CMPT307: Heapsort

Week 4-3

Xian Qiu

Simon Fraser University

xianq@sfu.ca

In Place Sorting

a sorting algorithm is in place if it stores $O(n)$ arrays

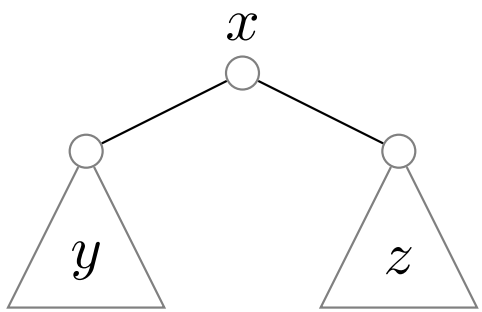
- ▷ insertion sort is in place $O(n^2)$
- ▷ what about merge sort?

MERGE-SORT(A,p,r)

```
1 if p < r then
2     q = ⌊(p + r)/2⌋;
3     MERGE-SORT(A,p,q);
4     MERGE-SORT(A,q + 1,r);
5     MERGE(A,p,q,r);
```

- ▷ merge sort is not in place $O(n \log n)$
- ▷ better algorithms?

Binary Search Trees

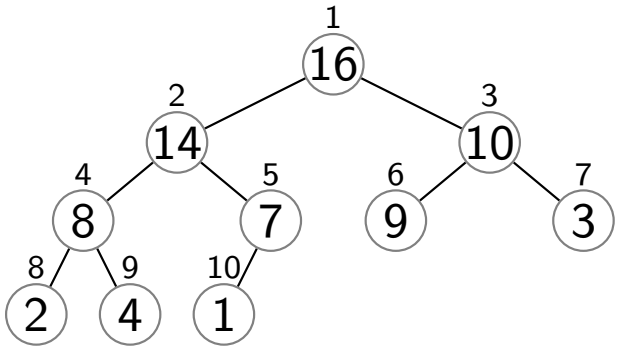


$y.\text{key} \leq x.\text{key} \leq z.\text{key}$

- having a BST, sorting costs only $\Theta(n)$
- running time is mainly bounded by building a BST
- classical BST: insertion is simple, may yield unbalanced tree
- red-black trees, AVL trees: yield (approximately) balanced tree, but insertion is complicated
- want a balanced tree, with simpler insertion

Heap

heap is an array object with attributes: length and heap-size



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

$\textsc{Parent}(i)$ {**return** $\lfloor i/2 \rfloor$;}

$\textsc{Left}(i)$ {**return** $2i$;}

$\textsc{Right}(i)$ {**return** $2i + 1$;}

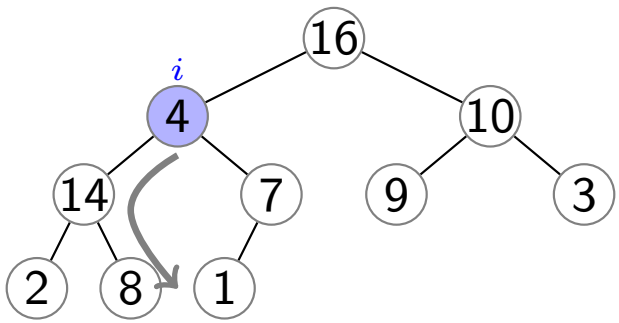
- heap forms a (nearly) complete binary tree
- max-heap: $A[\textsc{Parent}(i)] \geq A[i]$
- min-heap: $A[\textsc{Parent}(i)] \leq A[i]$

Properties

- $n \geq 1 + 2 + \ldots + 2^{h-1} + 1 = 2^h$
- $n \leq 1 + 2 + \ldots + 2^h = 2^{h+1} - 1$
- $n = \lfloor \log n \rfloor$
- how many leaves?
  - node n is the last leave and $\lfloor n/2 \rfloor$ is the last internal node
  - # leaves $= \lceil n/2 \rceil$

Maintaining Heap

- ▷ given array A and index i, assume the subtrees rooted at $\textsc{Left}(i)$ and $\textsc{Right}(i)$ are max-heaps
- ▷ make the subtree rooted at i a heap



Maintaining Heap

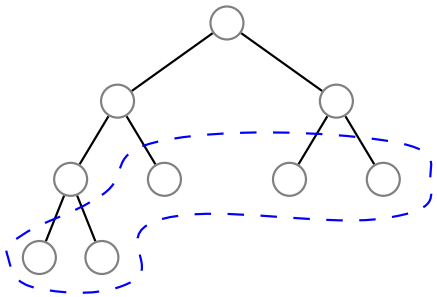
MAX-HEAPIFY(A,i)

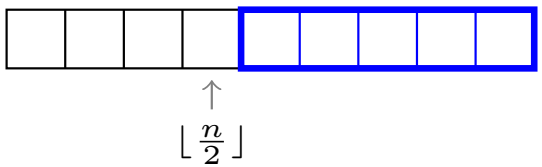
```
1  l = LEFT(i);
2  r = RIGHT(i);
   // find arg max{A[l], A[r], A[i]} for l, r, i ≤ A.heap-size
3  if l ≤ A.heap-size and A[l] > A[i] then
4      largest = l ;
5  else if r ≤ A.heap-size and A[r] > A[largest] then
6      largest = r ;
   // if A is not a max-heap
7  if largest ≠ i then
8      exchange A[i] with A[largest];
9      MAX-HEAPIFY(A, largest);
```

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

Building Heap

- ▷ to convert an array A into a heap
- ▷ use MAX-HEPIFY in a bottom-up manner



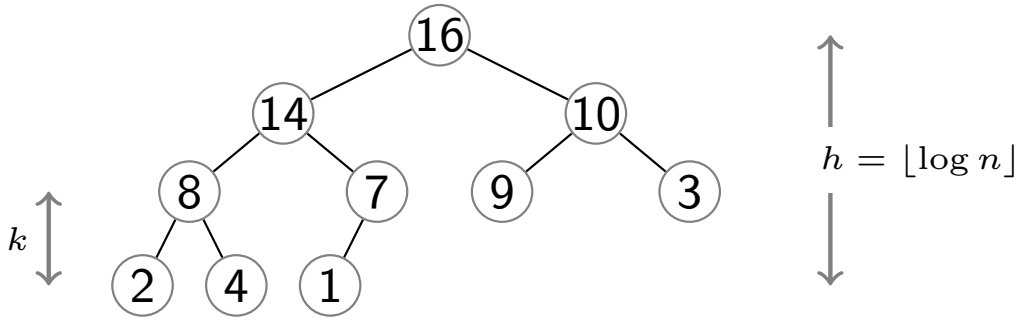


BUILD-MAX-HEAP(A)

```
1 A.heap-size = A.length;
2 for i = ⌊A.length/2⌋ down to 1 do
3     MAX-HEAPIFY(A,i);
```

- ▷ running time?

Tighter Bound

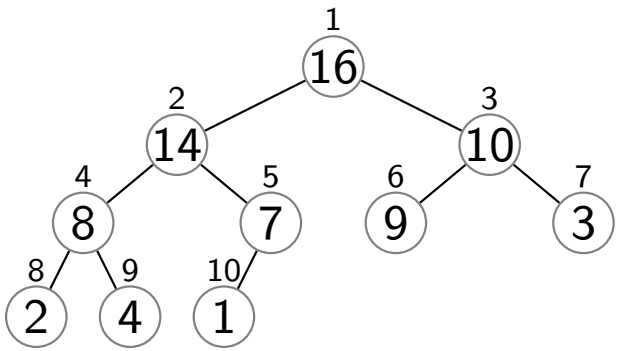


- ▷ $[\#\text{ nodes at height } k] = [\#\text{ nodes at depth } h-k] \leq 2^{h-k}$
- ▷ $2^h \leq n \Rightarrow 2^{h-k} \leq \frac{n}{2^k}$

$$T(n) = \sum_{k=0}^{\lfloor \log n \rfloor} \frac{n}{2^k} \cdot O(h) = O\left(n \sum_{k=0}^{\lfloor \log n \rfloor} \frac{k}{2^k} \right)$$

$$\sum_{k=0}^{\infty} \frac{k}{2^k} = \frac{1/2}{(1-1/2)^2} = 2 \quad \Rightarrow \quad T(n) = O(n)$$

Heapsort



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

HEAPSORT(A)

```
1 BUILD-MAX-HEAP(A);
2 for i = A.length down to 2 do
3     exchange A[1] with A[i];
4     A.heap-size = A.heap-size − 1;
5     MAX-HEAPIFY(A, 1);
```

heapsort is in place and runs in $O(n \log n)$

Priority Queue

Priority queue

- each element has a key (priority)
- dequeue an element with the maximum key

max-priority queue supports the following operations

- INSERT(S,x): inserts x into S
- MAXIMUM(S): returns the element with the largest key
- EXTRACT-MAX(S): returns MAXIMUM(S) and remove it
- INCREASE-KEY(S,x,k): increase x.key to k ($k \geq$ current key)

can use max-heap to implement max-priority queue

Operations

HEAP-MAXIMUM(A) {**return** $A[1]$;}

HEAP-EXTRACT-MAX(A)

```
1 max = A[1] ;                                   // assume A.heap-size ≥ 1
2 A[1] = A[A.heap-size];
3 A.heap-size = A.heap-size − 1;
4 MAX-HEAPIFY(A, 1);
5 return max;
```

Operations

HEAP-INCREASE-KEY(A, i, key)

```
1 A[i] = key ;                                   // assume A[i] < key
2 while i > 1 and A[PARENT(i)] < A[i] do
3 |   exchange A[i] with A[PARENT(i)];
4 |   i = PARENT(i);
```

HEAP-INSERT(A, key)

```
1 A.heap-size = A.heap-size + 1;
2 A[A.heap-size] = −∞;
3 HEAP-INCREASE-KEY(A, A.heap-size, key);
```