04-630
Data Structures and Algorithms for Engineers

Lecture 15: Priority Queues

Adopted and Adapted from Material by:
David Vernon: vernon@cmu.edu ; www.vernon.eu

Agenda

- Last Class: Trees!
  - Block versus variable encoding
  - Optimal Versus Greedy heuristics
  - Prefix Trees – Huffman encoding

- Today: More Trees!
  - Priority queues
  - Binary heap
  - Applications
    - Heapsort

Priority Queues

- Many applications require algorithms to process items in a specific order (e.g., relative importance).
  - One option: Use a list, sort it, and process in the resultant order(ascending or descending)
- Priority queues are more flexible
  - They allow new elements to be added at arbitrary intervals
  - More efficient to add the new element in a priority queue than to ***add*** to a list and ***re-sort***

Priority Queues

$O(\log n)$

Main priority queue operations

– *Insert (Q, x)*

  Given an item x with a key k, insert it into the priority queue Q

– Find_Minimum(Q) or Find_Maximum(Q)

  Return a pointer to the item whose key value is smaller / larger than any other key in the priority queue

– Delete_Minimum(Q) or DeleteMaximum(Q)

  Remove the item from the priority queue Q *whose key is minimum or maximum*

heap_min

heap_max

Priority Queues

Possible implementations

- Unsorted array
- ***Sorted array***: inserting new elements is slow.
- Balanced binary search tree
- ***Binary heap***: suitable when you know upper bound of elements in priority queue (since size of array needs to be specified upfront)….but can mitigate this too using dynamic arrays.

Priority Queues

Possible implementations

	Unsorted array	Sorted array	Balanced tree
Insert(Q, x)	$O(1)$	$O(n)$	$O(\log n)$
Find-Minimum(Q)	$O(1)$	$O(1)$	$O(1)$
Delete-Minimum(Q)	$O(n)$	$O(1)$	$O(\log n)$

log n

- How do we get $O(1)$ for Find-Minimum(Q) in all three cases?
- Use a variable to store a pointer/index to the minimum entry in each of these structures
  - Update the pointer on each insertion (if necessary)
  - Update it on each deletion, requiring the new minimum to be found (but the cost of this can be folded into the cost of the deletion because it is the same cost as the deletion)

Binary Heap

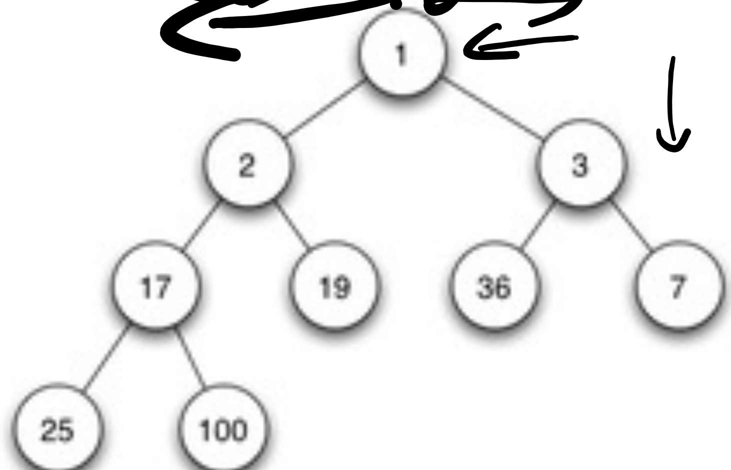
Binary Tree

Heaps maintain partial order on the set of elements

- Weaker than sorted order (& so it is efficient)
- Stronger than random order (& so min/max element can be quickly identified)
- "Heap" refers to being "top of the heap"
  - ***Root Dominates Children***: is greater than (or less than) everything under it

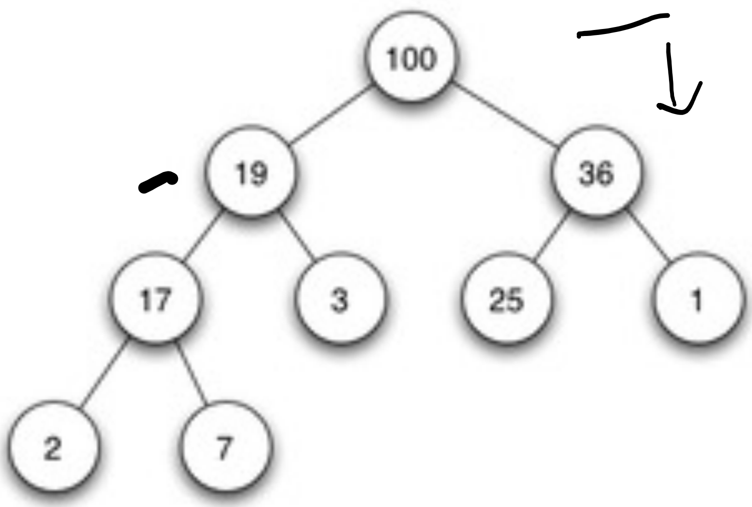
less greater

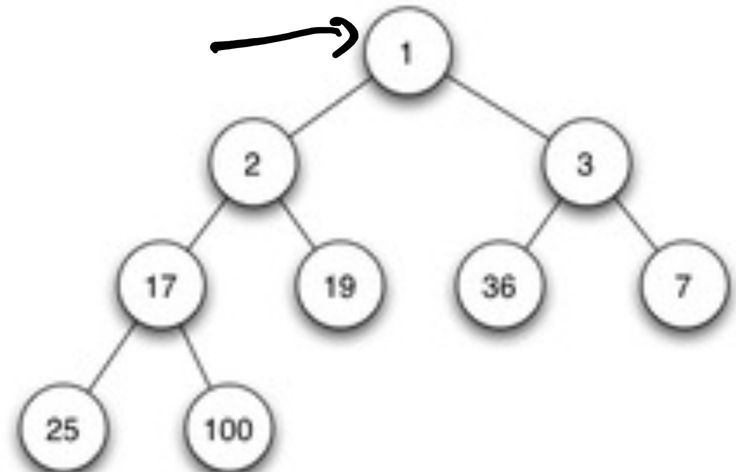
Min-heap == less than
Max-heap == greater than



Heap-labelled tree is a binary tree (not a binary search tree)

Binary Heap





Max-heap

Min-heap

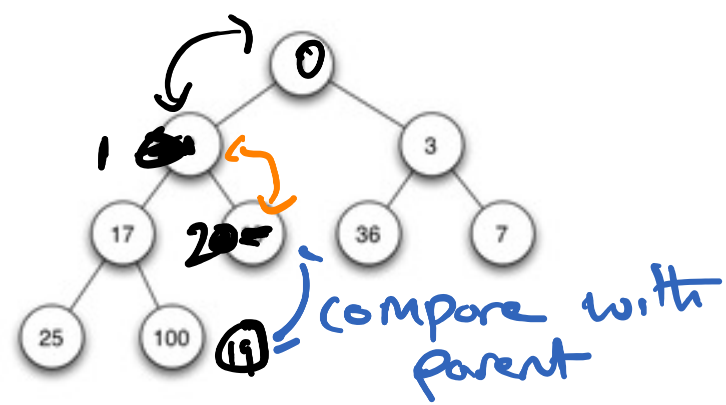
Min-heap == root less than children
Max-heap == root greater than children

Binary Heap

A ***binary heap*** is a ***binary tree*** that satisfies two special shape and heap properties:

– all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right

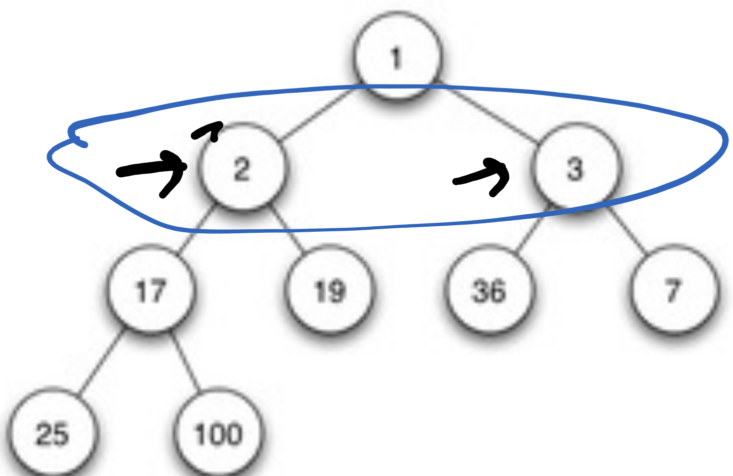
– each node is "greater than or equal to" each of its children (in the case of a max-heap) according to some comparison predicate which is fixed for the entire data structure



Binary Heap

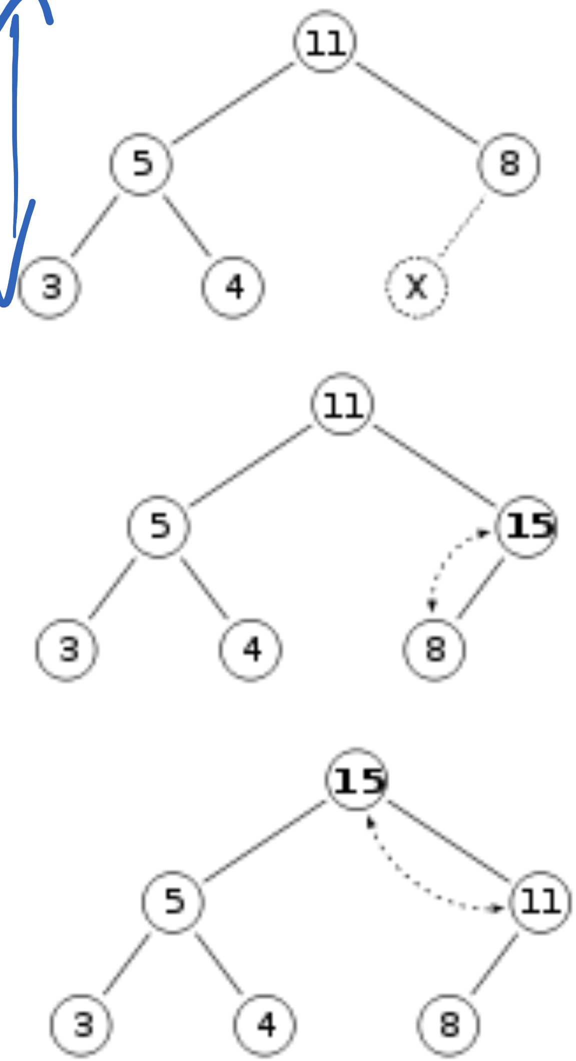
The order of siblings is not specified

- Two children can be freely interchanged
- As long as it doesn't violate the shape and heap properties



Adding to a heap

Add 15 to max-heap

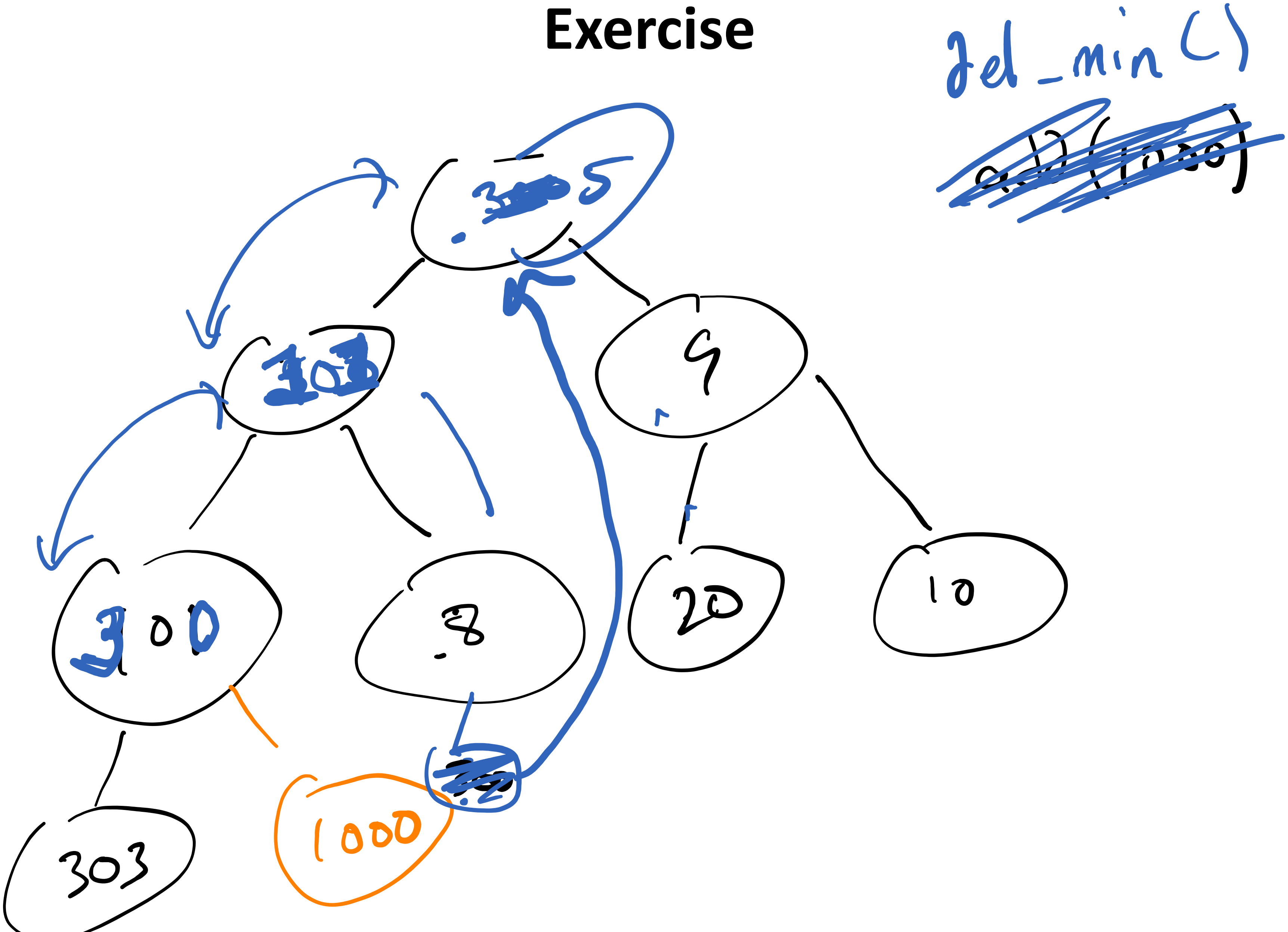


– Algorithm: upheap / heapify-up / sift-up

- Add element to bottom level
- Compare the added element with its parent;
  if they are in correct order, stop
- If not, swap the element with its parent and return to previous step

– O(log n)

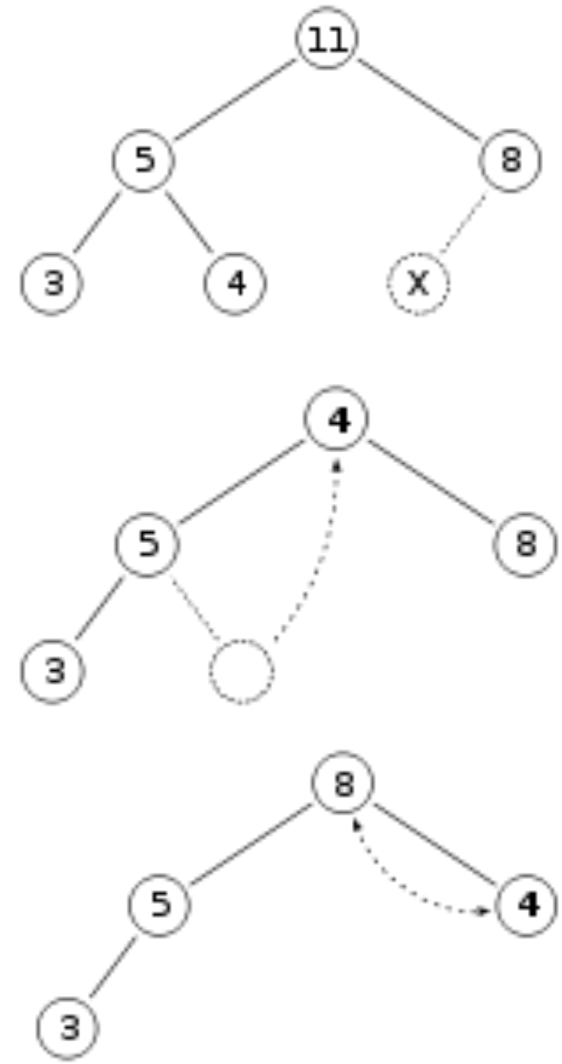
Exercise



Delete Root from max-heap

Deleting the root from a heap

- Effectively extracting the maximum element in a max-heap
  (or extracting the minimum element in min-heap)
- Algorithm: downheap / heapify-down / sift-down
  - Replace root with last element on the bottom level
  - Compare the swapped element with
    - The larger child (max-heap)
    - The smaller child (min-heap)
  - if they are in correct order, stop
  - If not, swap the element with the child and return to previous step
- O(log n)



Binary Heap

Implementation as a binary tree data structure

Problem: finding adjacent element on last level

- Find it algorithmically (takes time)
- Use additional links between siblings:
  threading the tree (takes space)

Binary Heap

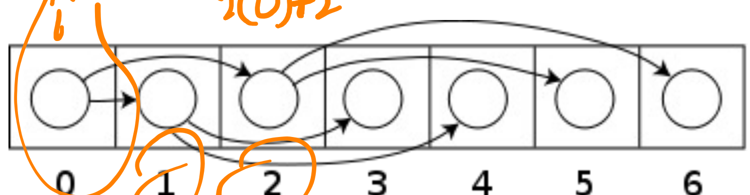
Implementation as an array

- Represent a binary tree WITHOUT any pointers by using an array of keys and a mapping function
- Use formula to find parents and children of a node
  - Node at index i has children at indices $2i + 1$ and $2i + 2$
  - Node at index i has parent at index $(i - 1)/2$

0 $2(0)+1$

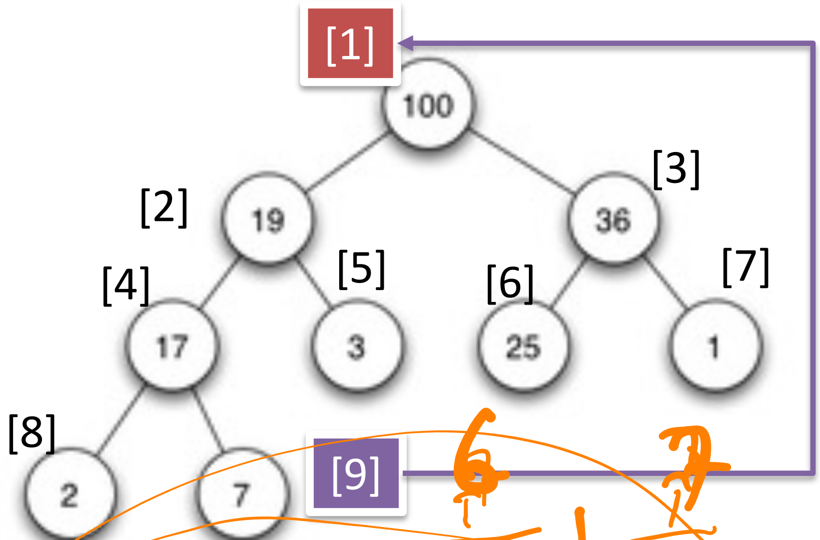
$2(0)+2$

i



Deleting and moving last leaf item: max- & min-heaps

Delete 100, move 7, bubble down



$2(3)+0$ | $2(3)+1$

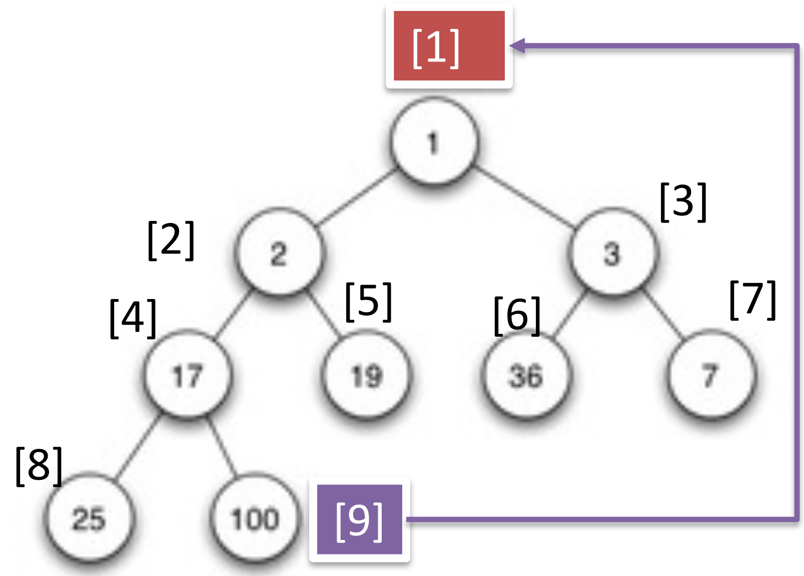
7

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
[illegible]	19	36	17	3	25	1	2	7

36

Max-heap

Delete 1, move 100, bubble down



left child

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	2	3	17	19	36	7	25	100

right child

Min-heap

Recall: for index k, left_child_index=2k, right_child_index=2k+1

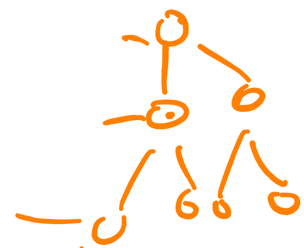
Applications of Priority Queues

- Implementation of:
  - Sorting data, e.g., heapsort
  - Dijkstra's shortest path algorithms.
  - Huffman coding used in data compression.
  - minimum spanning tree algorithms e.g., Prim's algorithm.
  - solutions to fetch the next best or next worst element.
  - best first search algorithms e.g., A* search (fetches next best).

Example application: Heapsort

quick
max
bubble
selection
insertion
selection

- The heap priority queue can be used to create a very efficient sorting algorithm: heapsort
- Construct the priority queue: $O(n \log n)$
- Repeatedly extract the minimum: $O(n \log n)$
- Overall complexity is $O(n \log n)$ worst-case
- This is the best that can be expected from any sorting algorithm
- Also, it is an in-place sort, meaning it uses no extra memory in addition the array containing the elements is to be sorted



$O(\log n)$

Heapsort

```
heapsort(item_type s[], int n) {

    int i;                /* counters */
    priority_queue q; /* heap for heapsort */

    make_heap(&q,s,n); //populate priority queue

    for (i=0; i<n; i++)
        s[i] = extract_min(&q);//repeatedly extract the root
}
```

n

log n

Priority Queues Versus AVL-Trees

- Complete tree as possible
  - Different rules for insert/delete
  - Different goals
  - Different notions/definition of Balance

Partial order: is greater than (or less than) everything under it

Delete Root: Replace it with last element on the bottom level
* Compare the swapped elements to reheap

Add Node: to bottom level as next child
* Compare the added element with its parent and swap to reheap

Order: Greater than left but less than right

Delete Node with two children: Replace with smallest node in right subtree

Add Node: traverse to bottom level based on order then add to location then rebalance.

Heap versus AVL (and BST)

Heap Tree

	Unsorted array	Sorted array	Balanced tree
Insert(Q, x)	$O(1)$	$O(n)$	$O(\log n)$
Find-Minimum(Q)	$O(1)$	$O(1)$	$O(1)$
Delete-Minimum(Q)	$O(n)$	$O(1)$	$O(\log n)$

S Tree

Binary Search
Insert: O(N)
Search: O(N)
Delete: O(N)

AVL Tree
Insert: O(logN)
Search: O(LogN)
Delete: O(logN)

- Heap looks muchhhhh better! Why not always use a heap?






Binary Tree

Binary Search Tree

AVL Tree (Balanced BST)

Red Black Tree

HuffMan (Prefix Tree)

Priority Heap

https://imgur.com/gallery/ENrzhSV/comment/1050925723

https://iconscout.com/icon/deadpool-4411850

Acknowledgement

Adopted and Adapted from Material by:

David Vernon: vernon@cmu.edu ; www.vernon.eu

Augmented by material from:

The Algorithm Design Manual 2nd Edition: by Steven Skiena

Introduction to Algorithms, 3rd Edition, Thomas H. Cormen et al. (2009)