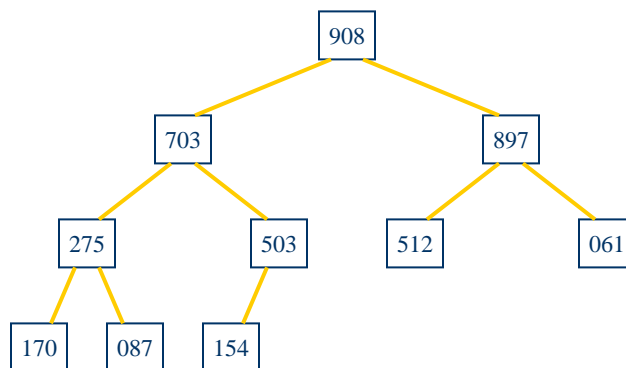


Heap Sort

CLRS Reading: Sections 6.1, 6.2, 6.3, 6.4, pages 123 -- 138
Problem Set: Assignment #2 due Friday, February 15

E - 1

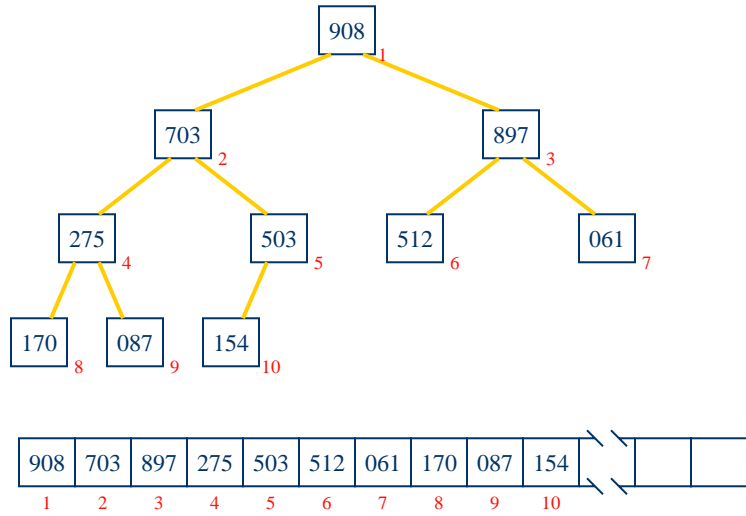
Sort of Sorted*



*Nearly complete binary tree satisfying the heap condition.

E - 2

Heaps



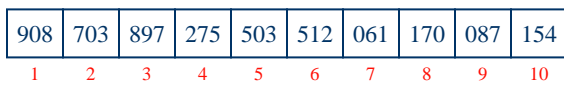
heap-size[A]

A

length[A] E - 3

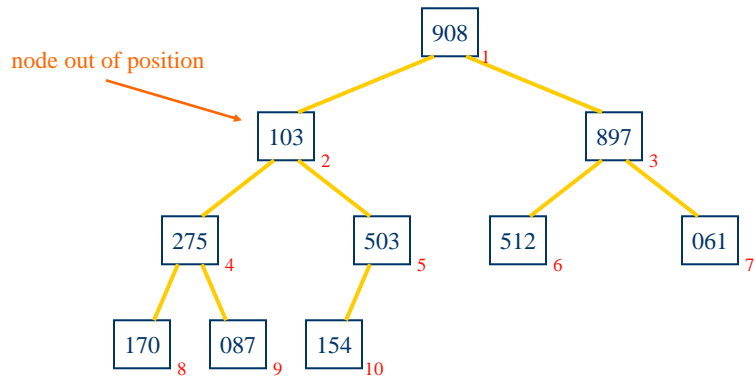
Families Reunited

- The array implementation of heaps make it particularly easy for parents to find children and vice versa.



E - 4

Maintaining the Heap Property



E - 5

Max-Heapify

908	103	897	275	503	512	061	170	087	154
1	2	3	4	5	6	7	8	9	10

```

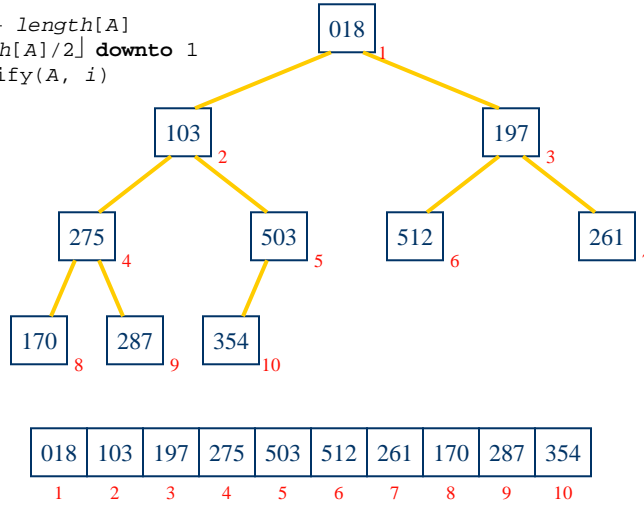
Max-Heapify(A, i)
  l ← Left(i)
  r ← Right(i)
  if l ≤ heap-size[A] and A[l] > A[i]
    then largest ← l
    else largest ← i
  if r ≤ heap-size[A] and A[r] > A[largest]
    then largest ← r
  if largest ≠ i
    then exchange A[i] ↔ A[largest]
       Max-Heapify(A, largest)
  
```

E - 6



Build-Max-Heap

```
Build-Max-Heap(A)  
  heap-size[A] ← length[A]  
  for i ← ⌊length[A]/2⌋ downto 1  
    do Max-Heapify(A, i)
```

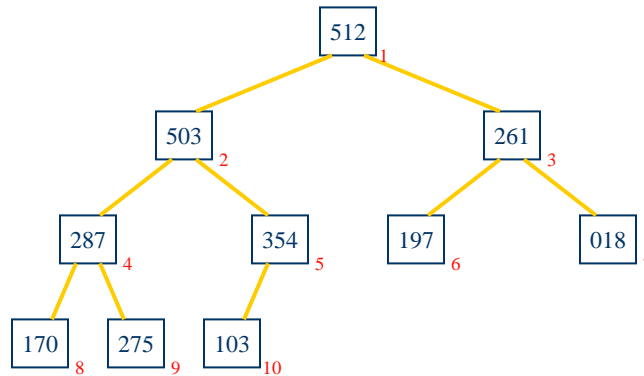


E-7



The Cost of Building a Heap

Lemma. An n -element heap has height $\lceil \lg n \rceil$ and has at most $\lceil n/2^{h+1} \rceil$ nodes of height h .



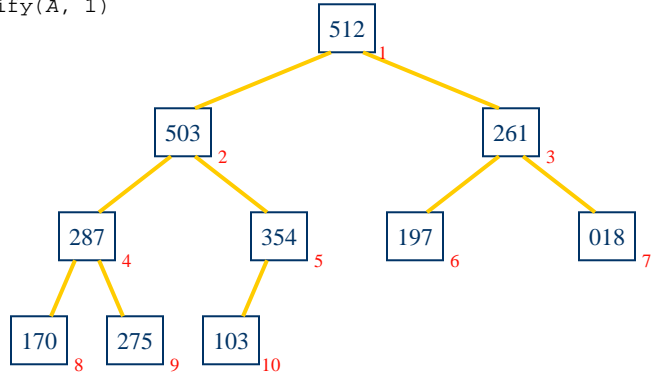
E-8



Heapsort(

512	503	261	287	354	197	018	170	275	103
1	2	3	4	5	6	7	8	9	10

```
Heapsort(A)
  Build-Max-Heap(A)
  for i ← length[A] downto 2
    do exchange A[1] ↔ A[i]
       heap-size[A] ← heap-size[A]-1
       Max-Heapify(A, 1)
```



E-9