Priority Queues

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

Priority Queues

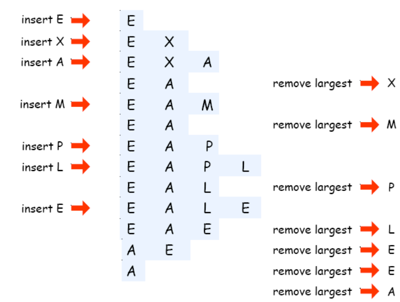
Data. Items that can be compared.

Basic operations.
- Insert.
- Remove largest.

} defining ops

- Copy.
- Create.
- Destroy.
- Test if empty.

} generic ops



Priority Queue Applications

Applications.
- Event-driven simulation. [customers in a line, colliding particles]
- Numerical computation. [reducing roundoff error]
- Data compression. [Huffman codes]
- Graph searching. [Dijkstra's algorithm, Prim's algorithm]
- Computational number theory. [sum of powers]
- Artificial intelligence. [A* search]
- Statistics. [maintain largest M values in a sequence]
- Operating systems. [load balancing, interrupt handling]
- Discrete optimization. [bin packing, scheduling]
- Spam filtering. [Bayesian spam filter]

Generalizes: stack, queue, randomized queue.

Priority Queue Client Example

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

Constraint. Not 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

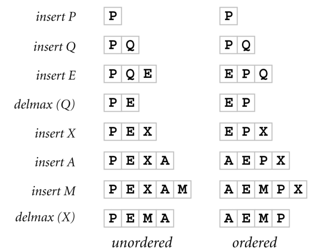
```
MinPQ<String> pq = new MinPQ<String>();

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

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

Priority Queue: Elementary Implementations

Two elementary implementations.

Implementation	Insert	Del Max
unordered array	1	N
ordered array	N	1

worst-case asymptotic costs for PQ with N items



Challenge. Implement both operations efficiently.

Priority Queue: Unordered Array Implementation

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

   public UnorderedPQ(int maxN) {
      pq = (Item[]) new Comparable[maxN];
   }

   public boolean isEmpty() { return N == 0; }

   public void insert(Item x) { pq[N++] = x; }

   public Item delMax() {
      int max = 0;
      for (int i = 1; i < N; i++)
         if (less(max, i)) max = i;
      exch(max, N-1);
      return pq[--N];
   }
}
```

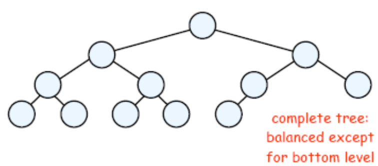
no generic array creation

Binary Heap

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

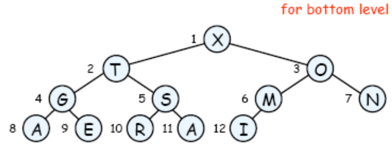
Binary tree.

- Empty or
- Node with links to left and right trees.



Heap-ordered binary tree.

- Keys in nodes.
- No smaller than children's keys.



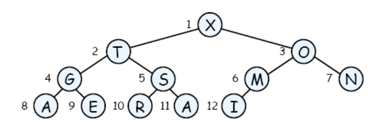
Array representation.

- 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

Binary Heap Properties

Property A. Largest key is at root.



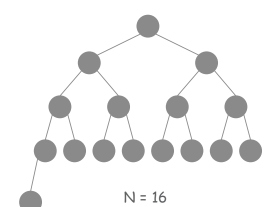
Property B. Can use array indices to move through tree.

- 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

Property C. Height of N node heap is $1 + \lfloor \lg N \rfloor$.

height only increases when N is a power of 2



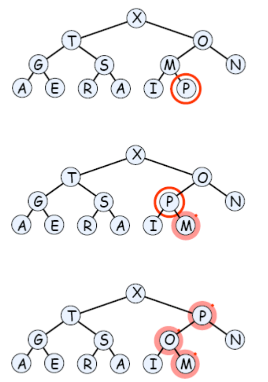
Promotion In a Heap

Scenario. 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



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

Peter principle: node promoted to level of incompetence.

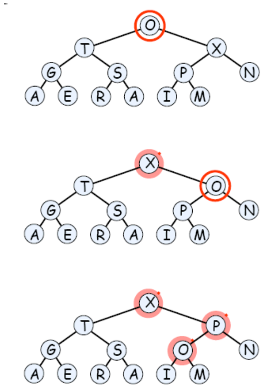
Demotion In a Heap

Scenario. 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) {
   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



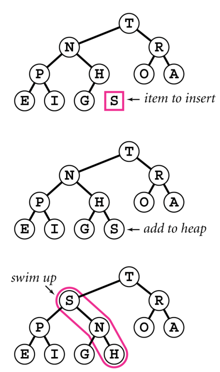
1	2	3	4	5	6	7	8	9	10	11	12	13
O	T	X	G	S	P	N	A	E	R	A	I	M
X	T	P	G	S	O	N	A	E	R	A	I	M

Power struggle: better subordinate promoted.

Insert

Insert. Add node at end, then promote.

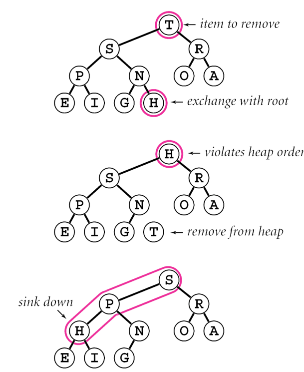
```
public void insert(Item x) {
   pq[++N] = x;
   swim(N);
}
```



Remove the Maximum

Remove max. Exchange root with node at end, then demote.

```
public Item delMax() {
   Item max = pq[1];
   exch(1, N--);
   sink(1);
   pq[N+1] = null;
   return max;
}
```



Binary Heap: Skeleton

```
public class MaxPQ<Item extends Comparable> {
   private Item[] pq;
   private int N;

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

   public void insert(Item x) { }          PQ ops
   public Item delMax()       { }

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

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


Binary Heap Considerations

Minimum oriented priority queue. Replace `less()` with `greater()` and implement `greater()`.

Array resizing. Add no-arg constructor, and apply repeated doubling.

O(log N) amortized time per op

Immutability of keys. We assume client does not change keys while they're on the PQ. Best practice: make keys immutable.

Other operations.

- Remove an arbitrary item.
- Change the priority of an item.
- Can implement using `sink()` and `swim()` abstractions, but we defer.

Priority Queues Implementation Cost Summary

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
binary heap	lg N	lg N	1

worst-case asymptotic costs for PQ with N items

Hopeless challenge. Make all ops O(1). Why hopeless?

Digression: Heapsort

First pass: build heap.

- Insert items into heap, one at at time.
- Or can use faster bottom-up method; see book.

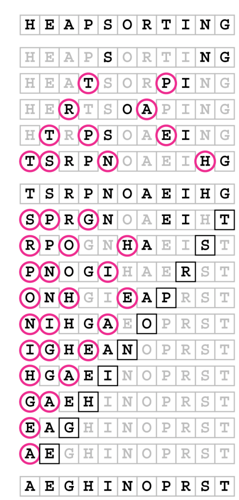
```
for (int k = N/2; k >= 1; k--)
   sink(a, k, N);
```

Second pass: sort.

- Remove maximum items, one at a time.
- Leave in array, instead of nulling out.

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

Property D. At most 2 N lg N comparisons.



Significance of Heapsort

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

Not mergesort? Linear extra space. — challenge for bored: in-place merge
Not quicksort? Quadratic time in worst case. — challenge for bored: 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.

Introsort

Introsort. [David Musser 1997] Run quicksort, but switch over to heapsort if things are not going well.

Q1. How would you define "not going well" ?
Q2. How would you detect it ?

In the wild: g++ STL uses introsort.

combo of quicksort, heapsort, and insertion

Sorting Summary

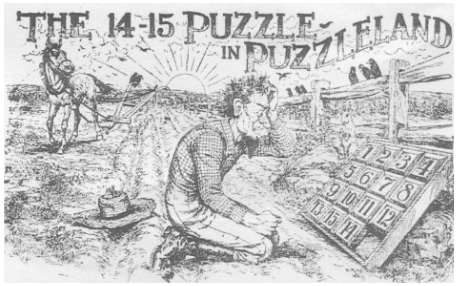
	In-Place	Stable	Worst	Average	Best	Remarks
Bubble sort	X	X	N^2 / 2	N^2 / 2	N	never use it
Selection sort	X		N^2 / 2	N^2 / 2	N^2 / 2	N exchanges
Insertion sort	X	X	N^2 / 2	N^2 / 4	N	use as cutoff for small N
Shellsort	X		$N^{3/2}$	$N^{3/2}$	$N^{3/2}$	with Knuth sequence
Quicksort	X		N^2 / 2	2N ln N	N 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

Key Comparisons

A* Algorithm

Sam Loyd's 15-Slider Puzzle

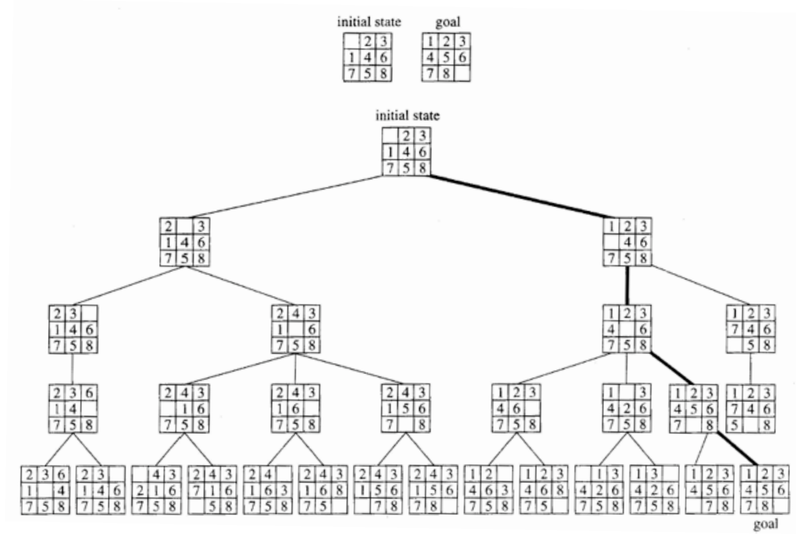
15 puzzle.
- Legal move: slide neighboring tile into blank square.
- Challenge: sequence of legal moves to put tiles in increasing order.
- Win $1,000 prize for solution.



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

Sam Loyd

Breadth First Search of 8-Puzzle Game Tree

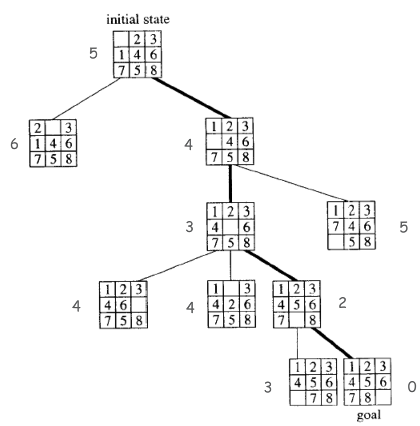


Pictures from *Sequential and Parallel Algorithms* by Berman and Paul.

A* Search of 8-Puzzle Game Tree

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

Implement A* algorithm with PQ.



Slider Puzzle: Unsolvable Instances

Unsolvable instances.

1	2	3
4	5	6
8	7	

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

8-slider invariant. Parity of number of pairs of pieces in reverse order.

3	1	2
4	5	6
8	7	

1-3, 2-3, 7-8
odd

3	1	2
4	5	6
8		7

1-3, 2-3, 7-8
odd

3	1	2
4		6
8	5	7

1-3, 2-3, 7-8, 5-8, 5-6
odd

Event-Driven Simulation

Molecular Dynamics Simulation of Hard Spheres

Goal. Simulate the motion of N moving particles that behave according to the laws of elastic collision.

Hard sphere model.

- Moving particles interact via elastic collisions with each other, and with fixed walls.
- Each particle is a sphere with known position, velocity, mass, and radius.
- No other forces are exerted.

temperature, pressure, diffusion constant

motion of individual atoms and molecules

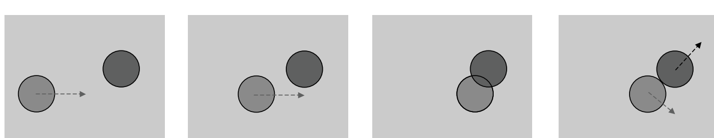
Significance. Relates macroscopic observables to microscopic dynamics.

- Maxwell and Boltzmann: derive distribution of speeds of interacting molecules as a function of temperature.
- Einstein: explain Brownian motion of pollen grains.

Time-Driven Simulation

Time-driven simulation.

- Discretize time in quanta of size dt.
- Update the position of each particle after every dt units of time, and check for overlaps.
- If overlap, roll back the clock to the time of the collision, update the velocities of the colliding particles, and continue the simulation.

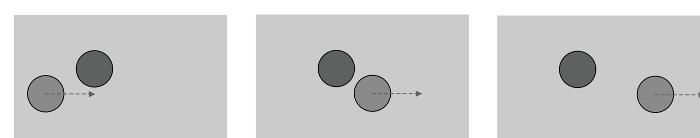


t | t + dt | t + 2 dt (collision detected) | t + Δt (roll back clock)

Time-Driven Simulation

Main drawbacks.

- N^2 overlap checks per time quantum.
- May miss collisions if dt is too large and colliding particles fail to overlap when we are looking.
- Simulation is too slow if dt is very small.



t | t + dt | t + 2 dt

Event-Driven Simulation

Event-driven simulation.

- Between collisions, particles move in straight-line trajectories.
- Focus only on times when collisions occur.
- Maintain priority queue of collision events, prioritized by time.
- Remove the minimum = get next collision.

Collision prediction. Given position, velocity, and radius of a particle, when will it collide next with a wall or another particle?

Collision resolution. If collision occurs, update colliding particle(s) according to laws of elastic collisions.

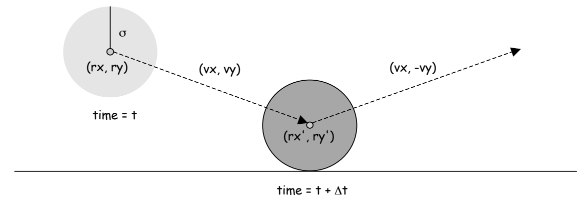
Particle-Wall Collision

Collision prediction.

- Particle of radius σ at position (rx, ry), moving with velocity (vx, vy).
- Will it collide with a horizontal wall? If so, when?

$$\Delta t = \begin{cases} \infty & \text{if } vy = 0 \\ (\sigma - ry)/vy & \text{if } vy < 0 \\ (1 - \sigma - ry)/vy & \text{if } vy > 0 \end{cases}$$

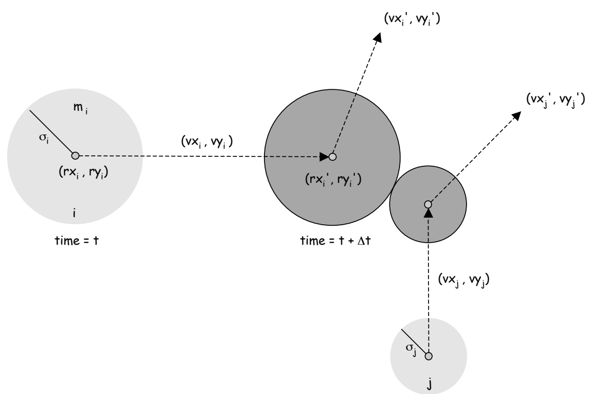
Collision resolution. (vx', vy') = (vx, -vy).



Particle-Particle Collision Prediction

Collision prediction.

- Particle i: radius σ_i, position (rx_i, ry_i), velocity (vx_i, vy_i).
- Particle j: radius σ_j, position (rx_j, ry_j), velocity (vx_j, vy_j).
- Will particles i and j collide? If so, when?



Particle-Particle Collision Prediction

Collision prediction.

- Particle i: radius σ_i, position (rx_i, ry_i), velocity (vx_i, vy_i).
- Particle j: radius σ_j, position (rx_j, ry_j), velocity (vx_j, vy_j).
- Will particles i and j collide? If so, when?

$$\Delta t = \begin{cases} \infty & \text{if } \Delta v \cdot \Delta r \ge 0 \\ \infty & \text{if } d < 0 \\ -\dfrac{\Delta v \cdot \Delta r + \sqrt{d}}{\Delta v \cdot \Delta v} & \text{otherwise} \end{cases}$$

$$d = (\Delta v \cdot \Delta r)^2 - (\Delta v \cdot \Delta v)\,(\Delta r \cdot \Delta r - \sigma^2) \qquad \sigma = \sigma_i + \sigma_j$$

$$\Delta v = (\Delta vx,\ \Delta vy) = (vx_i - vx_j,\ vy_i - vy_j)$$
$$\Delta r = (\Delta rx,\ \Delta ry) = (rx_i - rx_j,\ ry_i - ry_j)$$

$$\Delta v \cdot \Delta v = (\Delta vx)^2 + (\Delta vy)^2$$
$$\Delta r \cdot \Delta r = (\Delta rx)^2 + (\Delta ry)^2$$
$$\Delta v \cdot \Delta r = (\Delta vx)(\Delta rx) + (\Delta vy)(\Delta ry)$$

Particle-Particle Collision Prediction

Collision resolution. When two particles collide, how does velocity change?

$$
\begin{aligned}
vx_i' &= vx_i + Jx / m_i \\
vy_i' &= vy_i + Jy / m_i \\
vx_j' &= vx_j - Jx / m_j \\
vy_j' &= vx_j - Jy / m_j
\end{aligned}
$$

Newton's second law (momentum form)

$$Jx = \frac{J\,\Delta rx}{\sigma}, \quad Jy = \frac{J\,\Delta ry}{\sigma}, \quad J = \frac{2\,m_i\,m_j\,(\Delta v \cdot \Delta r)}{\sigma(m_i + m_j)}$$

impulse due to normal force
(conservation of energy, conservation of momentum)

Event-Driven Simulation

Initialization. Fill PQ with all potential particle-wall and particle-particle collisions.

potential since collision may not happen if some other collision intervenes

Main loop.

- Delete the impending event from PQ (min priority = t).
- If the event in no longer valid, ignore it.
- Advance all particles to time t, on a straight-line trajectory.
- Update the velocities of the colliding particle(s).
- Predict future particle-wall and particle-particle collisions involving the colliding particle(s) and insert events onto PQ.