CS2040C Data Structures and Algorithms

Priority Queue ADT

Outline

- What is a priority queue?
- What are the operations supported?
- heap (max-heap, min-heap)
  - heapInsert (O(log N))
  - heapDelete (O(log N))
  - heapRebuild (O(log N))
  - heapify (O(N))
- heapSort (O(N log N))
- STL priority queue

What is a Priority Queue?

A special form of queue from which items are removed according to their designated priority and not the order in which they entered

Examples

- A “to-do” list with priorities
- Scheduling jobs in OS
- Queue at A&E of the hospital

- Go to Takashimaya (1)
- Play tennis (5)
- Prepare CS2040C lecture slides (6)
- Go to department tea (3)
- …

Priority Queue (Example)

Job requests to programmers are:

job 1: clerk

job 2: lecturer

job 3: head

job 4: dean

Items entered into the queue in sequential order but will be removed in the order #4, #3, #2, #1.

Priority queue operations

- Create an empty priority queue
- Insert an item with a given key
- Remove the item with maximum key
- Determine whether a priority queue is empty

Unsorted list implementation

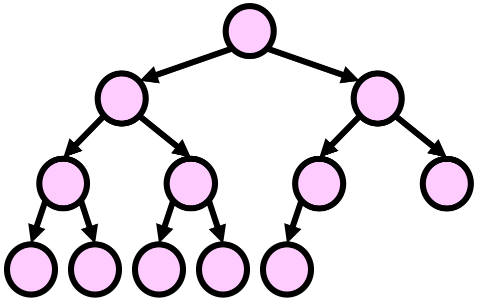
- **Insertion**: add the element to end of a list (O(1))
- **Deletion**: traverse the list to find the element of maximum key and remove it (O(n))

What are the running times if we use **sorted** list?

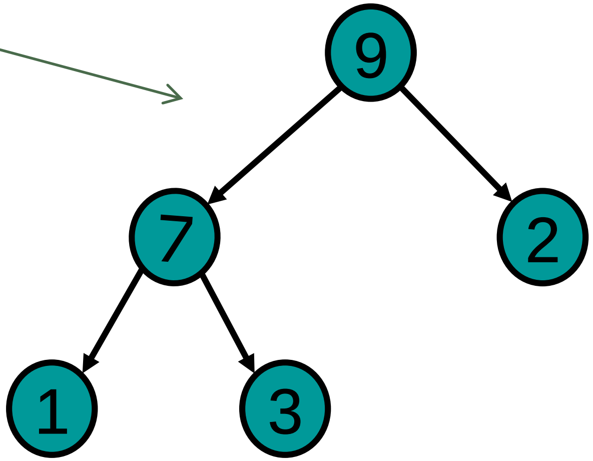
Heap

Heap is the most appropriate data structure for realizing the priority queue ADT

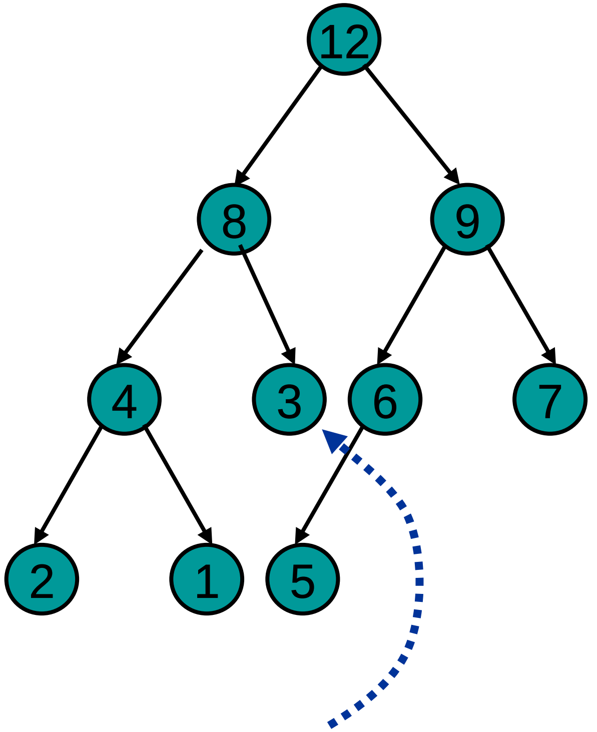
Definition



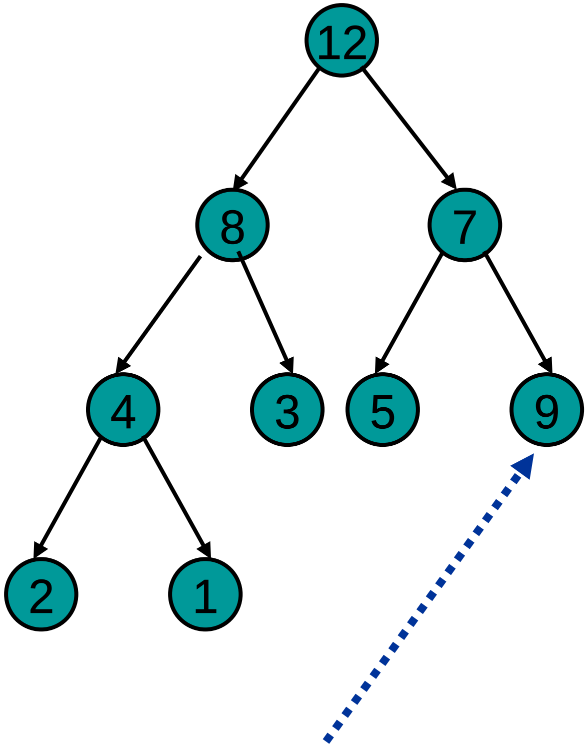
- A (binary) **heap** is a complete binary tree that satisfies the **heap property**:
  - for every node *v*, the search key in *v* is greater than or equal to those in the children of *v.*
- Also called *max-heap*
- If the search key in *v* is less than or equal to those in the children of *v*, it is called a *min-heap*



Negative examples

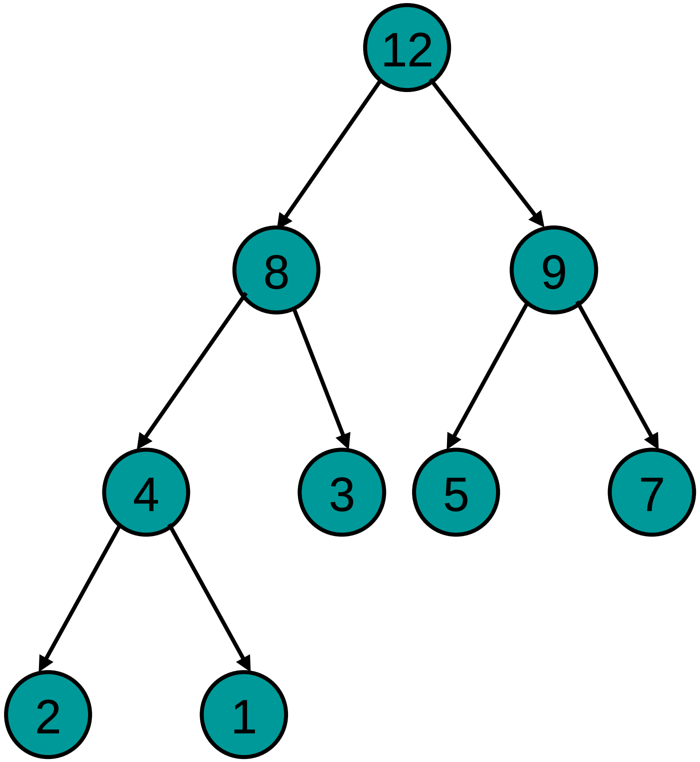


not complete



fail heap property

Representation using arrays



0	12
1	8
2	9
3	4
4	3
5	5
6	7
7	2
8	1

left(i) = 2*i+1

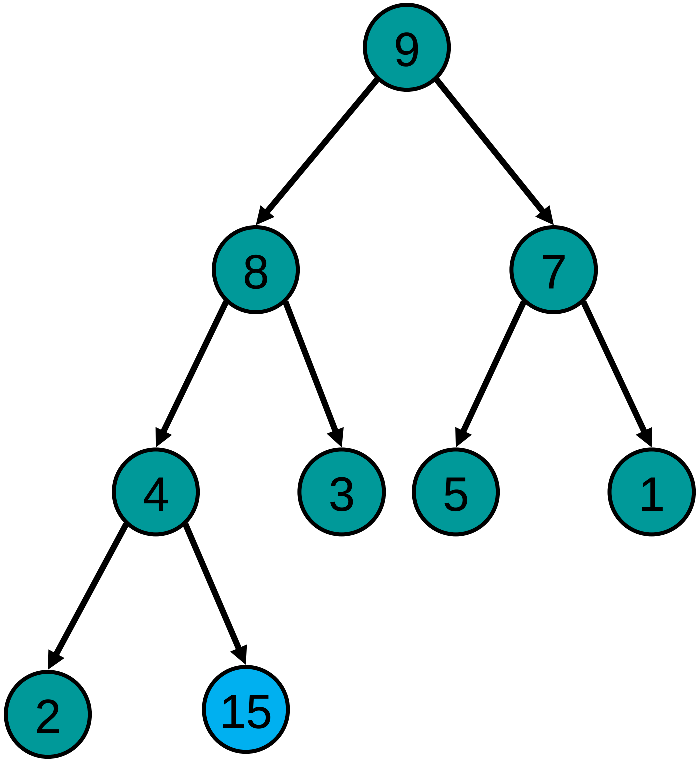
right(i) = 2*i+2

parent(i) =

floor((i-1)/2)

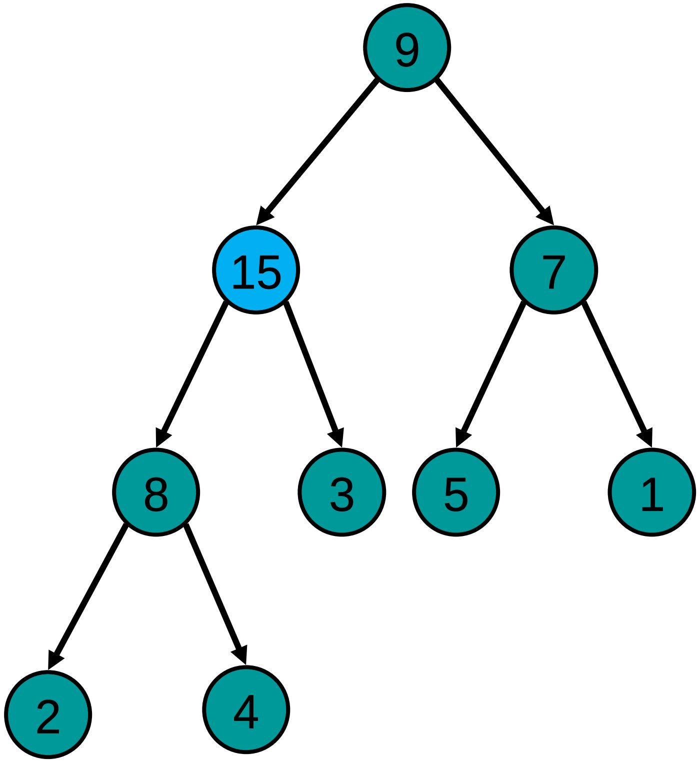
(i > 0)

Insert an item

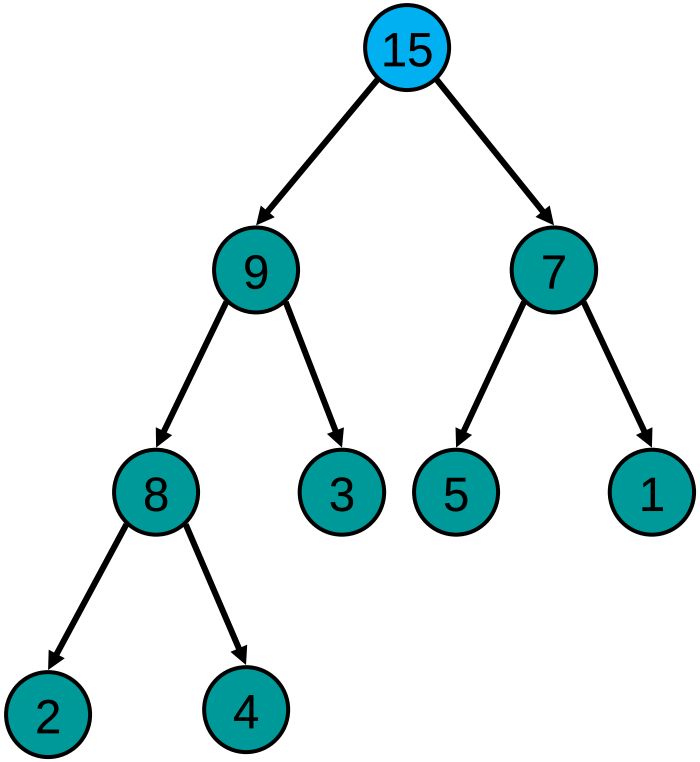


0	9
1	8
2	7
3	4
4	3
5	5
6	1
7	2
8	15
9	

Re-establish heap property

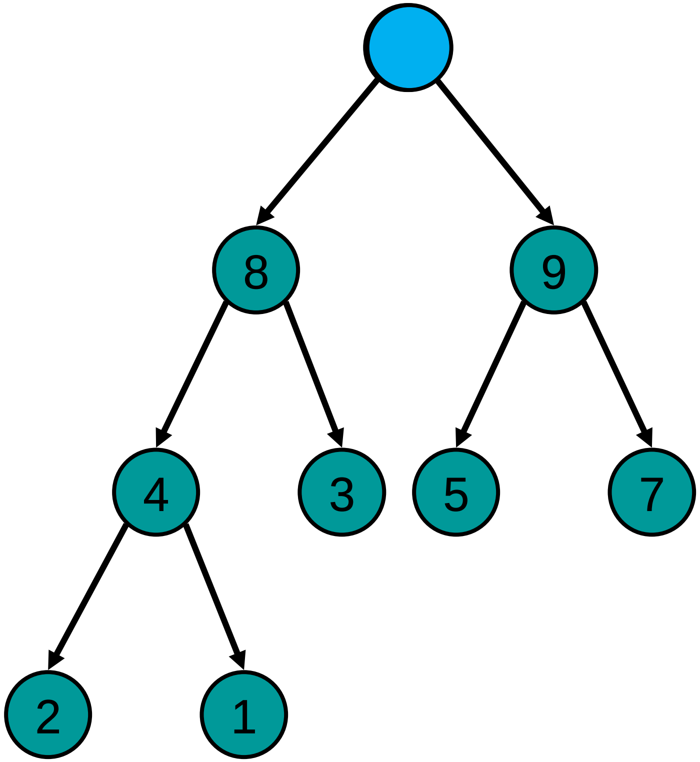


Re-establish heap property

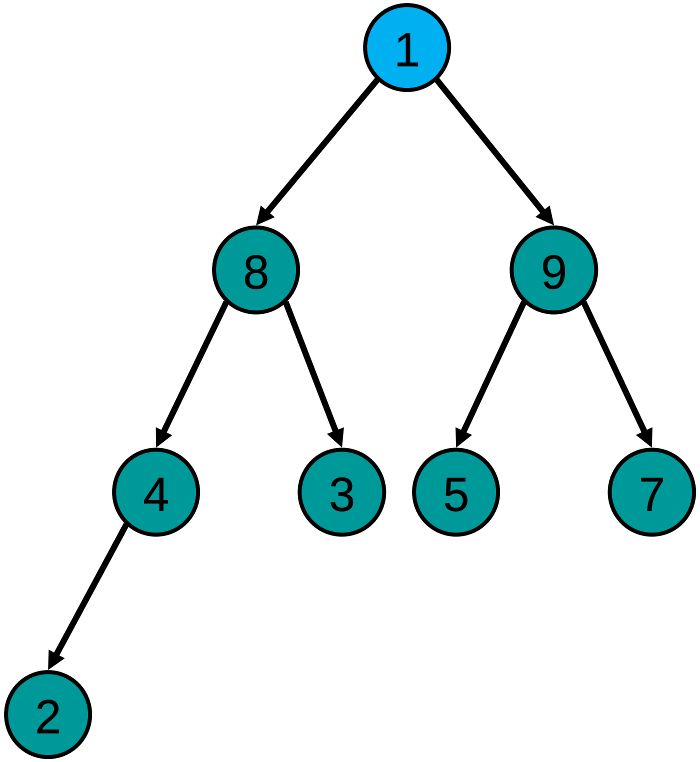


bubble up

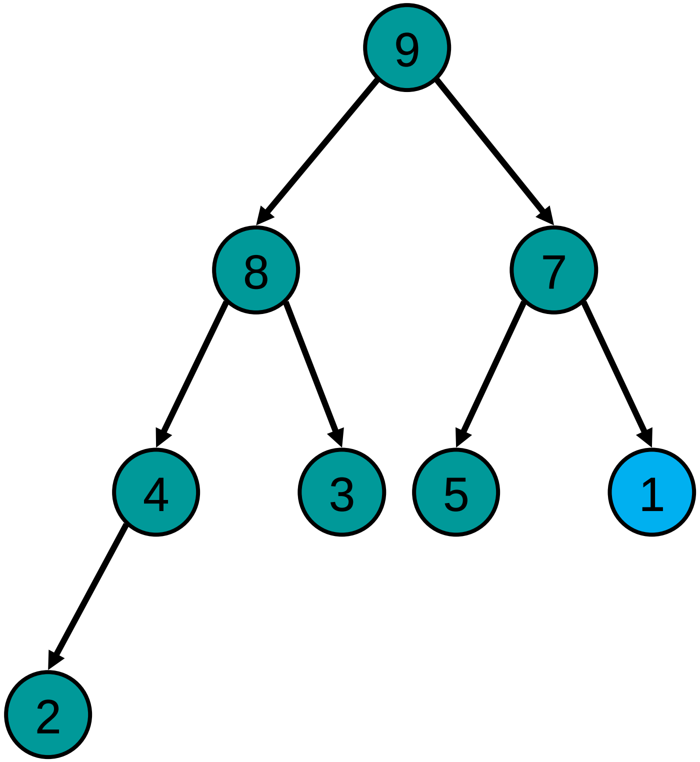
Remove the max item



Re-establish heap property



Re-establish heap property



bubble down

heapInsert (bubble up)

```
void heapInsert(int newItem) throw (HeapException) {
      if (size < MAX_HEAP) {
        items[size] = newItem;
        int place = size;
        int parent = (place - 1)/2;
        while ( (parent >= 0) && (items[place] > items[parent]) {
            int temp = items[place];          // swap
            items[place] = items[parent];     //
            items[parent] = temp;             //
            place = parent;
            parent = (place - 1)/2;
        }
        ++size;
      }
      else                 // Should extend items[] dynamically
        throw HeapException("HeapException: Heap full");
}
```

heapDelete

```
int heapDelete() throw (HeapException) {
   if (!heapIsEmpty()) {
      int rootItem = items[0];
      items[0] = items[--size];
      heapRebuild(0);
      return rootItem;
   }
   else
       throw HeapException ("HeapException: Heap empty");
}
```

heapRebuild (bubble down)

```
void heapRebuild(int root) {
        int child = 2 * root + 1;                  // left child
        if (child < size) {                        // there is a left child
            int rightChild = child + 1;            // right child
            if ( (rightChild < size) &&            // there is a right child
                 (items[rightChild] > items[child]) )
                child = rightChild;                // choose bigger child
            if ( items[root] < items[child] ) {    // trickle down
                int temp = items[root];            // swap
                items[root] = items[child];        //
                items[child] = temp;               //
                heapRebuild(child);
            }
        }
}
```

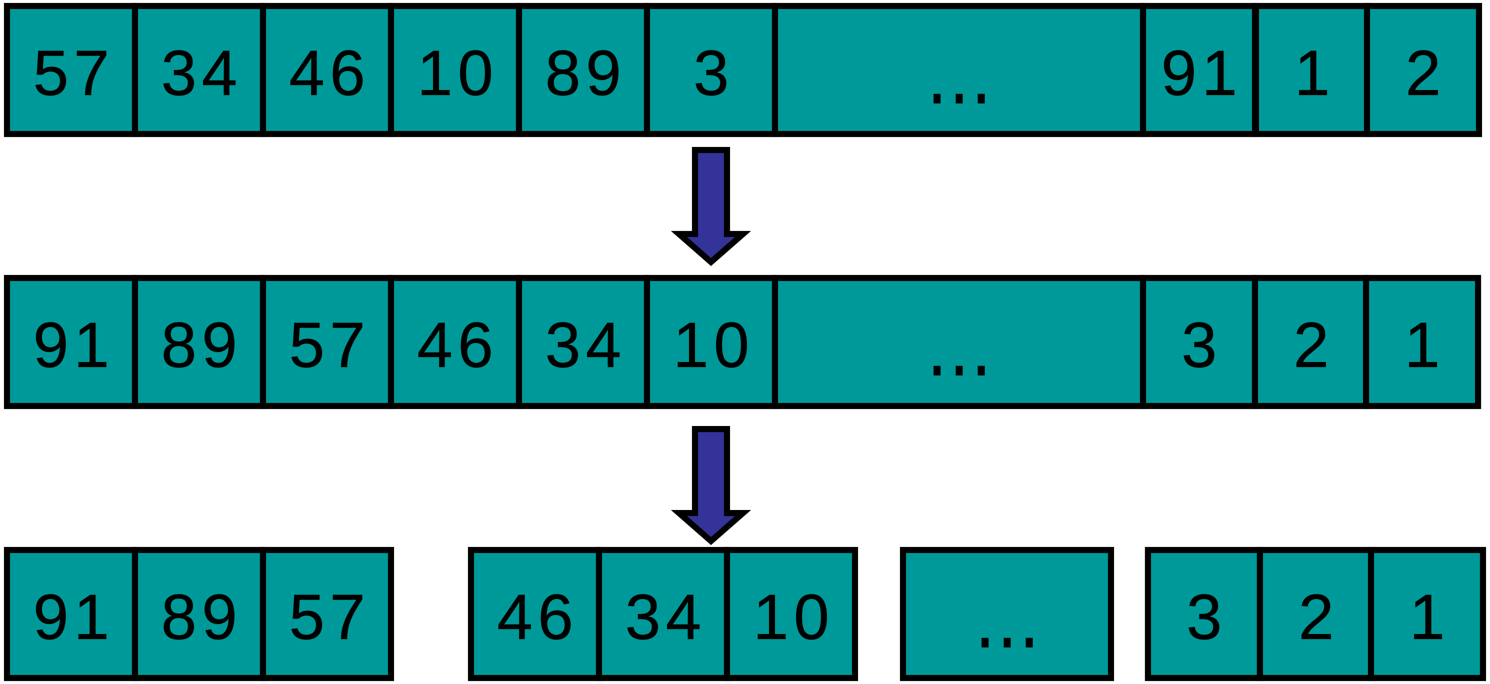
Running time

- How many calls to **heapRebuild**?
- Go down 1 level after each call to **heapRebuild**
- number of calls = height of tree
- Worst case running time:
  $O(h) = O(\log n)$

Application: Displaying ranked web pages

- Suppose we are searching for something and the searched pages are to be displayed
- The search results are to be displayed 10 pages at a time in order of decreasing page rank scores

Solution: Sort the pages by score



57 | 34 | 46 | 10 | 89 | 3 | ... | 91 | 1 | 2

91 | 89 | 57 | 46 | 34 | 10 | ... | 3 | 2 | 1

91 | 89 | 57   46 | 34 | 10   ...   3 | 2 | 1

- Sort the web pages according to their scores
- Traverse the sorted list (10 at a time)

Running times

- Sorting $O(n \log n)$
  n: total number of pages
- Traversing $O(k)$
  k: number of pages requested
- Total running time

$$O(n \log n) + O(k)$$

$\le O(n \log n) + O(n)$, since $k<n$

$= O(n \log n)$

Can we do better?

Maybe

Idea

- Build a heap of scores
- Remove the top 10 pages at a time

Is it more efficient?

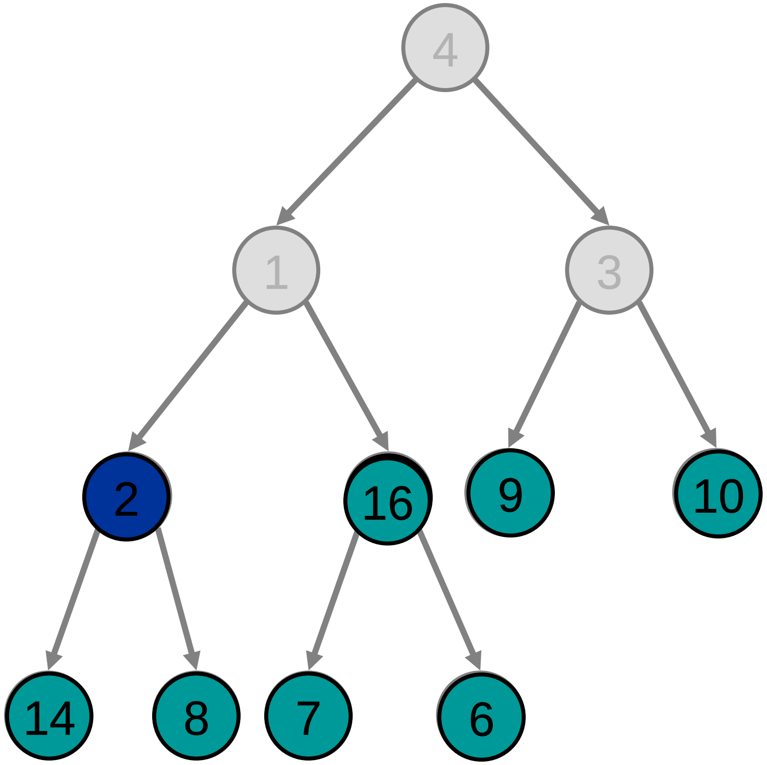
Heap Construction

Heap construction

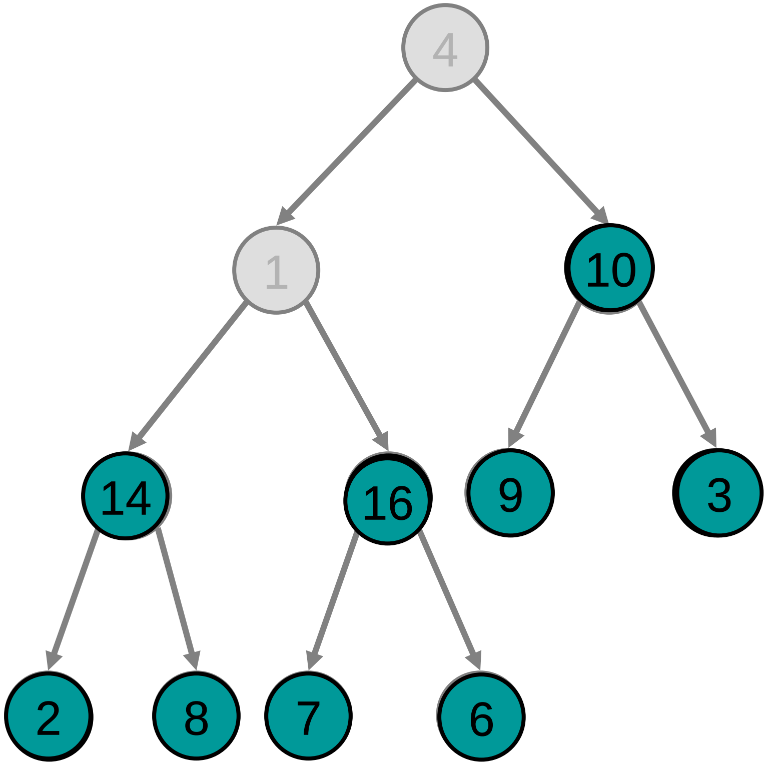
- **heap property**: for every node *v*, the search key in *v* is greater than those in the children of *v*
- We can build the heap by inserting nodes one at a time into the heap
- Better to build the heap recursively from bottom up *(on the existing array representation)*
- We start by building a heap of one node from the leaves. Then we increase the height of the heap. At every step, we check if the heap property is maintained. If not, we just bubble down.

Heap construction example

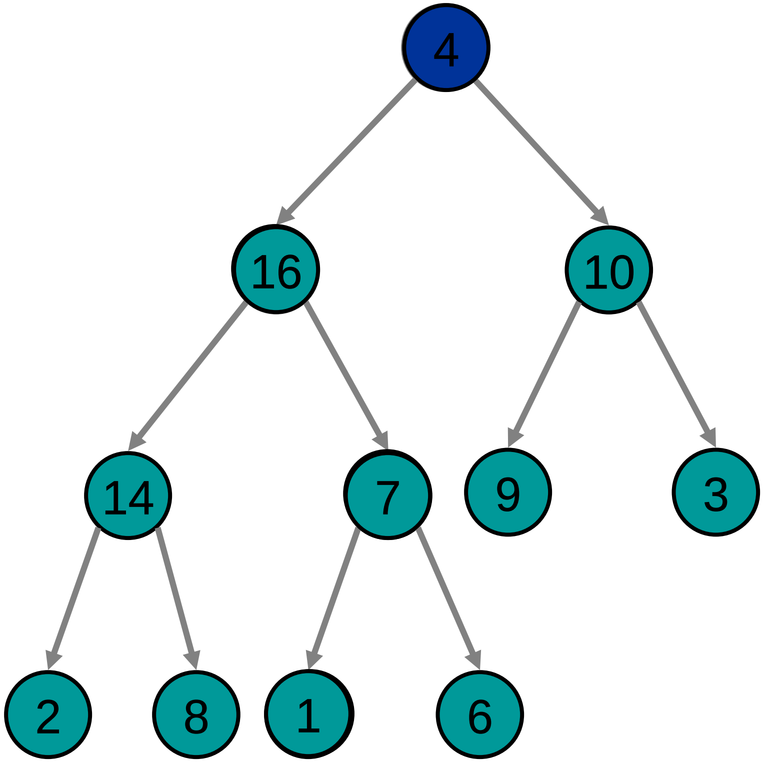
0	4
1	1
2	3
3	2
4	16
5	9
6	10
7	14
8	8
9	7
10	6



Heap construction example

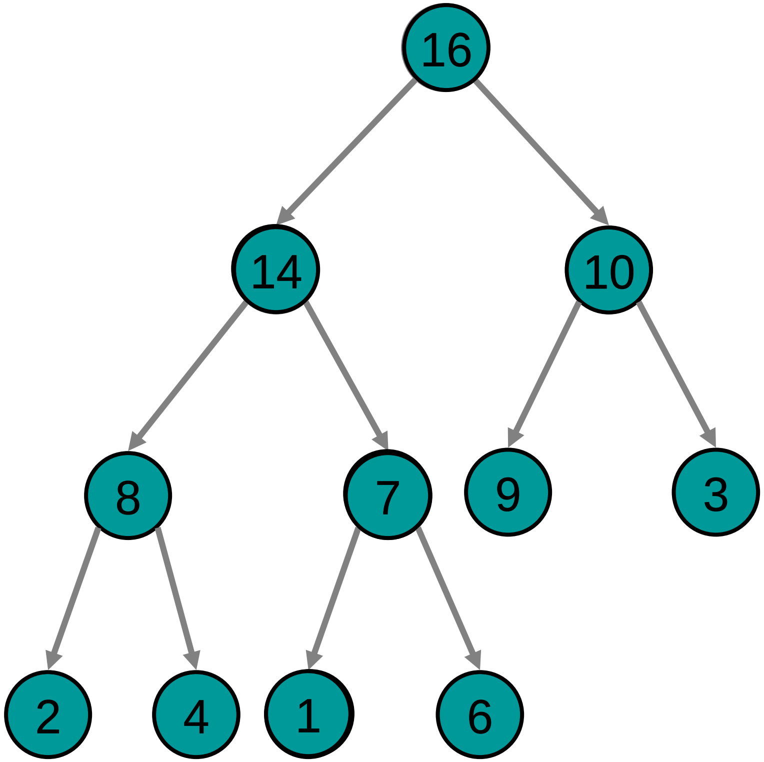


Heap construction example



Heap construction example

0	16
1	14
2	10
3	8
4	7
5	9
6	3
7	2
8	4
9	1
10	6



Code

```
void heapify() {
    for (int i = size/2; i >= 0; i--)
            heapRebuild(i);
}
```

Note: i can start from the last internal node at size/2 - 1

Running time

Rough count: $$n/_2 \cdot O(\log n) = O(n \log n)$$

More accurate count of no. of calls to **heapRebuild**, level by level from bottom up:

level	No. of calls	Each call requires
2	$n/2^2$	O(2)
3	$n/2^3$	O(3)
4	$n/2^4$	O(4)

...

Total complexity:

$2 \times n/2^2 + 3 \times n/2^3 + 4 \times n/2^4 + \ldots$

$< n(2/2^2 + 3/2^3 + 4/2^4 + \ldots)$

$< n(3/2) =$ O(n)

Web page ranking revisited

- Build a heap *O(n)*
- Retrieve top k pages *O(k log n)*
- Total running time
  O(n) + O(k log n)
  - If k=n, then O(n log n)
  - If k=20, O(n) + O(20 log n) = O(n)

Heapsort

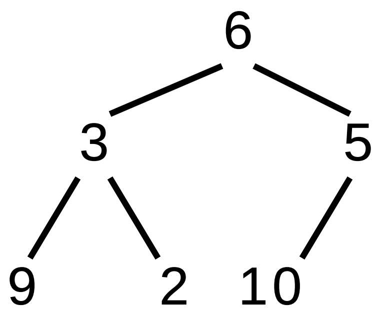
Uses a heap to sort an array a[0..n] of items

- Transform the array into a heap
- Execute n steps to turn the heap into a sorted array:
    - In step k, k =1..n:
        - The array has been partitioned into two regions: the heap region a[0..n-k+1] and the sorted region a[n-k+2..n]
        - swap a[0] with a[n-k+1]
        - heapRebuild a[0..n-k]

Heapsort - Example

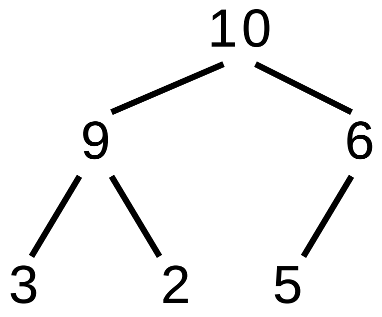
Original array

6	3	5	9	2	10



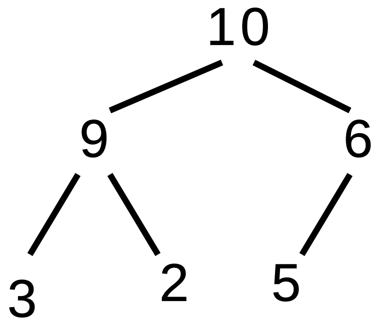
After heap construction

10	9	6	3	2	5



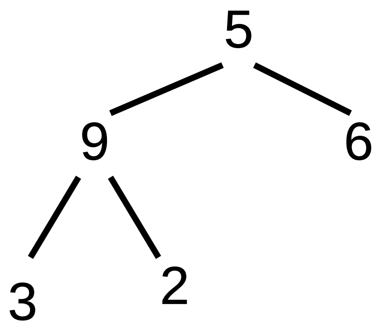
Heapsort - Example

Step 1



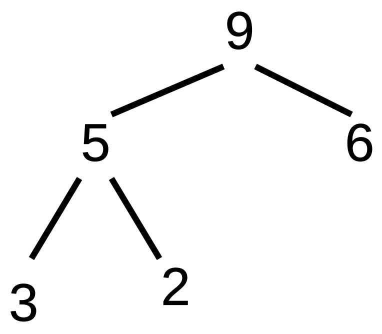
Swap

10	9	6	3	2	5



heap					sorted
5	9	6	3	2	10

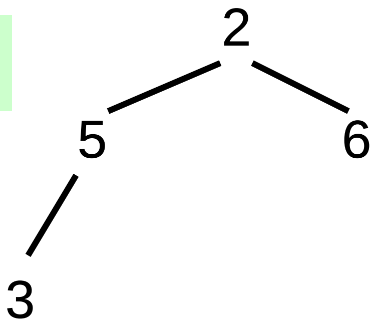
heapRebuild



heap					sorted
9	5	6	3	2	10

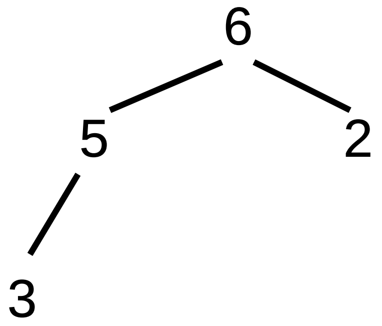
Heapsort - Example

Step 2



swap

heap				sorted	
2	5	6	3	9	10



heapRebuild

heap				sorted	
6	5	2	3	9	10

Sorted

heap	sorted				
2	3	5	6	9	10

Is it in place?

Is it stable?

Complexity?

STL priority_queue

To use STL priority_queue,

```
#include <queue>
```

To create an empty priority queue.

e.g. `priority_queue<int> pq;`

```
bool empty() const;
```

Check whether the priority queue is empty. Return *true* if it is empty, and *false* otherwise.

```
void pop();
```

Remove the item of highest priority from the queue

Precondition: The priority queue is not empty

Postcondition: The priority queue has 1 less element

STL priority_queue (cont'd)

void **push** (const T& item);

Insert the argument item into the priority queue.

Postcondition: The priority queue contains a new element

int **size**() const;

Return the number of items in the priority queue.

T& **top**();

Return a reference to the item having the highest priority.

Precondition: The priority queue is not empty

Priority Queue Example

```
#include <queue>
using namespace std;

int main() {
   priority_queue<int> pq;
   pq.push(10);
   pq.push(20);
   while (!pq.empty()) {
      cout << pq.top() << ” ”;       // 20 10
      pq.pop();
   }
   return 0;
}
```

Midsemester Test

- 3rd October 10am (wed lecture slot)
- Venue: Icube Auditorium (opposite current lecture room)
- Format of test:
  - 80 minutes (5 sections)
  - Covers everything up to Week 5
  - Short answer questions, algorithms (pseudocode/ C++)