LABORATORY 10 : Heaps

OBJECTIVES

- to understand heap properties and operations

BACKGROUND

Heaps : Heaps are a special form of complete binary tree.

A max (min) heap is a complete binary tree in which the key value in each node is larger (smaller) than the key values in its children (if any).

Examples:

```
              14                        9                       25
            /    \                    /   \                    /
         12        7               6        3              52
       /   \      /              /
    10       8   6             5
```

The key in root of a min heap is the smallest key in the tree, while that in root of a max heap is largest.

Heaps are used in applications where a priority queue is needed

Priority Queue (PQ) :

- element to be removed from the queue is the one with highest priority
- element with arbitrary priority can be inserted into the PQ at any time

Examples:

Scheduling jobs in multiuser environment

- short jobs run before longer jobs
- academic jobs run before student jobs
- user kills a job; scheduler removes job from queue

External sorting

- implementing greedy algorithms (i.e., repeatedly finding a minimum)

Priority Queues

Priority Queue Specification

A priority queue maintains a set of elements under insertions and deletions, where a deletion removes only an element with highest priority (i.e., max heap)

Priority Queue operations include

`is_empty()`	`is_empty()` returns true if the priority queue is empty; otherwise it returns false
`find_max()`	`find_max()` returns an element in the priority queue with highest priority
`insert(e)`	insert element e into a priority queue
`delete()`	delete an element with highest priority from the priority queue

Heap Implementation of Priority Queue

- Max heap is a complete binary tree: any minimum height tree with nodes on lowest level in their leftmost positions and in which each parent is greater than either of its children.

- A complete binary tree of height h has between 2^h and 2^{h+1} -1 nodes. Thus, height of a complete binary tree is $\lfloor \log n \rfloor$ which is O(log n).

- Array (sequential) representation: for any element in array position i, the left child is located at position 2i and right child at position 2i + 1.

Characteristics of static (array) implementation

- no pointers
  -- simple and fast traversal
  -- estimate of heap size required in advance

Heap Operations

- Insert:
  1. Place new element in next available location
  2. If heap condition is violated, reheap (percolate up).
     --percolating up d levels uses d+1 assignments

- Delete:
  1. Find minimum (or maximum) element and remove it, leaving "hole" at root (i.e. "prune" the tree)
  2. Percolate this element down until heap is rebuilt.

LABORATORY 10 : In-lab

1. In an array representation of a binary heap, for an item in position *i*, where are the parent, left child, and right child located?
2. For **max** heap, show the result of inserting 7, 20, 2, 15, 5, 21, 6, 12, 1, 8, 3, 9, 10 and 24, one at a time, into an initially empty heap. **Show the heap after each insertion!**
3. Show the heap from the previous question after deleting seven most maximum elements.
4. Show heap after inserting 99, 1, 55, 16, 28, 33, 599, 19 and 0, one at a time, into the heap in the previous question.

5. What is the greatest number of nodes that can appear at level *n* of a heap?
6. What is the maximum number of nodes that can appear in a heap of height *n*?
7. Given a heap of size *n* what is the minimum number of comparisons required to find the second largest element in the max heap?
8. Given a heap of size *n* what is the minimum number of comparisons required to find the smallest element in the max heap?
9. What is the running time of the `insert` operation on a heap containing `n` elements ?   Give result in big-oh and explain how you come up with it.

LABORATORY 10 : Post-lab

1. Learn about binary heap implementation from https://runestone.academy/runestone/books/published/pythonds/Trees/BinaryHeapImplementation.html

2. Read Chapter 9 (Goodrich's textbook).

3. Write pseudocode and find worst case running time (big O) of the following operations.
   - `build_heap : build a priority queue from a given set of items`
   - `find_max   : find the largest item in the priority queue`
   - `delete     : remove the largest item from the priority queue`

4. Implement the three functions in 3 and test their running times to see whether the actual running times confirm your analysis.

Submission:

as stated in Canvas