Data Structures

Binary Heaps

Yossi Azar & Rani Hod

Fall 2023

Required ADT

- Maintain items with keys subject to

- Insert(x,Q), Delete(x,Q)
- min(Q), Delete-min(Q)
- Decrease-key(x,Q,Δ)

Required ADT

- Decrease-key(x,Q,Δ): Can be simulated by Delete(x,Q), $x.key \leftarrow x.key - \Delta$, insert$(x,Q)$

We can use **AVL** trees to implement all operations in $O(\log n)$ time

We want to implement Decrease-key in $O(1)$ (amortized) time

Motivation

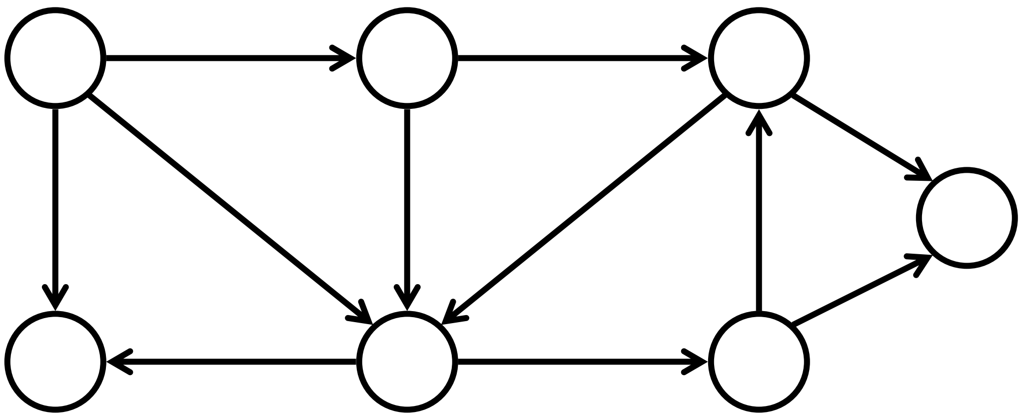
- Dijkstra's algorithm for single source shortest path
- Prim's algorithm for minimum spanning trees

Motivation

- Want to find the shortest route from New York to San Francisco

- Model the road-map with a graph

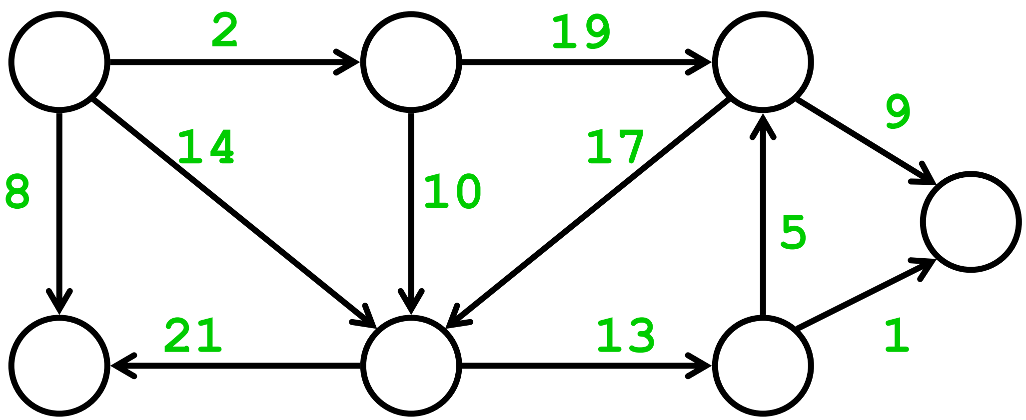
A Graph G=(V,E)



V is a set of vertices

E is a set of edges (pairs of vertices)

Model driving distances by weights on the edges

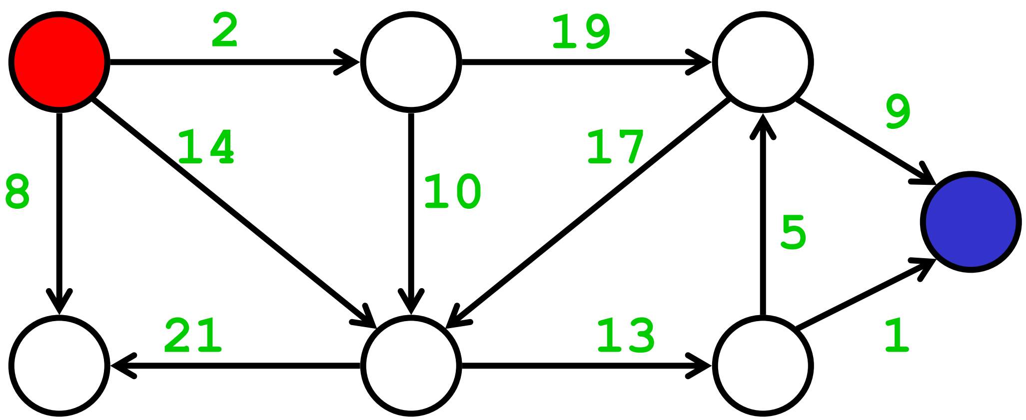


V is a set of vertices

E is a set of edges (pairs of vertices)

w is a weight function

Source and destination



V is a set of vertices

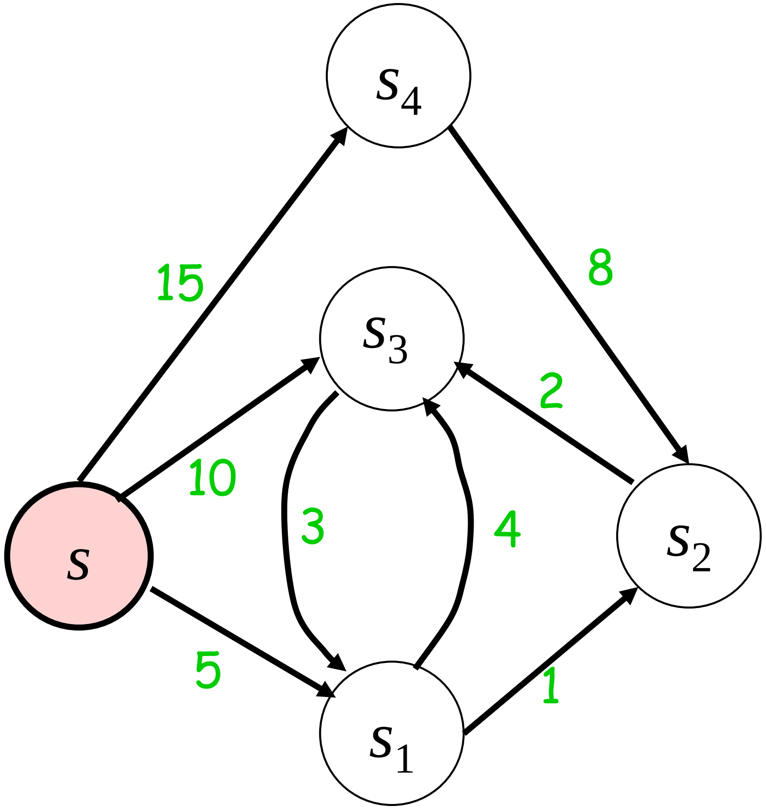
E is a set of edges (pairs of vertices)

w is a weight function

Dijkstra's algorithm

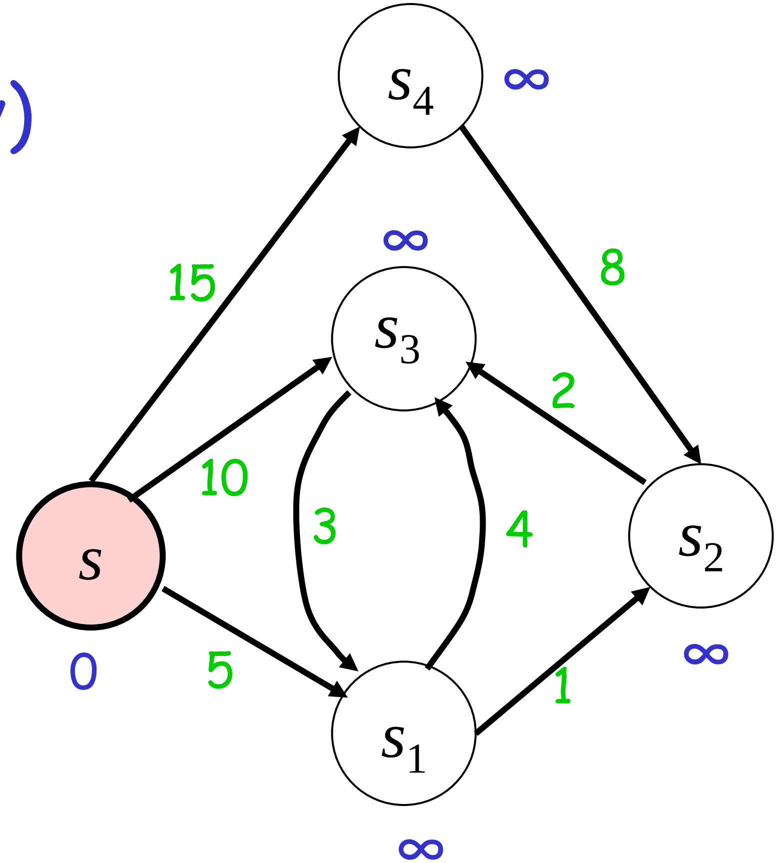
- Assume all weights are non-negative
- Finds the shortest path from some fixed vertex s to every other vertex

Example



Example

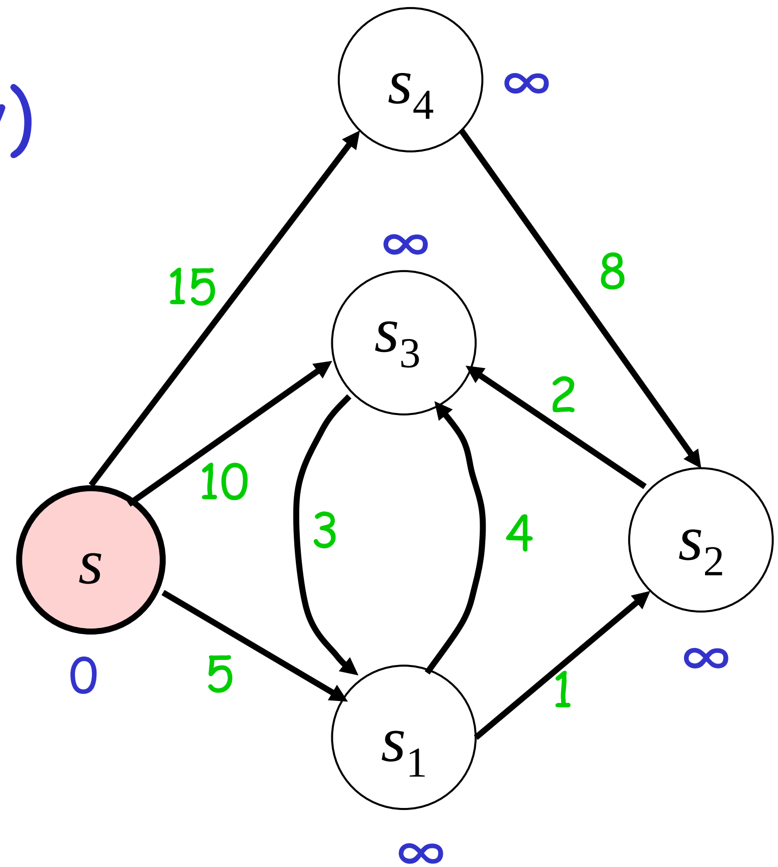
Maintain an upper bound d(v) on the shortest path to v



Maintain an upper bound d(v) on the shortest path to v

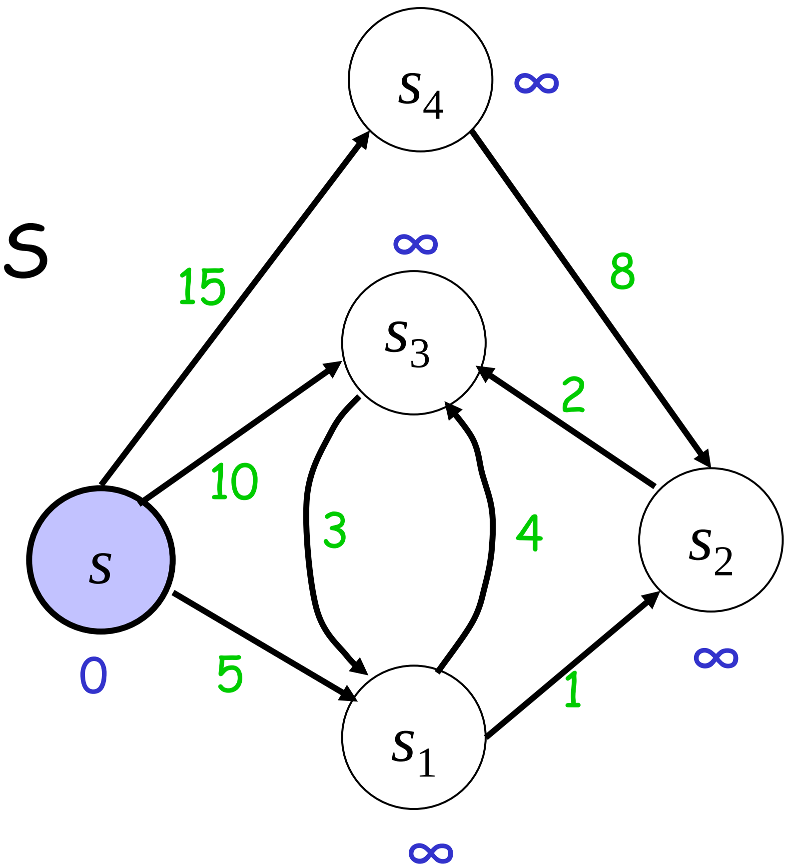
A node is either scanned (in S) or labeled (in Q)

Initially S = ∅ and Q = V



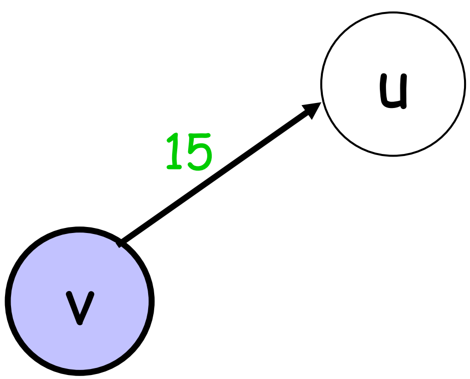
Initially S = ∅ and Q = V

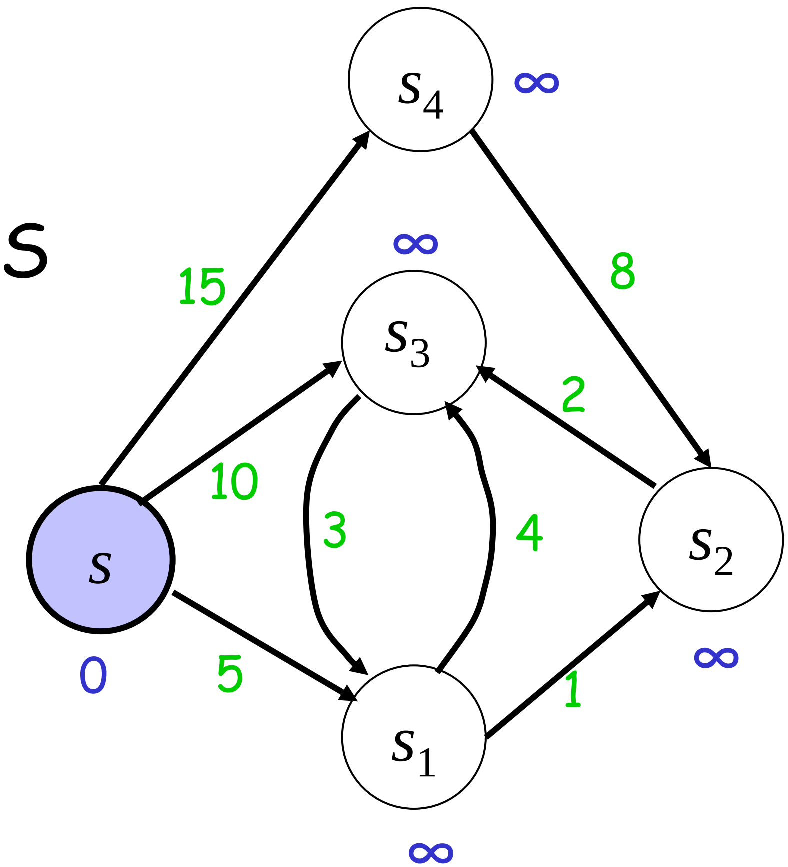
Pick a vertex v in Q with minimum d(v) and add it to S



Initially S = ∅ and Q = V

Pick a vertex v in Q with minimum d(v) and add it to S





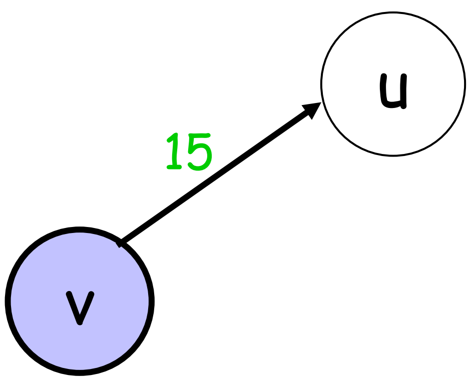
For every edge (v,u) where u in Q: relax(v,u)

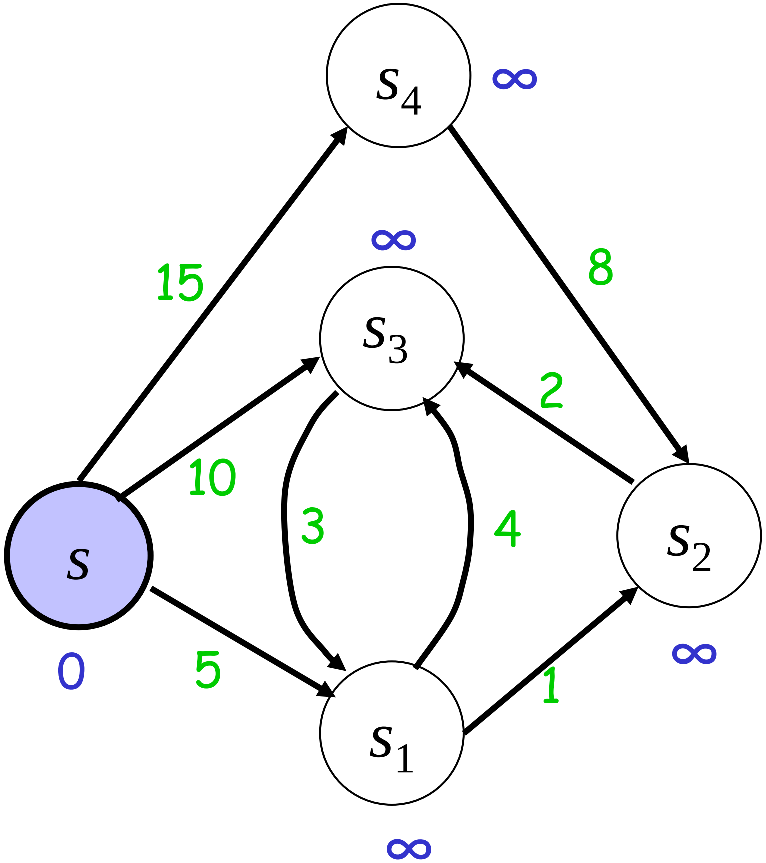
Relax(v,u)

If $d(v) + w(v,u) < d(u)$ then

  $d(u) \leftarrow d(v) + w(v,u)$

  $\pi(u) \leftarrow v$

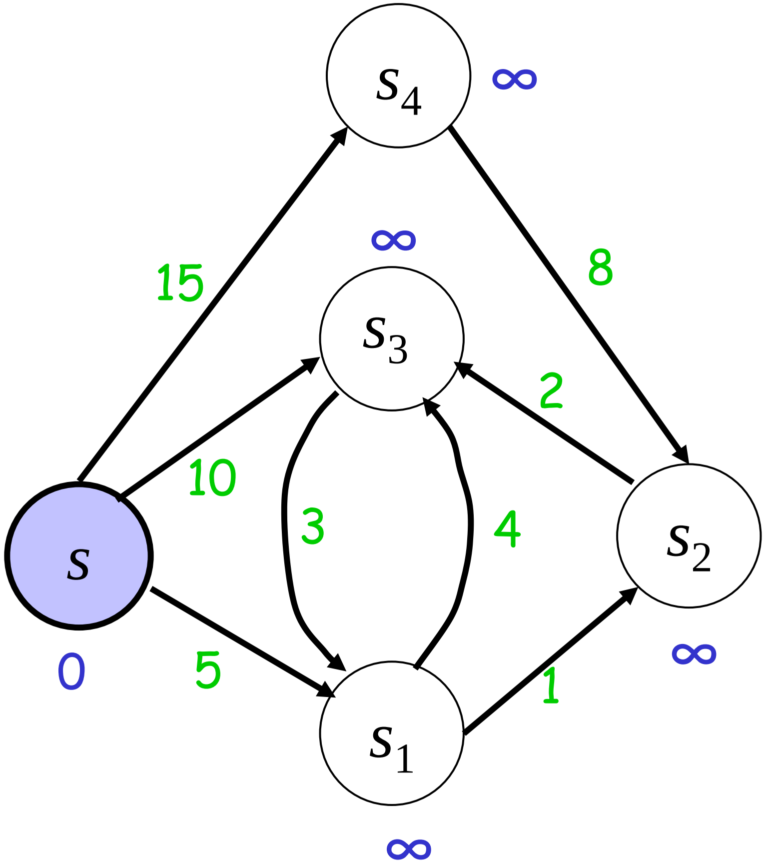




For every edge (v,u) where u in Q: relax(v,u)

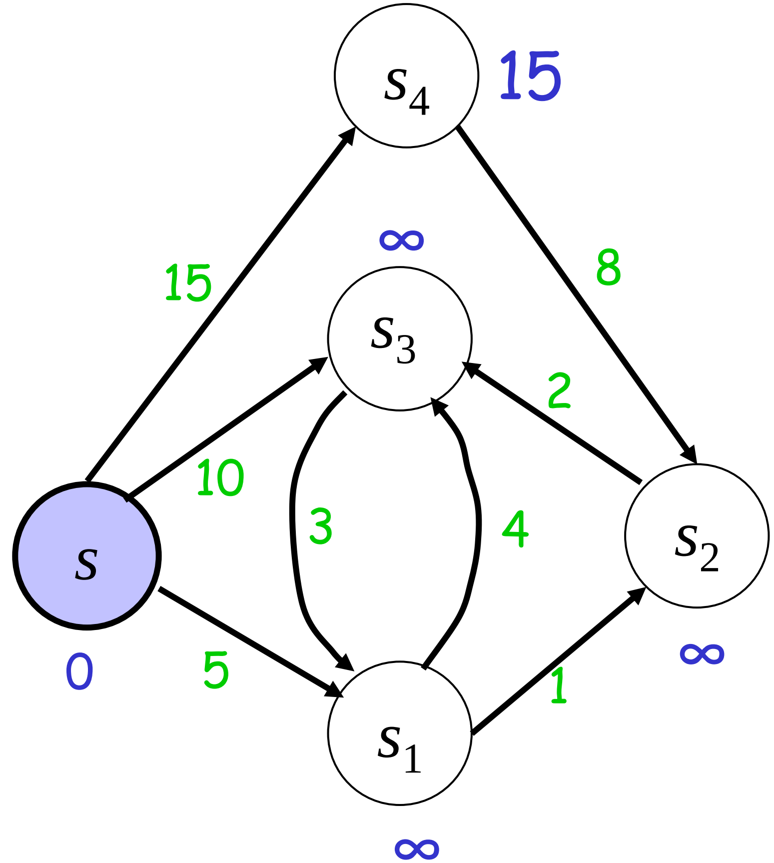
$S = \{s\}$

$\mathrm{Relax}(s, s_4)$



$S = \{s\}$

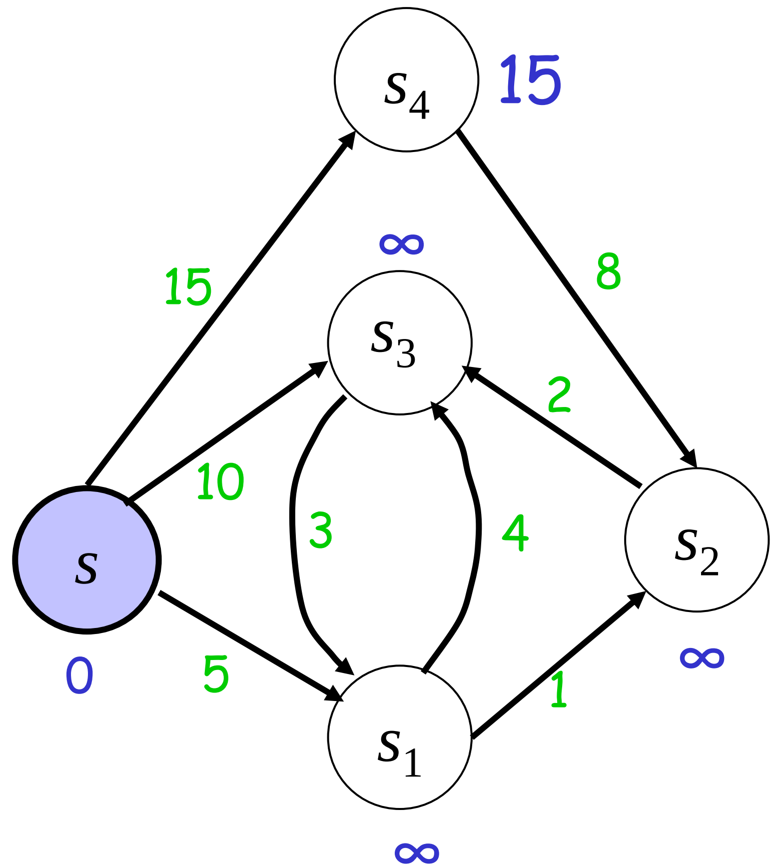
Relax(s,s_4)



$S = \{s\}$

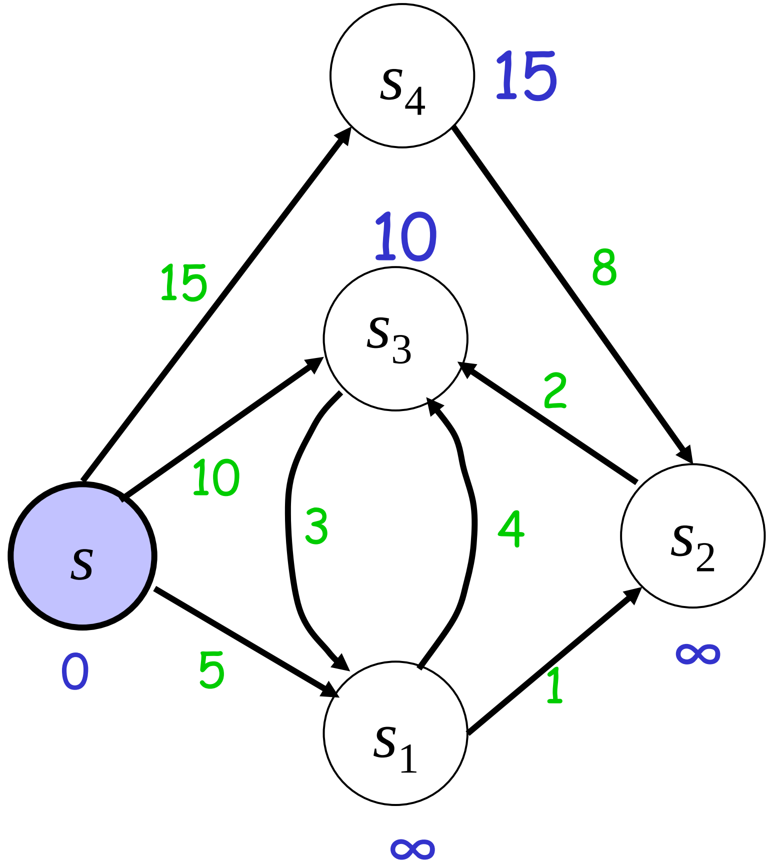
Relax(s, s_4)

Relax(s, s_3)



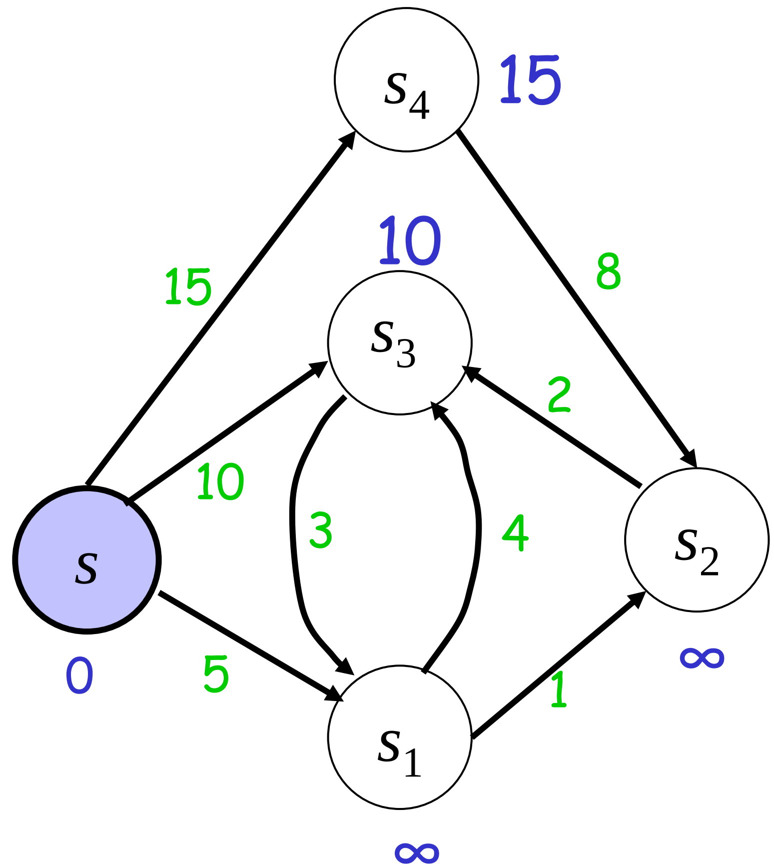
$S = \{s\}$

Relax(s,s_4)
Relax(s,s_3)

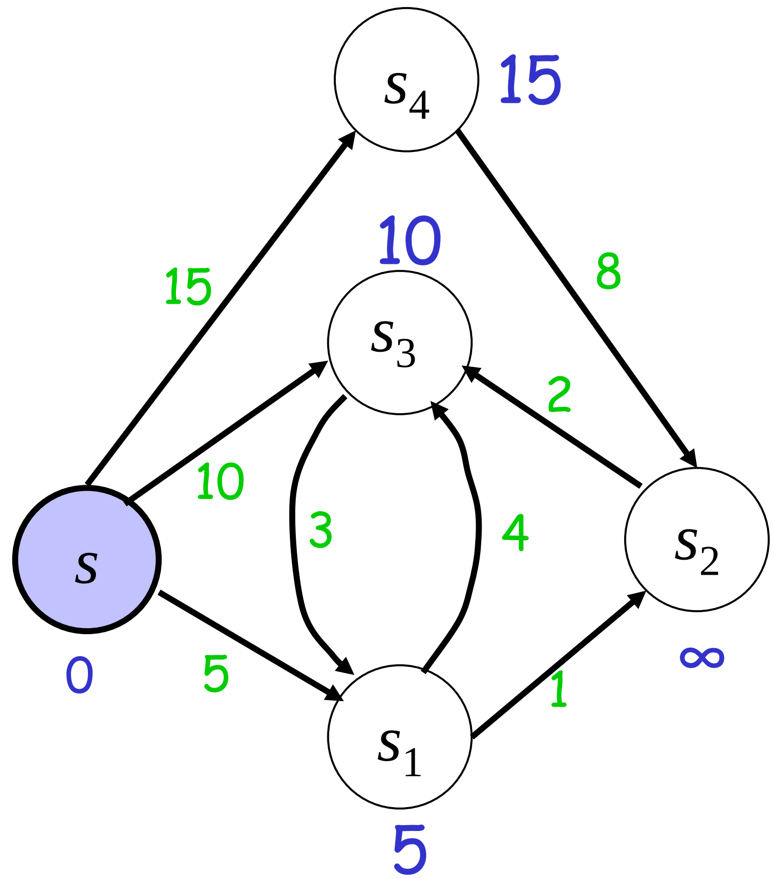


$S = \{s\}$

$\text{Relax}(s, s_4)$
$\text{Relax}(s, s_3)$
$\text{Relax}(s, s_1)$



$S = \{s\}$



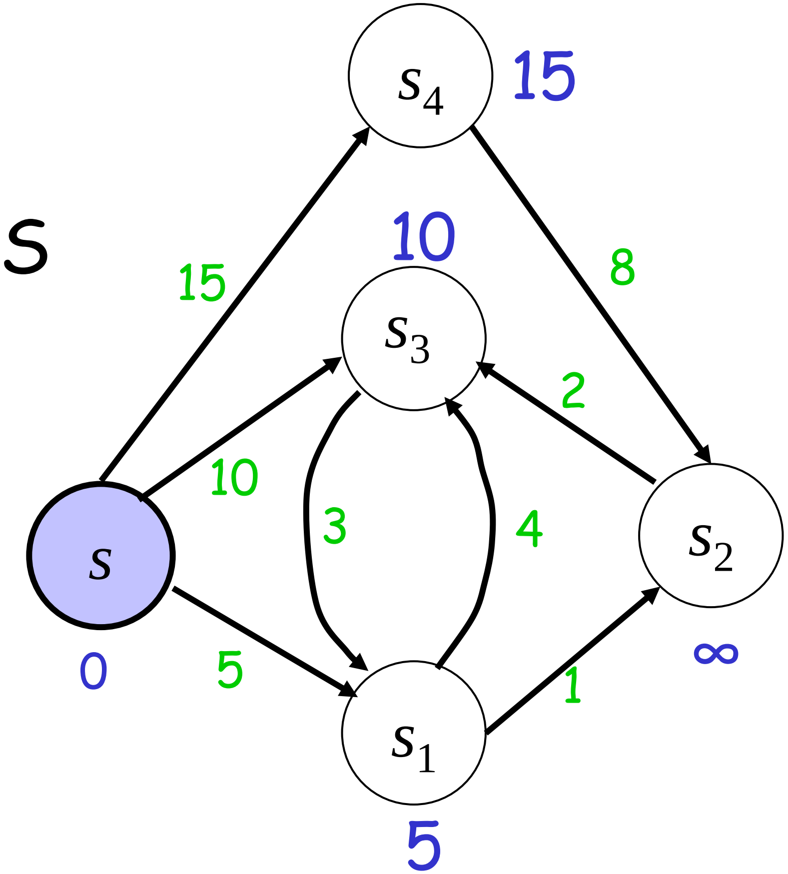
Relax(s,s_4)

Relax(s,s_3)

Relax(s,s_1)

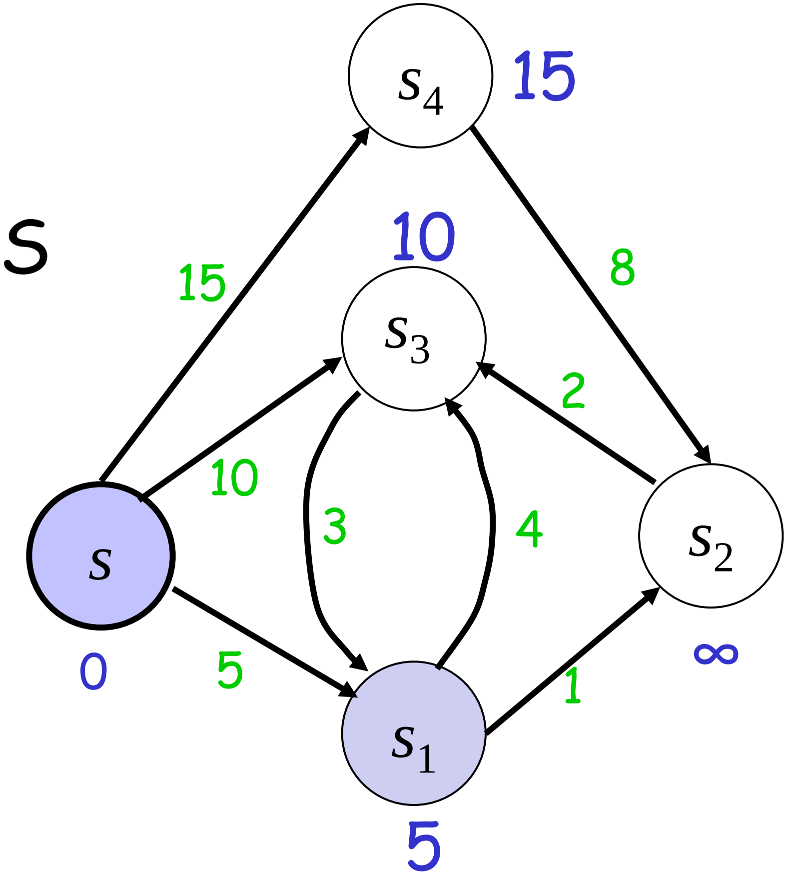
$S = \{s\}$

Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1\}$

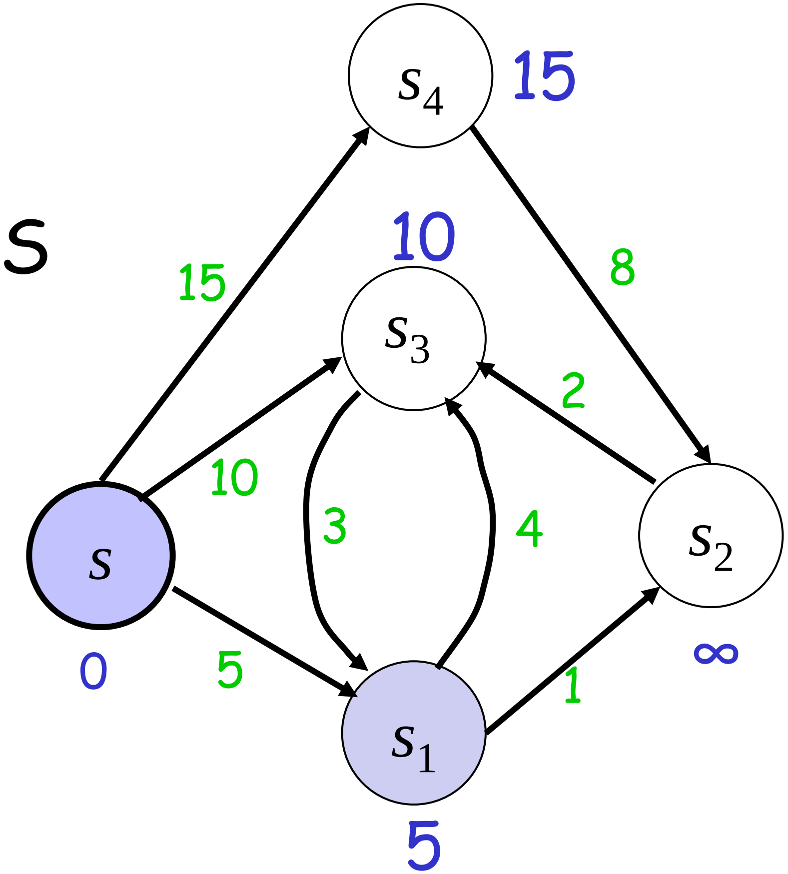
Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S

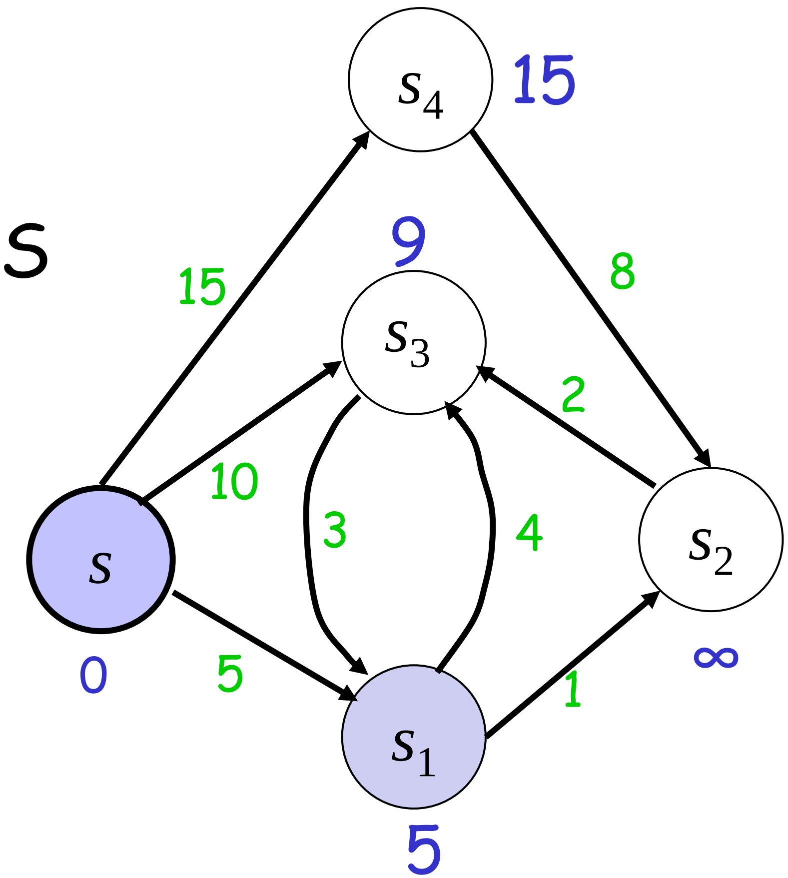
Relax(s_1,s_3)



$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S

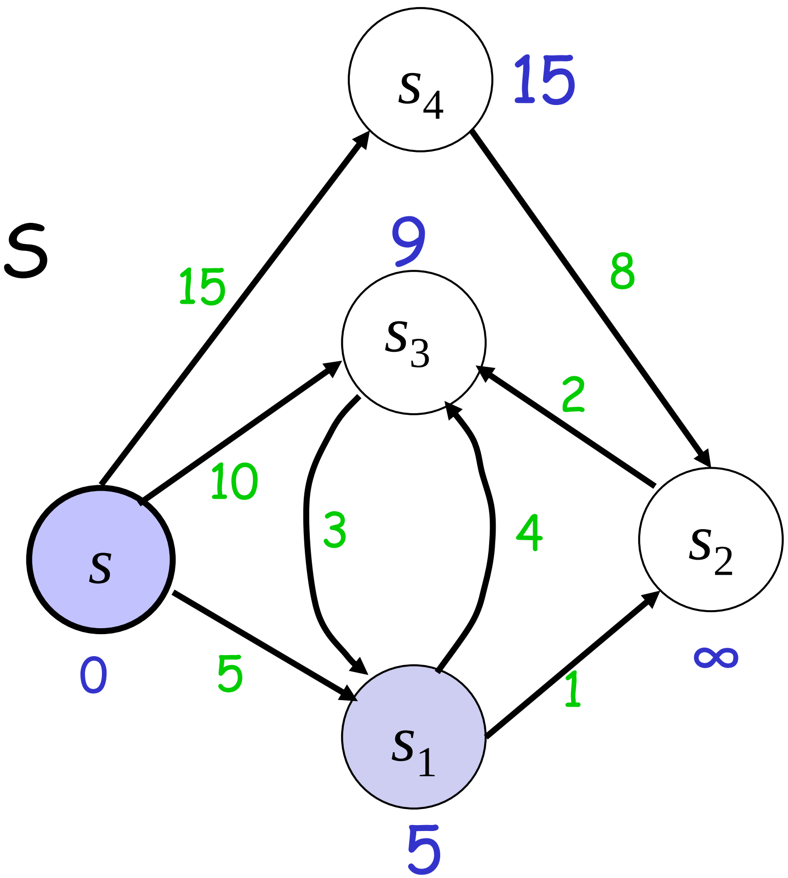
Relax(s_1,s_3)



$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S

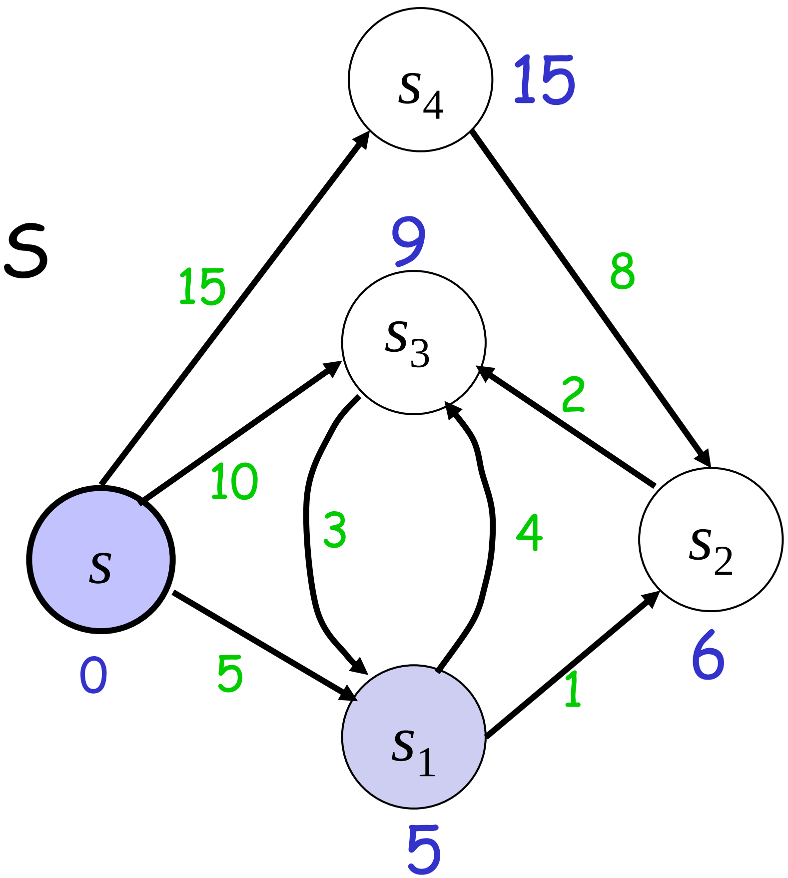
Relax(s_1,s_3)
Relax(s_1,s_2)



$S = \{s, s_1\}$

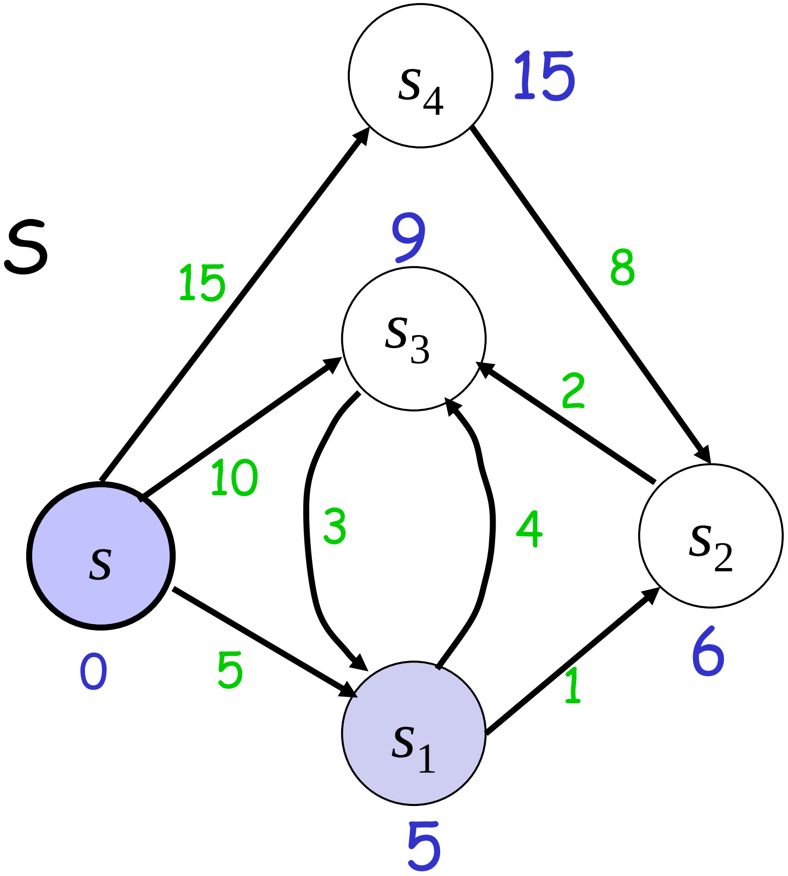
Pick a vertex v in Q with minimum d(v) and add it to S

Relax(s_1,s_3)
Relax(s_1,s_2)



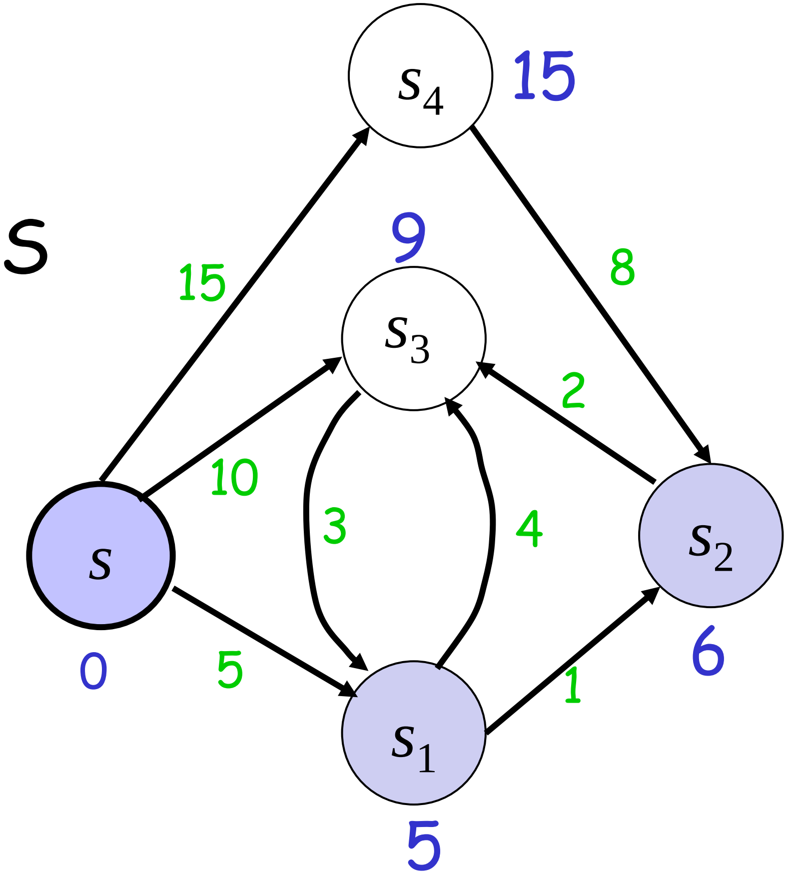
$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2\}$

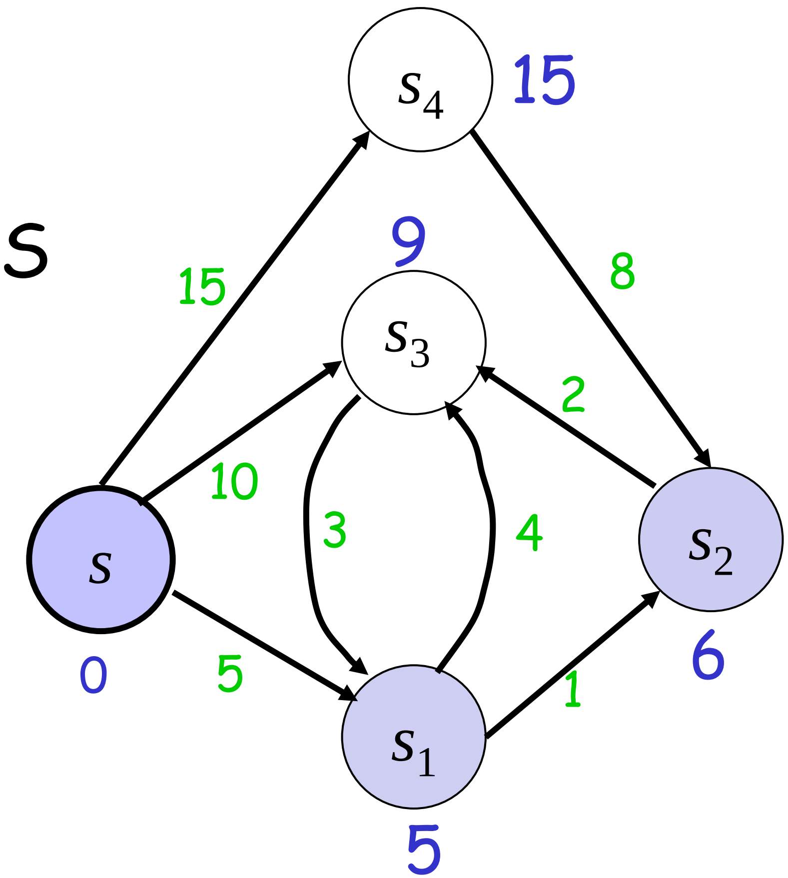
Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2\}$

Pick a vertex v in Q with minimum d(v) and add it to S

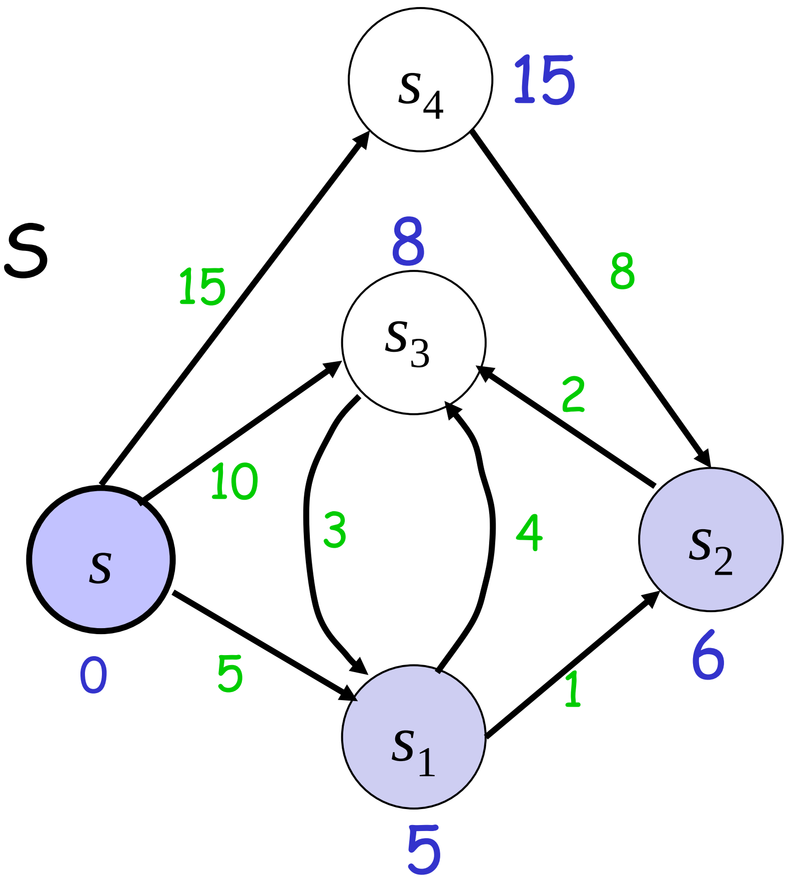
Relax(s_2,s_3)



$S = \{s, s_1, s_2\}$

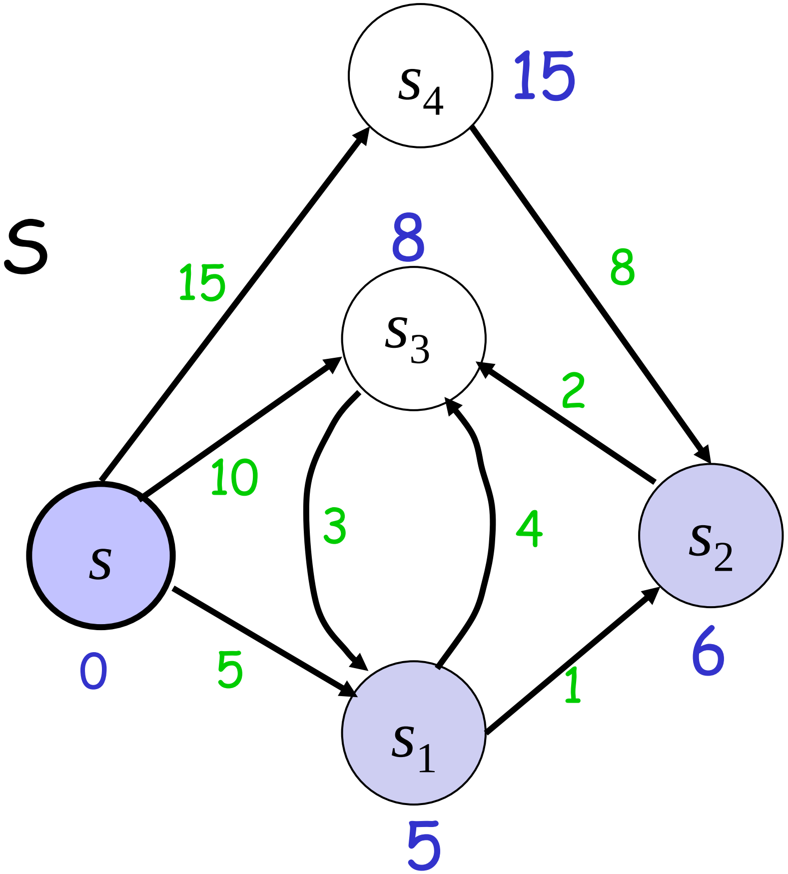
Pick a vertex v in Q with minimum d(v) and add it to S

Relax(s_2,s_3)



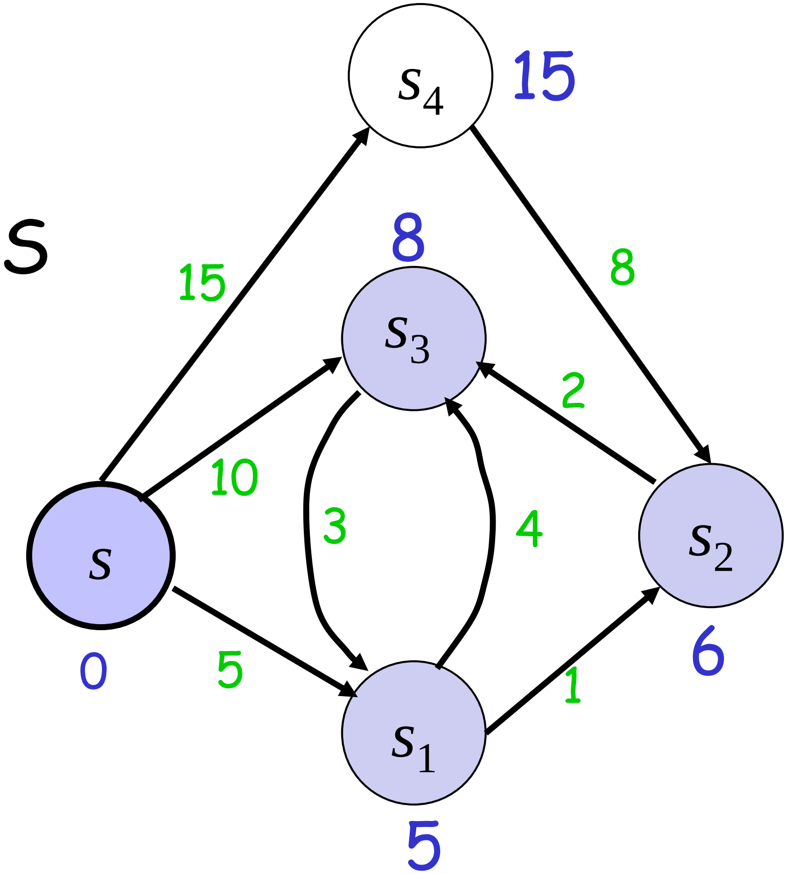
$S = \{s, s_1, s_2\}$

Pick a vertex v in Q with minimum d(v) and add it to S



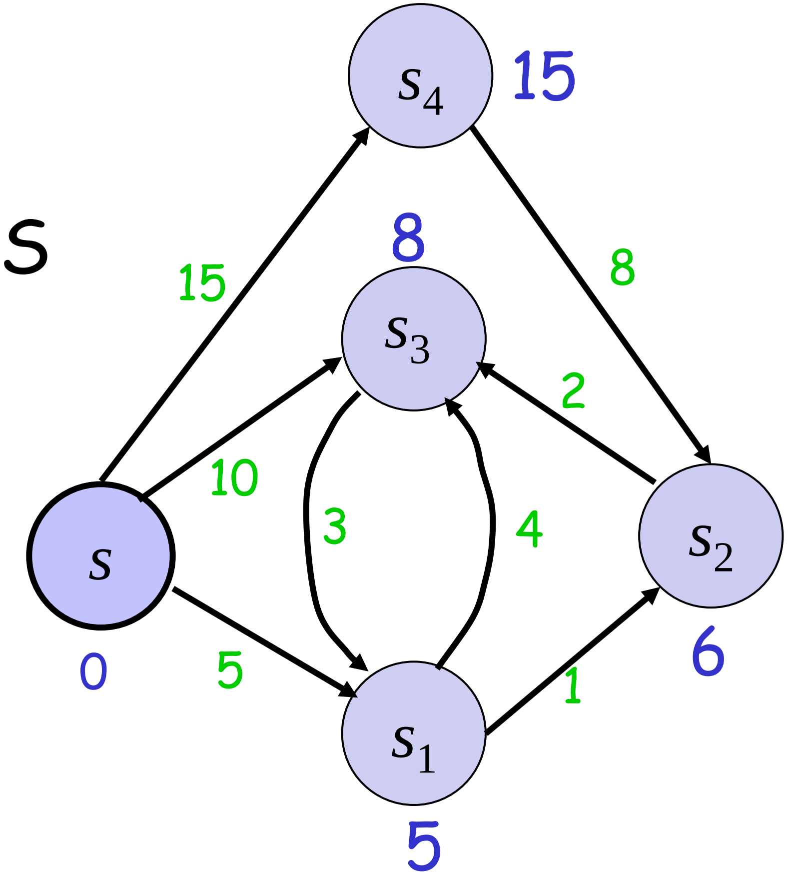
$S = \{s, s_1, s_2, s_3\}$

Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2, s_3, s_4\}$

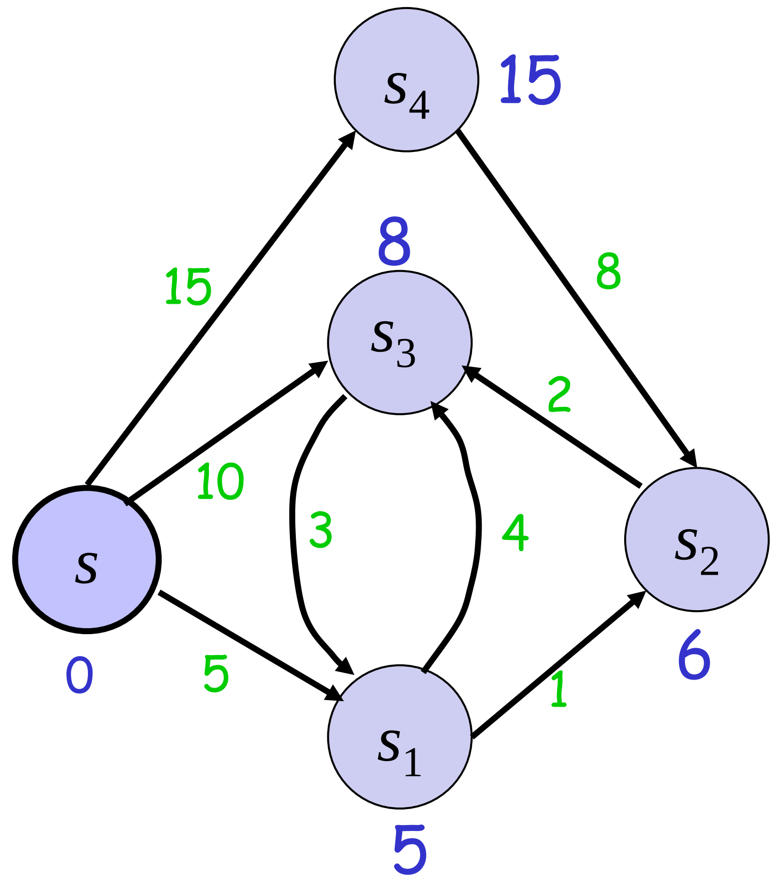
Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2, s_3, s_4\}$

When $Q = \varnothing$ then the d() values are the distances from s

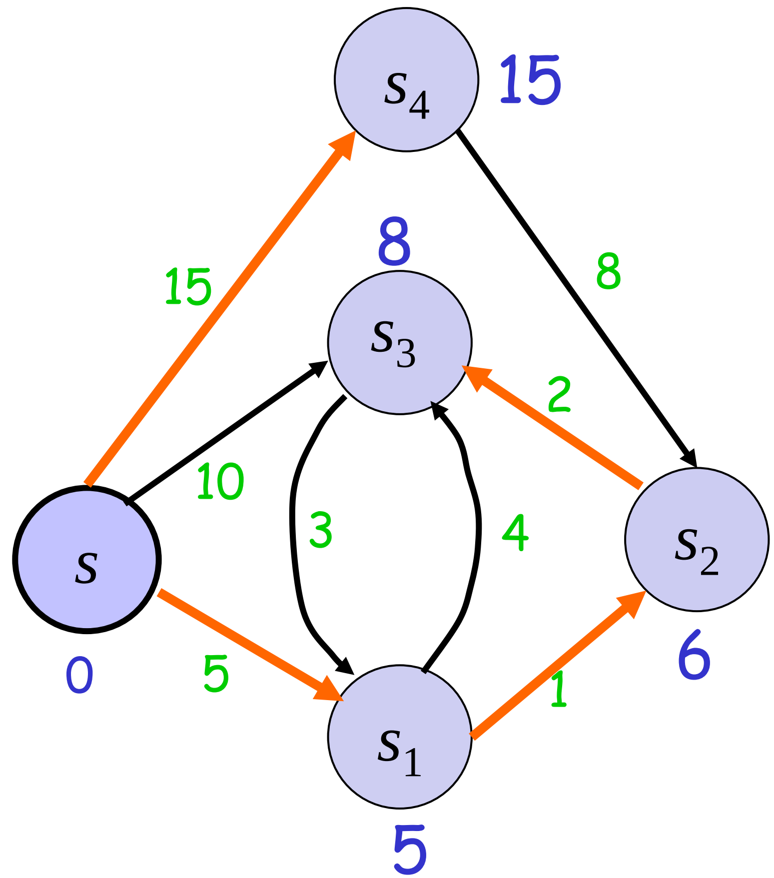
The π function gives the shortest path tree



$S = \{s, s_1, s_2, s_3, s_4\}$

When $Q = \varnothing$ then the d() values are the distances from s

The π function gives the shortest path tree



Implementation of Dijkstra's algorithm

- We need to find efficiently the vertex with minimum d() in Q
- We need to update d() values of vertices in Q

Required ADT

- Maintain items with keys subject to

- Insert(x,Q)
- min(Q)
- Delete-min(Q)
- Decrease-key(x,Q,Δ)

How many times we do these operations ?

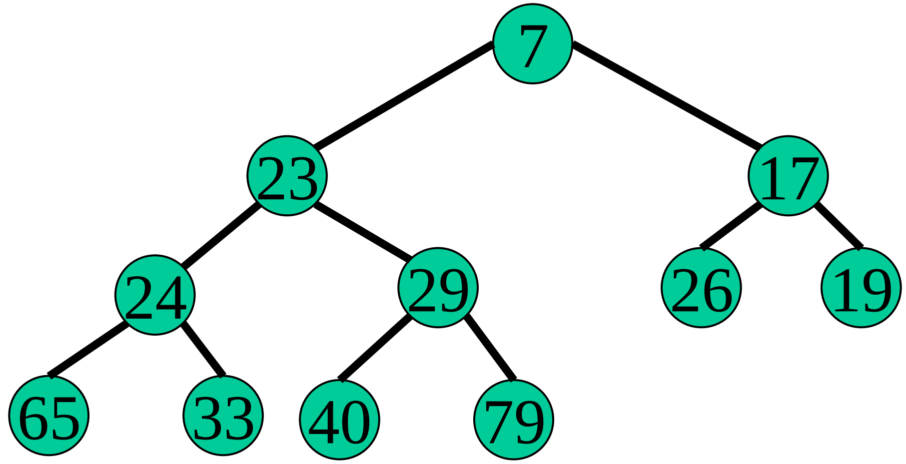
- Insert(x,Q) $n = |V|$
- min(Q) n
- Deletemin(Q) n
- Decrease-key(x,Q,Δ): Can simulate by Delete(x,Q), insert(x-Δ,Q) $m = |E|$

Total running time:

$O(m \log n) \rightarrow O(m + n \log n)$

Heap

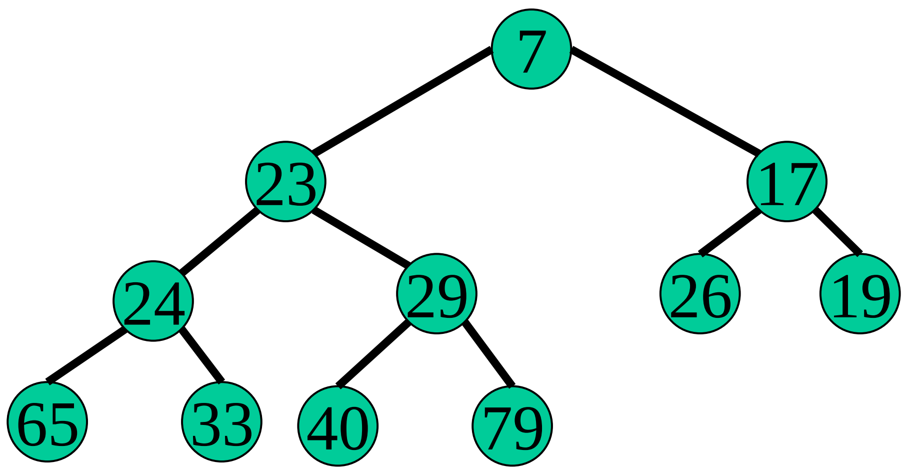
- Heap is
  - An almost complete binary tree
  - The keys at the children of v are greater than the key at v



Heap-ordered tree

Array Representation

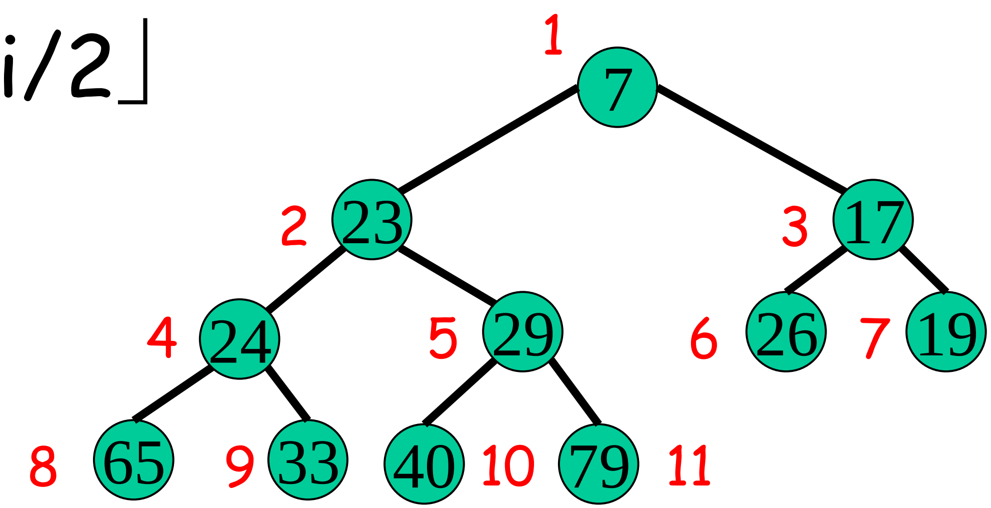
- Representing a heap with an array
  - Get the elements from top to bottom, from left to right



Q	7	23	17	24	29	26	19	65	33	40	79

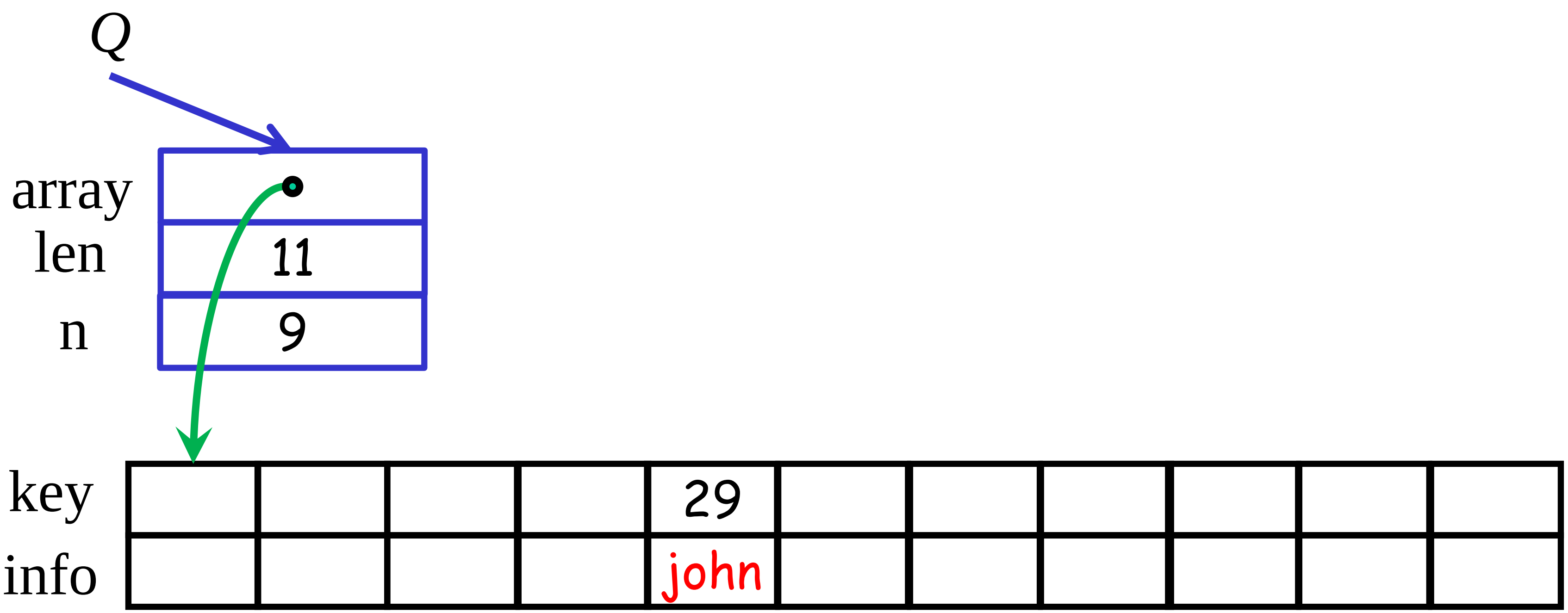
Array Representation

- Left(i): 2i
- Right(i): 2i+1
- Parent(i): $\lfloor i/2 \rfloor$



	1	2	3	4	5	6	7	8	9	10	11
Q	7	23	17	24	29	26	19	65	33	40	79

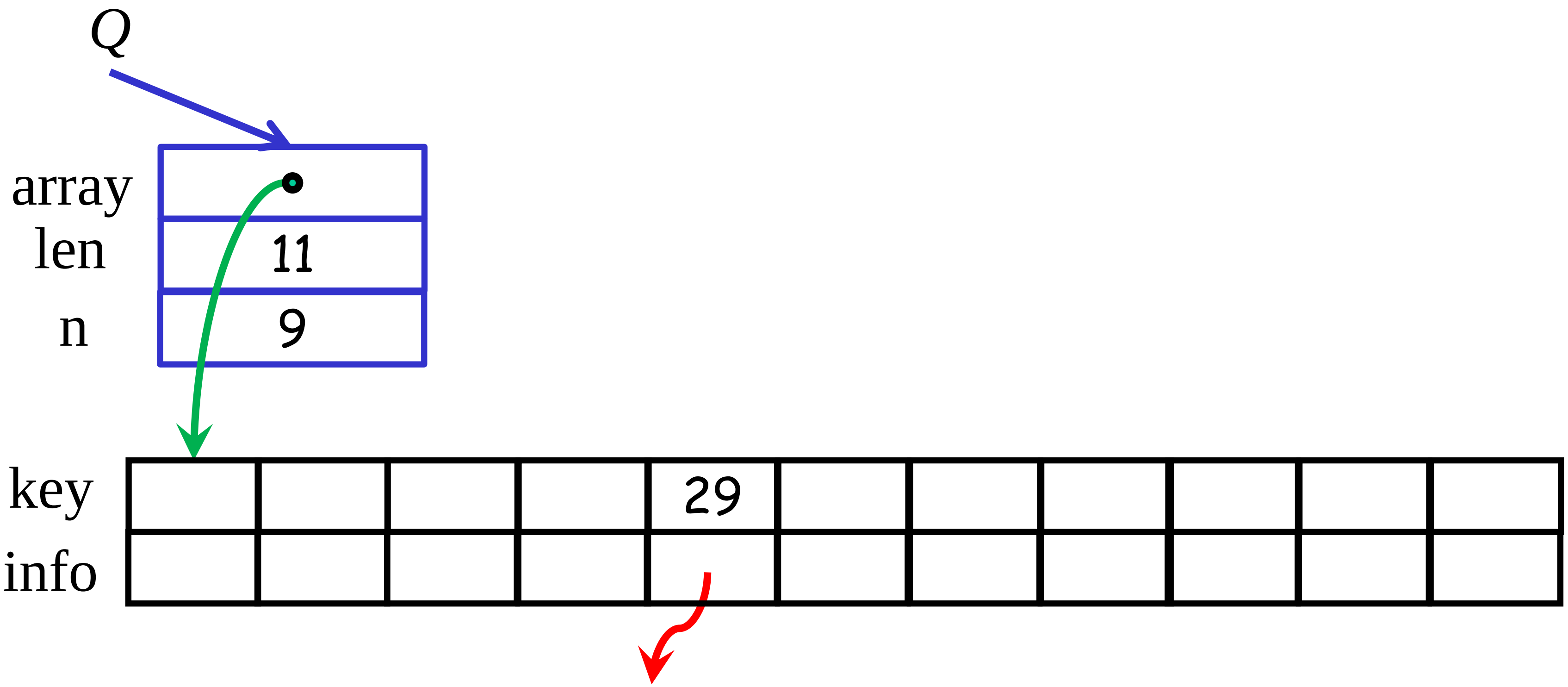
Heap Representation I



Store associated information directly in the array

Should only be used only for succinct information

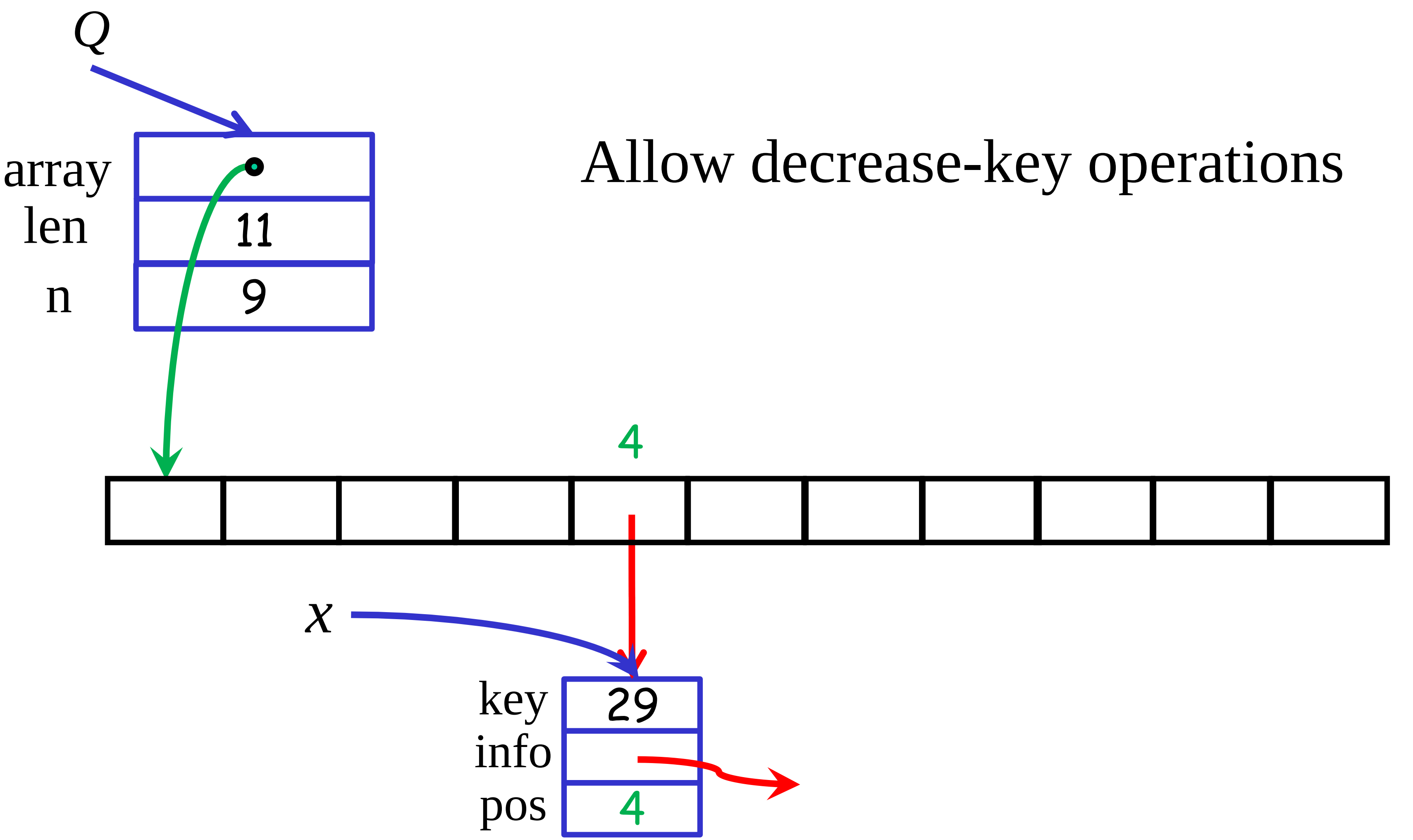
Heap Representation II



Store pointers to associated information

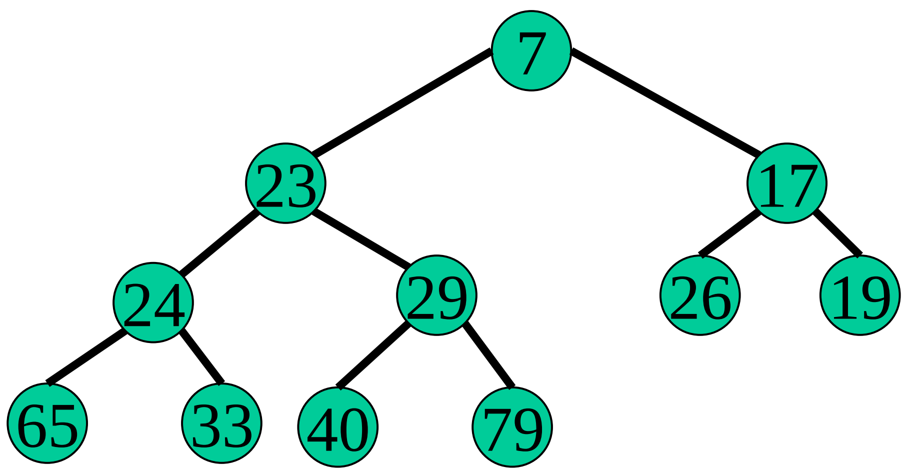
Heap Representation III

Allow decrease-key operations



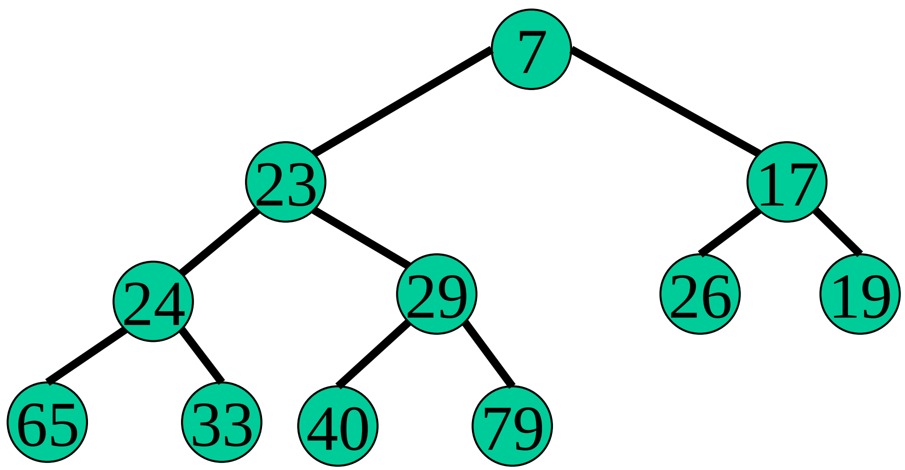
Find the minimum

- Return Q[1]



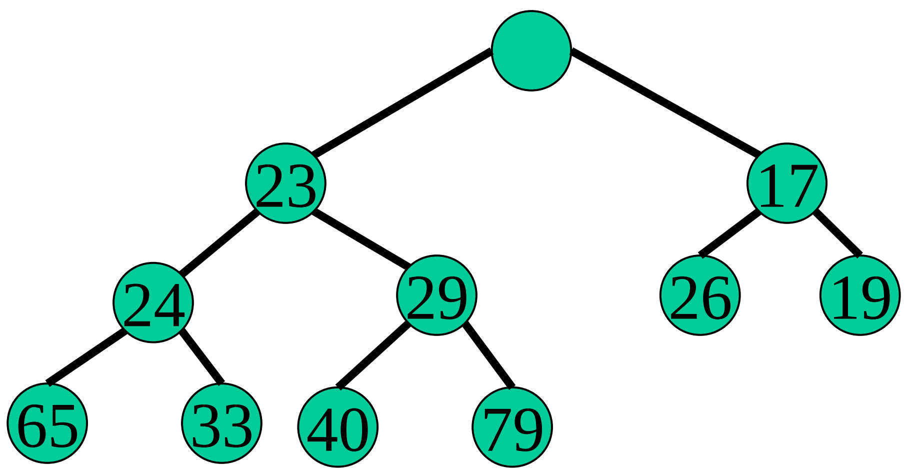
Q	7	23	17	24	29	26	19	65	33	40	79

Delete the minimum



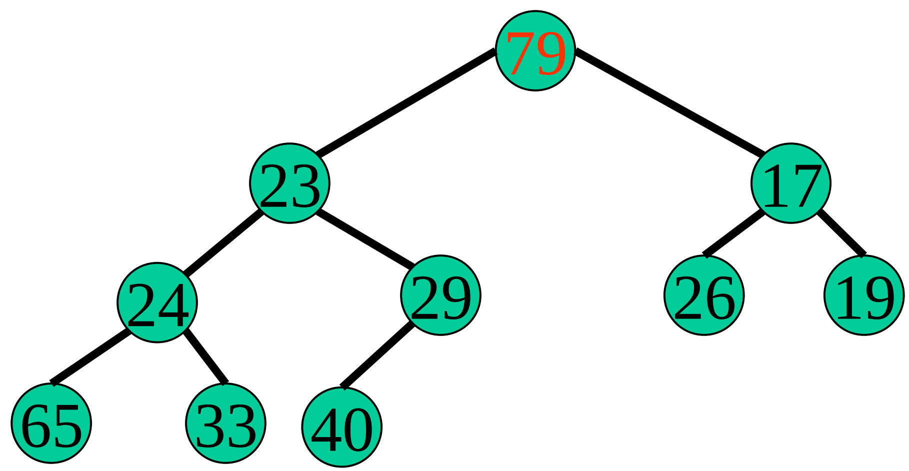
Q										
7	23	17	24	29	26	19	65	33	40	79

Delete the minimum



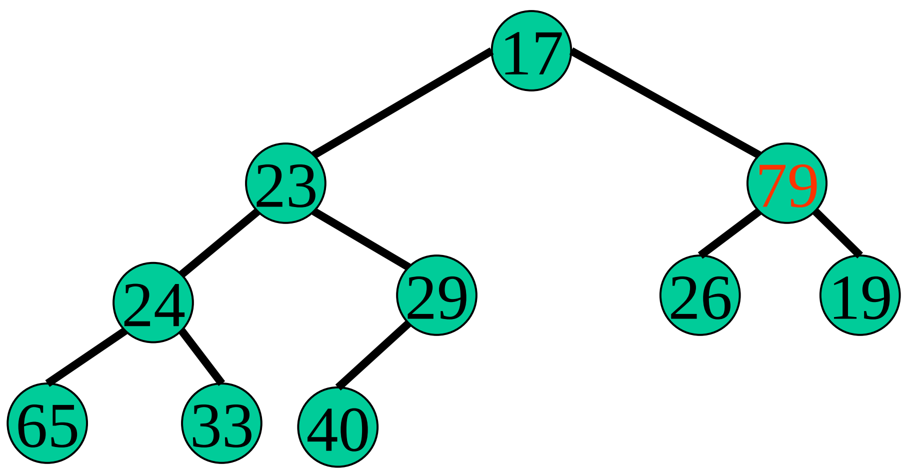
Q											
		23	17	24	29	26	19	65	33	40	79

Delete the minimum



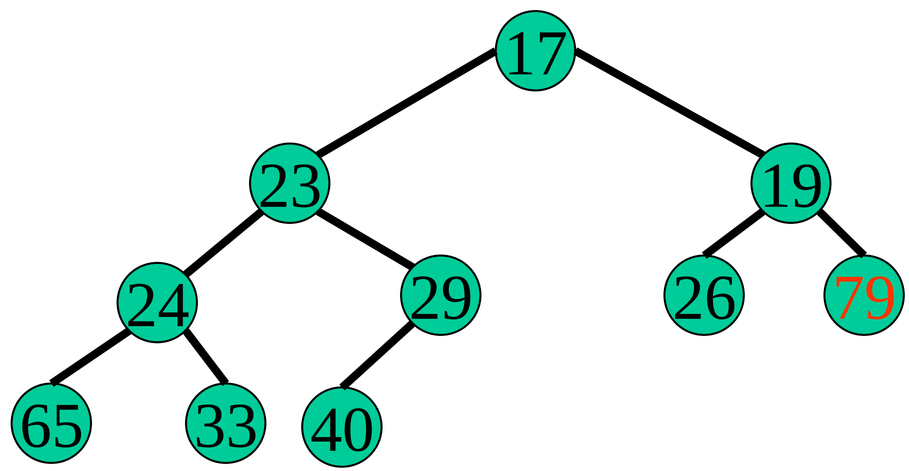
Q	79	23	17	24	29	26	19	65	33	40	

Delete the minimum



Q											
	17	23	79	24	29	26	19	65	33	40	

Delete the minimum



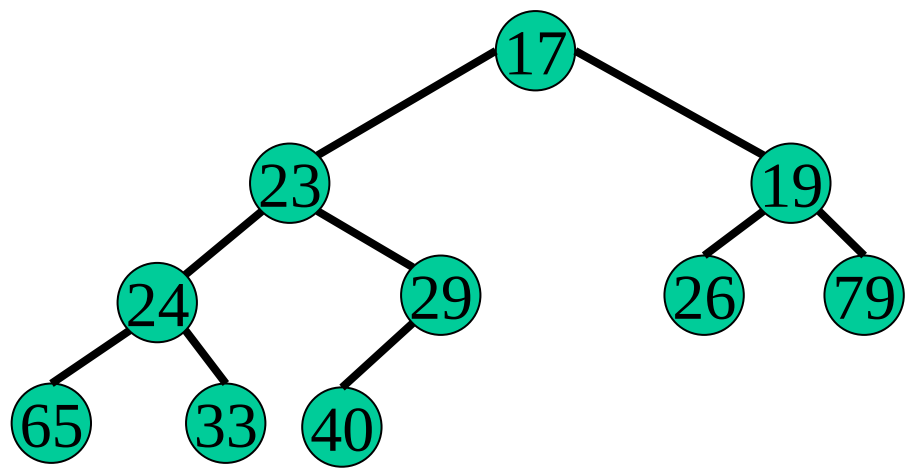
Q											
	17	23	19	24	29	26	79	65	33	40	

Delete the minimum

Q[1] ← Q[size(Q)]

size(Q) ← size(Q)-1

Heapify-down(Q,1)



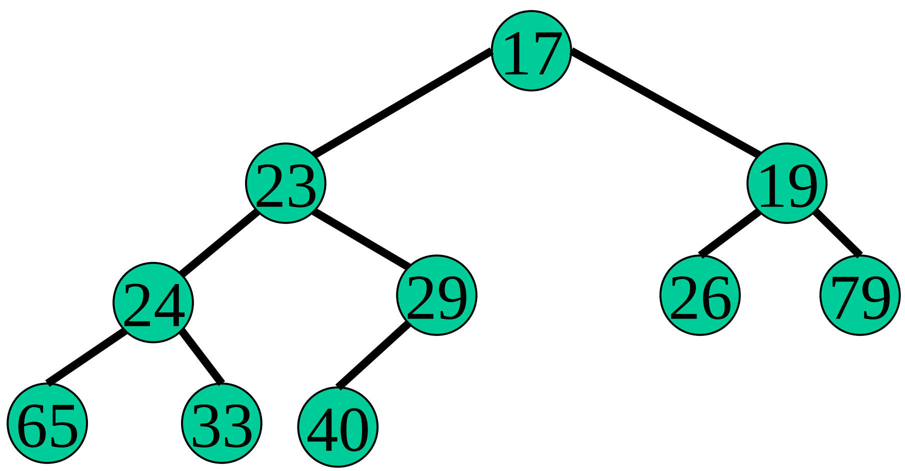
Q											
	17	23	19	24	29	26	79	65	33	40	

Heapify-down(Q,i)

- **Heapify-down(*Q*, *i*)**
- l ← left(i)
- r ← right(i)
- smallest ← i
- **if** l < size(Q) and Q[l] < Q[smallest]
  then smallest ← l
- **if** r < size(Q) and Q[r] < Q[smallest]
  then smallest ← r
- **if** smallest > i **then**
  Q[i] ↔ Q[smallest]
  Heapify-down(Q, smallest)

Inserting an item

Insert(15,Q)



Q	17	23	19	24	29	26	79	65	33	40	

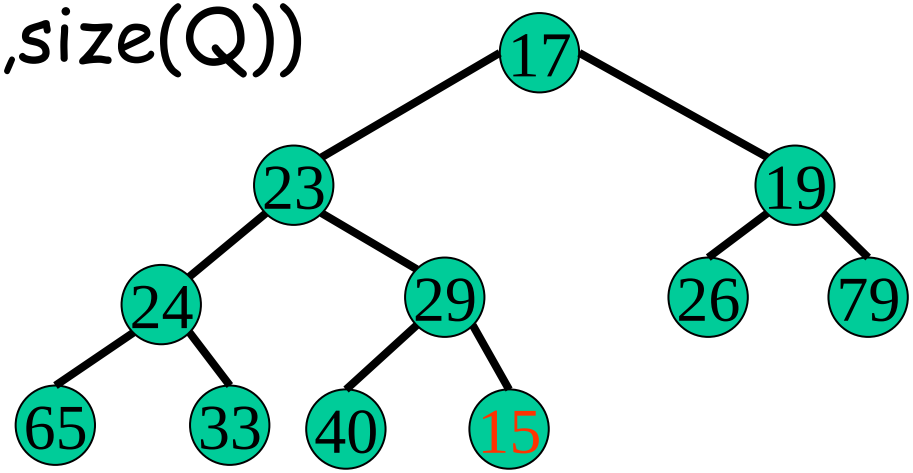
Inserting an item

Insert(k,Q) :

size(Q) ← size(Q) + 1

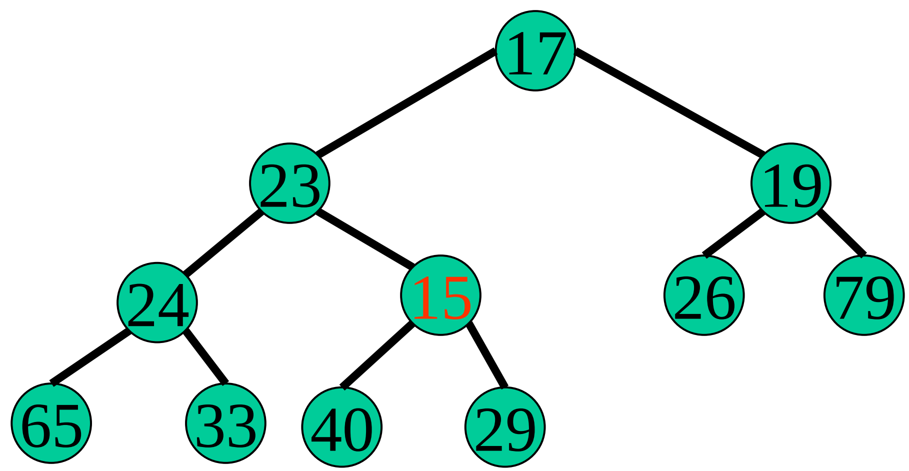
Q[size(Q)] ← k

Heapify-up(Q,size(Q))



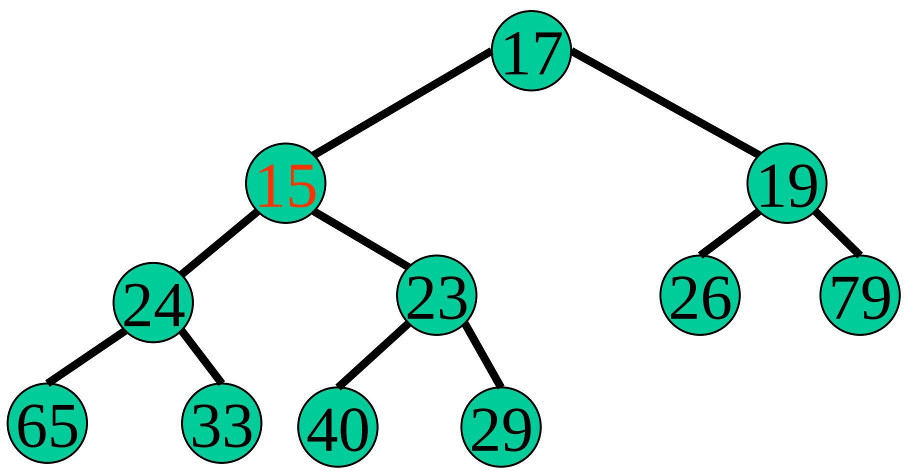
Q											
	17	23	19	24	29	26	79	65	33	40	15

Heapify-up



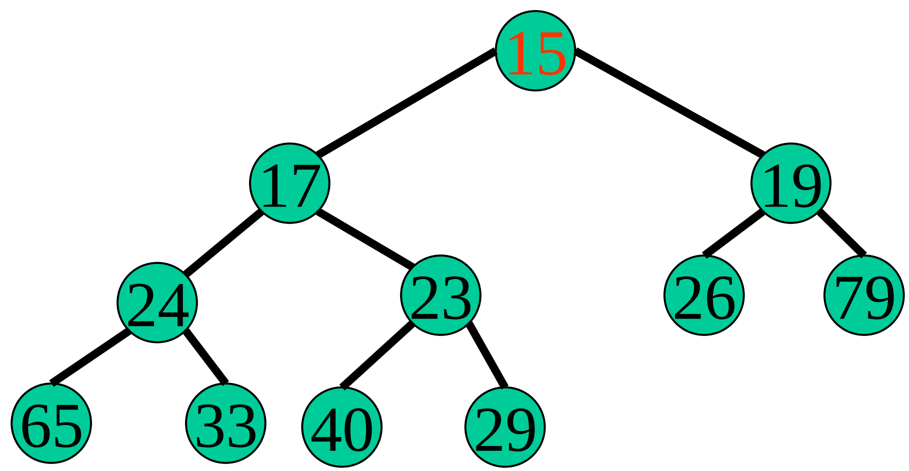
Q	17	23	19	24	15	26	79	65	33	40	29

Heapify-up



Q											
Q	17	15	19	24	23	26	79	65	33	40	29

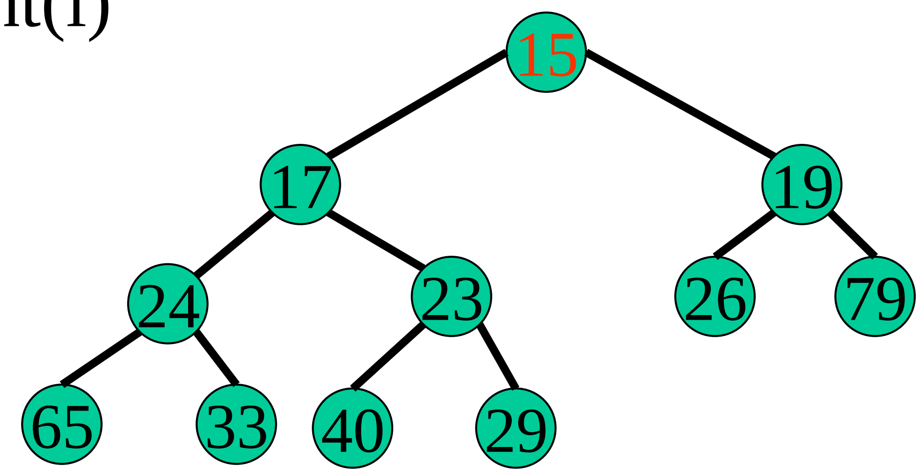
Heapify-up



Q	15	17	19	24	23	26	79	65	33	40	29

Heapify-up

- **Heapify-up(Q, i)**
- **while** $i > 1$ and $Q[i] < Q[parent(i)]$ **do**
- $Q[i] \leftrightarrow Q[parent(i)]$
- $i \leftarrow parent(i)$



Q											
Q	15	17	19	24	23	26	79	65	33	40	29

Other operations

- Decrease-key(x,Q,Δ)
- Delete(x,Q)

- Can implement them easily using heapify

Binary heaps - Summary

Binary heaps perform
all heap operations in $O(\log n)$ time

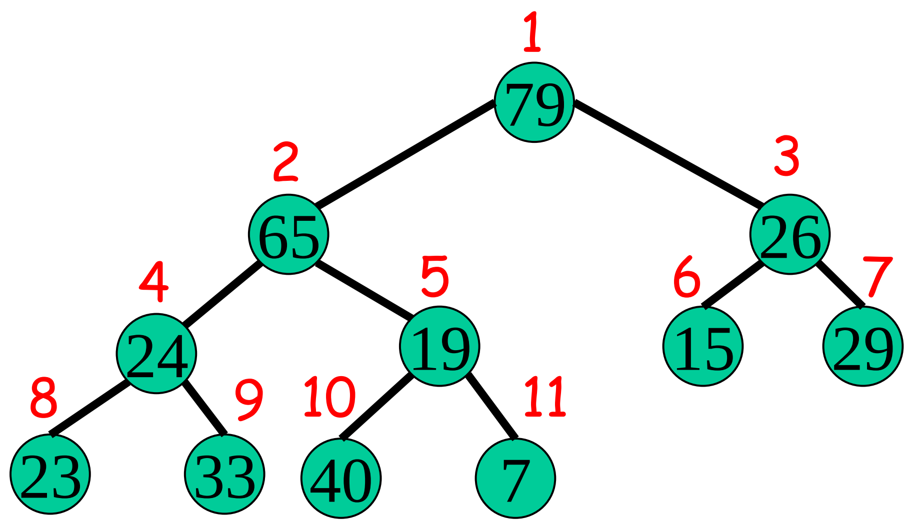
Binary heaps are
very efficient in practice

They do not achieved our goal of
implementing decrease-key in $O(1)$ time

Heapsort (Williams, Floyd, 1964)

- Put the elements in an array
- Turn the array into a heap
- Do a delete-min and put the deleted element at the last position of the array
- Reverse the array, or use a max-heap

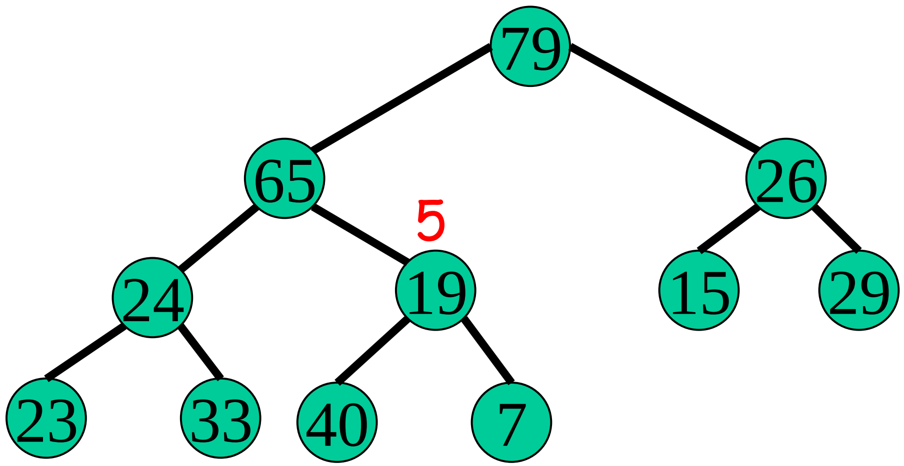
Turn an array into a heap



	1	2	3	4	5	6	7	8	9	10	11
Q	79	65	26	24	19	15	29	23	33	40	7

Turn an array into a heap

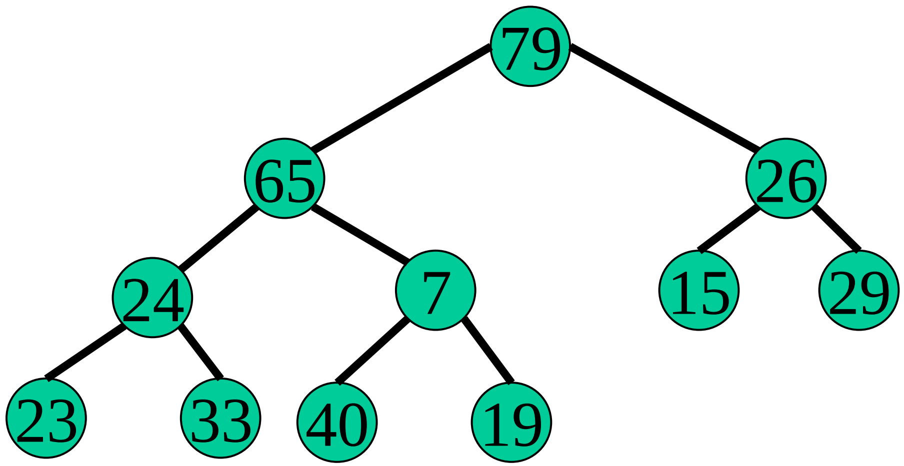
Heapify-down(Q,5)



	1	2	3	4	5	6	7	8	9	10	11
Q	79	65	26	24	19	15	29	23	33	40	7

Turn an array into a heap

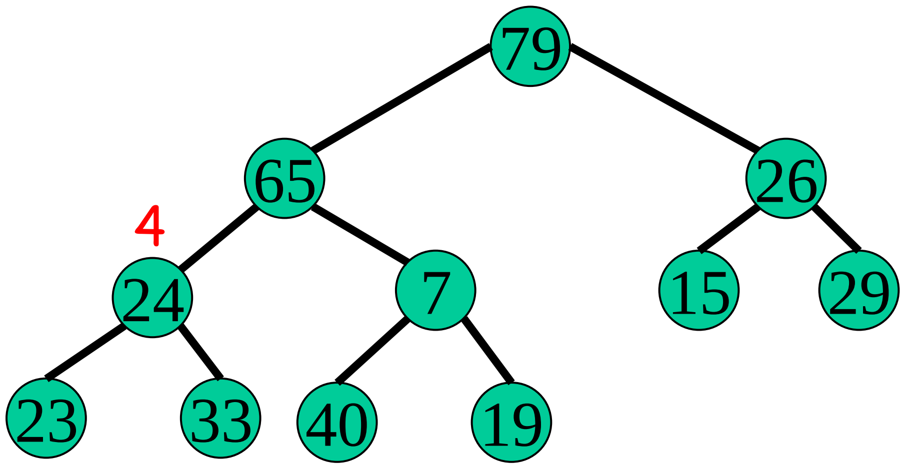
Heapify-down(Q,5)



	1	2	3	4	5	6	7	8	9	10	11
Q	79	65	26	24	7	15	29	23	33	40	19

Turn an array into a heap

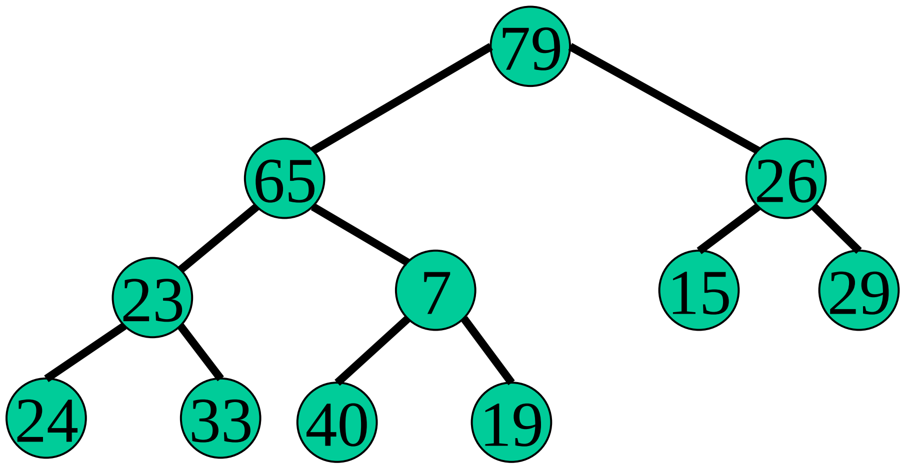
Heapify-down(Q,4)



	1	2	3	4	5	6	7	8	9	10	11
Q	79	65	26	24	7	15	29	23	33	40	19

Turn an array into a heap

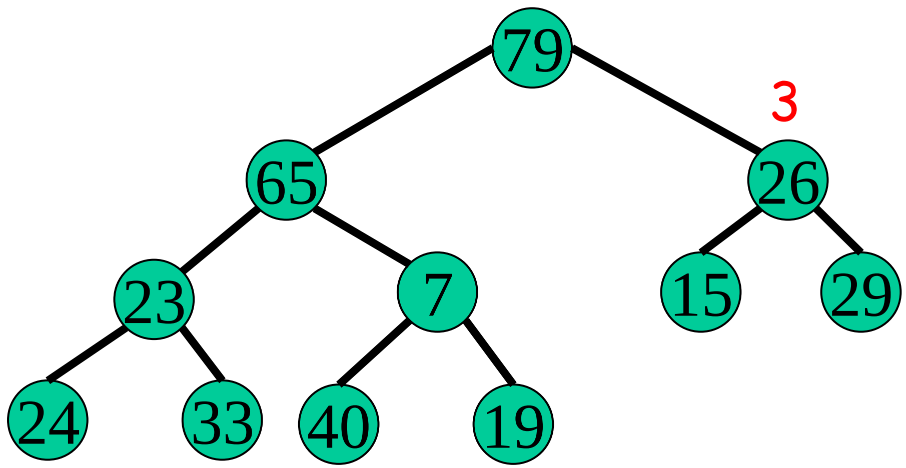
Heapify-down(Q,4)



	1	2	3	4	5	6	7	8	9	10	11
Q	79	65	26	23	7	15	29	24	33	40	19

Turn an array into a heap

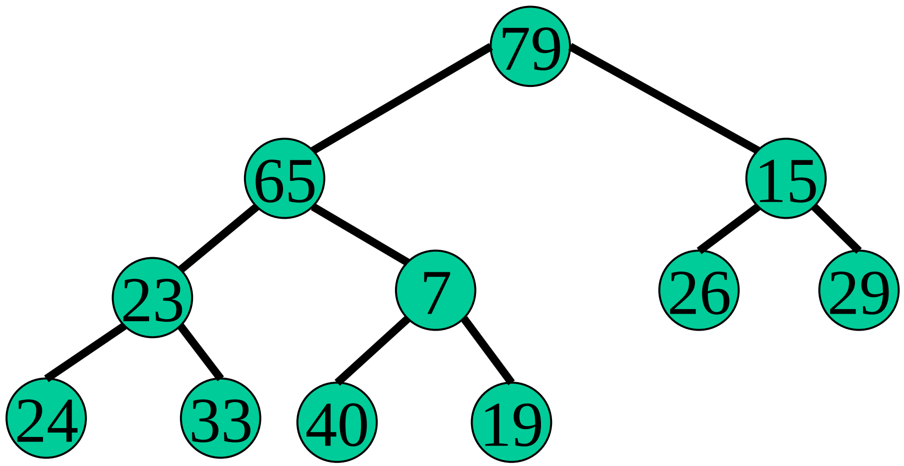
Heapify-down(Q,3)



	1	2	3	4	5	6	7	8	9	10	11
Q	79	65	26	23	7	15	29	24	33	40	19

Turn an array into a heap

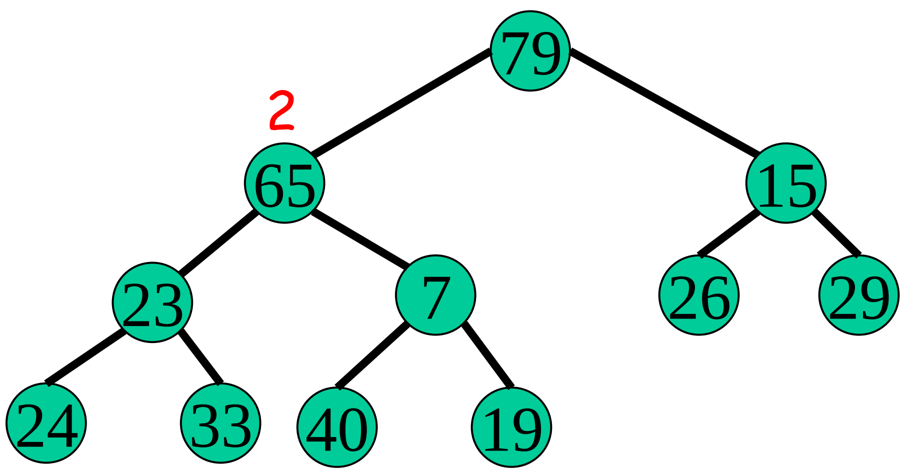
Heapify-down(Q,3)



	1	2	3	4	5	6	7	8	9	10	11
Q	79	65	15	23	7	26	29	24	33	40	19

Turn an array into a heap

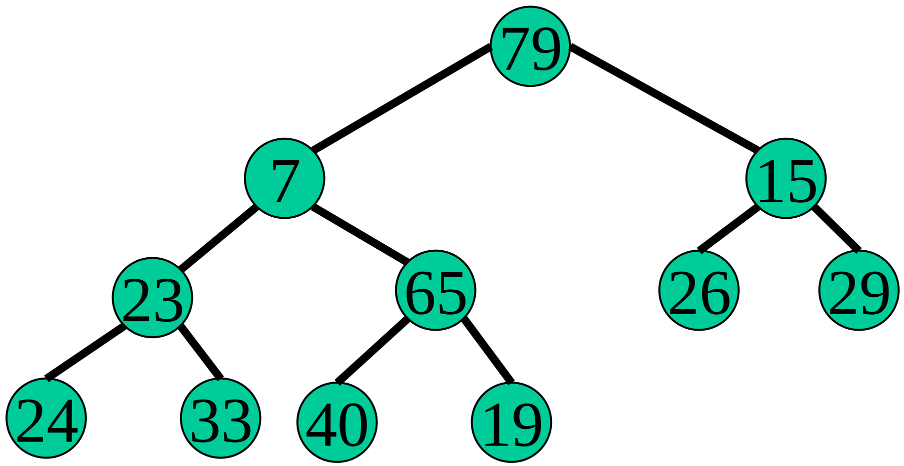
Heapify-down(Q,2)



	1	2	3	4	5	6	7	8	9	10	11
Q	79	65	15	23	7	26	29	24	33	40	19

Turn an array into a heap

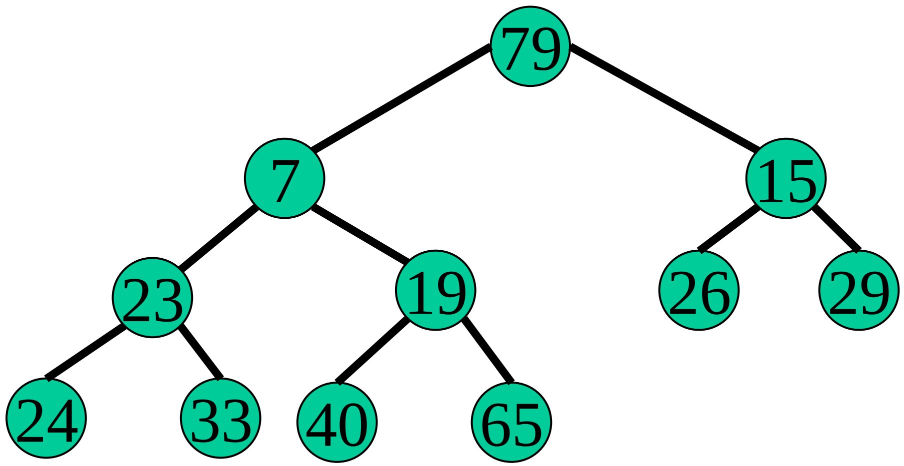
Heapify-down(Q,2)



	1	2	3	4	5	6	7	8	9	10	11
Q	79	7	15	23	65	26	29	24	33	40	19

Turn an array into a heap

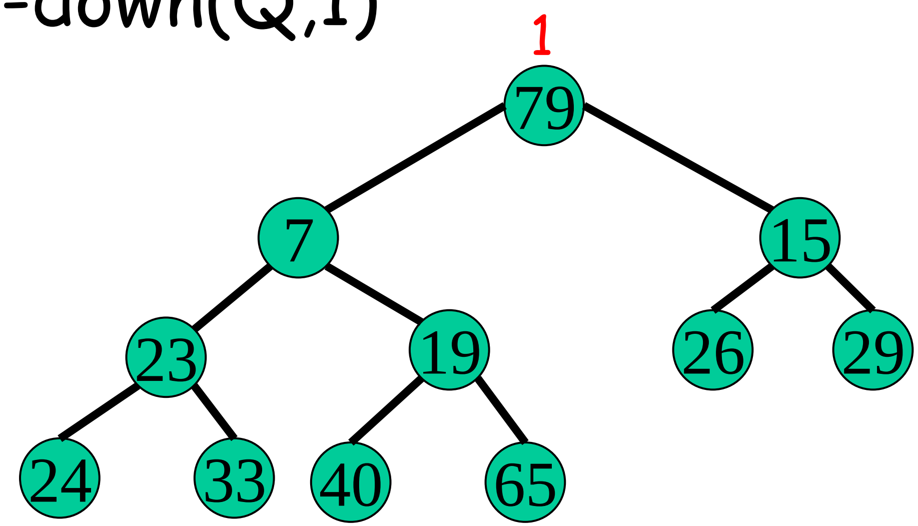
Heapify-down(Q,2)



	1	2	3	4	5	6	7	8	9	10	11
Q	79	7	15	23	19	26	29	24	33	40	65

Turn an array into a heap

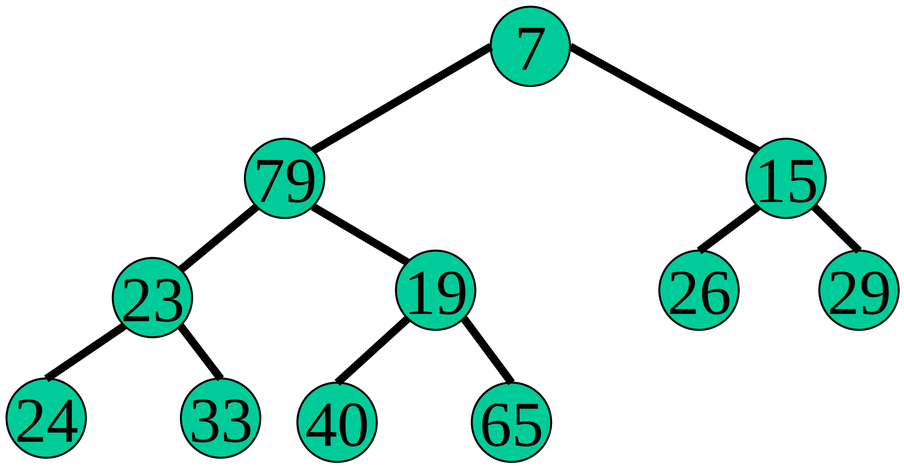
Heapify-down(Q,1)



	1	2	3	4	5	6	7	8	9	10	11
Q	79	7	15	23	19	26	29	24	33	40	65

Turn an array into a heap

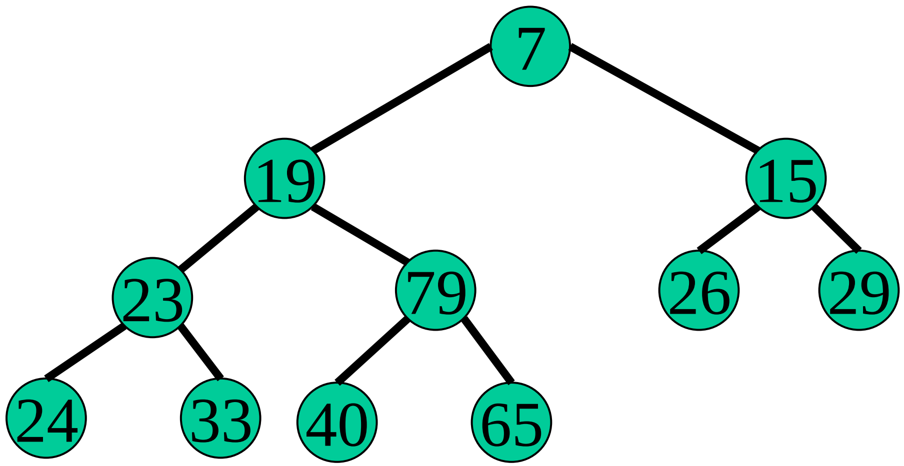
Heapify-down(Q,1)



	1	2	3	4	5	6	7	8	9	10	11
Q	7	79	15	23	19	26	29	24	33	40	65

Turn an array into a heap

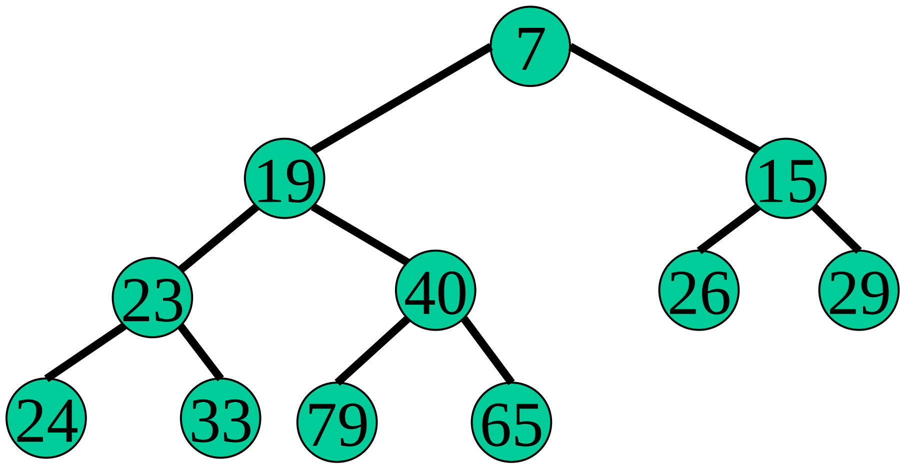
Heapify-down(Q,1)



	1	2	3	4	5	6	7	8	9	10	11
Q	7	19	15	23	79	26	29	24	33	40	65

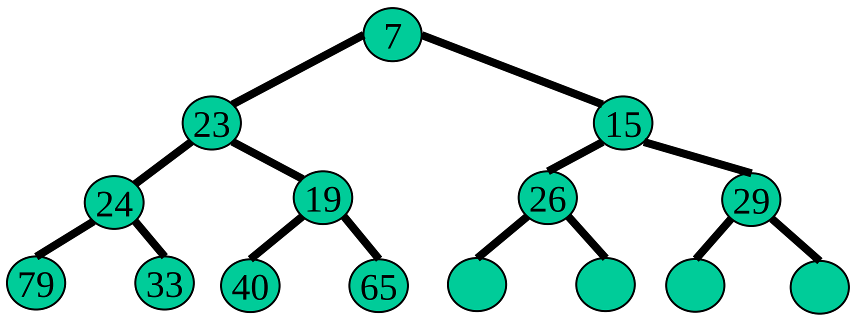
Turn an array into a heap

Heapify-down(Q,1)



	1	2	3	4	5	6	7	8	9	10	11
Q	7	19	15	23	40	26	29	24	33	79	65

How much time does it take to build the heap this way ?



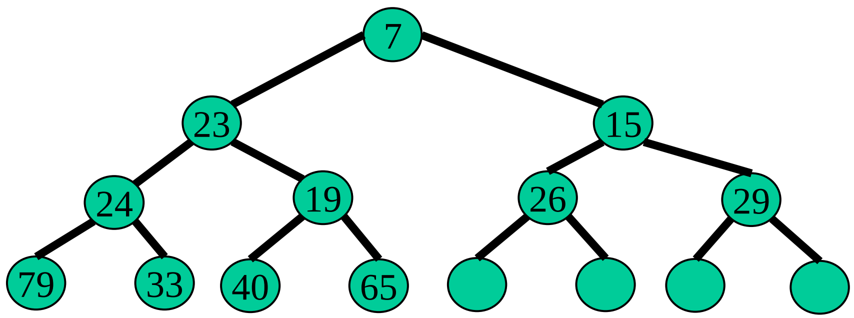
At most n/2 nodes heapified at height 1

At most n/4 nodes heapified at height 2

At most n/8 nodes heapified at height 3

Total time = $\frac{n}{2} + 2\frac{n}{4} + 3\frac{n}{8} + \ldots + H = \sum_{h=1}^{H} h\frac{n}{2^h}$

How much time does it take to build the heap this way ?



At most n/2 nodes heapified at height 1

At most n/4 nodes heapified at height 2

At most n/8 nodes heapified at height 3

Total time = $\displaystyle\sum_{h=1}^{H} h \frac{n}{2^h} < n \sum_{h=1}^{\infty} \frac{h}{2^h} = O(n)$

Heapsort

- We can build the heap in linear time
- We still have to delete-min the elements one by one in order to sort. That will take O(n log n)

An example

- Given an array of length n, find the k smallest numbers.