Priority Queues

Priority Queue ADT

Binary heaps

Heapsort

Reference: Chapter 6, Algorithms in Java, 3rd Edition, Robert Sedgewick.

Client, Implementation, Interface

Separate interface and implementation so as to:

- Build layers of abstraction.
- Reuse software.
- Ex: stack, queue, symbol table.

Interface: description of data type, basic operations.
Client: program using operations defined in interface.
Implementation: actual code implementing operations.

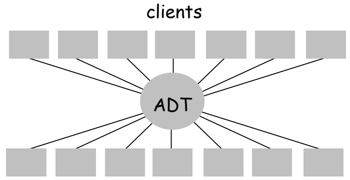
Benefits.

- Client can't know details of implementation, so has many implementation from which to choose.
- Implementation can't know details of client needs, so many clients can re-use the same implementation.
- Performance: use optimized implementation where it matters.
- Design: creates modular, re-usable libraries.

Abstract Data Types

Idealized scenario.

- Design general-purpose ADT useful for many clients.
- Develop efficient implementation of all ADT functions.
- Each ADT provides a new level of abstraction.



Total cost depends on:

- ADT implementation. ⬅ algorithms and data structures
- Client usage pattern. ⬅ might need different implementations for different clients

Priority Queues

Records with keys (priorities) that can be compared.

Basic operations.

- Insert. ⬅ PQ ops
- Remove largest.
- Create. ⬅ generic ADT ops
- Test if empty.
- Copy.
- Destroy.

⬆ not needed for one-time use, but critical in large systems when writing in C or C++

insert E ➡	E				
insert X ➡	E	X			
insert A ➡	E	X	A		
	E	A			remove largest ➡ X
insert M ➡	E	A	M		
	E	A			remove largest ➡ M
insert P ➡	E	A	P		
insert L ➡	E	A	P	L	
	E	A	L		remove largest ➡ P
insert E ➡	E	A	L	E	
	E	A	E		remove largest ➡ L
	A	E			remove largest ➡ E
	A				remove largest ➡ E
					remove largest ➡ A

Priority Queue Applications

Applications.

- Event-driven simulation. — customers in a line, colliding particles
- Numerical computation. — reducing roundoff error
- Data compression. — Huffman codes
- Graph searching. — shortest path, MST
- Computational number theory. — sum of powers
- Artificial intelligence. — A* search
- Statistics. — maintain largest M values in a sequence
- Operating systems. — task scheduling, interrupt handling
- Discrete optimization. — bin packing heuristics
- Spam filtering. — Bayesian spam filter

Priority Queue Client Example

Problem: Find the largest M of a stream of N elements.
Ex 1: Fraud detection - isolate $$ transactions.
Ex 2: File maintenance – find biggest files or directories.

Possible constraint: may not have enough memory to store N elements.
Solution: Use a priority queue.

Operation	time	space
sort	N lg N	N
elementary PQ	M N	M
binary heap	N lg M	M
best in theory	N	M

```
PQ pq = new PQ();

while(!StdIn.isEmpty()) {
   String s = StdIn.readString();
   pq.insert(s);
   if (pq.size() > M)
      pq.delMax();
}

while (!pq.isEmpty())
   System.out.println(pq.delMax());
```

Ex: top 10,000 in a stream of 1 billion.

- Not possible without good algorithm.

Unordered Array Priority Queue Implementation

```
public class PQ {
   private Comparable[] pq;    // pq[i] = ith element
   private int N;              // number of elements on PQ

   public PQ() { pq = new Comparable[8]; }                  constructor

   public boolean isEmpty() { return N == 0; }              is the PQ empty?

   public void insert(Comparable x) {
      pq[N++] = x;                                          insert element x into PQ
   }

   public Comparable delMax() {
      int max = 0;
      for (int i = 1; i < N; i++)                           remove and return max
         if (less(pq[max], pq[i])) max = i;                 element from PQ
      exch(pq, max, N-1);
      return pq[--N];
   }
}
```


Implementation Details

What if I don't know the max capacity of the PQ ahead of time?

- Double the size of the array as needed.
- Add following code to `insert` before updating array.

```
if (N >= pq.length) {
   Comparable[] temp = new Comparable[2*N];
   for (int i = 0; i < N; i++)
        temp[i] = pq[i];
   pq = temp;
}
```

Memory leak.

- Garbage collector only reclaims memory if there is no outstanding reference to it.
- When deleting element N-1 from the priority queue, set:

```
pq[N-1] = null;
```

Priority Queues Implementation Cost Summary

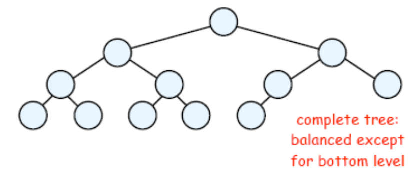
Worst-Case Asymptotic costs for PQ with N items			
Operation	Insert	Remove Max	Find Max
ordered array	N	1	1
ordered list	N	1	1
unordered array	1	N	N
unordered list	1	N	N

Can we implement all operations efficiently?

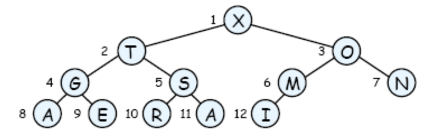
Heap

Heap: Array representation of a heap-ordered complete binary tree.

Binary tree.
- null or
- Node with links to left and right trees.



Heap-ordered binary tree.
- Keys in nodes.
- No smaller than children's keys.

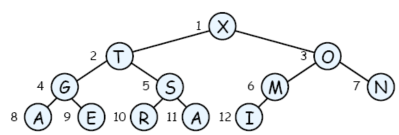


Array representation.
- Take nodes in level order.
- No explicit links needed since tree is complete.

1	2	3	4	5	6	7	8	9	10	11	12
X	T	O	G	S	M	N	A	E	R	A	I

Heap Properties

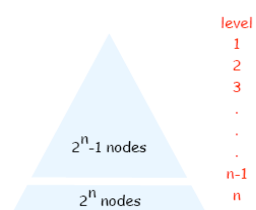
Largest key is at root.



Use array indices to move through tree.
- Note: indices start at 1.
- Parent of node at k is at k/2.
- Children of node at k are at 2k and 2k+1.

1	2	3	4	5	6	7	8	9	10	11	12
X	T	O	G	S	M	N	A	E	R	A	I

Length of path in N-node heap is at most ~ lg N.
- n levels when $2^n \le N < 2^{n+1}$.
- $n \le \lg N < n+1$.
- ~ lg N levels.

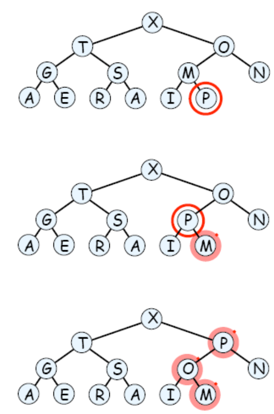


Promotion (Bubbling Up) In a Heap

Suppose that exactly one node is bigger than its parent.

To eliminate the violation:
- Exchange with its parent.
- Repeat until heap order restored.

```
private void swim(int k) {
   while (k > 1 && less(k/2, k)) {
      exch(k, k/2);
      k = k/2;
   }
}
```
parent of node at k is at k/2



Peter principle: node promoted to level of incompetence.

1	2	3	4	5	6	7	8	9	10	11	12	13
X	T	O	G	S	M	N	A	E	R	A	I	P
X	T	P	G	S	O	N	A	E	R	A	I	M

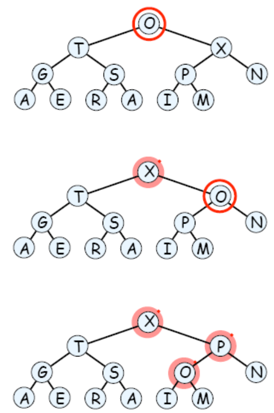
Demoting (Sifting Down) In a Heap

Suppose that exactly one node is smaller than a child.

To eliminate the violation:
- Exchange with larger child.
- Repeat until heap order restored.

```
private void sink(int k, int N) {
   while (2*k <= N) {
      int j = 2*k;
      if (j < N && less(j, j+1)) j++;
      if (!less(k, j)) break;
      exch(k, j);
      k = j;
   }
}
```

children of node at k are 2k and 2k+1



Power struggle: better subordinate promoted.

1	2	3	4	5	6	7	8	9	10	11	12	13
O	T	X	G	S	P	N	A	E	R	A	I	M
X	T	P	G	S	O	N	A	E	R	A	I	M

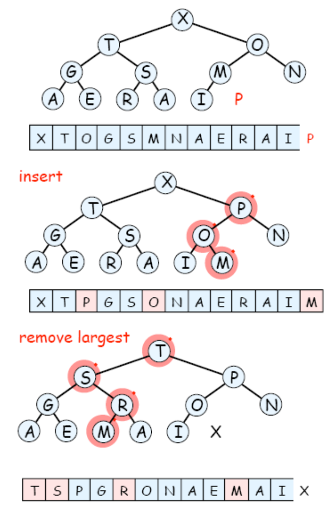
Insert and Delete Max

Insert. Add node at end, then promote.

```
public void insert(Comparable x) {
   pq[++N] = x;
   swim(N);
}
```

Remove largest. Exchange root with node at end, then sift down.

```
public Comparable delMax() {
   exch(1, N);
   sink(1, N-1);
   return pq[N--];
}
```



Heap Based Priority Queue in Java

```
public class PQ {
   private Comparable[] pq;              ← exactly as in array-based PQ
   private int N;

   public PQ()               { }         ← same as array-based PQ, but
   public boolean isEmpty()  { }           allocate one extra element in array
   public int size()         { }

   public void insert(Comparable x)  { }     ← PQ ops
   public Comparable delMax()        { }

   private void swim(int k)          { }     ← heap helper functions
   private void sink(int k, int N)   { }

   private boolean less(int i, int j) { }    ← helper functions
   private void    exch(int i, int j) { }
}
```

Expansion: double size of array as needed.
Memory leak: when deleting element `N`, set `pq[N] = null`.

Priority Queues Implementation Cost Summary

Worst-Case Asymptotic costs for PQ with N items			
Operation	Insert	Remove Max	Find Max
ordered array	N	1	1
ordered list	N	1	1
unordered array	1	N	N
unordered list	1	N	N
heap	lg N	lg N	1

Hopeless challenge: get all ops O(1).

Digression: Heapsort

First pass: build heap.
- Add item to heap at each iteration, then sift up.
- Or can use faster bottom-up method; see book.

```
for (int k = N / 2; k >= 1; k--) {
   sink(k, N);
```

Second pass: sort.
- Remove maximum at each iteration.
- Exchange root with node at end, then sift down.

```
while (N > 1) {
   exch(1, N);
   sink(1, --N);
}
```

in the heap

not in the heap

E	X	A	M	P	L	E
E	X	A	M	P	L	E
X	E	A	M	P	L	E
X	E	A	M	P	L	E
X	M	A	E	P	L	E
X	P	A	E	M	L	E
X	P	L	E	M	A	E
X	P	L	E	M	A	E
P	M	L	E	E	A	X
M	E	L	A	E	P	X
L	E	E	A	M	P	X
E	A	E	L	M	P	X
E	A	E	L	M	P	X
A	E	E	L	M	P	X
A	E	E	L	M	P	X

Significance of Heapsort

Q: Sort in N log N worst-case without using extra memory?
A: Yes. Heapsort.

Not mergesort? Linear extra space. ← challenge for bored: design in-place merge
Not quicksort? Quadratic in worst case. ← challenge for bored: design O(N log N) worst-case quicksort

Heapsort is OPTIMAL for both time and space, BUT
- Inner loop longer than quicksort's.
- Makes poor use of cache memory.

In the wild: g++ STL uses introsort.

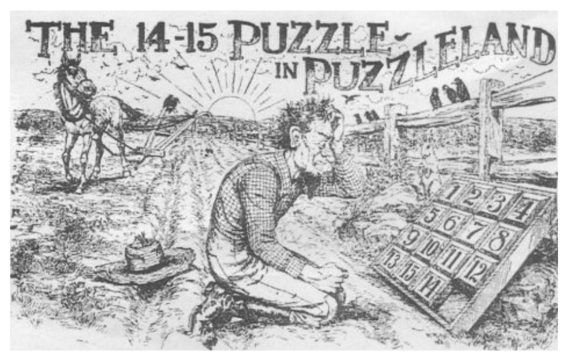
combo of quicksort, heapsort, and insertion

Sorting Summary

			# Key Comparisons			
	In-Place	Stable	Worst	Average	Best	Remarks
Bubble Sort	X	X	$N^2 / 2$	$N^2 / 2$	N	never use it
Selection Sort	X		$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
Insertion Sort	X	X	$N^2 / 2$	$N^2 / 4$	N	use as cutoff for small N
Shellsort	X		$N^{3/2}$	$N^{3/2}$	$N^{3/2}$	with Knuth sequence
Quicksort	X		$N^2 / 2$	2N ln N	N lg N	fastest in practice
Mergesort		X	N lg N	N lg N	N lg N	N log N guarantee, stable
Heapsort	X		2 N lg N	2 N lg N	N lg N	N log N guarantee, in-place

Sam Loyd's 15-Slider Puzzle

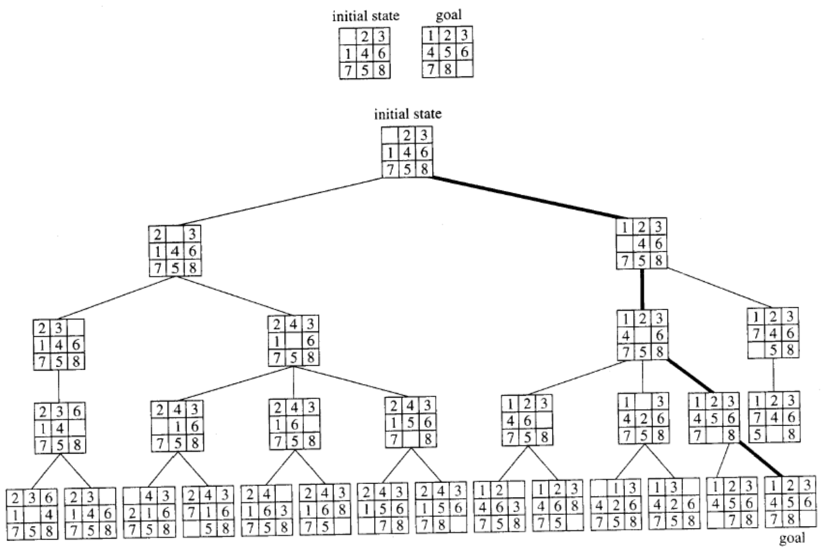
15 puzzle.
- Legal move: slide neighboring tile into blank square.
- Challenge: sequence of legal moves to put tiles in increasing order.
- Win $1000 prize for solution.



Sam Loyd

http://www.javaonthebrain.com/java/puzz15/

Breadth First Search of 8-Puzzle Game Tree

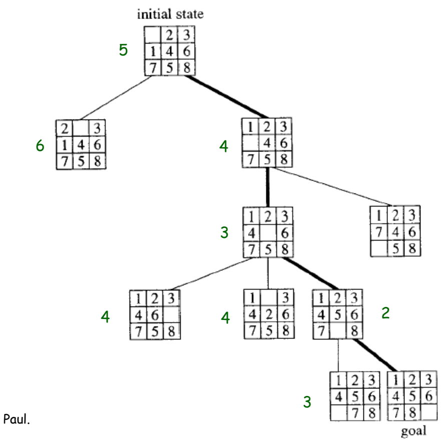


A* Search of 8-Puzzle Game Tree

Priority first search.

- Basic idea: explore positions in more intelligent order.
- Ex 1: number of tiles out of order.
- Ex 2: sum of Manhattan distances + depth.

Implement A* algorithm with PQ.



Pictures from *Sequential and Parallel Algorithms* by Berman and Paul.

Event-Based Simulation

Challenge: animate N moving particles.

- Each has given velocity vector.
- Bounce off edges and one another upon collision.

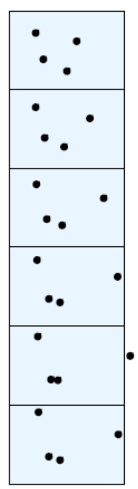
Example applications: molecular dynamics, traffic, ...

Naive approach: t times per second

- Update particle positions.
- Check for collisions, update velocities.
- Redraw all particles.

Problems:

- N^2t collision checks per second.
- May miss collisions!



Event-Based Simulation

Approach: use PQ of events with time as key.

- Put collision event on PQ for each particle (calculate time of next collision as priority)
- Put redraw events on PQ (t per second).

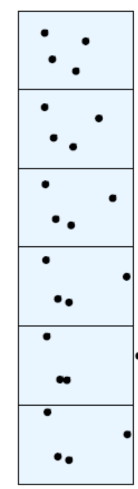
Main loop: remove next event from PQ.

- Redraw: update positions and redraw.
- Collision: update velocity of affected particles and put new collision events on PQ.

More PQ operations needed:

- may need to remove items from PQ .
- may want to join PQs for different sets of events (Ex: joi national for air traffic control).

More sophisticated PQ interface needed



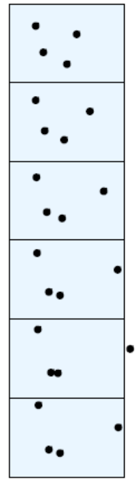
More Priority Queue Operations

Indirect priority queue.

- Supports deletion of arbitrary elements.
- Use symbol table to access binary heap node, given element to delete.

Binomial queue.

- Supports fast join.
- Slightly relaxes heap property to gain flexibility.



Priority Queues Implementation Cost Summary

	Worst-Case Asymptotic costs for PQ with N items				
Operation	Insert	Remove Max	Find Max	Change Key	Join
ordered array	N	1	1	N	N
ordered list	N	1	1	N	N
unordered array	1	N	N	1	N
unordered list	1	N	N	1	1
heap	lg N	lg N	1	lg N	N
binomial queue	lg N	lg N	lg N	lg N	lg N
best in theory	1	lg N	1	1	1