CS211. Prelim 2, Spring 2004. Sample answers

1. (a) Worst-case time for n = k+1-h: O(n*n);
Average-case time: O(n log n)

(b)
```
public static void quicksort(int[] b, int h, int k) {
    if (h+1 – k  < 10)
        { insertionsort(b, h, k); return; }
    medianOf3(b, h, k); // It is ok to leave this out
    int j= partition(b, h, k);
    // { b[h..j–1] <= b[j] <= b[j+1..k] }
    if (j – h <= k – j) {
        quicksort(b, h, j–1);
        quicksort(b,j+1, k);
    }
    else {
        quicksort(b,j+1, k);
        quicksort(b, h, j–1);
    }
}
```

> Sorting the larger partition using a tail-recursive call reduces space to O(log n) if the language implements tail recursion nicely.

2. To save space, we omit the method specs

```
public DList(){
  sentinel= new DNode(null, null, null);
  sentinel.next= sentinel;
  sentinel.prev= sentinel;
  current= sentinel;
}

public void insert(Object i){
  DNode temp= new
     DNode(current, i, current.next);
  current.next.prev= temp;
  current.next= temp;
  current= temp;
}

public void remove(){
  if (current == sentinel)
    throw new NoSuchElementException();
  current.next.prev= current.prev;
  current.prev.next= current.next;
  if (current.prev != sentinel)
    current= current.prev;
  else current= current.next;
}

private class DNode {
  public Object value;  // Value in the node
  public DNode next;    // next node
  public DNode prev;    // previous node
  /** Constructor: a node with value v,
      successor n, and precedessor p */
  public DNode(DNode p, Object v, DNode n) {
    value= v; next= n; prev= p;
  }
}
```

3.
```
public static LNode inorder (TNode root,
                             LNode head) {
    if (root.left != null) {
      head= inorder(root.left, head);
    }
    head.next= new LNode();
    head= head.next;
    head.item= root.data;
    if (root.right != null) {
      head= inorder(root.right, head);
    }
    return head;
  }
```

4a. Function f(n) is O(n) iff there are positive constants c and n0 such that f(n) <= c*n for n ≥ n0.

or

f(n) is O(n) iff there is a positive constant c uch that f(n) <= c*n for all but a finite number of positive n.

4b. The method of question 3 is O(n) for a tree with n nodes. Each recursive call processes 1 node of the tree, and all the operations in the method body (except the recursive calls themselves) take constant time k (say). Since n calls are made in total, the time is k*n for some positive constant k.

4c. A heap is a binary tree that satisfies:

(1) T is complete, i.e. with the nodes numbered in breadth-first order, if node n exists, so do nodes 0..n-1.
(2) The value of each node n of T is at least the values of its children.

Note: we have specified a max-heap; in a min-heap, the value of each node would be at most the value of its children.

5a. Suppose we are looking for object ob in a hashtable h of size s. If object ob hashes to x then **linear probing** says to probe cells with index x, (x+1)% s, (x+2) % s, (x+3) % s, … until ob or an empty cell is found.

5b. (Without having to draw the diagram)
after a: {null, (1,T), (8,T), null, (11,T), null, (13,T)}
after b: {null, (1,T), (8,F), null, (11,T), null,(13,T)}
after c: {null, (1,T),(15,T), null,(11,T), null,(13,T)}
or {null, (1,T),(8,F),(15,T),(11,T), null,(13,T)}