Heap Sort

cse2011

section 8.3.5 of textbook

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

- Temporary array T of size at least *n*+1, which will work as a heap.

- 2 steps:

1. Create a heap T using the elements of A
   - Insert each element A[i] into T using T.insert(A[i])
2. Call T.deleteMin() *n* times to move the elements from T to A, one by one.

Sorting Code

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

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)

In-place Heap Sort

Heap Sort Comparison

<u>Using a temp heap T</u>

(1) for (i = 0; i++; i < n)
   T.*insert*(A[i]); *// O(nlogn)*

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

Note: <u>min</u> heap

<u>In place (no temp array)</u>

(1) *buildHeap* on A; *// O(n)*

(2) repeat
   deleteMax; *copyMax*;
until the heap is empty;

Note: <u>max</u> heap

In-place Heap Sort

- The heap sort algorithm we just discussed requires a temporary array T (a min heap).
- In-place heap sort <u>uses only one array</u>, the original array storing the inputs.
- 2 steps:

1. Transform the original array to a ***max*** heap using *buildHeap* procedure (“heapify”)
2. Call *delete**Max***() n times to get the array sorted.