \( O(n) \)
\[ T(n) = cn - cn+c \sum_i (k_i n + q_i) + an \]

so \( T(n) \) is \( O(n) \) if:
\[ -cn+c \sum_i (k_i n + q_i) + an \leq 0 \]
\[ an \leq c(n (1 - \sum_i k_i) - \sum_i q_i) \]
Select

\[ a_n \leq c(n(1 - \sum_i k_i) - \sum_i q_i) \]
\[ a_n/(n(1-\sum_i k_i) - \sum_i q_i) \leq c \]

// Pick \( n > 2(\sum_i q_i/(1 - \sum_i k_i)) \)
\[ c \geq a \times 2(\sum_i q_i/(1-\sum_i k_i))/(\sum_i q_i) \]
\[ c \geq 2 a / (1- \sum_i k_i) \]

Done as \( \sum_i k_i < 1 \)
Select runs in:
\[ T(n) = T(\text{ceiling}(n/5)) + T(7n/10 + 6) + O(n) \]

By the previous proof this is \( O(n) \):
\[ \text{ceiling}(n/5) + 7n/10 + 6 \leq n/5 + 1 + 7n/10 + 6 = 9n/10 + 7 \]
\[ \sum_{k} k = 9/10 < 1, \text{ done} \]
Select

Does this work for making:

(1) $n/3$ groups of 3?

(2) $n/7$ groups of 7?

(3) $n/9$ groups of 9?