Implementation of priority queue: heap

Maximum: return $A[1]$

takes $\Theta(1)$ time

Extract-Max:

Heap-Extract-Max($A$)

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


$n = n - 1$

Heapify($A, 1$)

return $\text{max}$

Running time: $\Theta(\lg n)$ time
Implementation of priority queue: heap

Insert:
Heap-Insert(A,x)

\[ i=n+1 \]
\[ \text{while } i>1 \text{ and } A[\text{Parent}(i)]<x \]
\[ A[i]=A[\text{Parent}(i)] \]
\[ i=\text{Parent}(i) \]
\[ A[i]=x \]

Running time: \( \Theta(\lg n) \) time
Elementary data structures

stack operations: Stack-empty(S)    Push(S,x)    Pop(S)

queue operations: Enqueue(Q,x)    Dequeue(Q)

variations: FIFO    LIFO

linked lists operations: List-Search(L,k)    List-Insert(L,x)

                   List-Delete(L,x)

Priority Queue: data structure for maintaining set S of elements, each
               with an associated key and supporting operation
               Insert(S,x): insert element x into S
               Maximum(S): return the element of S with largest key
               Extract-Max(S): remove and return the element of S with largest key

Implementation of priority queue as list

               Insert    takes   O(1) time
               Maximum   takes   O(n) time
               Extract-Max    takes   O(n) time
Lecture 7: Selection, Priority Queue

given n inputs, want to find i-th smallest value

If sort first, costs $\Omega(n \lg n)$
Can we do $O(n)$ time?

Randomized-Select(A,p,r,i)
  if $p = r$ then return $A[p]$
  $q = \text{Randomized-partition}(A,p,r)$
  $k = q-p+1$
  if $i \leq k$ then
    return Randomized-Select(A,p,q,i)
  else
    return Randomized-Select(A,q+1,r,i-k)

Analysis: assuming that partition always gets a split into
  $n/b$ and $n-n/b = n(b-1)/b = n/c$  ($c = b/(b-1)$ ) for constants $b$ and $c$
  assume $n/b$ is the bigger side
  $T(n) = T(n/b) + \Theta(n) = \Theta(n)$