

0.1 Lecture 7 - Heapsort

priority queue

1. insert min in $O(\log n)$
2. extract min in $O(\log n)$

Definition 0.1. min-heap

1. children is as large as the parent
2. add new element to the next available position at the lowest level

Algorithm 1 Extract min

Input: A

Output: \min

```

1:  $\min \leftarrow A[i]$ 
2:  $\text{ASSIGN}(A[1], A[n])$ 
3:  $\text{ASSIGN}(A[n], \infty)$ 
4:  $j \leftarrow 1$ 
5:  $l \leftarrow A[2j]; r \leftarrow A[2j + 1]$ 
6: while  $A[j] > \min(A[l], A[r])$  do                                 $\triangleright$  heapify
7:   if  $l > r$  then
8:      $\text{SWAP}(A[j], A[2j])$ 
9:      $j \leftarrow 2j$ 
10:  else
11:     $\text{SWAP}(A[j], A[2j + 1])$ 
12:     $j \leftarrow 2j + 1$ 
13:  end if
14:   $l \leftarrow A[2j]; r \leftarrow A[2j + 1]$ 
15: return  $\min$ 

```

Definition 0.2. heap-sort

build a binary heap and insert elements, extract min in $O(n \log n)$

Remark 0.3 — How to support decrease-key in $O(\log n)$?

0.2 Lecture 8 - Sorting in linear time

Claim 0.4. Properties of sorting

1. Comparison based sorting have $\Omega(n \log n)$
2. $h \geq \log n$

0.2.1 Counting sort

Algorithm 2 Counting sort

Input: A, k **Output:** B

```

1:  $C[0 \dots k] \leftarrow 0$ 
2: for  $i \leftarrow 1$  to  $n$  do
3:    $C[A[i]] \leftarrow C[A[i]] + 1$ 
4: end for
5: for  $i \leftarrow 1$  to  $k$  do
6:    $C[i] \leftarrow C[i] + C[i - 1]$ 
7: end for
8: for  $i \leftarrow n$  to  $1$  do
9:    $idx \leftarrow C[A[i]]$ 
10:   $B[idx] \leftarrow A[i]$ 
11:   $C[A[i]] \leftarrow C[A[i]] - 1$ 
12: end for
13: return  $B$ 

```

0.2.2 Radix sort

Definition 0.5 (Radix sort). use counting sort to sort digits from least significant to most significant

Proof of correctness.

Assume we already sorted the lower digits $0 \dots k - 1$. We need to sort the most significant digit. WLOG, suppose we have a digit d , by definition, all numbers with d as the k -th digit are sorted but not necessarily in continuous positions. Counting sort simply puts numbers with the same k -th digit in the continuous order.

Run-time analysis.

Counting sort is run d times and total run time is $\Theta(d(n + k))$

Decision model on list merging.

1. number of leafs = $k!$
2. number of possible permutations = $(k!)^{\frac{n}{k}}$
3. height = $\log((k!)^{\frac{n}{k}}) = \Theta(n \log k)$