Counting sort

- aka pigeonhole sort
- Diff. problem: n students take a quiz, score from 1 to 10. List of scores:7,2,5,…
- Want to know: # 1's , # 2's …Solution?
- How to use idea for sorting?: first look at simpler version
- Stable Sorting: relative order of equal elements does not change

Counting sort

```
COUNTING-SORT(A, B, k)
    for i ← 0 to k
        do C[i] ← 0
    for j ← 1 to length[A]
        do C[A[j]] ← C[A[j]] + 1
    ▷ C[i] now contains the number of elements equal to i.
    for i ← 1 to k
        do C[i] ← C[i] + C[i − 1]
    ▷ C[i] now contains the number of elements less than or equal to i.
    for j ← length[A] downto 1
        do B[C[A[j]]] ← A[j]
           C[A[j]] ← C[A[j]] − 1
```

Preprocessing

- Willing to spend some time upfront to transform data to save time later. Eg ?
- Eg: sorting. To save time for what ?
- Searching – binary search faster, can only be done on a sorted array
- Eg: Counting sort
  - Creating the array C[ ]

Problem time bounds

- Avg case, worst case always same ?
- No – quick sort
- Problem time bounds: time bound for a problem independent of algorithm.
- Sort faster than $O(n^2)$?
- Yes – merge sort.
- Sort faster than $O(n \log n)$?
- No i.e. no comparison-based algorithm can sorter faster.

Lower bounds for sorting

- Comparison based sort:

numbers are black boxes.

  - 7, 9,5,12 same as
  - 500, 800,2,10000

- Decision tree: model of computation.
- Lower bd for searching:
- Lower bd for sorting:
- Do these apply to counting sort?
- No : is not comparison-based

Review of data structures

- Linked list: singly linked, doubly linked, circular
- Stack: LIFO, push, pop, program stacks.
- Queue: FIFO, enqueue, dequeue.
- Tree: parent, child, root, descendent, ancestor, sibling, height, depth, path, full, complete.
- Binary Tree: Lchild, Rchild

Review of data structures

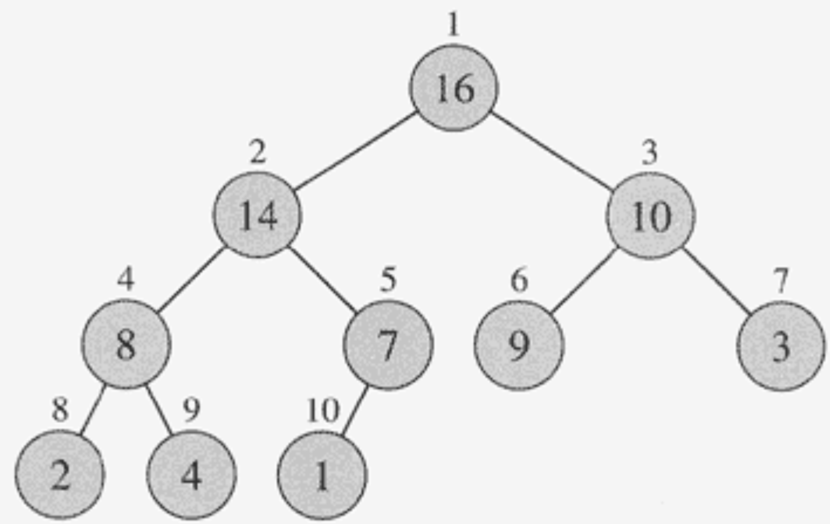
- Binary Search Tree:
  - L <= parent <= R
  - Useful for searching. How much time?
- depends on height. How much is height ?
- O(n) in worst case – Eg ?
- Want a small height tree. Complete tree?
- No – need flexibility for insertions/deletions
  - While guaranteeing small height. What works ?
- Balanced Binary Search Tree?
  - AVL, Red-Black.

Review: Priority Queues

- What do we want: Given elements with keys, order amongst keys, want to carry out the following operations on Q:
  - Insert (x,Q)
  - ExtractMax(Q)
  - Maximum(Q)
- Eg: prioritized jobs on a computer

Heaps

- Structural Property: Complete binary tree
- Heap Property (Max-Heap): parent >= child
- Eg [CLR Fig 6.1]



(a)

Heaps

- Insert: How to do ?
- Insert at bottom right (after rightmost elt in bottom row)
  - reheapify by moving up
- ExtractMax: How to do ?
- Exchange with bottom right (rightmost elt in bottom row)
  - reheapify by moving down
- Not a search tree !

Heap time analysis

- Height of tree ?
- O (log n)
- Time for Insert ?
- O (log n)
- Time for ExtractMax ?
- O (log n)

Heap implementations [Fig 6.1]

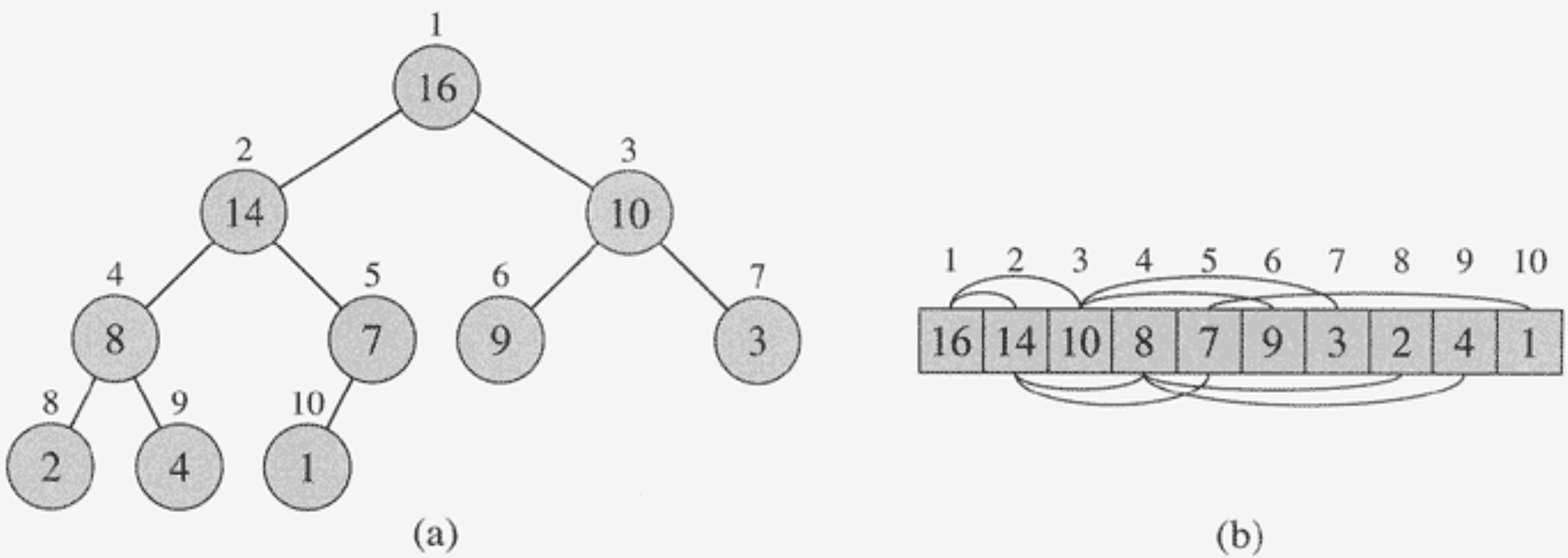


Figure 6.1 A max-heap viewed as **(a)** a binary tree and **(b)** an array. The number within the circle at each node in the tree is the value stored at that node. The number above a node is the corresponding index in the array. Above and below the array are lines showing parent-child relationships; parents are always to the left of their children. The tree has height three; the node at index 4 (with value 8) has height one.

Heap implementations

- How to implement a heap?
- Can do as a binary tree using pointers in the usual way.
- Can also implement with arrays.
  - Lchild(i) = 2i
  - Rchild(i) = 2i + 1
  - Parent(i) = $\lfloor i/2 \rfloor$

Heap sort

- MinHeap: parent < child
- Heap Sort:

for i from 1 to n

    Insert(A[i],Q)

for i from 1 to n

    A[i] = ExtractMin(Q)

- Time ?
- O(n log n)
- Can build heap faster: in time O(n), bottom up
  – Will change time analysis of heap sort ?

More Math Preliminaries:

- Permutations and Combinations
  - order matters. Permutation: Yes. Combination: No.
- Permutations Eg: # 2 letter words can you make with 3 letters a,b,c.
- Permutations : nPk (or ${}^{n}P_{k}$): number of ways of arranging k out of n items
  - nPk = ?
- nPk = n!/(n-k)!

More Math Preliminaries:

- Combinations Eg: #2 member committees can you make with 3 people x,y,z.
- Combinations : nCk (or ${}^{n}C_{k}$): number of ways of picking k out of n items
  - order does not matter.
  - nCk = ?
- nCk = n!/k!(n-k)!

More Math Preliminaries:

- Proof by induction : sum of first n numbers.
- Prime number : divisible only by 1 and itself : 2,3,5,7…
  - can't be written as product of other numbers
  - eg. 2,3,5,7, 11, 13 are prime, 4,6,8,9,10 are not
  - 4,6,8,9,10 are composite numbers
- Proof by contradiction : no largest prime.