Some things you didn't know about priority queues

Mike Atkinson

CS Seminar, Carleton University, 4 October 2012
Joint work with Michael Albert

Outline of talk

1. Two Things
2. Things you did know
3. Things you may not know
4. Recent Things

Thing One: hashing

- We can model hashing by thinking of parking n cars on a 1-way street with n parking spaces
- The cars arrive in order
- Each car has a preferred space it tries first
- If that space is full it tries the next available spaces one by one, parking at the first free space
- If it reaches the end of the street having failed to park we have *failure*
- If all cars can park we have *success*

Thing One: hashing

- We can model hashing by thinking of parking n cars on a 1-way street with n parking spaces
- The cars arrive in order
- Each car has a preferred space it tries first
- If that space is full it tries the next available spaces one by one, parking at the first free space
- If it reaches the end of the street having failed to park we have *failure*
- If all cars can park we have *success*

Thing One: hashing

- We can model hashing by thinking of parking n cars on a 1-way street with n parking spaces
- The cars arrive in order
- Each car has a preferred space it tries first
- If that space is full it tries the next available spaces one by one, parking at the first free space
- If it reaches the end of the street having failed to park we have *failure*
- If all cars can park we have *success*

Example

Suppose $n = 5$ and the preferences are

Car	1	2	3	4	5
Preferred space	3	4	1	4	3

Thing One: hashing

- We can model hashing by thinking of parking n cars on a 1-way street with n parking spaces
- The cars arrive in order
- Each car has a preferred space it tries first
- If that space is full it tries the next available spaces one by one, parking at the first free space
- If it reaches the end of the street having failed to park we have *failure*
- If all cars can park we have *success*

Example

Suppose $n = 5$ and the preferences are

Car	1	2	3	4	5
Preferred space	3	1	1	3	2

3 cars

Preference functions

111	112	113	121	122	123	131	132	133
211	212	213	221	222	223	231	232	233
311	312	313	321	322	323	331	332	333

3 cars

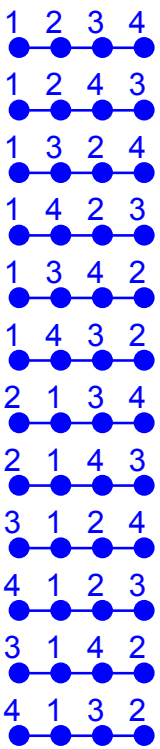
Successful functions

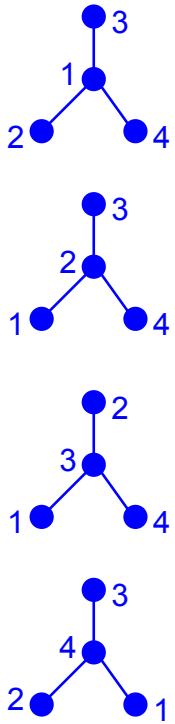
111	112	113	121	122	123	131	132	133
211	212	213	221	222	223	231	232	233
311	312	313	321	322	323	331	332	333

- With n cars there are n^n preference functions
- But only $(n+1)^{n-1}$ successful ones (parking functions)

- With n cars there are n^n preference functions
- But only $(n+1)^{n-1}$ successful ones (parking functions)

Thing two: labeled (unrooted) (unordered) trees

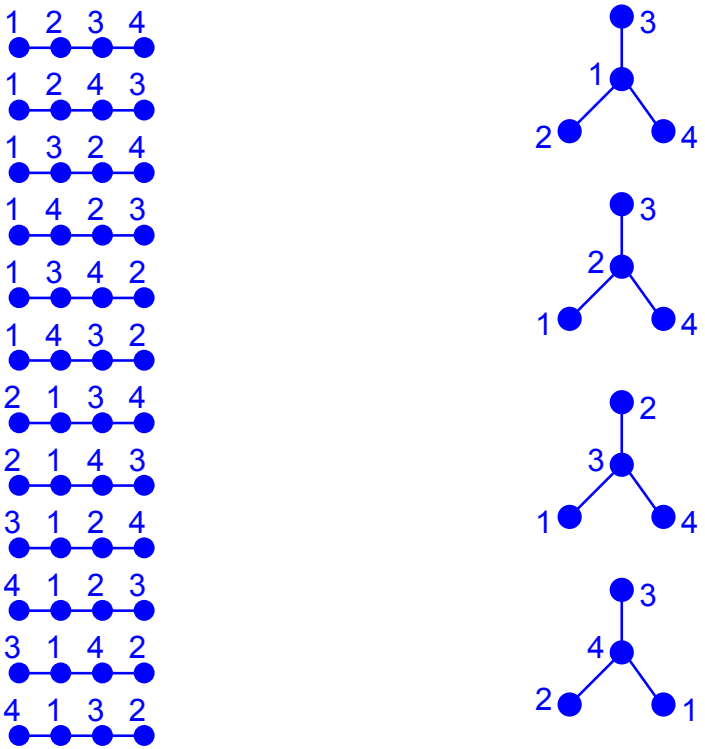




All the trees on 4 labeled nodes

There are $(n+1)^{n-1}$ labeled trees on $n+1$ nodes.

Thing two: labeled (unrooted) (unordered) trees



All the trees on 4 labeled nodes

There are $(n+1)^{n-1}$ labeled trees on $n+1$ nodes.

What is a priority queue?

A priority queue is a container which can contain priorities (or data items which have a priority). There are two main operations that can be applied to the container:

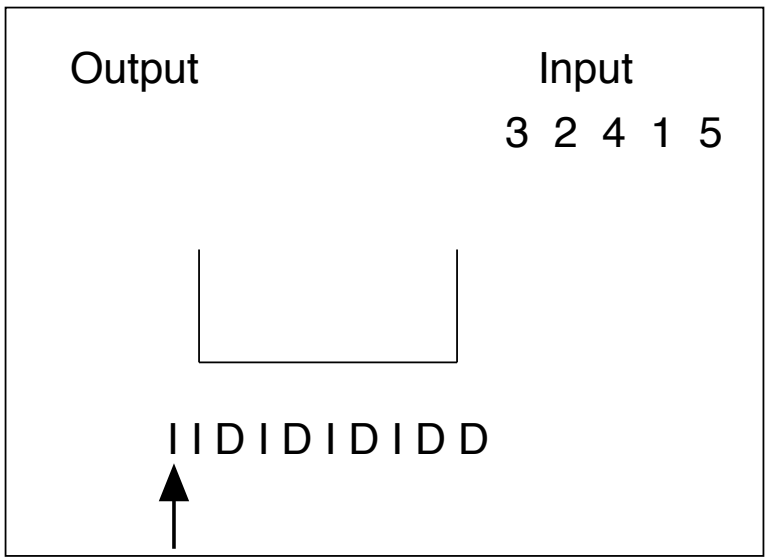
Insert	Insert a new item into the container
Delete-Min	Delete the item of smallest priority from the container

. . . and you also knew

- Several implementations: heap, binary search tree
- Insert and Delete-Min operations in log n time
- Applications to scheduling and sorting - particularly Heapsort

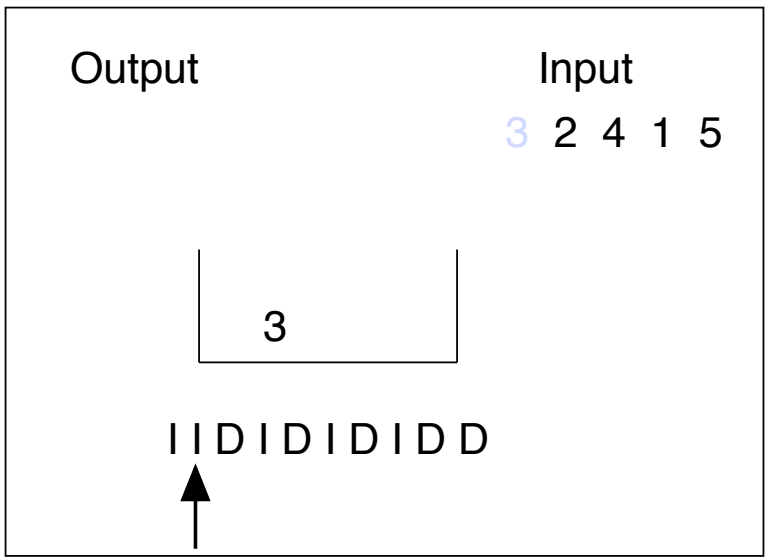
Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



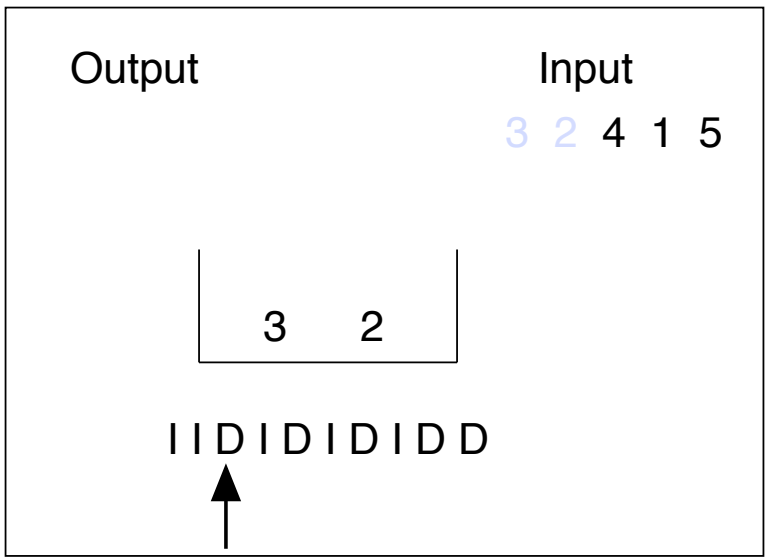
Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



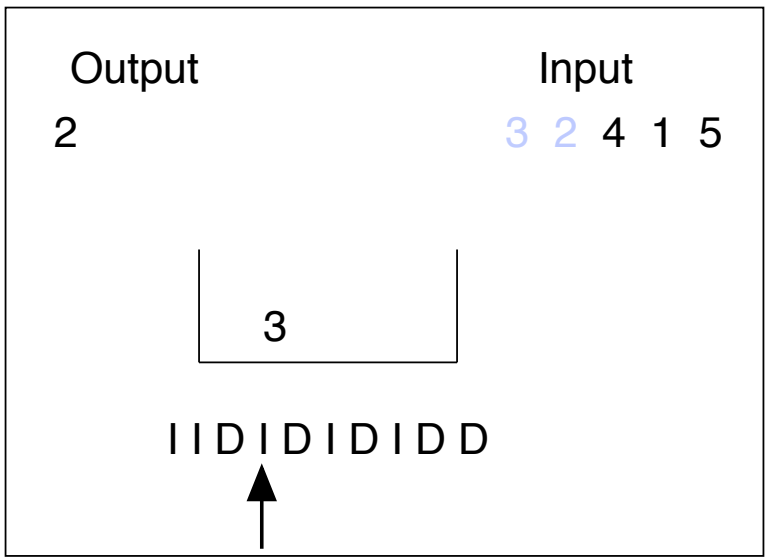
Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



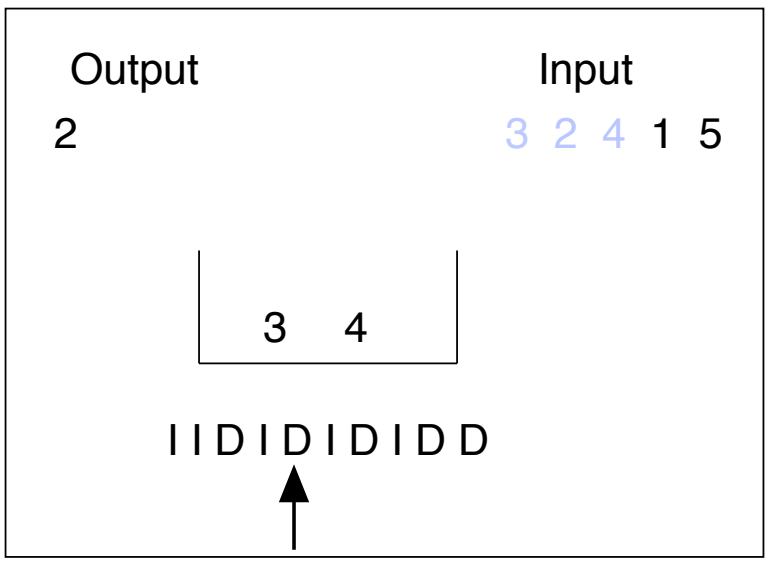
Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



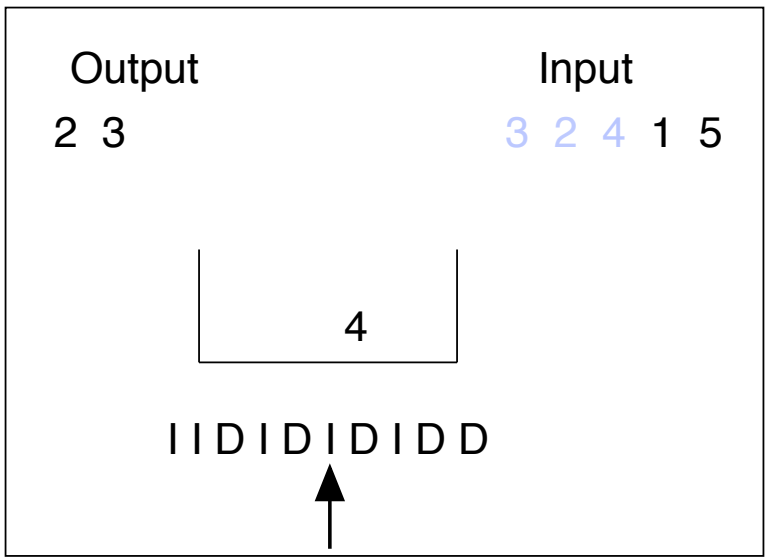
Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



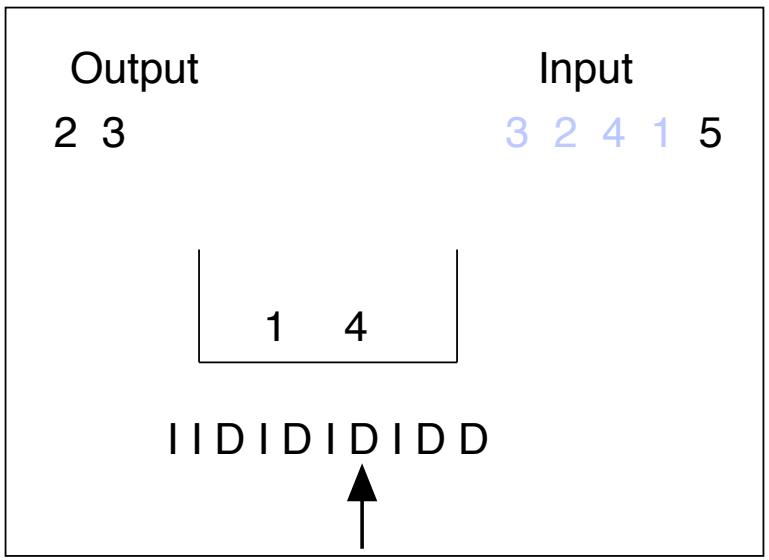
Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



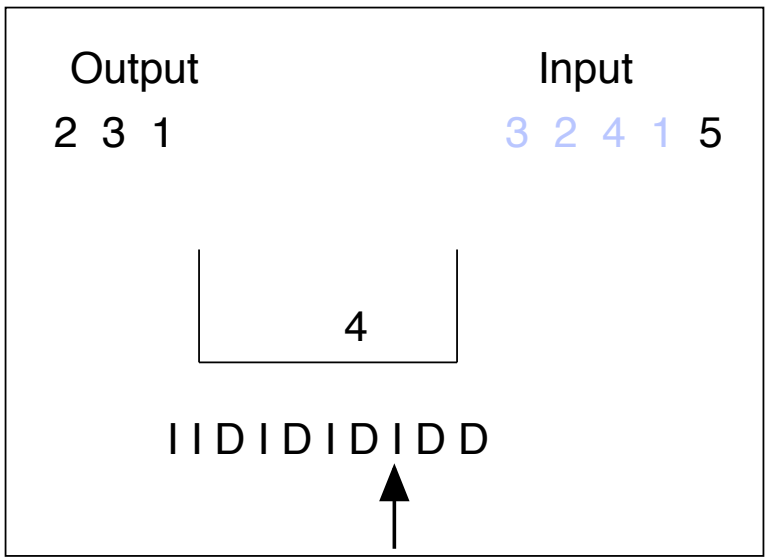
Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



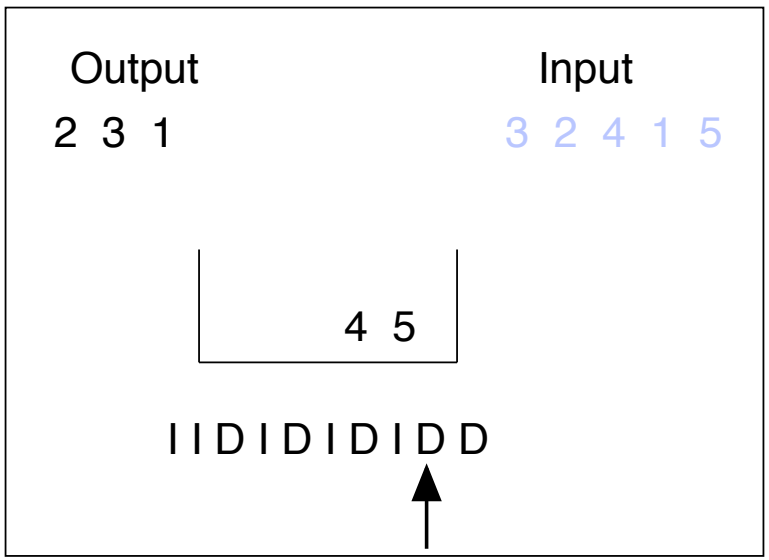
Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



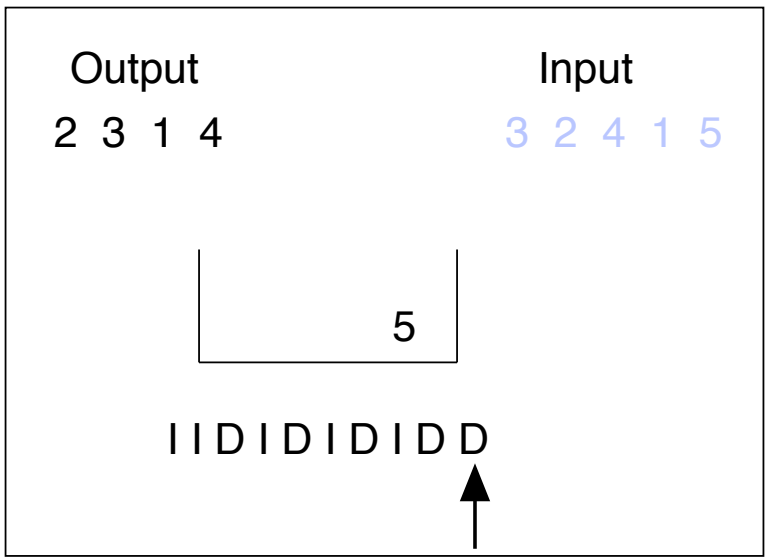
Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



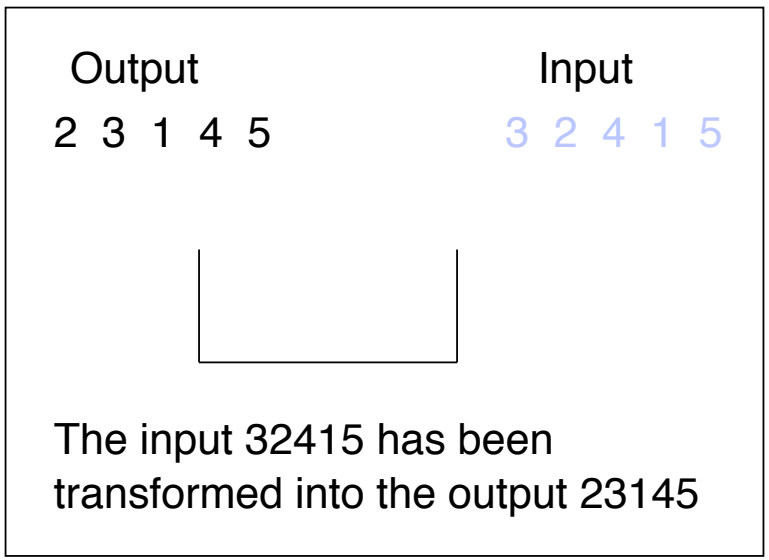
Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



Priority queue computations

A priority queue computation is a sequence of Insert (I) and Delete-Min (D) operations that begins and ends with the priority queue in the empty state such as $IIDIIDIDDD$.
It takes a sequence of input items and produces a sequence of output items



Allowable pairs

- We have just seen a priority queue computation that transformed the input sequence 32415 into the output sequence 23145.
- $(32415, 23145)$ is an *allowable pair*
- Not every pair of permutations is an allowable pair
  - $(12, 21)$ is not allowable
  - $(321, 132)$ is not allowable

Serendipity

I was sitting at my desk at Carleton in February 1992 ....

The number of allowable pairs

Theorem

There are $(n+1)^{n-1}$ allowable pairs of length n.

There are direct correspondences with parking functions and trees

Lots of related results

- Algorithm for computing the number of outputs that a given input of length n can produce: $O(n^4)$
- Algorithm for computing the number of inputs that can produce a given output of length n: $O(n)$
- What happens if we place a bound on the capacity of the priority queue?
- What happens if we allow the priorities to have duplicates?
- What happens if we chain the output of a priority queue so that it is the input to a second priority queue?
- What if we allow any number of priority queues in series?

Lots of related results

- Algorithm for computing the number of outputs that a given input of length n can produce: $O(n^4)$
- Algorithm for computing the number of inputs that can produce a given output of length n: $O(n)$
- What happens if we place a bound on the capacity of the priority queue?
- What happens if we allow the priorities to have duplicates?
- What happens if we chain the output of a priority queue so that it is the input to a second priority queue?
- What if we allow any number of priority queues in series?

Lots of related results

- Algorithm for computing the number of outputs that a given input of length n can produce: $O(n^4)$
- Algorithm for computing the number of inputs that can produce a given output of length n: $O(n)$
- What happens if we place a bound on the capacity of the priority queue?
- What happens if we allow the priorities to have duplicates?
- What happens if we chain the output of a priority queue so that it is the input to a second priority queue?
- What if we allow any number of priority queues in series?

Lots of related results

- Algorithm for computing the number of outputs that a given input of length n can produce: $O(n^4)$
- Algorithm for computing the number of inputs that can produce a given output of length n: $O(n)$
- What happens if we place a bound on the capacity of the priority queue?
- What happens if we allow the priorities to have duplicates?
- What happens if we chain the output of a priority queue so that it is the input to a second priority queue?
- What if we allow any number of priority queues in series?

Lots of related results

- Algorithm for computing the number of outputs that a given input of length n can produce: $O(n^4)$
- Algorithm for computing the number of inputs that can produce a given output of length n: $O(n)$
- What happens if we place a bound on the capacity of the priority queue?
- What happens if we allow the priorities to have duplicates?
- What happens if we chain the output of a priority queue so that it is the input to a second priority queue?
- What if we allow any number of priority queues in series?

Lots of related results

- Algorithm for computing the number of outputs that a given input of length n can produce: $O(n^4)$
- Algorithm for computing the number of inputs that can produce a given output of length n: $O(n)$
- What happens if we place a bound on the capacity of the priority queue?
- What happens if we allow the priorities to have duplicates?
- What happens if we chain the output of a priority queue so that it is the input to a second priority queue?
- What if we allow any number of priority queues in series?

Closure: new allowable pairs from old

- We know that $(32415, 23145)$ is allowable
- Fix on a subset of the priorities e.g. $\{1, 3, 4\}$
- So $(32415, 23145)$
- So $(341, 314)$ is also allowable
- i.e. $(231, 213)$ is allowable

This "downward closure" property suggests a connection with Permutation Pattern Classes

Closure: new allowable pairs from old

- We know that $(32415, 23145)$ is allowable
- Fix on a subset of the priorities e.g. $\{1, 3, 4\}$
- So $(32415, 23145)$
- So $(341, 314)$ is also allowable
- i.e. $(231, 213)$ is allowable

This "downward closure" property suggests a connection with Permutation Pattern Classes

Closure: new allowable pairs from old

- We know that $(32415, 23145)$ is allowable
- Fix on a subset of the priorities e.g. $\{1, 3, 4\}$
- So $(32415, 23145)$
- So $(341, 314)$ is also allowable
- i.e. $(231, 213)$ is allowable

This "downward closure" property suggests a connection with Permutation Pattern Classes

Closure: new allowable pairs from old

- We know that $(32415, 23145)$ is allowable
- Fix on a subset of the priorities e.g. $\{1, 3, 4\}$
- So $(32415, 23145)$
- So $(341, 314)$ is also allowable
- i.e. $(231, 213)$ is allowable

This "downward closure" property suggests a connection with Permutation Pattern Classes

Closure: new allowable pairs from old

- We know that $(32415, 23145)$ is allowable
- Fix on a subset of the priorities e.g. $\{1, 3, 4\}$
- So $(32415, 23145)$
- So $(341, 314)$ is also allowable
- i.e. $(231, 213)$ is allowable

This "downward closure" property suggests a connection with Permutation Pattern Classes

Closure: new allowable pairs from old

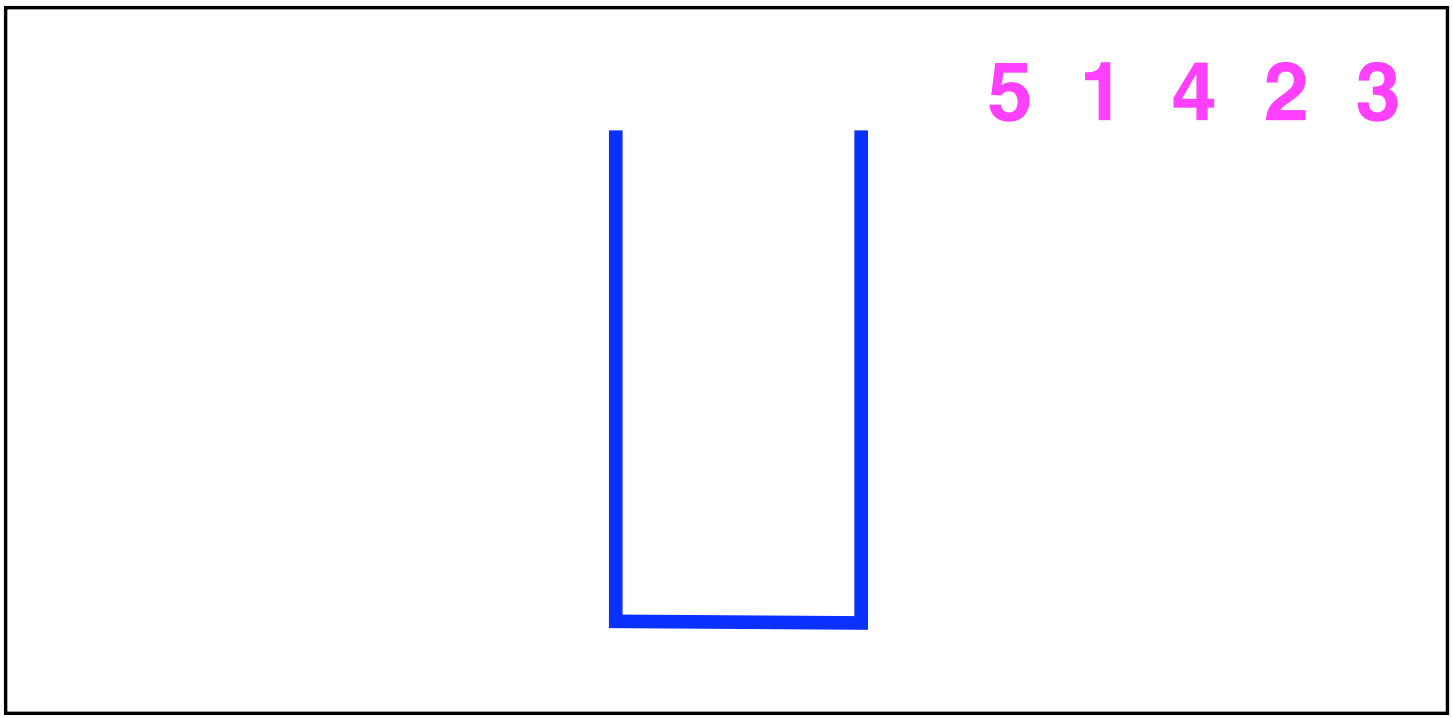
- We know that $(32415, 23145)$ is allowable
- Fix on a subset of the priorities e.g. $\{1, 3, 4\}$
- So $(32415, 23145)$
- So $(341, 314)$ is also allowable
- i.e. $(231, 213)$ is allowable

This "downward closure" property suggests a connection with Permutation Pattern Classes

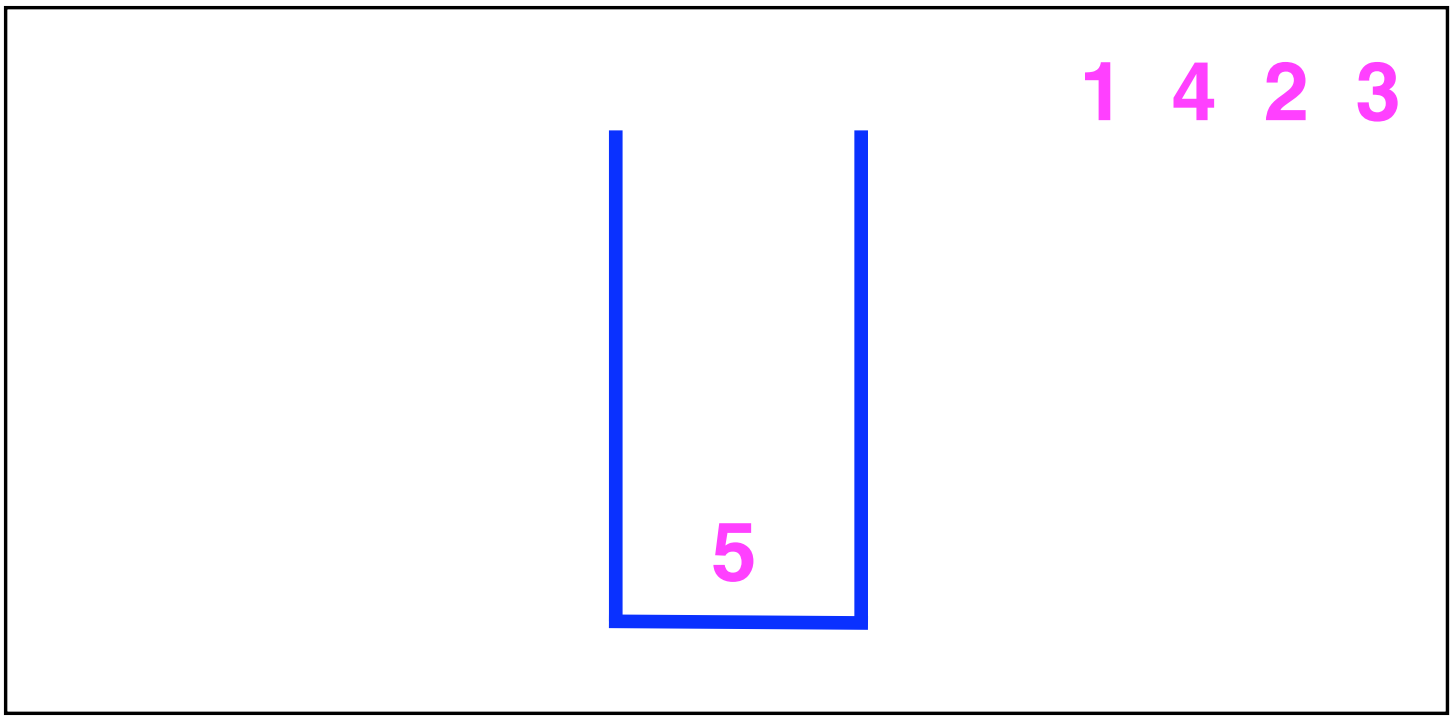
Pattern classes

- A pattern class $\mathcal{X}$ of permutations is a set of permutations with a downward closure property like the one we just saw (but single permutations rather than pairs).
- Eg. if $426315 \in \mathcal{X}$ then (considering the red elements 461) also $231 \in \mathcal{X}$. (We say 426315 *contains* 231).
- Every pattern class is defined by a set of avoided (= not contained) permutations.
- e.g. The set of permutations that a stack can sort is a pattern class

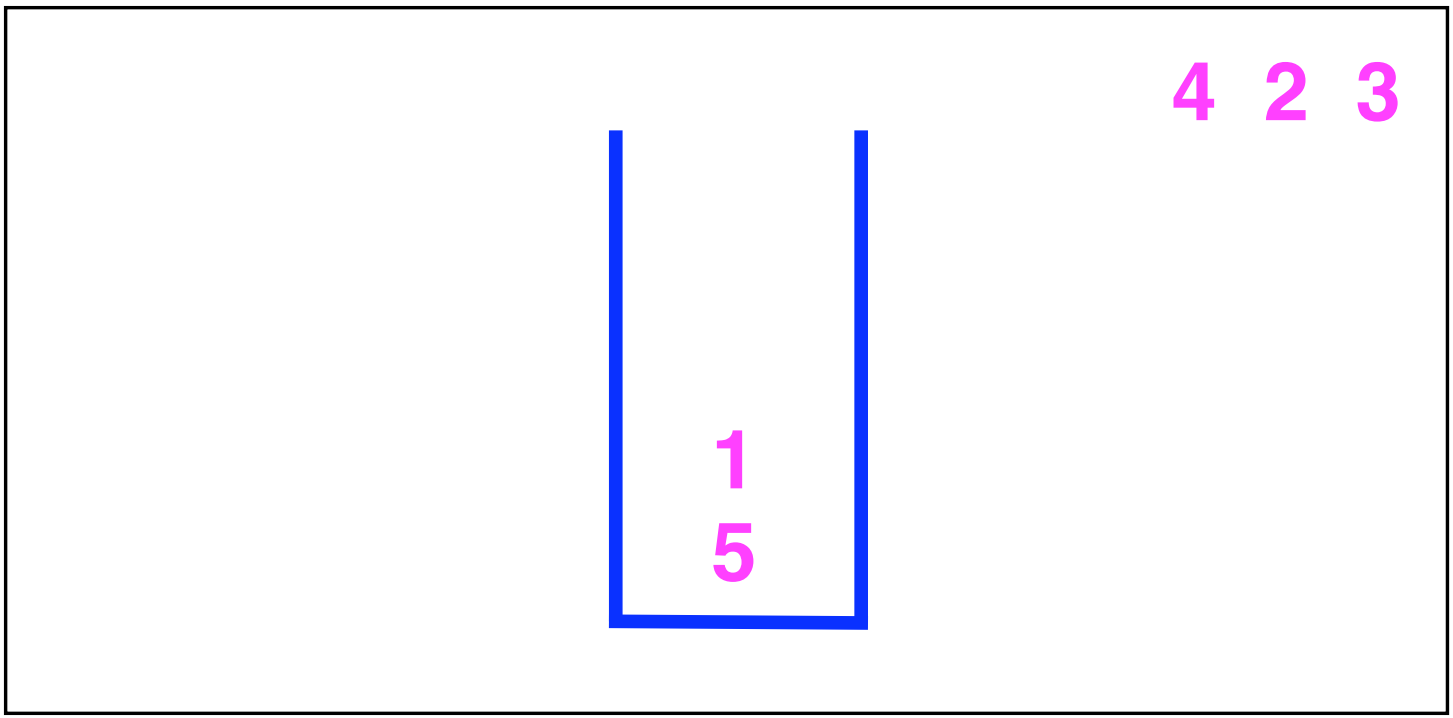
Sorting a permutation with a stack



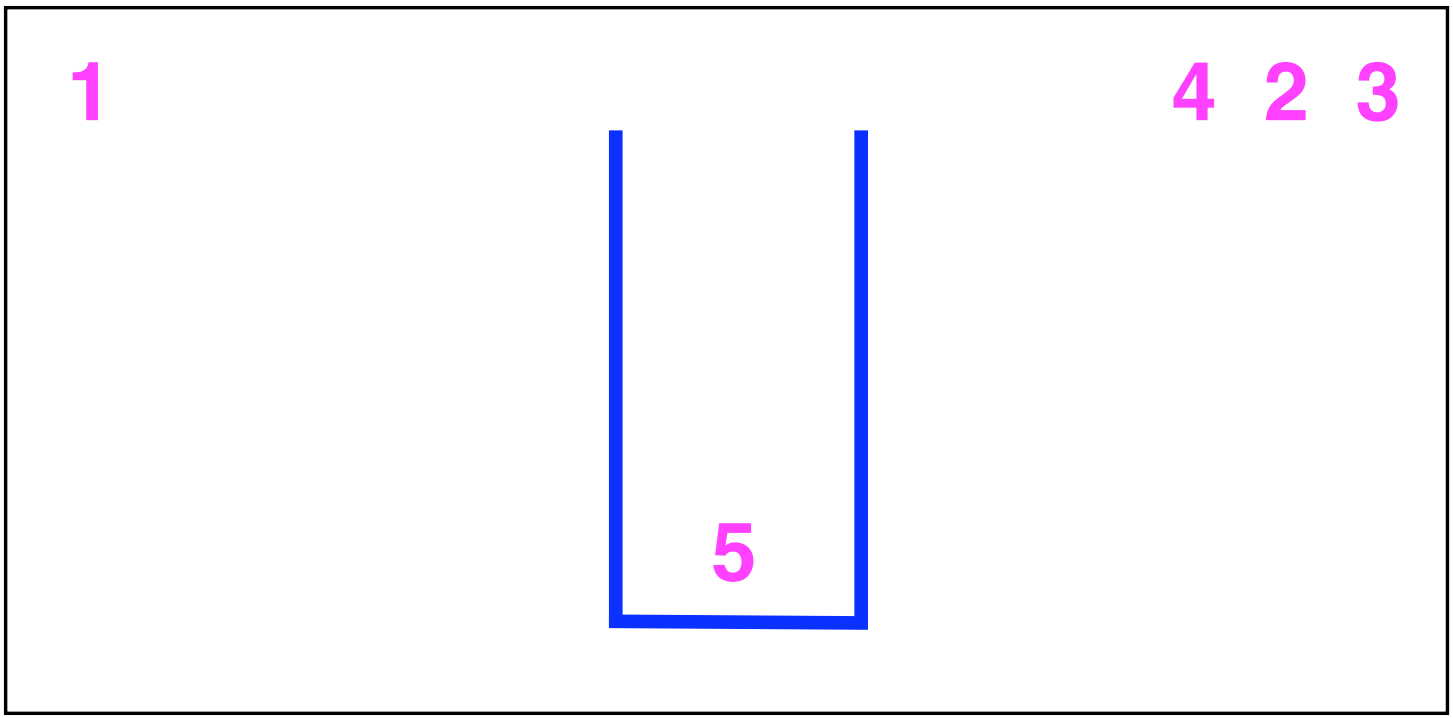
Sorting a permutation with a stack



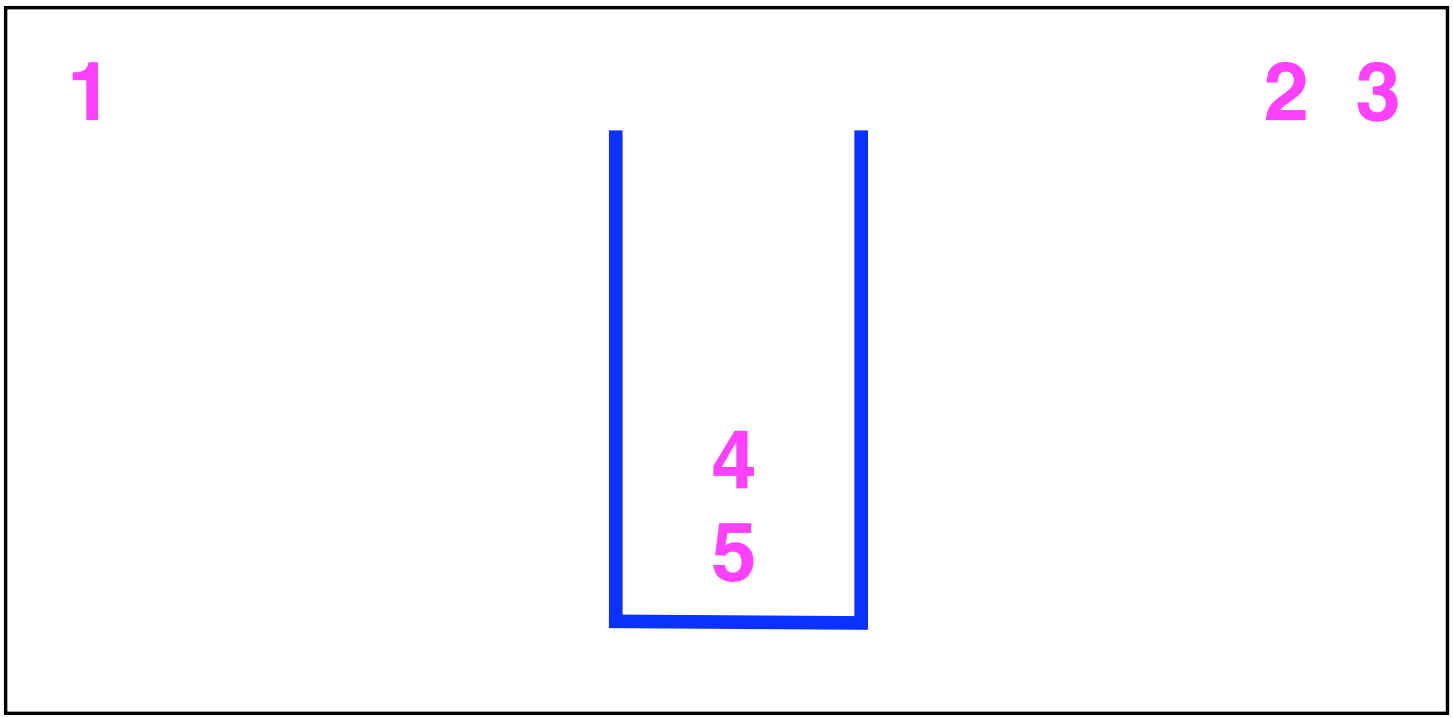
Sorting a permutation with a stack



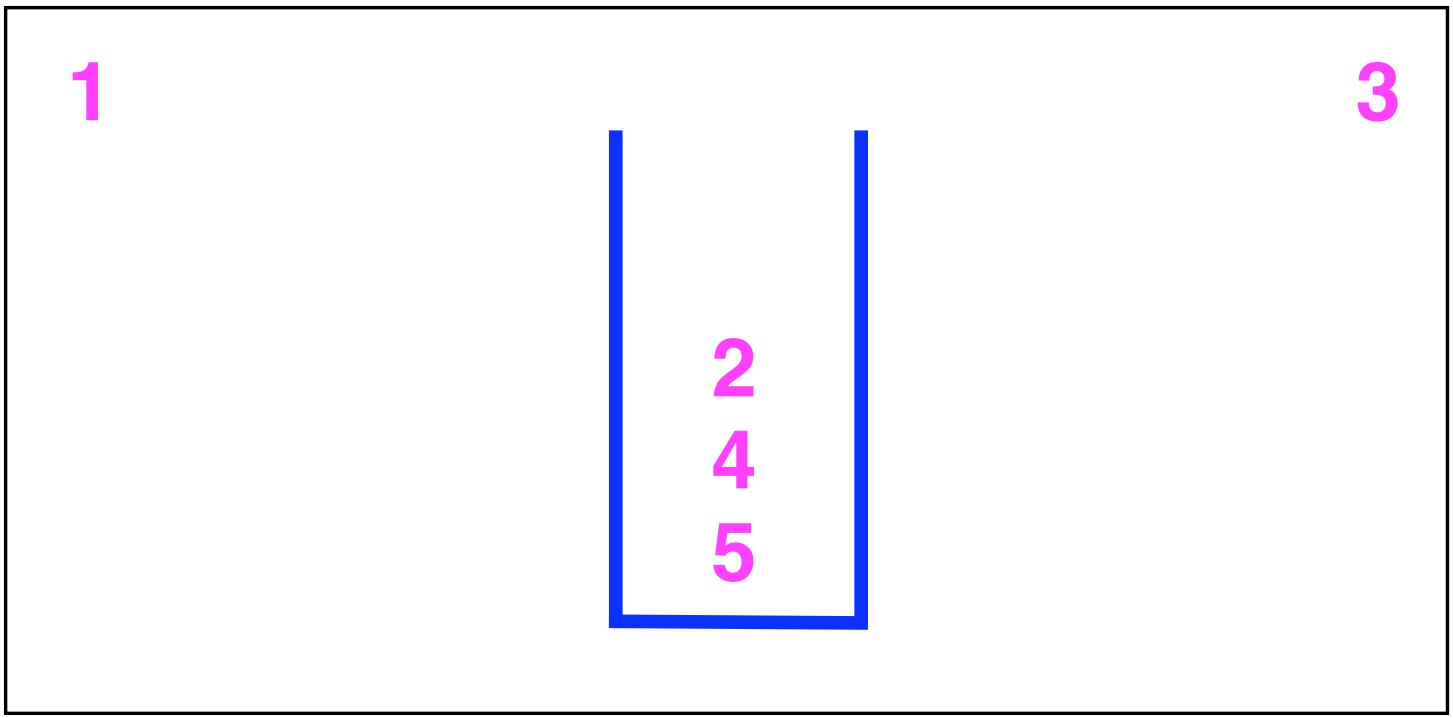
Sorting a permutation with a stack



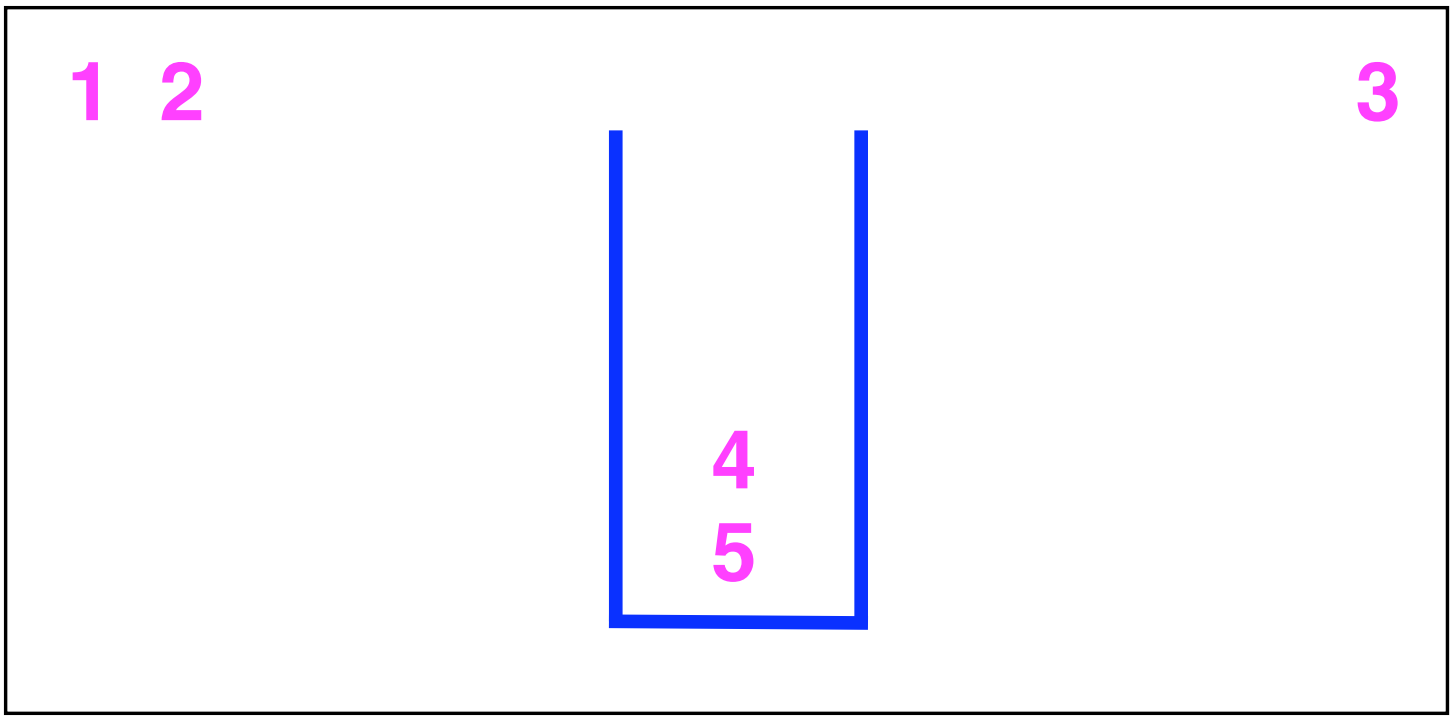
Sorting a permutation with a stack



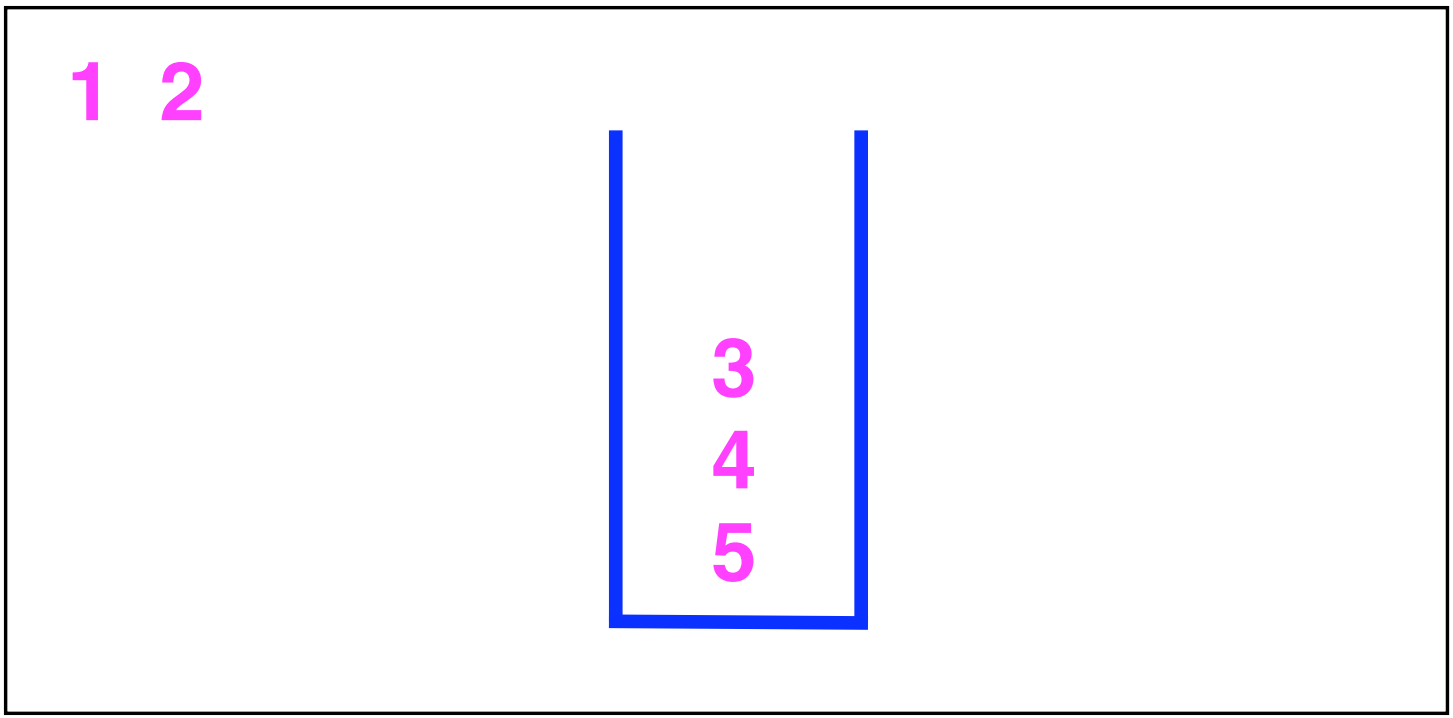
Sorting a permutation with a stack



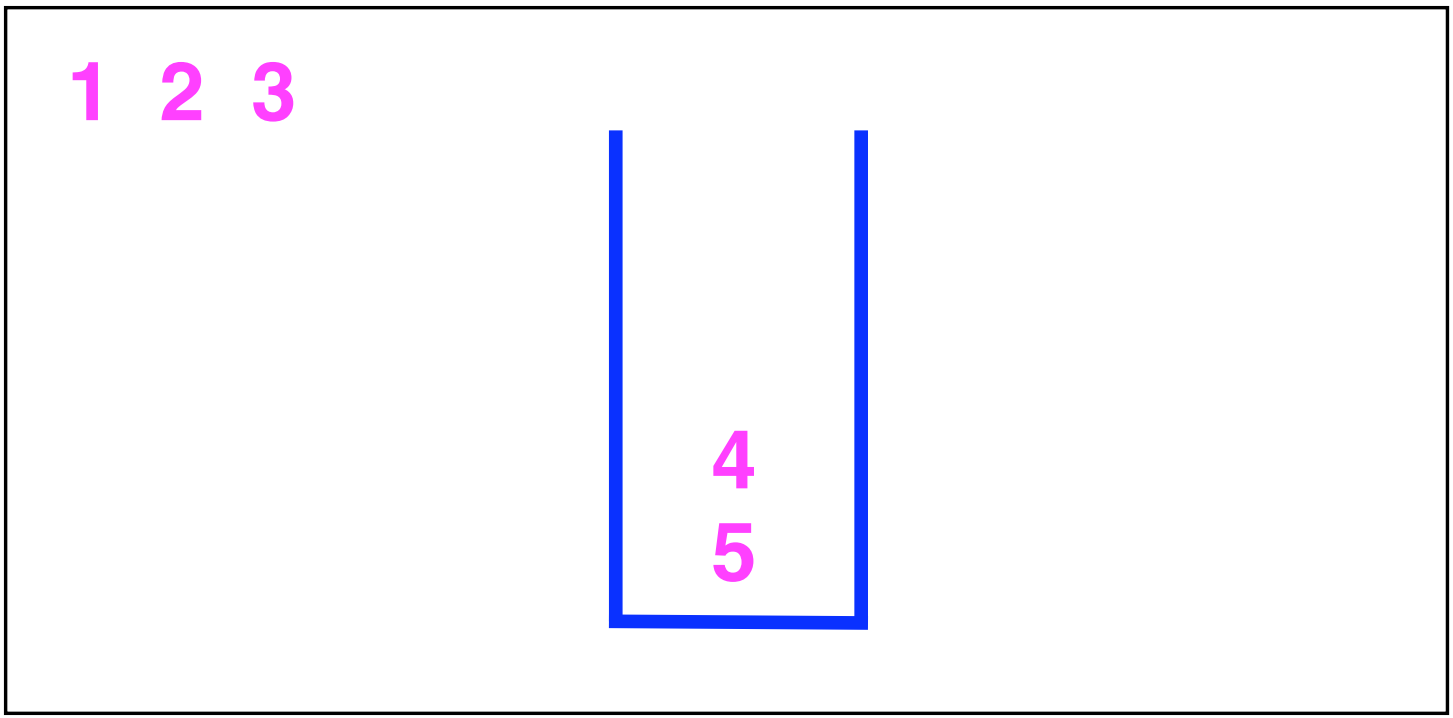
Sorting a permutation with a stack



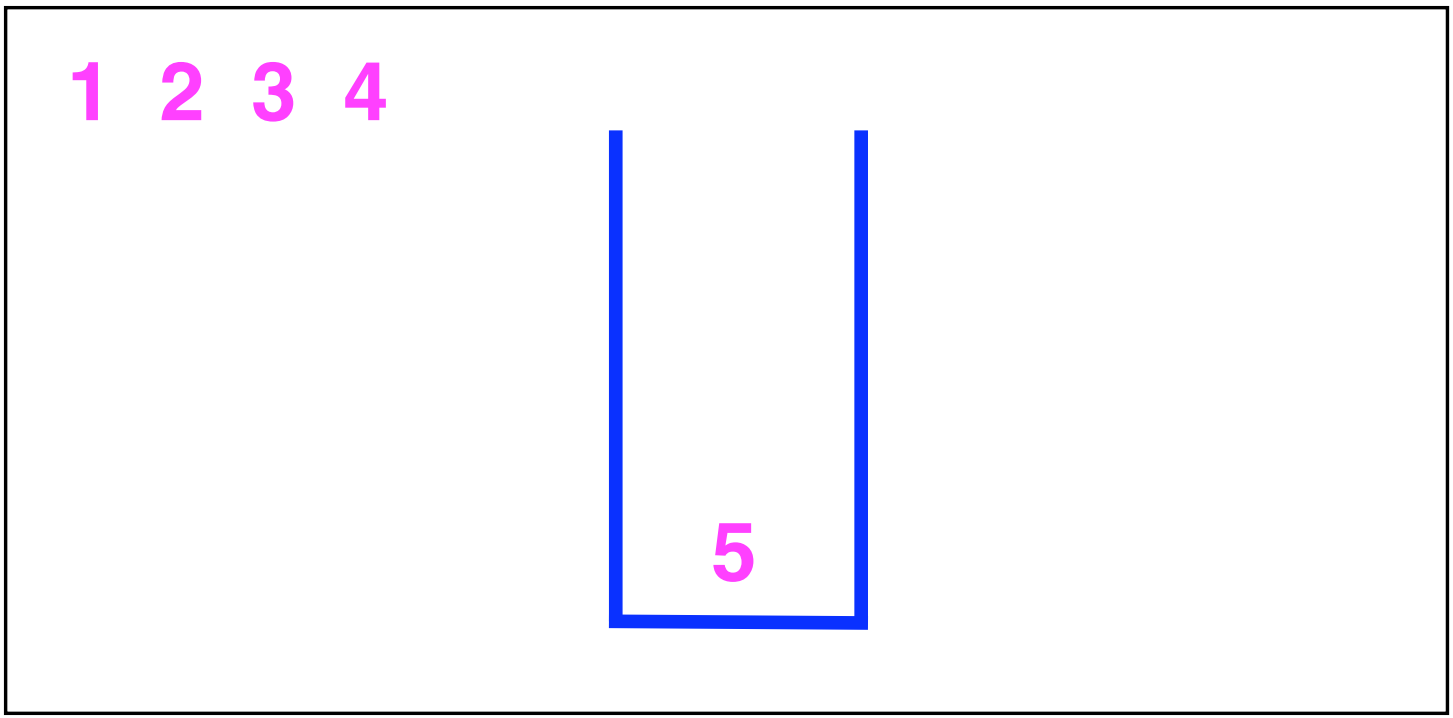
Sorting a permutation with a stack



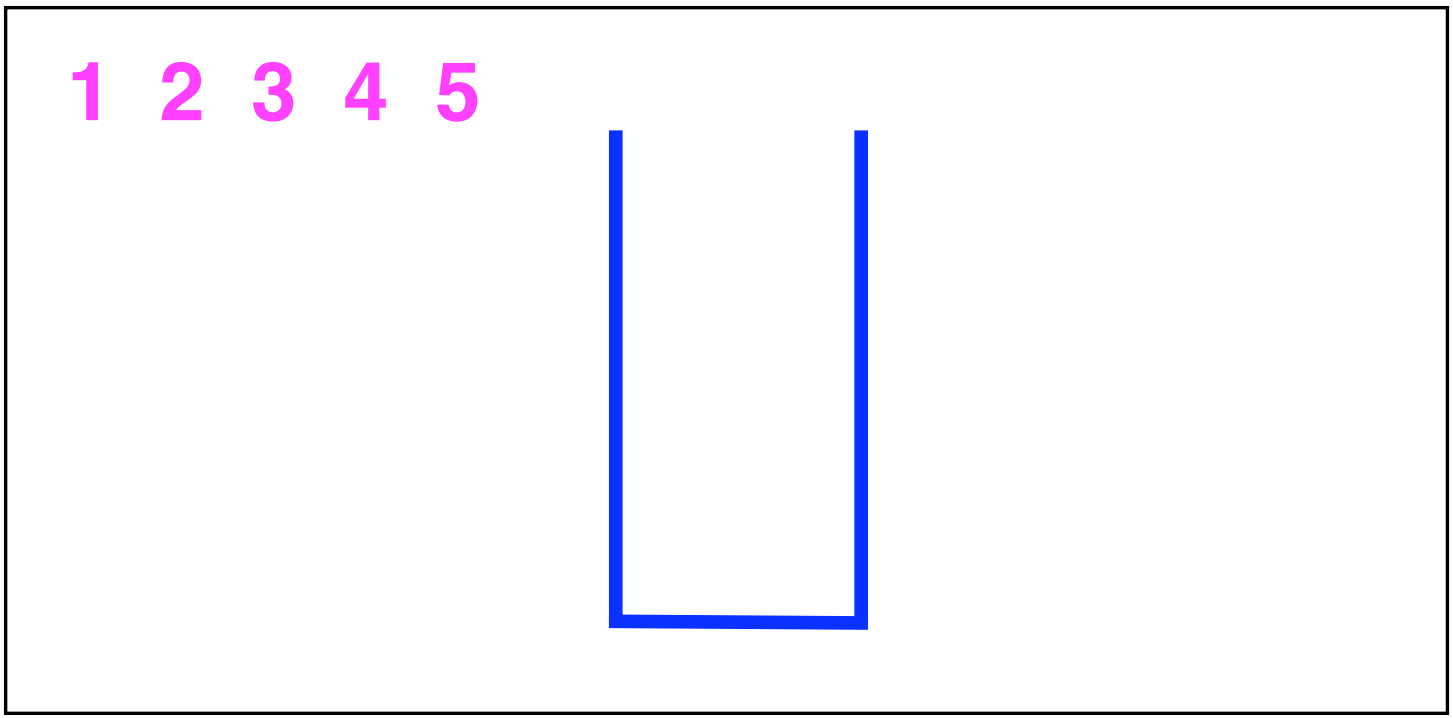
Sorting a permutation with a stack



Sorting a permutation with a stack



Sorting a permutation with a stack



Knuth theorem

Theorem

A permutation can be sorted via a stack if and only if it avoids the permutation 231*.*

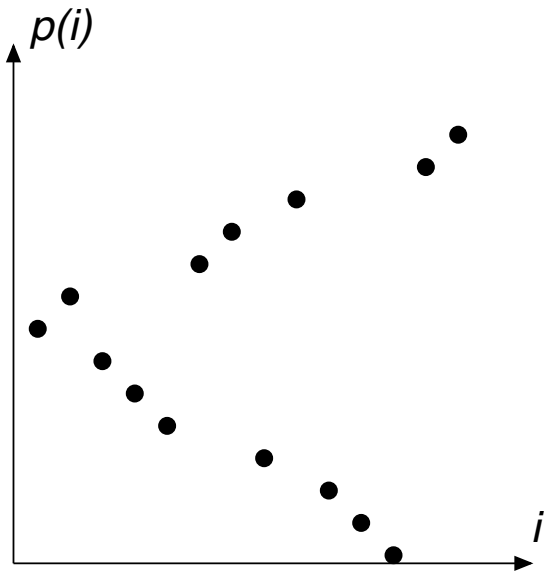
This was the first of very many connections between Pattern classes and various structures in Computer Science.

Conjuring with $\mathrm{Av}(132, 312)$

♠ ♢ ♡ ♣ ♢ ♠ ♣ ♢ ♡ ♣ ♠ ♡ ♢ ♢ ♠ ♡ ♣ ♠ ♣ ♢ ♣ ♡

A hint at the reveal

- The initial deck is rigged before the cut and shuffle
- The shuffle produces a permutation p in $\mathrm{Av}(132, 312)$



The graph of a permutation $p \in \mathrm{Av}(132, 312)$

How are priority queues and pattern classes connected?

- There ought to be a connection between allowable pairs and pattern classes because each have a similar looking downward closure property (one on pairs of permutations, the other on single permutations)
- And by following definitions we get

Theorem

If a priority queue is presented with the permutations of a fixed pattern class $\mathcal{X}$ as a set of inputs then the set of all possible outputs is also a pattern class $\mathcal{X}^*$.

So we have a map $\mathcal{X} \longrightarrow \mathcal{X}^*$ defined on the set of all pattern classes.

Most of the results are about this $\mathcal{X} \longrightarrow \mathcal{X}^*$ map.

How are priority queues and pattern classes connected?

- There ought to be a connection between allowable pairs and pattern classes because each have a similar looking downward closure property (one on pairs of permutations, the other on single permutations)
- And by following definitions we get

Theorem

If a priority queue is presented with the permutations of a fixed pattern class $\mathcal{X}$ as a set of inputs then the set of all possible outputs is also a pattern class $\mathcal{X}^*$.

So we have a map $\mathcal{X} \longrightarrow \mathcal{X}^*$ defined on the set of all pattern classes.

Most of the results are about this $\mathcal{X} \longrightarrow \mathcal{X}^*$ map.

How are priority queues and pattern classes connected?

- There ought to be a connection between allowable pairs and pattern classes because each have a similar looking downward closure property (one on pairs of permutations, the other on single permutations)
- And by following definitions we get

Theorem

If a priority queue is presented with the permutations of a fixed pattern class $\mathcal{X}$ as a set of inputs then the set of all possible outputs is also a pattern class $\mathcal{X}^*$.

So we have a map $\mathcal{X} \longrightarrow \mathcal{X}^*$ defined on the set of all pattern classes.

Most of the results are about this $\mathcal{X} \longrightarrow \mathcal{X}^*$ map.

How are priority queues and pattern classes connected?

- There ought to be a connection between allowable pairs and pattern classes because each have a similar looking downward closure property (one on pairs of permutations, the other on single permutations)
- And by following definitions we get

Theorem

If a priority queue is presented with the permutations of a fixed pattern class $\mathcal{X}$ as a set of inputs then the set of all possible outputs is also a pattern class $\mathcal{X}^*$.

So we have a map $\mathcal{X} \longrightarrow \mathcal{X}^*$ defined on the set of all pattern classes.

Most of the results are about this $\mathcal{X} \longrightarrow \mathcal{X}^*$ map.

Simple examples of the $\mathcal{X} \longrightarrow \mathcal{X}^*$ map

- Suppose $\mathcal{X}$ is the set of all increasing permutations $12\cdots n$ (one permutation of every length). No priority queue computation can disorder $12\cdots n$ so $\mathcal{X}^* = \mathcal{X}$.
- Suppose $\mathcal{X}$ is the set of all decreasing permutations $n\cdots 21$ (one permutation of every length). Priority queue computations now behave just as though the priority queue was a stack. So (Knuth's theorem) $\mathcal{X}^*$ is the set of permutations that avoid 132.

Simple examples of the $\mathcal{X} \longrightarrow \mathcal{X}^*$ map

- Suppose $\mathcal{X}$ is the set of all increasing permutations $12\cdots n$ (one permutation of every length). No priority queue computation can disorder $12\cdots n$ so $\mathcal{X}^* = \mathcal{X}$.
- Suppose $\mathcal{X}$ is the set of all decreasing permutations $n\cdots 21$ (one permutation of every length). Priority queue computations now behave just as though the priority queue was a stack. So (Knuth's theorem) $\mathcal{X}^*$ is the set of permutations that avoid 132.

Simple examples of the $\mathcal{X} \longrightarrow \mathcal{X}^*$ map

- Suppose $\mathcal{X}$ is the set of all increasing permutations $12\cdots n$ (one permutation of every length). No priority queue computation can disorder $12\cdots n$ so $\mathcal{X}^* = \mathcal{X}$.
- Suppose $\mathcal{X}$ is the set of all decreasing permutations $n\cdots 21$ (one permutation of every length). Priority queue computations now behave just as though the priority queue was a stack. So (Knuth's theorem) $\mathcal{X}^*$ is the set of permutations that avoid 132.

The boring main theorems

$\mathcal{X}$ defined by avoiding	$\mathcal{X}^*$ defined by avoiding
21	21
12	132

$\mathcal{X}$ defined by avoiding	$\mathcal{X}^*$ defined by avoiding
321	321
312	3142, 4132
231	2431
213	2143
132	1432
123	13254, 14253, 15243

The boring main theorems

$\mathcal{X}$ defined by avoiding	$\mathcal{X}^*$ defined by avoiding
21	21
12	132

$\mathcal{X}$ defined by avoiding	$\mathcal{X}^*$ defined by avoiding
321	321
312	3142, 4132
231	2431
213	2143
132	1432
123	13254, 14253, 15243

The boring main theorems. Stay awake, almost done!

$\mathcal{X}$ defined by avoiding	$\mathcal{X}^*$ defined by avoiding
132, 321	321, 2143, 2413
213, 321	321, 2143, 2413
231, 312	2413, 2431, 3142, 4132
231, 321	231, 321
123, 132	1423, 1432, 13254
...	About 50 further cases!!

But if $\mathcal{X}$ is defined by avoiding 2431 there is no *finite* avoided list defining $\mathcal{X}^*$

The boring main theorems. Stay awake, almost done!

$\mathcal{X}$ defined by avoiding	$\mathcal{X}^*$ defined by avoiding
132, 321	321, 2143, 2413
213, 321	321, 2143, 2413
231, 312	2413, 2431, 3142, 4132
231, 321	231, 321
123, 132	1423, 1432, 13254
...	About 50 further cases!!

But if $\mathcal{X}$ is defined by avoiding 2431 there is no *finite* avoided list defining $\mathcal{X}^*$