- 书面作业讲解
  - TC第7.1节练习2
  - TC第7.2节练习4
  - TC第7.3节练习2
  - TC第7.4节练习2
  - TC第7章问题4、5
  - TC第8.1节练习3、4
  - TC第8.2节练习4
  - TC第8.3节练习4
  - TC第8.4节练习2
  - TC第8章问题2
  - TC第9.1节练习1
  - TC第9.3节练习5、7

TC第7.1节练习2

- Modify PARTITION so that $\lfloor q=(p+r)/2 \rfloor$ when all elements in the array A[p..r] have the same value.
  - 方法1：if (A[j]==x) count++;
  - 方法2：if (A[j]==x) flag=true;
  - 方法3：if (A[p]==A[r]) return $\lfloor q=(p+r)/2 \rfloor$; 行不行?

TC第7.3节练习2

- How many calls are made to RANDOM?
  - Worst case: $\Theta(n)$
  - Best case: $T(n)=T(\lfloor n/2\rfloor)+T(\lceil n/2\rceil)+1$，$T(n)=\Theta(n)$

TC第7.4节练习2

- 再次强调：要用数学归纳法严格证明，不能只用递归树来估计。这是态度问题！

- 教材P180，$T(n)=\min(\ldots)+\Theta(n)$

TC第7章问题4

- (a) 如何严格证明？
  - 数学归纳法
  - loop invariant
- (b) Stack depth is Θ(n).
  - 单调增
  - 单调减行不行？
- (c) the worst-case stack depth is Θ(lgn).
  - 在小半区间上递归，在大半区间上尾递归
  - 找中位数作为pivot，行不行？

TC第8.1节练习4

- Hint: It is not rigorous to simply combine the lower bounds for the individual subsequences.

- $2^h \geq (k!)^{n/k}$

TC第8.2节练习4

- return C[b]-C[a-1]，有没有问题？

- if (a>0) return C[b]-C[a-1]
  else return C[b]

TC第8章问题2

- (b) Give an algorithm that satisfies criteria 1 and 3 above.
  - 类似quicksort的parititon（pivot=0）
  - counting sort行不行？
  - bucket sort行不行？
- (e) How to modify counting sort so that it sorts the records in place in O(n+k) time?

```
CC = arraycopy(C);
for (j=A.length; j>=1; j--)
   while (j != C[A[j]])
      if (C[A[j]]<j<=CC[A[j]]) {
         break;
      } else {
         swap (A[C[A[j]]], A[j]);
         C[A[j]]--;
      }
```

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	2	4	7	7	8

TC第9.3节练习7

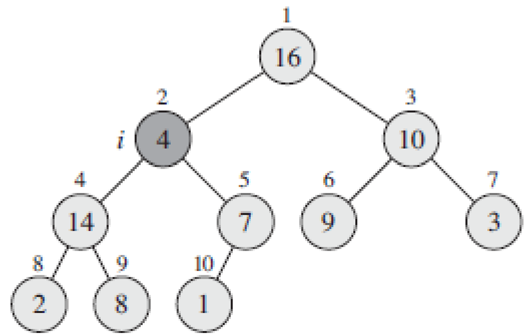
- O(n)-time algorithm determines the k numbers in S that are closest to the median of S.
  1. 找中位数 O(n)
  2. 每个数减去中位数、取绝对值 O(n)
  3. 选k次最小值 O(kn)=O(n)，行不行？

  4. 选第k小的值 O(n)
  5. 选所有比它小的值 O(n)

- 教材答疑和讨论
  - TC第6章
  - SB第2章

问题1：heap和heapsort

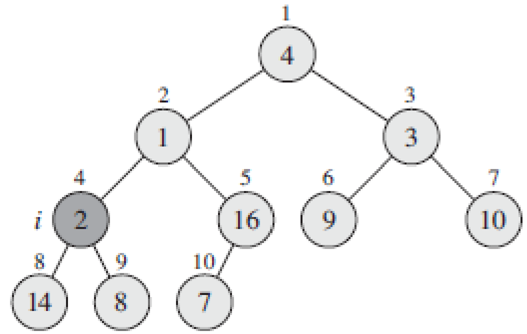
```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```



- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 它能给出它的运行时间吗？ $T(n) \le T(2n/3) + \Theta(1)$

问题1：heap和heapsort (续)

```
BUILD-MAX-HEAP(A)
  A.heap-size = A.length
  for i = ⌊A.length/2⌋ downto 1
      MAX-HEAPIFY(A, i)
```

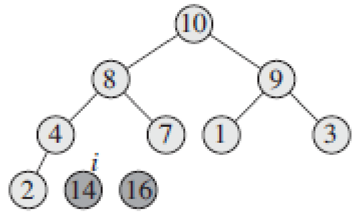


- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 它能给出它的运行时间吗？

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right)$$

问题1：heapsort和heapsort (续)

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size − 1
    MAX-HEAPIFY(A, 1)
```



- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 它能给出它的运行时间吗？

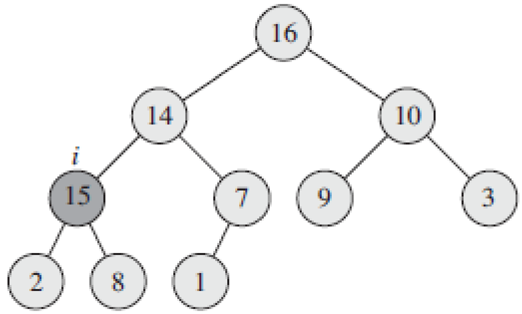
问题2：priority queue

```
HEAP-EXTRACT-MAX(A)
if A.heap-size < 1
    error "heap underflow"
max = A[1]
A[1] = A[A.heap-size]
A.heap-size = A.heap-size − 1
MAX-HEAPIFY(A, 1)
return max
```

- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 它能给出它的运行时间吗？

问题2：priority queue (续)

```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```



- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 它能给出它的运行时间吗？

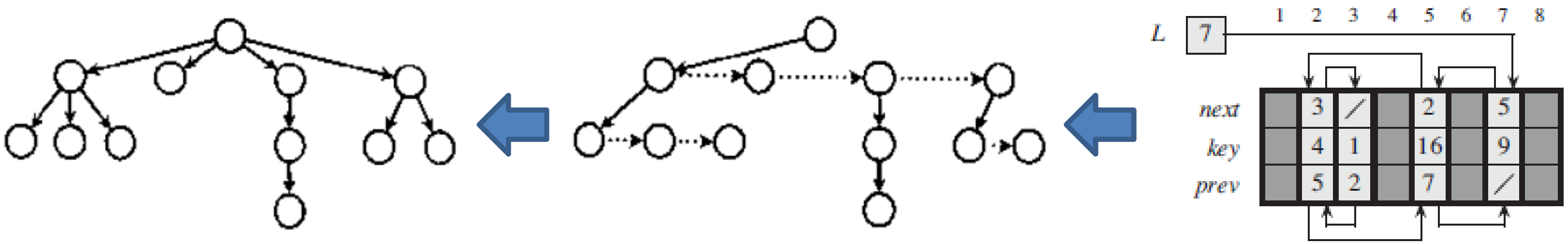
问题2：priority queue (续)

MAX-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)
```

- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 它能给出它的运行时间吗？

问题3：ADT

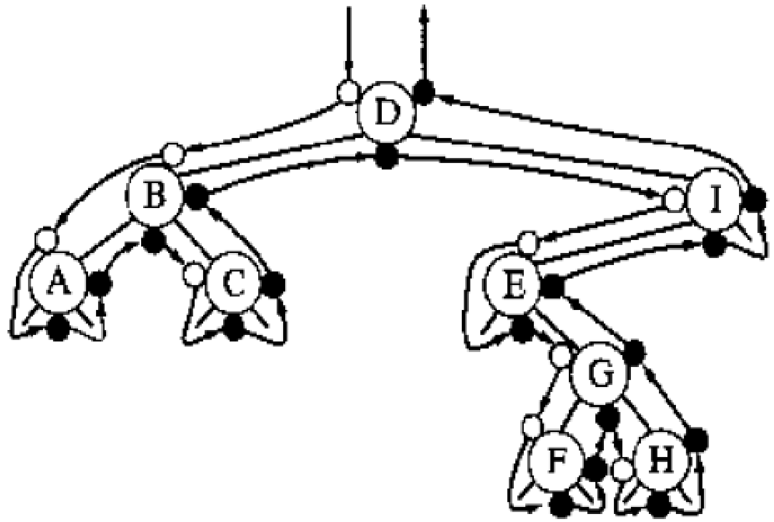
- priority queue ← heap ← array
- 

  结合这些例子，谈谈你对ADT的理解

- ADT和分层抽象对于算法的设计与分析有什么好处？
  – 设计：信息隐藏/数据封装、性能优化
  – 分析：正确性分析、性能分析

问题3：ADT (续)

```
void  traverse(BinTree T)
    if (T is not empty)
        Preorder-process root(T);
        traverse(leftSubtree(T));
        Inorder-process root(T);
        traverse(rightSubtree(T));
        Postorder-process root(T);
    return;
```



- 你理解binary tree的preorder/inorder/postorder了吗？
- 它们遍历的顺序分别是什么？

问题3：ADT (续)

- 你理解union-find (disjoint sets)了吗？

UnionFind create(int n)

Precondition: none.

Postconditions: If sets = create(n), then sets refers to a newly created object; find(sets, e) = e for $1 \leq e \leq n$, and is undefined for other values of e.

int find(UnionFind sets, e)

Precondition: Set $\{e\}$ has been created in the past, either by makeSet(sets, e) or create.

void makeSet(UnionFind sets, int e)

Precondition: find(sets, e) is undefined.

Postconditions: find(sets, e) = e; that is, e is the set id of a singleton set containing e.

void union(UnionFind sets, int s, int t)

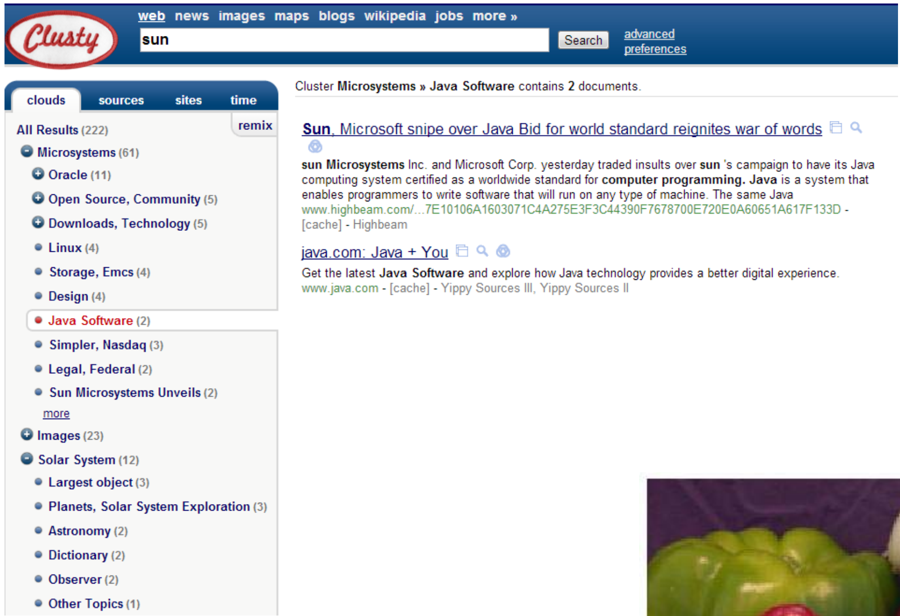
Preconditions: find(sets, s) = s and find(sets, t) = t, that is, both s and t are set ids, or "leaders." Also, $s \neq t$.

Postconditions: Let /sets/ refer to the state of sets before the operation. Then for all x such that find(/sets/, x) = s, or find(/sets/, x) = t, we now have find(sets, x) = u. The value of u will be either s or t. All other find calls return the same value as before the union operation.

- (linked) list
- (binary) tree
- stack
- queue
- heap
- priority queue
- union-find
- dictionary
- …

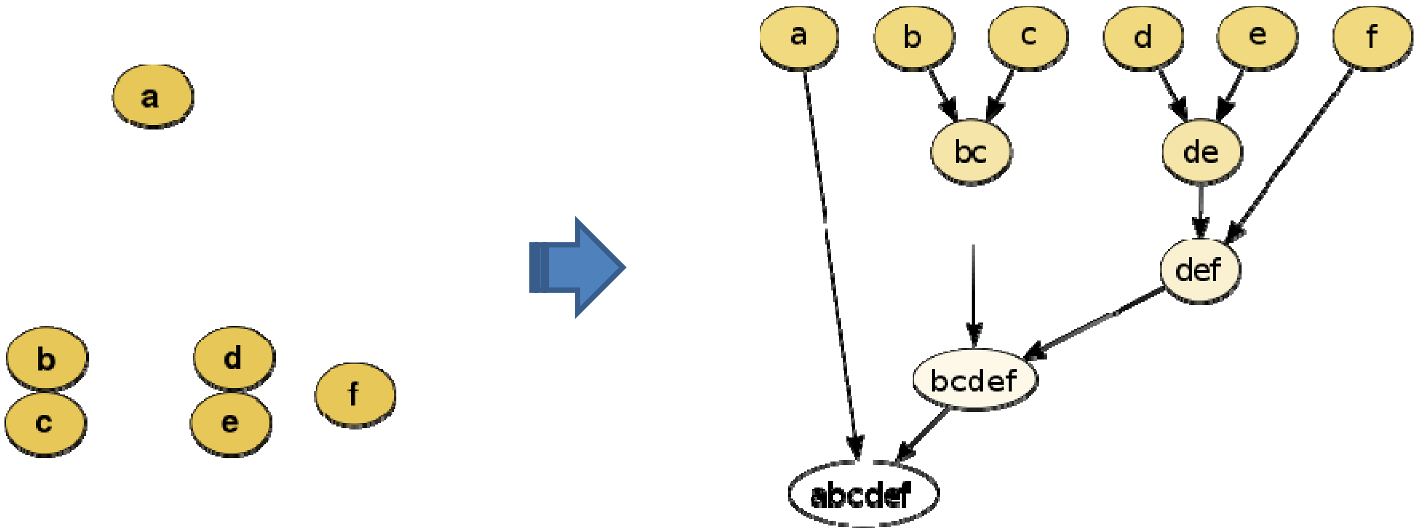
你准备好迎接一次综合挑战了吗？！

问题4： single-linkage agglomerative clustering



问题4：single-linkage agglomerative clustering (续)

- Agglomerative clustering
  - Each element starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- Single linkage
  - The distance between two clusters is computed as the distance between the two closest elements in the two clusters.



问题4： single-linkage agglomerative clustering (续)

- 请给出你的实现，使得以下操作较为高效
  - 生成hierarchy
  - 在生成过程中，用户可以
    - 浏览生成的hierarchy的结构
    - 监测任意element所属的cluster
    - 回退到任意步骤手工调整结果再继续

- priority queue + union-find

- binary out-tree
- union-find
- stack