Order Statistics, Data Structures, and Heaps

Oh my!

Administrivia…

- Today:
  - Order Statistics
  - Data Structures
  - Heap Sort
- Homework
  - Big-Oh notation
  - Recursion trees
- Feedback

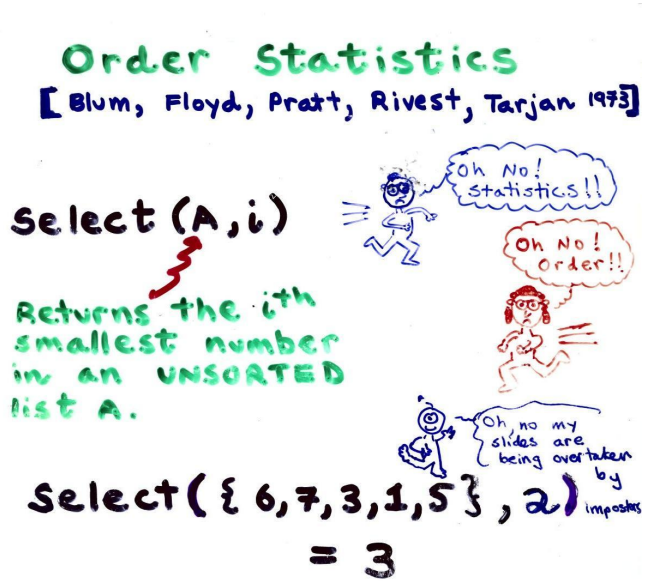
The selection problem

- ***i*th order statics** is the *i*th smallest element of a set of *n* elements
  - The **minimum** is the first order statistic (*i*=1)
  - The **maximum** is the last order statistic (*i*=n)
  - A **median** is the "halfway point" of the set
    - When *n* is odd, median is unique, at *i*=(*n*+1)/2
    - When *n* is even, there are two medians: *i*=*n*/2 and *i*=*n*/2+1

The ***selection problem***:

Input: A set A of n distinct numbers and a number i, with $1 \le i \le n$.

Output: The element $x \in A$ that is larger than exactly $i - 1$ other elements in A. In other words, the ith smallest element of A.



Solving the selection problem

- Obvious approach
  - Sort in O(n log n)
  - Select i^{th} element
- Is there a faster way?
  - In the expected case?
  - In the worst case?

The Cast…

Manuel Blum

Manuel Blum is a computer scientist who received the Turing Award in 1995 "In recognition of his contributions to the foundations of computational complexity theory and its application to cryptography and program checking". Wikipedia

Born: April 26, 1938 (age 74), Caracas

Spouse: Lenore Blum

Children: Avrim Blum

Education: Massachusetts Institute of Technology (1964), More

Awards: Turing Award

Robert Floyd

Robert W Floyd was an eminent computer scientist. His contributions include the design of the Floyd–Warshall algorithm, which efficiently finds all shortest paths in a graph, Floyd's cycle-finding ... Wikipedia

Born: June 8, 1936, New York

Died: September 25, 2001, Palo Alto

Books: The Language of Machines: an Introduction to Computability and Formal Languages

Education: University of Chicago (1958), University of Chicago (1953)

Awards: Turing Award

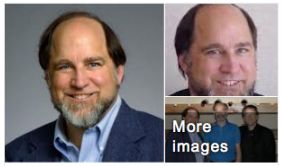
The Cast…

Vaughan Ronald Pratt

Vaughan Ronald Pratt, a Professor Emeritus at Stanford University, was one of the earliest pioneers in the field of computer science. Wikipedia

Born: 1944

Education: Stanford University (1972), University of Sydney (1970), University of Sydney (1967)



Ronald Rivest

Ronald Linn Rivest is a cryptographer. He is the Andrew and Erna Viterbi Professor of Computer Science at MIT's Department of Electrical Engineering and Computer Science and a member of MIT's Computer Science and Artificial Intelligence Laboratory. Wikipedia

Born: 1947, Schenectady

Books: Introduction to Algorithms

Education: Stanford University (1974), Yale University (1969)

Awards: Turing Award, Marconi Prize

The Cast…

Robert Tarjan

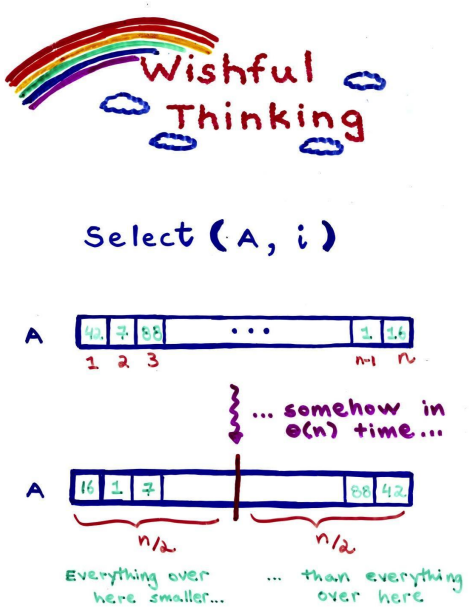
Robert Endre Tarjan is an American computer scientist. He is the discoverer of several graph algorithms, including Tarjan's off-line least common ancestors algorithm, and co-inventor of both splay trees and Fibonacci heaps. Wikipedia

Born: April 30, 1948 (age 64), Pomona

Books: Data Structures and Network Algorithms

Education: Stanford University (1972), Stanford University (1971), California Institute of Technology (1969)

Awards: Turing Award, Nevanlinna Prize



Worksheet:Think, pair, share

- Sketch an algorithm for select that is O(n) in expected case

Selection in Expected Linear Time

```
RANDOMIZED-SELECT(A, p, r, i)
  if p == r
      return A[p]
  q = RANDOMIZED-PARTITION(A, p, r)
  k = q − p + 1
  if i == k          // pivot value is the answer
      return A[q]
  elseif i < k
      return RANDOMIZED-SELECT(A, p, q − 1, i)
  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

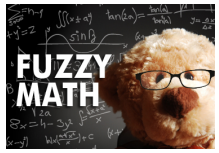
We've seen Randomized-Partition before… in Quicksort!

- After R-P call:

  $A[p \ldots q-1] \le A[q]$; • Pivot is k^{th} (k=q-p+1)element of $A[p..r]$

  $A[q+1 \ldots r] > A[q]$
- 3 cases:
  - i=k: done!
  - i<k: we want ith element of A[$p..q-1$]
  - i>k: we want (i-k)th element of A[$q+1...r$]

Random Select (rough) Analysis



- Worst-case: $O(n^2)$
- Expected-case?
  - Fact: 50% of all number fall in 25th to 75th percentile
  - Fact: A fair coin needs to be flipped twice in expectation to see a heads
  - Thus: in expectation, we shrink array to at most ¾ of it's original size
  - Ergo: In expectation, $T(n) \le T(3n/4) + O(n) \rightarrow T(n) = O(n)$

Linear worst case: Min / max

{17,10,4,37,12,28,27,14}

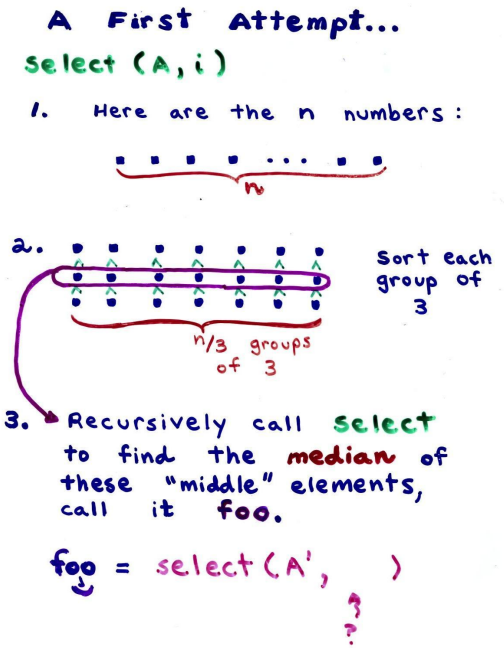
- Find min? max? both?
  - How many comparisons did it take?
  - *n*-1 *2 = 2n-2
- A more pair-fect way?
  - Maintain min/max
  - Process in pairs
    - Compare larger to max, smaller to min $3\lfloor n/2 \rfloor$

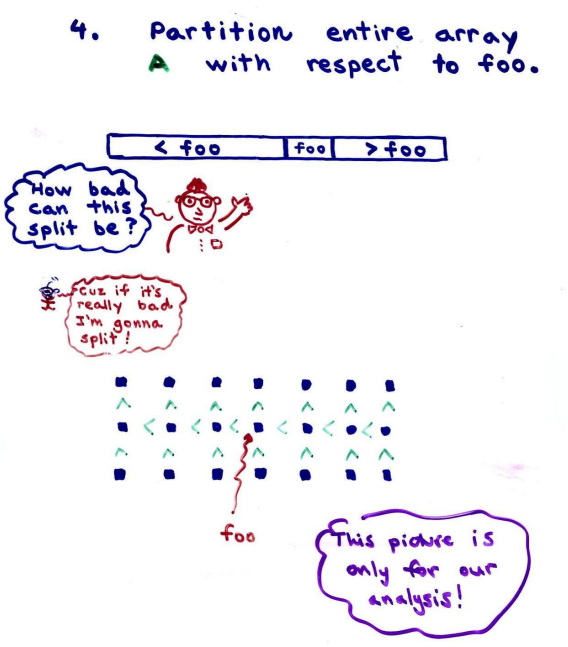
A first attempt

1. Divide n elements into groups of 3
2. Find median of n/3 groups
3. X = Find median of the n/3 medians
4. Partition on median of median
5. If i=k, return X

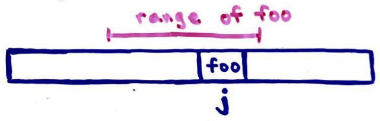
   If i<k, recurse on smaller partition

   If i>k, recurse on larger partition





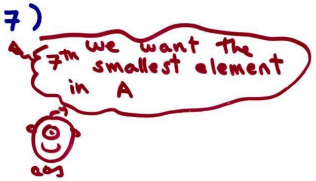
5. we know the location of foo in the partitioned array...



Ex

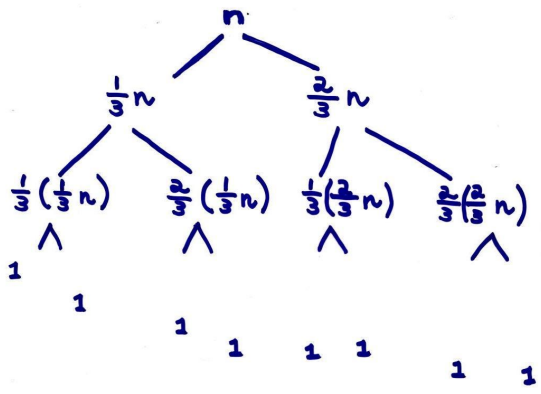
A = | 100 | 6 | 8 | 20 | 42 | 4 | 5 | 3 | 2 |
1 2 3 4 5 6 7 8 9

select (A, 7)



Worst case:

$$T(n) = T\left(\tfrac{1}{3}n\right) + T\left(\tfrac{2}{3}n\right) + c \cdot n$$



Analysis

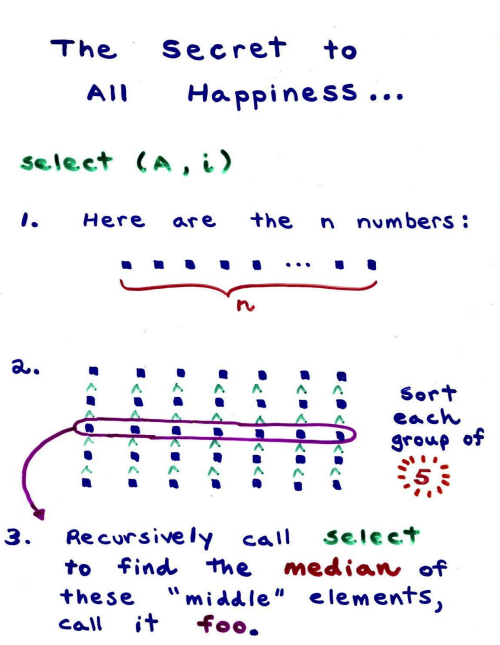
- Steps 1,2,4 all take O(n)
- Step 3: recurses on how many elements?
- Step 5: recurses on how many elements?

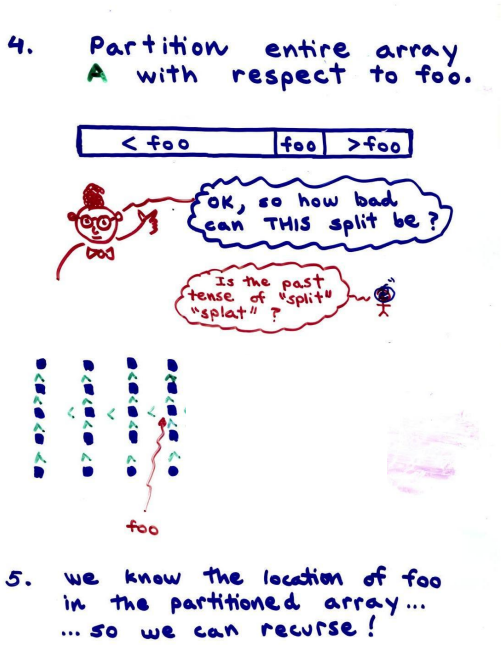
5: The secret of all happiness!!

1. Divide n elements into groups of n/5
2. Find median of n/5 groups
3. X = Find median of the n/5 medians
4. Partition on median of median
5. If i=k, return X

   If i<k, recurse on smaller partition

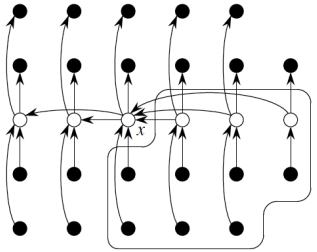
   If i>k, recurse on larger partition





Analysis

- Steps 1,2,4 all take O(n)
- Step 3: recurses on how many elements?
- Step 5: recurses on how many elements?



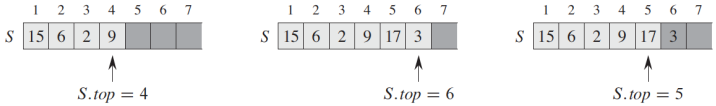
$$\sim T(n) \le T(n/5) + T(7n/10) + O(n)$$

Select vs. Sort

- Select / Randomized-Select are comparison based!
- Sorting is Ω(n log n) under-comparison based sort
- Linear-time sorting algorithm need to make assumptions
- Linear-time selection do NOT require assumptions by solving the problem without sort

Data Structures

Quick Review: Stacks



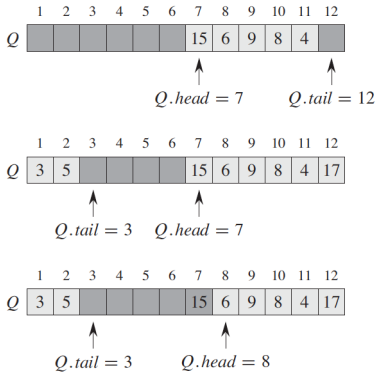
- LIFO
- All operations: O(1)

```
STACK-EMPTY(S)
1  if S.top == 0
2       return TRUE
3  else return FALSE
```

```
PUSH(S, x)
1
2
```

```
POP(S)
1  if STACK-EMPTY(S)
2       error "underflow"
3  else S.top = S.top − 1
4       return S[S.top + 1]
```

Quick Review: Queues

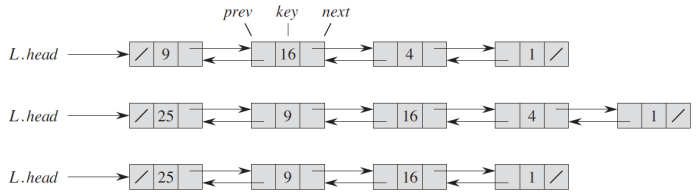


- FIFO
- All operations O(1)

```
ENQUEUE(Q, x)
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3       Q.tail = 1
4  else Q.tail = Q.tail + 1
```

```
DEQUEUE(Q)
1
2
3
4
5
```

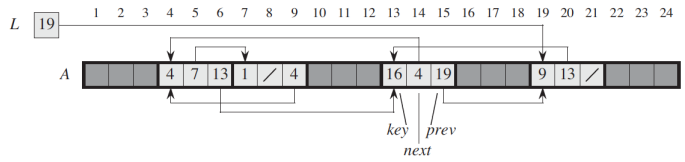
Quick Review: Linked Lists




```
LIST-SEARCH(L, k) O(n)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3       x = x.next
4  return x
```

```
LIST-INSERT(L, x) O(1)
1  x.next = L.head
2  if L.head ≠ NIL
3       L.head.prev = x
4  L.head = x
5  x.prev = NIL
```

```
LIST-DELETE(L, x) O(1)
1
2
3
4
5
```



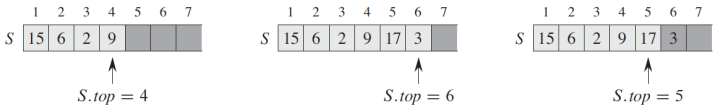
Worksheet: Fill in missing lines

PUSH(S, x)
1
2

DEQUEUE(Q)
1
2
3
4
5

LIST-DELETE(L, x)
1
2
3
4
5

Quick Review: Stacks



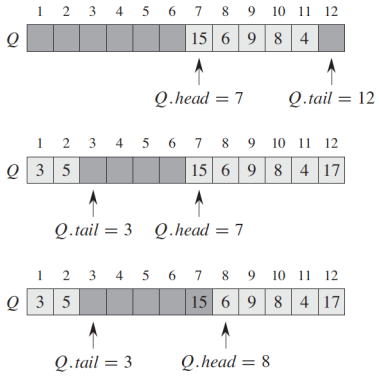
- LIFO
- All operations: O(1)

STACK-EMPTY(S)
1 **if** $S.top == 0$
2 **return** TRUE
3 **else return** FALSE

PUSH(S, x)
1
2

POP(S)
1 **if** STACK-EMPTY(S)
2 **error** "underflow"
3 **else** $S.top = S.top - 1$
4 **return** $S[S.top + 1]$

Quick Review: Queues

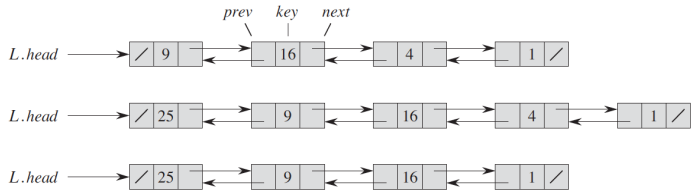


- FIFO
- All operations O(1)

ENQUEUE(Q, x)
1 $Q[Q.tail] = x$
2 **if** $Q.tail == Q.length$
3 $Q.tail = 1$
4 **else** $Q.tail = Q.tail + 1$

DEQUEUE(Q)
1
2
3
4
5

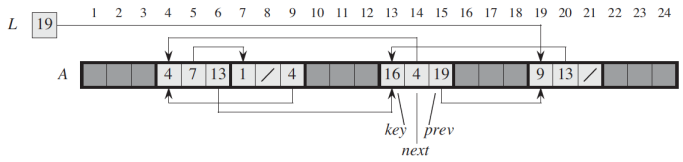
Quick Review: Linked Lists



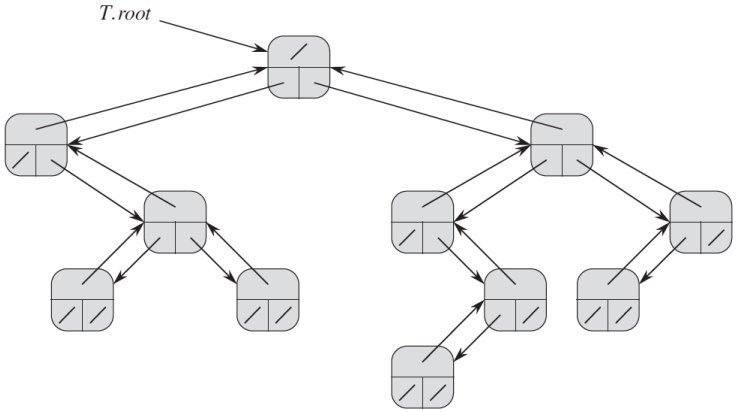
LIST-SEARCH(L, k) O(n)
1 $x = L.head$
2 **while** $x \neq$ NIL and $x.key \neq k$
3 $x = x.next$
4 **return** x

LIST-INSERT(L, x) O(1)
1 $x.next = L.head$
2 **if** $L.head \neq$ NIL
3 $L.head.prev = x$
4 $L.head = x$
5 $x.prev =$ NIL

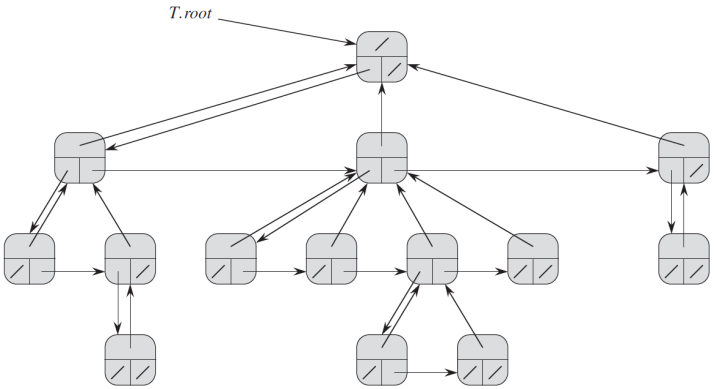
LIST-DELETE(L, x) O(1)
1
2
3
4
5



Quick Review: Binary Tree Representations



Quick Review: Tree Representations

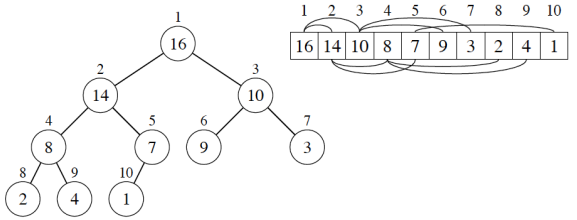


Heapsort

- O(n log n) worst cast
  - Like merge sort
- Sorts in place
  - Like insertion sort
- Combines the best of both algorithms!

Heap data structure

- Heap *A* is a nearly complete binary tree
  - **Height** of a node = # edges on simplest path to leaf
  - **Height** of a tree = height of root = Θ(*n*)
  - Can be stored as an array with root A[1]

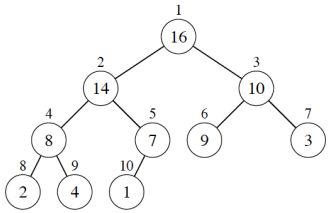


PARENT(i)
1 **return** $\lfloor i/2 \rfloor$

LEFT(i)
1 **return** $2i$

RIGHT(i)
1 **return** $2i + 1$

Heap Property

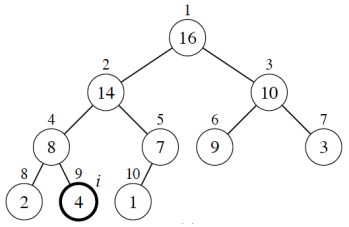


- **(Max-)heap property**: for all nodes *i*, excluding the root, $A[Parent(i)] \geq A[i]$
- For min-heaps, flip the sign (we'll use max-heaps)
- By induction and transitivity of ≤, max-heap property guarantees max element at root
- In general, can be *k*-ary trees rather than just binary

Maintaining the heap property

- Max-Heapify maintains the max-heap property
  - Assume left and right subtrees are max-heaps
  - Swap A[i] with largest child
  - Repeat until we hit a leaf
  - O(log *n*)

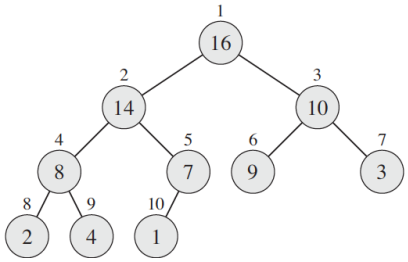
```
MAX-HEAPIFY(A, i, n)
  l = LEFT(i)
  r = RIGHT(i)
  if l ≤ n and A[l] > A[i]
      largest = l
  else largest = i
  if r ≤ n and A[r] > A[largest]
      largest = r
  if largest ≠ i
      exchange A[i] with A[largest]
      MAX-HEAPIFY(A, largest, n)
```



Building a Max Heap

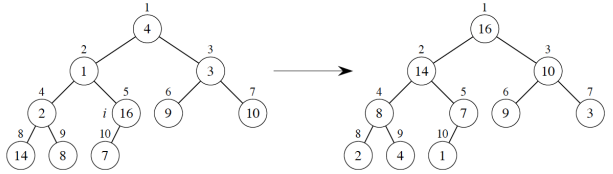
```
BUILD-MAX-HEAP(A, n)
  for i = ⌊n/2⌋ downto 1
      MAX-HEAPIFY(A, i, n)
```

	1	2	3	4	5	6	7	8	9	10
A	4	1	3	2	16	9	10	14	8	7



Building a heap

	1	2	3	4	5	6	7	8	9	10
A	4	1	3	2	16	9	10	14	8	7



- **Loop invariant**: at start of every iteration, each node *i+1,i+2, …,n* is root of max-heap
  - **Initial**: all leaf nodes
  - **Maintain**: by invariance, nodes higher than *i* are roots, *i* is made a root
  - **Termination**: when *i*=0, loop terminates and node 1 is root of max-heap

Analysis

- Simple bound: O(n) calls to Max-Heapify, which each take O(log n) → O(n log n)
- Tighter bound:
  - Max-Heapify is linear in height of node
  - Most nodes have small height

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right)$$

  - O(n)

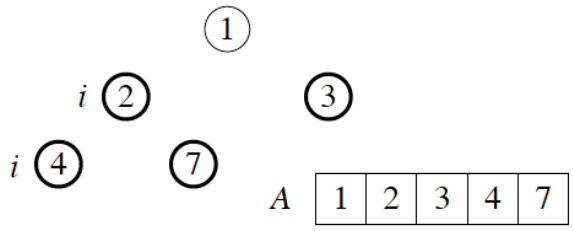
Heapsort

```
HEAPSORT(A, n)
  BUILD-MAX-HEAP(A, n)
  for i = n downto 2
      exchange A[1] with A[i]
      MAX-HEAPIFY(A, 1, i − 1)
```

- The idea:
  - Build a max-heap
  - Swap root to last position
  - "Discard" last node (by decreasing heap size)
    - Max-heapify new root of remaining heap
  - Rinse and repeat

Heapsort

```
HEAPSORT(A, n)
  BUILD-MAX-HEAP(A, n)
  for i = n downto 2
      exchange A[1] with A[i]
      MAX-HEAPIFY(A, 1, i − 1)
```



Heapsort Analysis

```
HEAPSORT(A, n)
  BUILD-MAX-HEAP(A, n)
  for i = n downto 2
      exchange A[1] with A[i]
      MAX-HEAPIFY(A, 1, i − 1)
```

- Build-Max-Heap: O(n)
- **for** loop: n-1 times
  - exchange O(1)
  - Max-Heapify O(log n)
- Total O(n log n)

Priority Queue as Heap

- Balances cost of insertion / extraction
  - Insert(S,*x*): O(log *n*)
  - Maximum(S): O(1)
  - Extract-Max(S): O(log *n*)
  - Increase-Key(S,*x*,*k*): O(log *n*)

Multi-select

- Earlier we discussed a select(S,i) procedure that runs in O(n) in the expected/worst case
- What if over time we knew we'd be calling select on S *n* times?
  - n O(n) → $O(n^2)$
  - Improvement: Store values in case of repeat query?
  - Improvement: Sort values up front?
    - O(n log n) + (n-1) O(1)
    - Average cost per select query? O(log n)