UNIVERSITY OF SOUTHAMPTON COMP1009W1

SEMESTER 2 EXAMINATION 2005/2006

DATA STRUCTURES AND ALGORITHMS

Duration: 120 mins

Answer THREE *questions*

This examination is worth 85%. The tutorials were worth 15%.

University approved calculators MAY be used.

Author: Adam Prügel-Bennett

Question 1

The algorithm for bubble sort to sort an array $\boldsymbol{a} = (a_0, a_1, \ldots a_{n-1})$ is given by

```
BubbleSort(a)
  for i←0 to n-2
    for j←0 to n-2-i
      if $a_{j+1} < a_j$
        swap $a_j$ and $a_{j+1}$
      endif
    endfor
  endfor
```

(a) Write a Java method to performing bubble sort on an array of integers.

Students haven't been shown details of bubble sort although they had a very quick description of it. Simple exercise showing understanding of pseudo-code.

```
    public static void sort(int[] a) {
      for (int i=0; i<a.length-1; i++)
         for (int j=0; j<a.length-i-1; j++)
            if (a[j+1]<a[j]) {
               int swap = a[j];
               a[j] = a[j+1];
               a[j+1] = swap;
            }
   }
```

(7 marks)

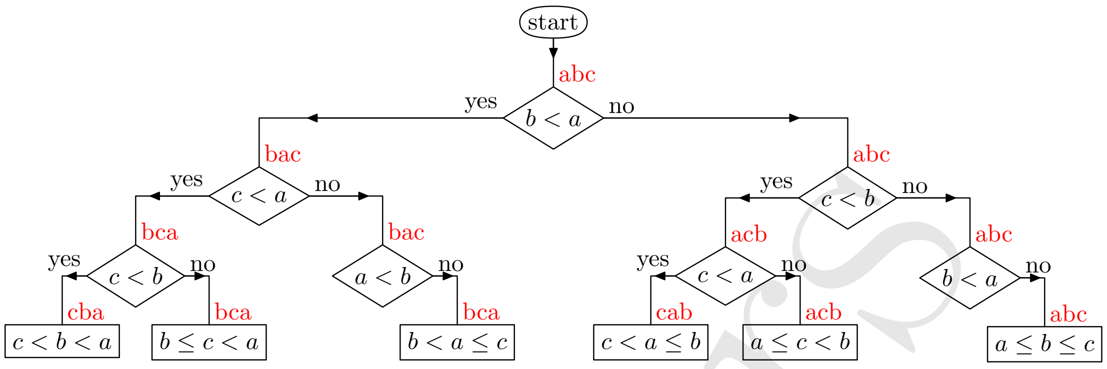
(b) What is the time complexity of bubble sort? Explain your answer.

Test simple loop counting. **The time complexity is $O(n^2)$ since there are two nested "for loops" which both are $O(n)$ long.**

(3 marks)

(c) Draw a decision tree showing how bubble sort would process an array of three elements (a, b, c).

Students have seen a decision tree for insertion sort. This is therefore quite a challenging question.



This is slightly unusual in that some of the queries are pointless.

(13 marks)

(d) Prove a lower bound on the time complexity for any sort algorithm that uses binary decisions?

This proof was given in the lectures, but it conceptually one of the hardest proofs in the course. The proof goes as follows.

- **Any sort using binary decisions can be represented as a binary decision tree.**
- **The leaves of the tree represents possible end states of the algorithm.**
- **There has to be an end state for every possible permutation of the original list, otherwise two permutations of a list would be sorted in the same way, but this would mean one of those permutations was incorrectly sorted (assuming the elements of the list are all different).**
- **There are $n!$ possible permutations of a list with n elements.**
- **A binary decision tree with $n!$ leaves must have depth at least $\log_2(n!) = \Theta(n \log(n))$.**

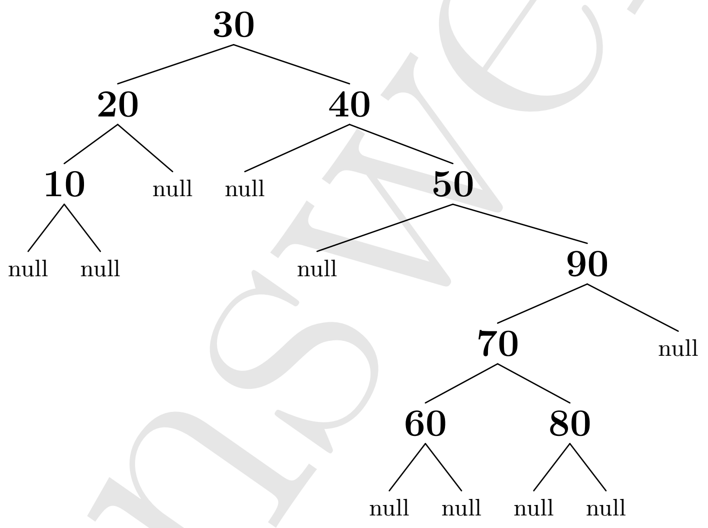
(10 marks)

Question 2

(a) Draw the binary search tree which results from inserting the following list of numbers into the tree

$$30, 40, 20, 50, 90, 10, 50, 70, 60, 80.$$

This should be fairly easy.

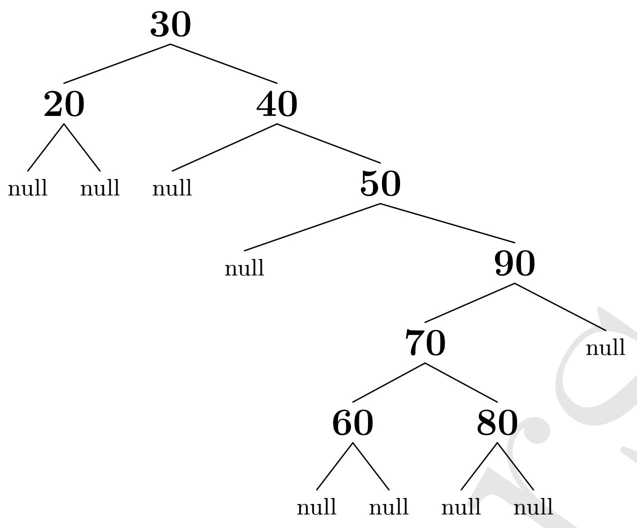


(5 marks)

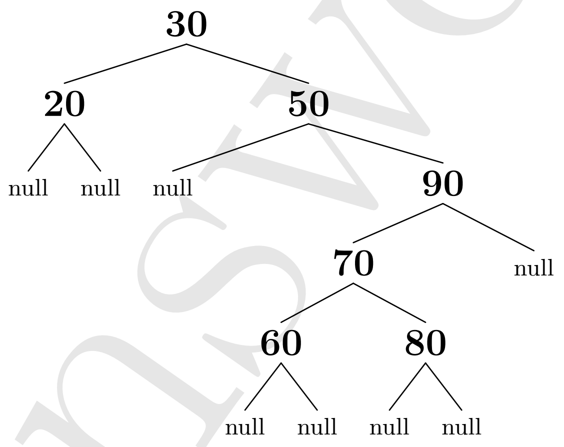
(b) Explain what happens when you delete 10, 40 and 30?

This is more challenging. Especially removing the 30 which has two children.

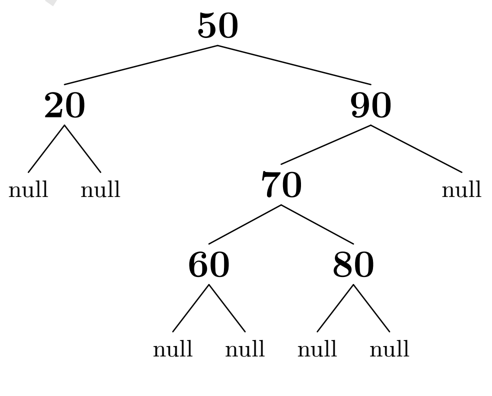
Removing 10 is straightforward as it has not children. We simply replace it with `null`.



Removing 40 is also straightforward. This has one child. We make the parent point at the child. The only tricky part of the implementation is determining if the current node is the left or right node of its parent.



The hard part is to remove a node with two children. Here we remove the head of the tree 30 and have to replace it with its immediate successor which in this case is 50. The immediate successor will have at most one child so we remove that node using the techniques we described above.



TURN OVER

(10 marks)

(c) Derive the typical and worst case time complexities for insertion. Briefly explain the strategies used to reduce the worst case time complexity.

This tests an understanding of the theoretical aspects underlying trees.

The time complexity for insertion depends on the depth of the search tree, which in turn depends on the number of entries n. In the typical case the tree is approximately balanced and the depth of any branch is $O(\log(n))$, which is therefore the typical case time complexity.

In the worst case the tree is completely unbalanced. In which case the tree turns into a linked list. This would happen for example if the entries were inserted in order. In this case each insertion would take $O(n)$ operations.

To avoid poorly balanced trees, the trees are rebalanced using rotation operations. The usual strategy is to maintain some measure of how unbalanced the tree becomes, when the tree is too unbalance a rotation takes place to re-balance the tree. Two common mechanisms to measure how poorly balanced a tree is are AVL trees and red-black trees. Both strategies ensure that the depth of the tree is $O(\log(n))$.

(10 marks)

(d) Explain why binary search trees are commonly used to represent sets. What alternative data structure is also used for this purpose? How do these data structures compare?

The requirement of a set is that it has fast search, insertion and deletion. A binary search tree performs all these operations in $O(\log(n))$ time.

The other commonly used data structure for representing sets are hash tables. This has constant time search, insertion and deletion, although a hash function evaluation is required and there is a hidden cost of collision avoidance when the table becomes heavily loaded.

One feature of binary search trees which may be an advantage is that the iterator visits each element in order.

(8 marks)

Question 3

(a) What are the properties of a heap?

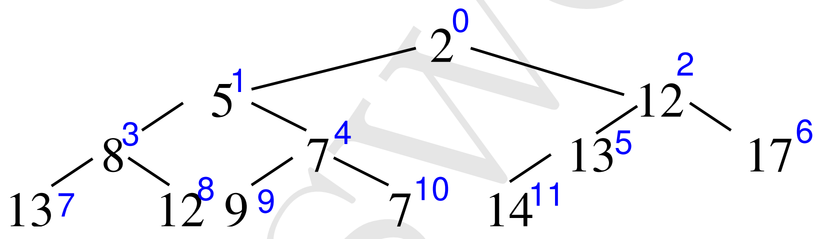
A heap is a complete binary tree (i.e. all the layers above the lowest layer are full) where the leaves on the lowest layer are as far left as possible. In addition, each node in the tree is less than any of its children.

(5 marks)

(b) Explain in detail how a heap is stored in memory?

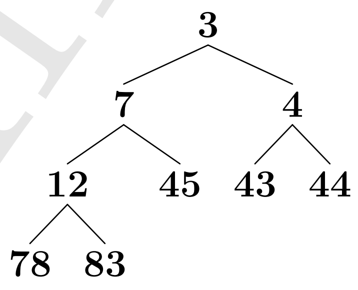
A heap is stored as an array. The root of the array is element 0. The children of element i are $2i+1$ and $2i-1$.

0	1	2	3	4	5	6	7	8	9	10	11
2	5	12	8	7	13	17	13	12	9	7	14



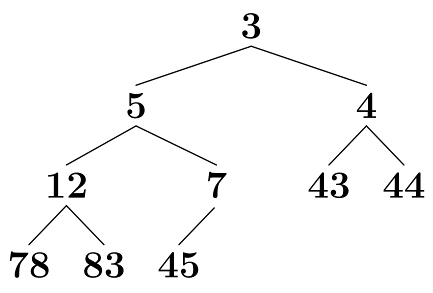
(5 marks)

(c) Consider the following heap



Explain what happens when we add an new element 5. Draw the new heap.

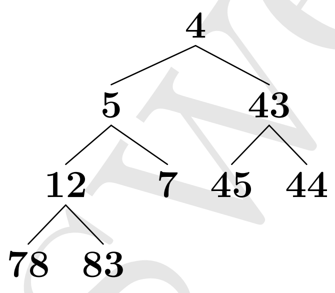
The element gets added to the bottom of the stack (i.e. left child of 45) and percolated up. That is it gets compared with its parents and swapped if it less than its parent. This happens iteratively to leave



(5 marks)

(d) Explain what happens when we remove the minimum element from the heap.

The minimum element, 3, is returned and replace with the last element in the heap (i.e. element 45). This is percolated down the heap by swapping with its smallest child iteratively. In this case it is swapped with 4 and 43.



(5 marks)

(e) Explain how a heap is used to perform sort? Is it a stable sort?

Heap sort requires putting elements from an array on a heap and taking them off into a new array.

```
public static <T> void sort(List<T> aList)
{
   PQ<T> aHeap = new HeapPQ<T>(aList.size());
   for (T element: aList)
      aHeap.add(element);

   aList.clear();
   while(aHeap.size() > 0)
      aList.add(aHeap.removeMin());
}
```

It won't be stable as their is no guarantee that the order of two identical elements will be preserved by the heap.

(5 marks)

(f) Prove the time complexity for heap sort.

We have to add n elements and remove n elements from the heap. The add and remove operations take at worst case the depth of the tree which will be $\lceil \log_2(n) \rceil$. Thus heap sort is $O(n \log(n))$

(4 marks)

(g) Give two other applications of heaps.

Heaps are used to implement priority queues. These are often used as part of a greedy algorithm (e.g. for performing Huffman coding). Priority queues are also frequently used in real time simulations.

(4 marks)

Question 4

(a) Assume we are hashing a two digit number d_2d_1 and we have a hash function

$$hash(d_2d_1) = (d_2 + 3d_1) \bmod 10.$$

Calculate the hash addresses for 12, 14, 31, 83, 17, 54, 22.

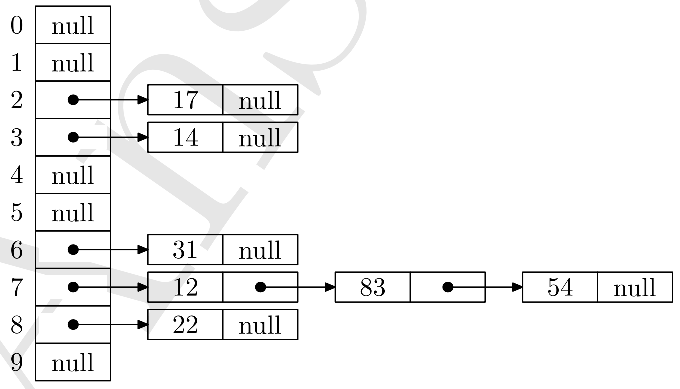
Simple calculation.

Number, x	**12**	**14**	**31**	**83**	**17**	**54**	**22**
Hash(x)	**7**	**3**	**6**	**7**	**2**	**7**	**8**

(3 marks)

(b) Show how the numbers above would be stored in a hash table of size 10 using separate chaining. Calculate the average number of comparisons needed to find an entry.

Tougher question, requiring students to have a solid understanding of how separate chaining works.

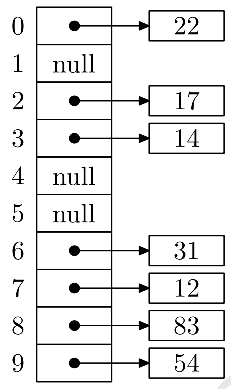


We have to count how many comparisons we have to make for each entry. The average number of comparisons need to find an entry is

$$\frac{5 \times 1 + 1 \times 2 + 1 \times 3}{7} = \frac{10}{7} \approx 1.43.$$

(7 marks)

(c) Show how the numbers above would be stored in the same size hash table using linear probing and calculated the average number of comparisons needed to find an entry.



The average number of comparisons need to find an entry is

$$\frac{4 \times 1 + 1 \times 2 + 2 \times 3}{7} = \frac{12}{7} \approx 1.71.$$

(7 marks)

(d) Explain the problem specific to linear probing and give an alternative method to implement a hash table using open addressing.

This is a test of knowledge taught in the course.

Linear probing leads to clustering of the the probes in which a contiguous section of the hash table is full. In the example above, locations 6-7-8-9-0 form such a cluster. Any time a new collision happens in this cluster the cluster size increases by one. This new entry will lie at the end of the cluster which may be some distance from where the collision occurred.

Two common approaches to overcome this problem are double hashing and quadratic probing. A description of either is sufficient. **In quadratic probing, at each collision a new entry in the hash table is chosen a distance 1, 4, 9, 16, etc. from the hash address.**

(7 marks)

(e) Explain why deletions cause problems for open hashing. What is the most commonly used fix?

Deletions cause a space in the hash table at a location which was once occupied. If this entry had caused a collision with some entry x then x's location will be elsewhere in the table. After the deletion, this collision will no longer occur so that the entry x will no longer be found when searched for.

TURN OVER

The usual fix is to replace the deleted entries by a dummy entry which indicates that a collision could have happened at this position. This dummy entry is considered in search, but ignored when iterating through the hash table and can be replace by a new insertion.

(7 marks)

(f) What hashing scheme is used by `HashSet<T>` in the Java collection class and why?

`HashSet<T>` uses separate chaining because it has better performance when the loading factor of the hash table becomes large, thus reducing the need to resize the hash table so often. In addition, separate chaining has no problems with deletion.

(2 marks)

END OF PAPER