CSE 326
Lecture 23

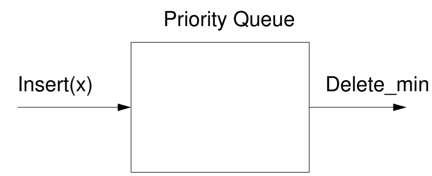
Spring Quarter 1995

Priority Queues

- Priority queues support the two operations

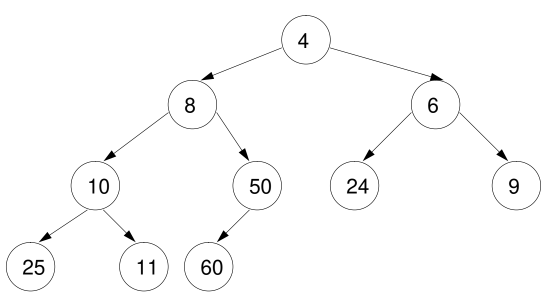
  1. Insert(x) which inserts the key (number x) into the priority queue

  2. Delete_min which removes the smallest item in the priority queue and return it.

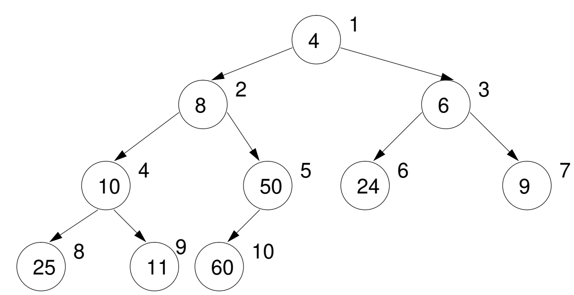


The Heap Data Structure

- A binary heap is a complete binary tree with the "heap property".
- Heap property: The key of the parent is smaller than the keys of its children.



Storage of a Complete Binary Tree



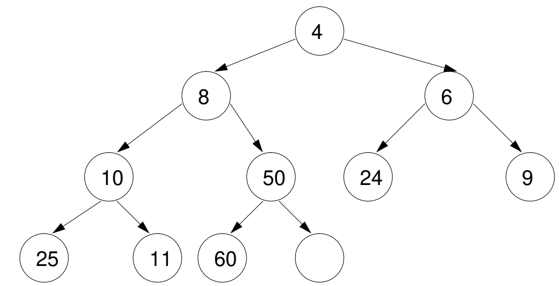
1	2	3	4	5	6	7	8	9	10		
4	8	6	10	50	24	9	25	11	60		

Finding Children and Parent

- We assume that there are n items in the priority queue and an array H[1..m] where m $\geq$ n.
- The root of the heap is in position 1.
- Children of i are in 2i and 2i+1 as long as 2i $\leq$ n and 2i+1 $\leq$ n.
- Parent of i is in i/2 as long as i > 1.

Insertion

- Suppose we want to insert the item 5. "Percolate Up"



Insertion Code

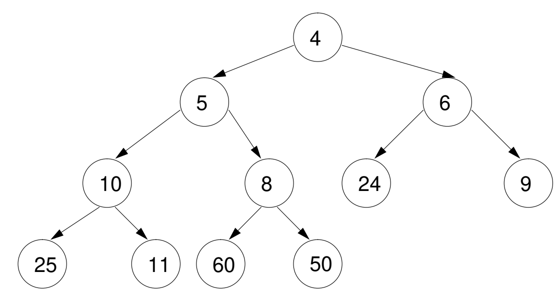
- Insert(x) inserts into the heap H.
- Insert(x) {

```
local integer i;
if n = m then "heap is full"
else {
   n := n + 1;
   i := n;
   while not(i/2=0) and H[i/2] > x do {
       H[i] := H[i/2];
          i := i/2 }
   H[i] := x }}
```

- Use of a sentinel in H[0] can speed up the code slightly. H[0] should be smaller than any element put into the priority queue ever.

Delete_Min

- "Percolate Down"



Delete_min Code

- Delete_min removes and returns the minimum in the heap and reheapifies.
- Delete_min {

```
local integer i, j ; local key x,y;
if n = 0 then "heap is empty"
else {
  i :=1;
  y := H[1]
  x := H[n];
  n := n-1;
  while 2i ≤ n do {
    if 2i + 1 ≤ n and H[2i+1] < H[2i] then j := 2i+1 else j := 2i;
    if H[j] < x then  {
      H[i] := H[j];
          i := j }
    else break }
  H[i] := x }
return(y)}
```


Performance of the Binary Heap

- Insert

  - worst case O(log n)
  - average case O(1). The average insert percolates up 1.6 levels

- Delete_min

  - worst case O(log n)
  - average case also O(log n). The average delete-min percolates to the bottom of the tree.

Other Heap Operations

- Decrease(i,d) decreases the value in H[i] by d and returns the new index of the item H[i].

  Simply decrease H[i] by d and percolate up from position i.

- Increase(i,d) increases the value of H[i] by d and returns the new index of the item H[i].

  Simply increase H[i] by d and percolate down from position i.

- Delete(i) deletes the item H[i] from the heap.

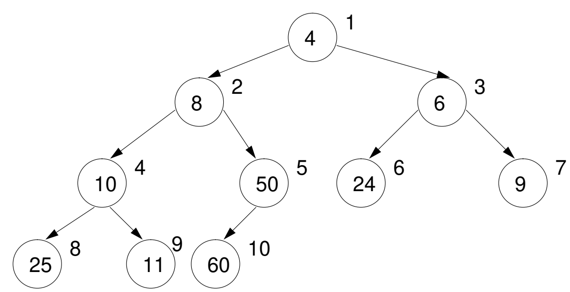
  First Decrease(i,infinity), then Delete_min.

- Build_heap builds a heap from scratch assumming items are stored in H[1..n] (not necessarily in heap order).

  Starting with i = n/2 and working down to 1 just percolate_down(i). Total time is O(n).

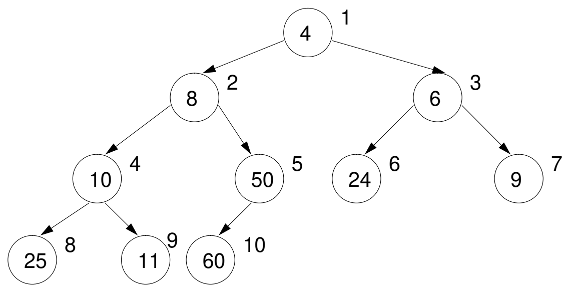
Decrease(i,d)

- Decrease(8,21)



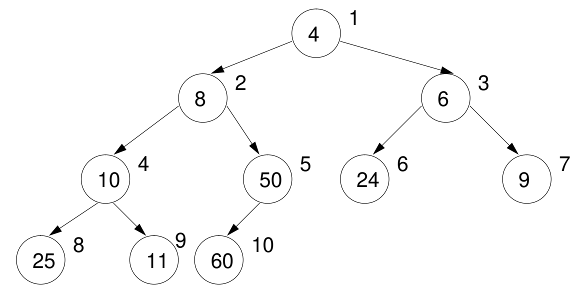
Increase(i,d)

- Increase(2,50)

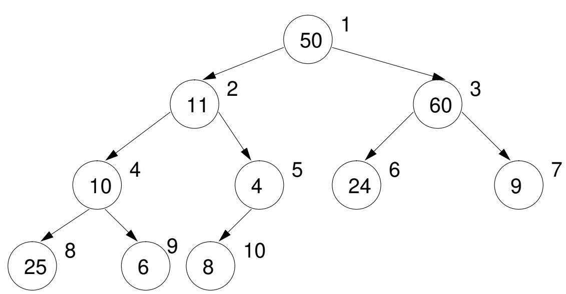


Delete(i)

- Delete(4)



Build_heap



Complexity of Build_Heap

- At most n/2 percolate 1 level
  at most n/4 percolate 2 levels
  at most n/8 percolate 3 levels

$$Total\ Time = 1 \cdot \frac{n}{2} + 2 \cdot \frac{n}{4} + 3 \cdot \frac{n}{8} + \ldots = \sum_{i=1}^{\log n} i \cdot \frac{n}{2^i} = n \cdot \sum_{i=1}^{\log n} \frac{i}{2^i} \le 2n$$

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

$$x \cdot \frac{d}{dx} \sum_{i=0}^{\infty} x^i = x \cdot \sum_{i=0}^{\infty} i \cdot x^{i-1} = \sum_{i=1}^{\infty} i \cdot x^i = \frac{x}{(1-x)^2}$$

Heap Sort

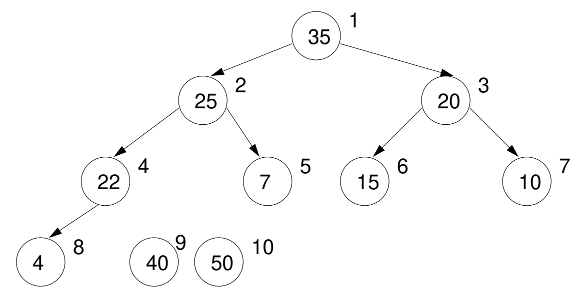
- Suppose we have an unordered array A[1..N] which we wish to sort.

  1. Build a max heap. (The largest element is in A[1].)
  2. For i = 1 to N - 1 do
     x := Delete_max;
     A[N-i+1] := x
     endfor

	1	2	3	4	5	6	7	8	9	10
After Build_heap	50	40	20	25	35	15	10	22	4	7
	40	35	20	25	7	15	10	22	4	50
	35	25	20	22	7	15	10	4	40	50

Heap Sort Step

1	2	3	4	5	6	7	8	9	10
35	25	20	22	7	15	10	4	40	50



25	22	20	4	7	15	10	35	40	50

Properties of Heap Sort

- Worst case time complexity O(n log n)
  Build_heap O(n)
  n Delete-max's for O(n log n)
- In-place sort - only constant storage beyond the array is needed.