Heap Sort

CSE 2011
Fall 2009

Heap Sort

- Consider a priority queue with n items implemented by means of a heap
  - the space used is $O(n)$
  - methods *insert* and *deleteMin* take $O(\log n)$ time
  - methods *size*, *isEmpty*, and *findMin* take time $O(1)$ time
- Using a heap-based priority queue, we can sort a sequence of n elements in $O(n \log n)$ time
- The resulting algorithm is called heap-sort
- Heap-sort is much faster than quadratic sorting algorithms, such as insertion-sort and selection-sort

Sorting Using a Heap

- Input: array A with n elements to be sorted
- Create a heap T
  - Example: www.cs.hut.fi/Opinnot/T-106.1220/heaptutorial/lisaaminen.html
- Sorting code

```
for (i = 0; i++; i < n)
    T.insert(A[i]);
for (i = 0; i++; i < n)
    A[i] = T.deleteMin();
```

3

Analysis of Heap Sort

- Stirling's approximation: $n! \approx n^n e^{-n} \sqrt{2\pi n}$
- Insertions
  log1 + log 2 + … + log n = log(n!) = O(nlogn)
- Deletions
  log1 + log 2 + … + log n = log(n!) = O(nlogn)
- Total = O(nlogn)

4

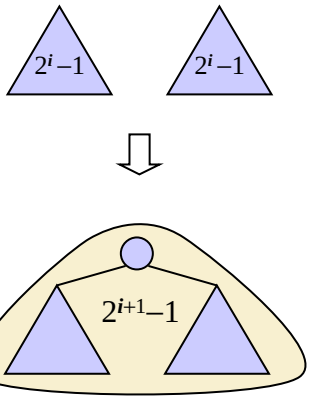
In-place Heap Sort

- The heap sort algorithm we just discussed requires a temporary array (the heap).
- In-place heap sort uses only one array, which is the original array storing the inputs.
- Needs to build a heap (in-place) from the set of input values $\Rightarrow$ *buildHeap* procedure

5

buildHeap

- Input: a non-heap binary tree stored in an array
- Output: a heap stored in the same array
- We can construct a heap storing n given keys in using a bottom-up construction with $\log n$ phases
- In phase i, pairs of heaps with 2^i-1 keys are merged into heaps with $2^{i+1}-1$ keys



6

Example

- See demo with max heaps at

www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/heapSort/heapSort.html

7

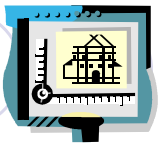
In-place Heap Sort

- The first step is to build a max heap using *buildHeap*
- Call *deleteMax* to remove the max item (the root). The heap size is reduced by one. The last entry of the heap is now empty. Store the item just removed into that location (*copyMax*).
- Repeat *deleteMax* and *copyMax* until the heap is empty.
- Examples: will be shown in class
- Demo/animation

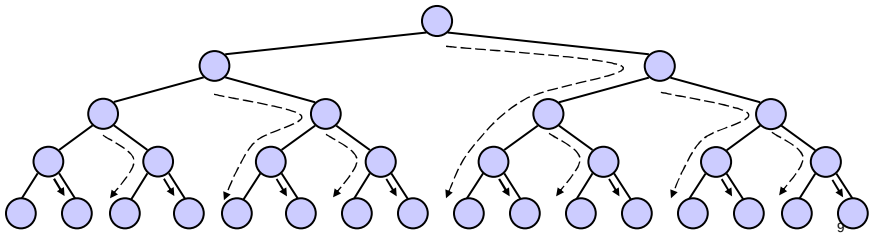
  www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/heapSort/heapSort.html
  www2.hawaii.edu/~copley/665/HSApplet.html

8

Analysis of *buildHeap*

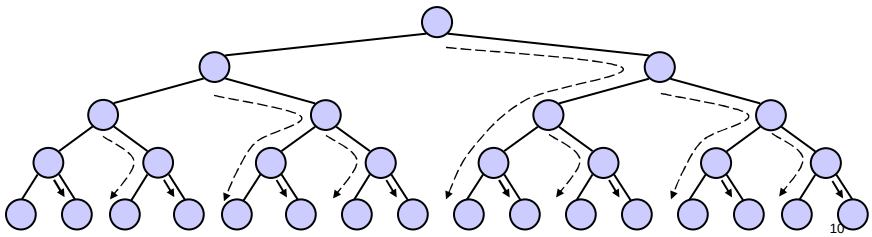


- Bottom-up heap construction runs in $O(n)$ time
- Bottom-up heap construction is faster than n successive insertions and speeds up the first phase of heap-sort



Analysis of *buildHeap* (2)

- Theorem: For the perfect binary tree of height h containing $n = 2^{h+1} - 1$ nodes, the sum of the heights of all the nodes is $2^{h+1} - 1 - (h + 1)$
- *buildHeap* thus runs in $O(n)$ time



Analysis of In-place Heap Sort

- Build a max heap using *buildHeap* $\Rightarrow O(\ )$
- Repeat
  - *deleteMax* $\Rightarrow O(\ )$
  - *copyMax* $\Rightarrow O(\ )$

  until the heap is empty $\Rightarrow n$ iterations

Total running time = $O(n \log n)$

More Heap Operations

- decreaseKey(p, d)
  - T[p] = T[p] **–** d, then percolate up
  - Example: system admin boosts the priority of their jobs
- increaseKey(p, d)
  - T[p] = T[p] **+** d, then percolate down
  - Example: penalizing misbehaved processes
- delete(p)
  - Perform decreaseKey(p, ∞) then deleteMin()
  - Example: removing a print job from the PQ
- Note: searching for index p takes O(n) time in the worst case, but we expect not to use the above methods very often.

Next time …

- Hash Tables (section 9.2)

13