Menu

• Priority Queues
• Heaps
• Heapsort
Priority Queue

A data structure implementing a set $S$ of elements, each associated with a key, supporting the following operations:

- $\text{insert}(S, x)$: insert element $x$ into set $S$
- $\text{max}(S)$: return element of $S$ with largest key
- $\text{extract\_max}(S)$: return element of $S$ with largest key and remove it from $S$
- $\text{increase\_key}(S, x, k)$: increase the value of element $x$’s key to new value $k$ (assumed to be as large as current value)

Lecture 3: time (mins)

41 46 49.1 56
Heap

- Implementation of a priority queue (more efficient than BST)
- An array, visualized as a nearly complete binary tree
- **Max Heap Property**: The key of a node is $\geq$ than the keys of its children
  
  (Min Heap defined analogously)
Heap as a Tree

root of tree: first element in the array, corresponding to $i = 1$

parent($i$) = $i/2$: returns index of node's parent

left($i$) = $2i$: returns index of node's left child

right($i$) = $2i+1$: returns index of node's right child
Heap Operations

build_max_heap: produce a max-heap from an unordered array

max_heapify: correct a single violation of the heap property in a subtree at its root

insert, extract_max, heapsort
Max_heapify

• Assume that the trees rooted at left($i$) and right($i$) are max-heaps

• If element $A[i]$ violates the max-heap property, correct violation by “trickling” element $A[i]$ down the tree, making the subtree rooted at index $i$ a max-heap
Max_heapify (Example)

MAX_HEAPIFY (A,2)
heap_size[A] = 10
Max_heapify (Example)

Call MAX_HEAPIFY(A,4)
because max_heap property
is violated
Max_heapify (Example)

Time=? $O(\log n)$
No more calls
Build_Max_Heap(A)

Converts $A[1...n]$ to a max heap

Build_Max_Heap(A):
    for $i=n/2$ downto 1
        do Max_Heapify(A,i)

Time=? $O(n)$

$T(n)=2T(n/2)+O(\log n)$ + Master Theorem
Heap-Sort

Sorting Strategy:

1. Build Max Heap from unordered array;
4. Discard node $n$ from heap (by decrementing heap-size variable)
5. New root may violate max heap property, but its children are max heaps. Run max_heapify to fix this.
6. Go to step 2.
Heap-Sort Demo


Max_heapify(A,1)
Heap-Sort

heap_size = 9
MAX_HEAPIFY (A, 1)

not part of heap
Heap-Sort

A

4 1 3 2 16 9 10 14 8 7
Heap-Sort

Sorting Strategy:

1. Build Max Heap from unordered array;
Heap-Sort

Sorting Strategy:

1. Build Max Heap from unordered array;

2. Find maximum element A[1];

3. Swap elements A[n] and A[1]:
   now max element is at the end of the array!
Heap-Sort

Sorting Strategy:

1. Build Max Heap from unordered array;

2. Find maximum element A[1];

3. Swap elements A[n] and A[1]:
   now max element is at the end of the array!

4. Discard node n from heap
   (by decrementing heap-size variable)
Heap-Sort

Sorting Strategy:

1. Build Max Heap from unordered array;


   now max element is at the end of the array!

4. Discard node $n$ from heap
   (by decrementing heap-size variable)

5. New root may violate max heap property, but its children are max heaps. Run max_heapify to fix this.
Heap-Sort

Heap-Sort

MAX_HEAPIFY(A, 1)
Heap-Sort

Heap-Sort

Running time:

after $n$ iterations the Heap is empty
every iteration involves a swap and a heapify operation;
thus it takes $O(\log n)$ time

Overall $O(n \log n)$