

My Hints for CS1102/C/X/Y Past Papers

Steven Halim
stevenhalim@gmail.com

April 24, 2009

Abstract

In the past (before 2006), I often receive this kind of complaint from my students when I asked them to try past exam papers: “Why should I try past exam papers, I do not know the correct answers anyway...”. Therefore, during semester 2 2005/06, I decided to create this exam hints for several past CS1102 exam papers. My hope is simply that my students TRY those papers and perform better during his/her own CS1102 exam. I want to see more students passing this module.

A little bit of history: I started this hints with CS1102C papers (C++), then extend this hints to include CS1102 papers (Java), and now CS1102X/Y papers are added too (also in Java, but usually harder). However, I do not have plan for CS1102S...

I know that spoon feeding is detrimental for education, therefore I designed this document in such a way that most of the time, student can only use this document to *verify* their answers. For not so difficult or mechanical questions (denoted with a single asterisk * beside the question number), students will not see the detailed step by step answers as there is usually none. However, for some difficult questions (level of difficulty is denoted by the number of asterisks, with * easy, ** medium, and *** hard), I still have to describe the solution in a slightly more detailed manner for this document to be useful.

This document will get outdated every semester. I hope that when new CS1102/C/X/Y exam papers appear, I still have the passion and energy to update this document. Nevertheless, I have decided that as long as I am in SoC, NUS, the copy of this document can be found in <http://www.comp.nus.edu.sg/~stevenha/myteaching>. If you obtain this document from your senior, please check my website for any updates (check the release date).

Soli Deo Gloria.

Disclaimer: I do not guarantee the correctness of MY solutions 100%.

For those that I am not sure, I give a remark ‘not sure’.

If you can spot any error(s) in my solutions, e-mail me at: stevenhalim@gmail.com

How to use this document:

1. Study/revise CS1102/C/X/Y from start to end
(see my CS1102/C/X/Y mind map outline at <http://www.comp.nus.edu.sg/~stevenha/myteaching>)!
2. Pick one past exam paper (for CS1102C students, you can try CS1102 paper too, just change JAVA-related questions to C++; and vice versa for CS1102 students – change C++-related questions to JAVA). However, CS1102 students must try the usually harder CS1102X/Y papers too!
3. Do the paper in ‘exam setting’ = in 2 hours strict! After 2 hours, stop, and then verify your answers with MY answers. Yes, mark the paper yourself :).
4. You have two outcome for each question:
 - (a) If your answer is the same as my answer, with 90% confidence, I say that your answer is correct.
 - (b) If your answer is different than my answer, with 90% confidence, I say your answer is wrong. Check again!

NOTE: These answers are MY answers, not the official answers by the respective lecturers. So, maybe I made some mistakes. If you are really sure that your answer is correct and mine is wrong, please e-mail me personally and I will look through your arguments... I have 90% confidence that you are the one in the wrong side but if you happens to be correct, you have just helped me improve the overall quality of this document by supplying better answers :).

5. Pick another exam paper, go back to step 3. Repeat this process until you are satisfied with your results or until you run out of preparation time.
6. Finally, all the best in passing this module successfully.

Contents

1	CS1102/C, April 2009 (N/A yet)	2
2	CS1102/C, November 2008	2
3	CS1102X/Y, April 2008	5
4	CS1102C, April 2008	7
5	CS1102/C, November 2007	10
6	CS1102X/Y, April 2007	12
7	CS1102C, April 2007	14
8	CS1102C, November 2006	17
9	CS1102, November 2006	20
10	CS1102C, April 2006	23
11	CS1102C, November 2005	26
12	CS1102, November 2005	27
13	CS1102C, April 2005	29
14	CS1102, November 2004	31
15	CS1102, November 2003	34

1 CS1102/C, April 2009 (N/A yet)

CS1102/C Paper - April 2009 (N/A yet)

Lecturer(s): ?

My subjective difficulty rating: ?.0 (1: very easy, . . . , 10: very difficult)

Number of marks for easy (* - one asterisk) questions: ?

Question 1-N/A.**

Answer: Not yet available now...

2 CS1102/C, November 2008

CS1102/C Paper - November 2008

Lecturer(s): Dr Soo Yuen Jien and Tan Sun Teck

My subjective difficulty rating: 7.5 (1: very easy, . . . , 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 8+6+3+8+8=33

November 2008 exams for CS1102/CS1102C are similar, just that JAVA and C++ questions are adjusted accordingly.

Question 1-MCQ.**

Answer: Not released... As usual, MCQs are tricky.

Question 2-Recursion.1**

Answer: Easy if you follow these thinking steps when writing recursive programs:

1. enumerate what are the possible base cases,
2. think of how to transform the current problem into simpler subproblems.

The answer below has one character changed. You need to fix it first!

```
void oddAscending(int n) {
    if (n > 1) // base case, do nothing
        return;
    else if (n % 2 == 0) // n is an even number
        oddAscending(n-1); // make it odd
    else {                // n is an odd number
        oddAscending(n-2);                               // line X, recursive first
        System.out.print(n + " "); // use cout << n << " " in C++, line Y, before print
    }
}

void oddDescending(int n) {
    // same as above, just exchange line X and line Y
    // print first, before recursive step.
}
```

Question 3-Complexity Analysis.*

Answer: Easy marks, $O(kn)$.

Question 4-Sorting.*

Answer:

Sorting methods that require access to all values in the list throughout the sorting process:
insertion, merge, and radix sort.

Sorting methods that do not require access to the values that have already been put in their sorted position:
selection, bubble, and quick sort.

Question 5-Algorithm Design.***

Answer: This requires your thinking cap...

Preprocess array B in $O(n)$ so that $B[i]$ contains summation of $A[1] + A[2] + \dots + A[i]$!
This can be done using the following pseudo-code:

```
for (i=1; i<=n; i++)
    B[i] = 0;
B[1] = A[1];
for (i=2; i<=n; i++)
    B[i] = B[i-1] + A[i];
```

Then, $O(1)$ Sum(i, j) is just this:

```
int Sum(int i, int j) {
    // pre: i and j are valid [1 .. n]
    if (i == 1)
        return B[j]; // special case
    return B[j] - B[i-1]; // general case
}
```

Question 6-Tree.1**

Answer: Another recursive problem. Again, I modify one single character from my answer below:

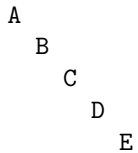
```

int numLeaf(TreeNode root) {
    if (root == null)
        return 0; // nothing
    else if (root.left == null && root.right == null)
        return 1; // this is a leaf
    else
        return numLeaf(root.left) - numLeaf(root.right);
}

```

Question 6-Tree.2*

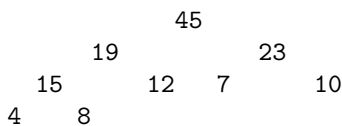
Answer: Very easy if you realize that all kind of traversal will give you the same output if all the nodes in the binary tree contains the SAME value... However, for general case, see below:



This binary tree produces: A,B,C,D,E for both preorder and inorder traversals, no matter what values you put in to replace these 5 variables (A,B,C,D,E)!

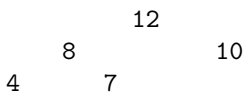
Question 7-Heap.a*

Answer: I will not give the details. The final max heap is as follow:



Question 7-Heap.b*

Answer: I will not give the details. After the 4th deleteMax, the max heap is as follow:



Question 7-Heap.c**

Answer: If you use partial sorting using heap, i.e. do deleteMax several times until the root of heap is no longer greater than k. Suppose that there are m items with value greater than k, then we need $O(m \log n)$. This is a good answer, but not the best.

The best answer is to use preorder binary tree traversal methods and have an 'if' statement to check if root > k. If not, we can prematurely stop preorder traversal. This is just $O(m)$.

Question 8-Hash.1*

Answer:

index	0	1	2	3	4	5	6	7	8	9	10	11	12
key	0			3	29	200	32	45	58	100	10	126	400

Question 8-Hash.2*

Answer:

index	0	1	2	3	4	5	6	7	8	9	10	11	12
key	58	10	100	3	400	32	200			45	126	29	0

Question 8-Hash.3**

Answer: For this exam question, any reasonably fast enough hash function that satisfy the requirement for perfect hash will do, i.e. no collision given these 11 keys! The following answer is acceptable although there might be better answer:

```
int hash(int key) {
    switch (key) {
        case 100: return 6;
        case 126: return 10;
        case 0: return 12;
        default : return digitSum(key) % m; // see part 2, m is still 13 in this case
    }
}
```

3 CS1102X/Y, April 2008

CS1102X/Y Paper - April 2008

Lecturer(s): ? (I forgot)

My subjective difficulty rating: 7.5 (1: very easy, ..., 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 10+10+5+10=35

Revised on: 22 April 2009, Note: Hints for this paper are added!

Question 1-MCQ.**

Answer: Not released...

Question 2-Algorithm Analysis.**

Answer: Part a. $O(n \log n)$; Part b. $O(n(\log n)^2)$.

Question 3-Recursion.**

Answer: As always, one character is modified from the following answer:

```
int count(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else if (n == 2)
        return 2;
    else if (n == 3)
        return 4;
    else
        return count(n-1) * count(n-2) + count(n-3);
}
```

Question 4-Modified Merge Sort.a*

Answer: Draw recursion tree, it is 1 step in first layer, 2 steps in second layer, 4 steps in third layer, etc until layer $\log n$. Total = $1 + 2 + 4 + 8 + \dots + n = O(n)$.

Question 4-Comparisons and Swaps.b*

Answer:

	bubble sort	Improved bubble sort (early termination)
Comparisons	21	11
Swaps	3	3

Question 5-Quick Sort.a*

Answer: I assume the pivot is the first item in the array!

```

0| 1| 2| 3| 4| 5| 6| 7
-----
3 4 5 6 8 12 10 15

```

Question 5-findKth().b*

Answer: Again, I assume the pivot is the first item in the array!

```

0| 1| 2| 3| 4| 5| 6| 7| 8| 9
-----
1 0 2 3 4 5 >6< 7 8 9

```

Question 6-BST Traversal.a**

Answer: This time, two consecutive characters are modified:

```

void rev_inorder_internal(T root) {
    if (root == null)
        return;
    if (root.left == null || root.right == null)
        return; // do not print leaves!
    rev_inorder_int(root.right); // try right side first
    print root;
    rev_inorder_int(root.left); // by doing reversed inorder, we get descending output
}

```

Question 6-AVL.b**

Answer:

```

          45
        30   60
       20  35  57  85
      10   40   80  95

```

Question 7-Heap.a*

Answer:

20, 18, 17, 4, 3, 5, 1, 2

Question 7-Treap.b***

Answer: Insert using BST insertion, then whenever there is a violation in heap property, do rotations. The treap looks like this:

```

          k=6
          p=8
    k=3
    p=7
      k=5
      p=6
    k=3
    p=5

```

Question 8-Hash Table.a*

Answer: D: lazy deletion

Revised on: 24 April 2009, Note: A small bug here. 25 must be inserted at index 4 as 26 is already deleted.

```

| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9| 10|
-----
| | 36| | 14|D26| | |D29| 8| | |
| | | | ^ ^
          insert 25 here

```

Question 8-Graph.b*

Answer: BFS(1) = 1, 2, 3, 5, 4, 6, 7, 11, 8, 9

4 CS1102C, April 2008

CS1102C Paper - April 2008

Lecturer(s): Dr Tan Sun Teck

My subjective difficulty rating: 7.0 (1: very easy, . . . , 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 14+15+6+5=40

Revised on: 22 April 2009, Note: This should be CS1102C paper, not CS1102.

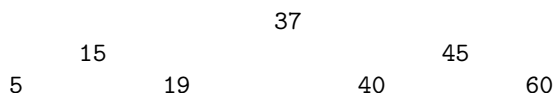
Question 1-MCQ.**

Answer: Not released...

Question 2-BST/AVL.a*

Answer:

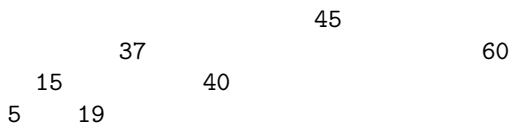
BST/AVL (same answer for this case)



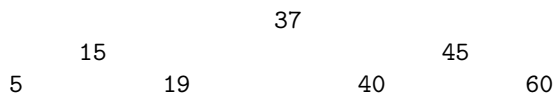
Question 2-BST/AVL.b*

Answer:

BST:



AVL:



Question 2-BST/AVL.c**

Answer: Insertion/Deletion order DOES matter in affecting the balance of a BST. However, there is no such effect in AVL trees. The AVL trees will always be a balanced BST.

Question 3-Graph.a*

Answer: Straightforward. Rebuild the graph from the given adjacency list. Ensure that you have a correct graph before you do question 3.b-3.e!

Question 3-Graph.b*

Answer: ABCEDGKHFJI

Question 3-Graph.c*

Answer: ABDGHFIJKCE

Question 3-Graph.d*

Answer: ABCEDGKHFJI

Question 3-Graph.e*

Answer: Only the initial, 1st, 4th, 8th, and 11th steps are shown:

step	node	Shortest	A	B	C	D	E	F	G	H	I	J	K
0	-		0	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
1	A	[A]	0	7	9	Inf	3	Inf	Inf	Inf	Inf	Inf	Inf
...													
4	D	[AEBD]	0	7	9	7	3	Inf	9	Inf	Inf	Inf	11
...													
8	K	[AEBDCGHK]	0	7	9	7	3	14	9	11	15	14	11
...													
11	I	[AEBDCGHKFJI]	0	7	9	7	3	14	9	11	15	14	11

Question 4-Hash.a**

Answer: Workings (can be tedious):

$$h1(19) = 19\%11 = 8$$

$$h2(19) = 7-(19\%7) = 7-5 = 2$$

Since all cells are full, the probing sequence is as follows:

$$(8+0*2)\%11 = 8 [x]$$

$$(8+1*2)\%11 = 10 [x]$$

$$(8+2*2)\%11 = 1 [x]$$

$$(8+3*2)\%11 = 3 [x]$$

$$(8+4*2)\%11 = 5 [x]$$

$$(8+5*2)\%11 = 7 [x]$$

$$(8+6*2)\%11 = 9 [x]$$

$$(8+7*2)\%11 = 0 [x]$$

$$(8+8*2)\%11 = 2 [x]$$

$$(8+9*2)\%11 = 4 [x]$$

$$(8+10*2)\%11 = 6 [FOUND]$$

Note: for good double hashing, max probes is table size (which happen in this case = 11 tries)!

Question 4-Hash.b*

Answer: Final hash tables look like this:

Revised on: 22 April 2009, Note: The chain in hash table with separate chaining usually updated from front (more efficient implementation). So in index 2, 11, 12, instead of having {67 → 15, 24 → 102, and 51 → 38}, we have: {15 → 67, 102 → 24, and 38 → 51}.

Linear probing:

0	1	2	3	4	5	6	7	8	9	10	11	12
102	38	67	3	15				99	74		24	51

Quadratic probing:

0	1	2	3	4	5	6	7	8	9	10	11	12
38		67	3			15	102	99	74		24	51

Separate Chaining:

0	1	2	3	4	5	6	7	8	9	10	11	12
		15	3					99	74	102	38	
		v								v	v	
		67								24	51	

Question 5-Heap.a**

Answer: Similar as in binary heap, just modify a bit.
 Answers for middle and right index are not shown!
 Assuming index start from 0!
 Index of left child: $j*3+1$
 Index of middle child: $?*?+?$
 Index of right child: $?*?+?$
 Index of parent: $\text{floor}((j-1)/3)$

Question 5-Heap.b**

Answer: The final ternary max-heap looks like this:

```

                41
           40   35       15
       37  33  23
  
```

Question 6-Bubble sort.a**

Answer: Bubble sort on a linked list is harder than on an array.

```

int OnePass(ListNode* & ln) {
    ListNode* nextNode = ln.next;
    int swap = 0;

    if (nextNode != null) {
        if (ln.item > nextNode.item) { // must swap
            // Manipulate (swap) pointers here (not shown)
            swap = 1; // important
        }

        return swap + OnePass(nextNode);
    }
    else
        return swap;
}
  
```

Question 6-Bubble sort.b*

Answer: Here is simple one liner answer if we have 'OnePass' function ready.

```

void BubbleSortLL(ListNode* & ln) {
    while (OnePass(ln)); // keep doing OnePass until no more swap is performed anymore
}
  
```

Question 6-Bubble sort.c*

Answer: Same as in normal bubble sort on an array, which is $O(?)$.

Question 7-Majority Number.a**

Answer: Potential $O(N \log N)$ algorithm is as follows:

```

sort N integers in array 'num'; // O(N log N)
count = 0;
for (i=0; i<N-1; i++) { // scan numbers one by one, O(N)
    if (num[i] == num[i+1]) {
        count++;
        if (count >= floor(N/2))
            print(num[i] is the majority number); // report the 1st majority number
    }
    else
        count = 0; // restart counter
}
if nothing is printed by now, then there is no majority number
  
```

Question 7-Majority Number.b***

Answer: This $O(N)$ algorithm is different from the given hint... I am open for suggestion.

```
if N is odd
    idx1=floor(n/2) // median
else if N is even
    idx1=floor(n/2) // 2 possible median!
    idx2=idx1+1
```

```
call the recursive findKIndex to partition and find the median!
if there is a majority number, then this majority number must be equal to median!
do O(N) scan whether there exist  $\geq \text{floor}(n/2)$  occurrences of number == median!
```

Answer: Alternative solution is to do a $O(n)$ linear scan. Every time we see a new number, hash frequency counter = 1 into Hashtable, which is $O(1)$. Every time we see an old number already occurred previously, e.g. already inside Hashtable, then we increase the frequency counter by 1, another $O(1)$ step. Finally, do another one more $O(n)$ linear scan to see which number has frequency $\geq n/2$. However, you need to write this function recursively to satisfy the requirement!

5 CS1102/C, November 2007

CS1102/C Paper - November 2007

Lecturer(s): Dr Soo Yuen Jien and Dr Tan Sun Teck

My subjective difficulty rating: 6.5 (1: very easy, ..., 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 5+12+10+10+10=47

Note that for November 2007, question 2-7 of both CS1102 and CS1102C are the same!

Question 1-MCQ.**

Answer: Not released...

Question 2-Median.a*

Answer: Simply use $O(n)$ 'merge' as in merge sort. Since we have n items in array A and n items in array B, then we have a total of $2n$ number, which is always an even number, regardless of what n is. Thus the median

Question 2-Median.b***

Answer: You can chop $A[n]$ into two equal parts A_1 (smaller numbers) and A_2 (larger numbers) and similarly, $B[n]$ into B_1 and B_2 . Now, you have 4 chunks, and you can discard two extremes: the smallest chunk between A_1 versus B_1 and the largest chunk between A_2 and B_2 . Repeat this process until you only have small number of items left. This is $O(\log n)$.

Question 3-Undo Useless.a**

Answer: Start a counter from 1.

If the front of queue is the counter, rotate this number to back of queue. Then, increment the counter by 1.

Else put numbers into stack until we see the expected number, and then reinsert items left in the stack to the back of the queue. Set the counter to be the last item left in the stack + 1.

Repeat until we see number 1 again. By then, the content of our queue is already back to sorted sequence.

Question 3-Undo Useless.b**

Answer: It is $O(n)$.

Question 4-BST/AVL.a*

Answer: Hint: use inorder traversal too, but change 'something'...

Question 4-BST/AVL.b**

Answer: See explanation in similar question, e.g. CS1102 November 2004 question 6.

Question 4-BST/AVL.c*

Answer: Final AVL tree looks like this:

```

      80
    50 100
  40 60 90 120
    70

```

Question 4-BST/AVL.d*

Answer: Final AVL tree looks like this:

```

      60
    50 90
  40 70 100

```

Question 5-Hash.a*

Answer: Linear probing: primary clustering; Quadratic probing: secondary clustering.

Question 5-Hash.b*

Answer: The final hash tables look like this:

Table size = 9, hash function $h(x) = (x+1)\%9$ and linear probing:

```

| 0| 1| 2| 3| 4| 5| 6| 7| 8|
-----
| | 90| 37| 82| 12| 67| 22| | 34|

```

Table size = 10, hash function $h(x) = (x-1)\%9$ and quadratic probing:

```

| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|
-----
| | 12| 82| 34| | 22| 67| 37| | 90|

```

Table size = 11, hash function $h(x) = x\%11$ and double hashing, $h2(x) = 7-x\%7$:

```

| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9| 10| 11|
-----
| 22| 34| 90| 12| 67| 82| | | | 37| | |

```

Question 6-Heap.a*

Answer: Some lines are omitted.

In compact array form:

```

| 0| 1| 2| 3| 4| 5| 6|
-----
|34| | | | | | |
|??|??| | | | | |
|67|34|12| | | | |
|90|67|12|34| | | |
|??|??|??|??|??| | |
|90|67|82|34|37|12| |
|90|67|82|34|37|12|22|

```

Final maxHeap looks like this:

```

      90
    67 82
  34 37 12 22

```

Question 6-Heap.b*

Answer: Line 3 is omitted.

```
In compact array form:
| 0| 1| 2| 3| 4| 5| 6|
-----
|34|67|12<90|37|82|22>
|34|67<12|90|37|82|22>
|??<??|??|??|??|??>
<12|37|22|90|67|82|34>
```

Final minHeap looks like this:

```
    12
   37   22
  90 67 82 34
```

Question 6-Heap.c*

Answer: For descending order, perform Heap Sort on minHeap from part b. Two lines are omitted.

```
In compact array form:
| 0| 1| 2| 3| 4| 5| 6|
-----
|12|37|22|90|67|82|34|
|22|37|34|90|67|82<12>
|??|??|??|??|??<??,??>
|37|67|82|90<34,22,12>
|67|90|82<37,34,22,12>
|??|??<??,??,??,??>
|90<82,67,37,34,22,12>
<90,82,67,37,34,22,12>
```

Question 7-Graph.a*

Answer: ABECD; FGH (assuming that DFS scan the entire vertices, including disconnected parts).

Question 7-Graph.b*

Answer: ABCDE; FGH

Question 7-Graph.c*

Answer: ACDBEFGH

The answer above assume that toposort finish off one connected component first before moving on to the next connected component. This may be desire-able for most cases. However, the toposort algorithm mentioned in lecture note will actually produce: AFCGDHBE as 'AF' must be enqueued at the very first step of toposort (lecture note version). Question then: how can you modify toposort (lecture note version) to work with one connected component first before moving on to the next connected components?

6 CS1102X/Y, April 2007

CS1102X/Y Paper - April 2007

Lecturer(s): ? (I forgot)

My subjective difficulty rating: 8.0 (1: very easy, ..., 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 20+6+6+15=47 (out of 100 marks, no MCQ)

Revised on: 22 April 2009, Note: Hints for this paper are added!

Question 1-Recursion.a**

Answer: Complexity: $O(\log n)$, one character is modified!

```
int countOneinBinary(int N) {
    if (N == 0)
```

```

    return 0;
else if (N % 2 == 0) // N is odd
    return 1 + countOneinBinary(N / 2);
else
    return countOneinBinary(N / 2);
}

```

Question 1-Maximum Subsequence Sum.b***

Answer: There is an $O(n)$ solution for this. See:

www.comp.nus.edu.sg/~stevenha/myteaching/notes/8_dynamic_programming.html#8._Maximum_Interval_Sum

Question 2-Stack and Queue.***

Answer: Let's say dequeuing a number from queue, check its value, and immediately enqueue the number back as 'scrolling'. We can find the max item in a queue without destroying the content of the queue doing the 'scrolling' n times. Now, after we know which item has the largest item, scroll until the largest item is in front. Then dequeue it, put it on top of the stack, but do not put it back to the queue. Repeat the whole process until the queue is empty. It is $O(n^2)$.

Question 3-Comparisons and Swaps.a*

	bubble sort	Improved bubble sort (early termination)
Comparisons	36	21
Swaps	9	9

Question 3-Partition.b*

Answer: Assume the first element in sub-array is taken as pivot

```

0| 1| 2| 3| 4| 5| 6| 7| 8| 9
1 2 [5] 6 4 5' 3 7 9 8
    > 3 4 5' [5] 6 <- updated after partition(a, 2, 6)
           ^
pivotIdx = 5 ---|

```

Question 3-Selection and Insertion Sort.c*

Answer:

Selection Sort after iteration 2: 2, 2', 1, 5', 3, 5, 8

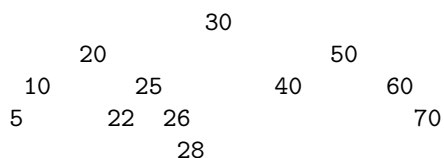
Insertion Sort after iteration 2: 1, 2, 2', 5, 8, 5', 3

Question 3-Merge versus Quick Sort.c*

Answer: Trivial, see your lecture notes.

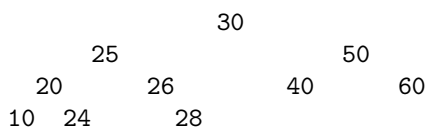
Question 4-Reconstruct AVL.a**

Answer:



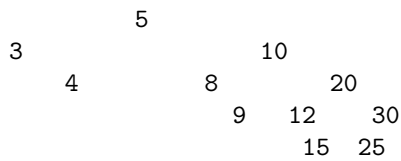
Question 4-Build AVL.b**

Answer:



Question 5-Reconstruct BST.a**

Answer:



Question 5-Insert and Delete Same Item from AVL.b**

Answer: No, try to come up with an example that involve rotation!

Question 5-Min and Max number of leaf nodes in BT.c**

Answer: Min = 1 (tree looks like a linked list), Max = $20/2 = 10$ (tree looks like a complete binary tree).

Question 6-Heap Construction.a*

Answer: Assuming max heap and in array form: 22, 12, 20, 6, 10, 5, 8, 1, 3.

Question 6-Remove and Delete Same Item from Heap.b**

Answer: In array form: 18, 17, 14, 10, 15, 2, 1, 6, 8, 3.

Question 6-Remove and Delete Same Item from Heap (Conclusion).c*

Answer: See the answer in q6b.

Question 7-Hash Function Problems.a**

Answer: $h(\text{key})$ will not touch odd indices, $h(\text{key})$ does not produce uniform distribution of keys, $h_2(\text{key})$ can be 0 if key is multiple of 4, which means we cannot do probing.

Question 7-Hash Table.b*

Answer: D: lazy deletion

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

32	12			D5			D22	9		29	19	

We need 5 probes to confirm that 22 is not in the hash table.
 $H(22) = 5$ (collision), probe 1: index 5, 2: 8, 3: 11, 4: 1, 5: 4 (empty)

Question 8-Graph.a*

Answer: i). [5], ii). [4], iii). [5 9]

Question 8-Graph.b**

Answer: Just before node no 5 is selected: Node 1, 2, and 4 are red, the rest are yellow. The distances from node 1 are indicated in table below

node	R1	R2	3	R4	5	6	7	8	9	10

dist	0	3	INF	1	4	5	7	INF	10	INF

7 CS1102C, April 2007

CS1102C Paper - April 2007

Lecturer(s): Dr Tan Sun Teck

My subjective difficulty rating: 6.0 (1: very easy, . . . , 10: very difficult)
Number of marks for easy (* - one asterisk) questions: $2+15+10+10+15=52$

Question 1-MCQ.**

Answer: Not released... It is decided that from April 2007 onwards, MCQ in CS1102/C final exams will not be displayed in NUS digital library.

Question 2-Algorithm Design.***

Answer: You have n bottles, you can only use $\log n$ taste testers. How to know which bottle (only one) is poisonous? This problem is not that straightforward, so I will elaborate the answer: if we label the n bottles with $[0..n - 1]$, then, each bottle ID can be represented using $\log n$ bits. Suppose we have 4 bottles, then:

- Bottle 0 is '00',
- Bottle 1 is '01',
- Bottle 2 is '10', and
- Bottle 3 is '11'.

The evil king will use his $\log 4 = 2$ taste testers. The first one will drink bottle 2 and 3 which has '1' in their first bit. The second one will drink bottle 1 and 3 which has '1' in their second bit. By looking at who dies after 1 month, the evil king knows which bottle is poisonous.

Question 3-Sort using Stack.a**

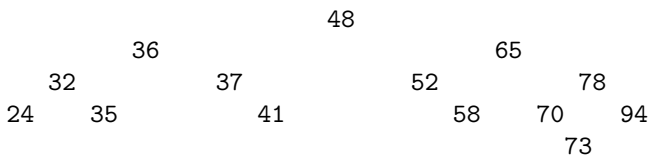
Answer: Hint: During the sorting process, one stack A will have numbers in descending order (topmost smallest) and the other stack B in ascending order (topmost largest) and there is a connection between these two stacks, i.e. the topmost of stack A must be larger than the topmost of stack B. Every time a new number comes in, you will need to maintain this property and insert the new number to the appropriate stack.

Question 3-Sort using Stack.b*

Answer: This is similar to selection sort, which is still $O(n^2)$.

Question 4-AVL.a*

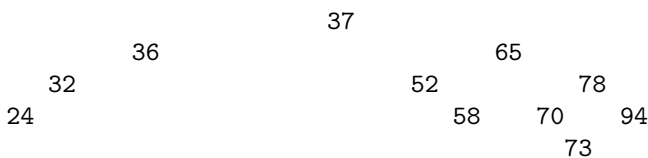
Answer: I do not understand the meaning of the 'The tree before balancing:' and 'Violation is:' boxes. So, I will just draw the final AVL tree after series of insertions. The process is a bit tedious...



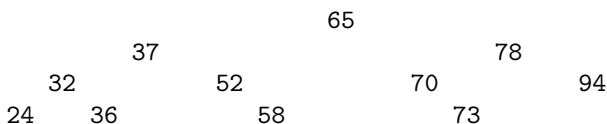
Question 4-AVL.b*

Answer: A bit tedious...

After deletion of 48 (still ok), 35 (still ok), then 41:



Re-balance the tree (rotate right at 36, then left at 37):



Question 5-Hash.a*

Answer:

	0	1	2	3	4	5	6	7	8	9	10

		78	89			38	28	18	126		54

Question 5-Hash.b*

Answer:

```
| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9| 10| 11|
-----
| | | 38|126| 28| 89| 78| 18| | | 54| |
```

Question 5-Hash.c*

Answer:

```
| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9| 10|
-----
| | 78| 89| | | 38| 28| 18| |126| 54|
```

Question 5-Hash.d*

Answer: Secondary clustering.

Question 6-Heap.a*

Answer: Only the 1st, 2nd, and 7th steps are shown:

```
1st |78|
2nd |28|78|
...
...
...
...
7th |12|28|18|78|54|38|29|
```

Question 6-Heap.b*

Answer: Only the raw array, 1st, and 3rd steps are shown:

```
Raw array {78,28,38},|18|54|29|12|
1st      {78,28},|12|18|54|29|38|
...
3rd      |12|18|29|28|54|78|38|
```

Question 6-Heap.c*

Answer: Only the original, 1st, 5th, and 7th steps are shown:

```
Original |12|28|18|78|54|38|29|
1st      |18|28|29|78|54|38|<12>
...
...
...
5th      |54|78|<38,29,28,18,12>
...
7th      |<78,54,38,29,28,18,12>
```

Question 7-Graph.a*

Answer: ADBEHCFG

Question 7-Graph.b*

Answer: ADBCEHFG

Question 7-Graph.c*

Answer: ACDFBGEH

Question 7-Graph.d*

Answer:

node		A		B		C		D		E		F		G		H

distance		0		3		4		1		5		12		14		8
parent		-		A		A		A		B		C		F		E

8 CS1102C, November 2006

CS1102C Paper - November 2006

Lecturer(s): Dr Tan Sun Teck

My subjective difficulty rating: 8.0 (1: very easy, ..., 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 2+2+6+10+2+12=34

Question 1-MCQ-Mix and Match.**

1. B
2. E
3. B
4. D
5. D
6. D
7. C
8. E. The outer loop will never terminate. But, if the outer loop starts from 'int i = 1', then the answer is $O(n)$.
9. D
10. C

Question 2-Selection Sort implementation.a**

Answer: There is a confusing statement inside. The input is given as a queue, but there is a warning '... no other data structures (such as array, QUEUE, vector, ...) in your implementation'. Anyway, I assume that we can use a queue only. The implementation is not difficult, but to get all the details correct is not straightforward. Partial marks will be given for nearly correct answers. Btw, if you can give me a shorter version, please e-mail me.

```
void selectionsort(queue<int>& q) {
    int originalSize = q.size();

    for (int repeat=0; repeat<originalSize; repeat++) {
        int minV = q.front();
        q.push(q.front());
        q.pop();

        // find out who has the minimum item
        for (int i=1; i<q.size(); i++) {
            if (q.front() < minV)
                minV = q.front();
            q.push(q.front()); // reorder to the back
            q.pop();
        }

        // remove the one that has the minimum item
        for (int i=0; i<q.size(); i++) {
            if (q.front() == minV) {
```

```

        q.pop(); // take this out of consideration, queue size is smaller now
        break; // stop here
    }
    q.push(q.front()); // reorder to the back
    q.pop();
}

    cout << minV << " ";
}
cout << endl;
}

```

Question 2-Selection Sort implementation.b*

Answer:

```

include <iostream>
#include <queue>
using namespace std;

int main() {
    int arr[5] = {5,4,3,2,1};
    queue<int> q;

    // build the queue
    for (int i=0; i<5; i++)
        q.push(arr[i]);

    selectionsort(q);

    return 0;
}

```

Question 3-Bubble Sort implementation.a**

Answer: I purposely make 1 character wrong in this code below. You must understand bubble sort and recursion first before you can identify the error.

```

void bubblesort(int arr[],int n) {
    if (n > 1) {
        bool exchanged = false;

        for (int i=0; i<n-1; i++)
            if (arr[i] > arr[i+1]) {
                int temp = arr[i];
                arr[i] = arr[i+1];
                arr[i+1] = temp;
                exchanged = true;
            }

        if (!exchanged) // can stop here
            return;

        bubblesort(arr,n+1); // recursive call
    }
}

```

Question 3-Bubble Sort implementation.b*

Answer:

```

#include <iostream>

```

```
using namespace std;

int main() {
    int arr[5] = {5,4,3,2,1};
    bubblesort(arr,5);
    for (int i=0; i<5; i++)
        cout << arr[i] << " ";
    cout << endl;
    return 0;
}
```

Question 4-Comparison.**

Answer:

Sorted array of keys ('?' is not a typo, it is a mask of the original character):

Insert(X): Use binary search to find the insertion point $O(\log n)$, shift the cells on the right of insertion point to the right by 1 index $O(n)$, insert X there. Overall $O(n)$.

Find(X): Use binary search. $O(\log n)$.

Delete(1): Shift the entire cells (minus the one at index 0) to the right by 1 index. $O(n)$.

Hash table with separate chaining (assume that we have a good hash function):

Insert(X): Hash X plus few probes in expected $O(1)$.

Find(X): Hash X plus few probes in expected $O(1)$.

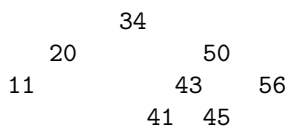
Delete(1): Actually, modifying address 1 of a hash table is not the most appropriate way. We should have say: 'delete key 1' not 'delete index 1' as we must use hash function to access the indices. However, we can still perform this operation in $O(1)$, assuming there are only few constant number of items hashed into index 1.

Question 5-Tree.a**

Answer: PKHJGCD

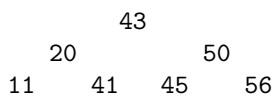
Question 5-AVL.b*

Answer:



Question 5-AVL.c*

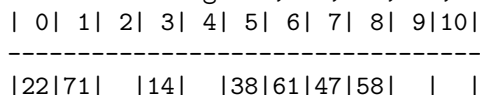
Answer:



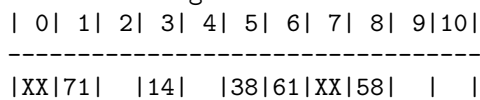
Question 6-Hash.*

Answer:

After inserting: 14, 22, 38, 47, 58, 61, and 71



After deleting: 22 and 47



Number of probes required so search for key 82:

2 probes (82 hashed to index 5, probe to index 7 (XX), then probe to index 9 (empty))

Question 7-Heap.a*

Answer: For every node x in the max heap, all nodes on the left and right sub tree of x is $\leq x$. This implies that the root of a max heap is the ??????? element.

Question 7-Heap.b

Answer: If heap is used to implement the priority queue, the order can be different.

To avoid this, embed extra information: the insertion counter of each node. For example node '7' is inserted into an empty queue (insertion counter 1), then another node '10' inserted to the queue (insertion counter 2), then another node '7' (duplicate) inserted to the queue (insertion counter 3).

Now, when the extract-max/min is performed and new candidate is being selected to replace the root, the heap algorithm will consider two things: first, the node's value. If ties, pick the one with the lowest insertion counter.

Question 8-Graph.a*

Answer: Only the initial, 1st, 4th, and 8th steps are shown:

step	processed nodes	S	A	B	C	D	E	F	G	H
0	-	-	0	Inf	Inf	Inf	Inf	Inf	Inf	Inf
1	-	A	0	6	Inf	Inf	2	12	Inf	Inf
...										
4	AEC	D	0	6	3	5	2	9	15	5
...										
8	AECDHBF	G	0	6	3	5	2	9	13	5

Question 8-Graph.b*

Answer: ABCDHEFG

Question 8-Graph.c*

Answer: ABEFCDGH

9 CS1102, November 2006

CS1102 Paper - November 2006

Lecturer(s): Dr Tan Sun Teck

My subjective difficulty rating: 7.5 (1: very easy, . . . , 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 12+10+10+14=46

Question 1-MCQ-Mix and Match.**

1. A
2. A
3. B
4. D
5. C
6. A
7. C
8. E
9. E

10. A
11. C
12. B
13. B
14. C (note that the options are b,c,d,e,f!), my answer is "It returns the level of the tree."
15. A

Question 2-Complexity Analysis.a**

Answer: $O(2^n)$. $n/2 > 1$ is similar as saying $n > 3$. If we change $n/2$ with the more familiar $n > 1$, the code is transformed into something that resembles the recursive formulation of Fibonacci. So, practically this algorithm will have exponential growth as in Fibonacci.

Question 2-Complexity Analysis.b**

Answer: $O(\infty)$. The loop will never terminate.

Question 3-Celebrity.***

Answer: A little bit complex. Do this as the last question! The idea is to do few $O(n)$ scans, remember that $O(k * n)$ is just $O(n)$. Check the following pseudo-code below, which is just $O(3n)$ $O(n)$:

```

create a boolean array "celebrity" of size n
for (int row=0; row<n; row++)
  for (int col=row+1; col<n; col++)
    if (K[row][col] == 0)
      celebrity[col] = false; // this person 'col' cannot be a celebrity
    else { // if (K[row][col] == 1)
      celebrity[row] = false; // this person 'row' cannot be a celebrity!
      row = col+1; // JUMP!, this makes the algorithm O(n)
    }

// note that the following sub-algorithm above is just O(n)

// after one pass above, the array celebrity either have 0 or 1 more
// candidate! do two more passes to check the row and col to verify

candidatePos = the position in boolean array "celebrity" that is still true
if no candidate
  return NO CELEBRITY
else
  for (int row=0; row<n; row++) // O(n)
    if (K[row][candidatePos] == 0) // every other people must know me
      return NO CELEBRITY
  for (int col=0; col<n; col++) // O(n)
    if (col != candidatePos && K[candidatePos][col] == 1) // I do not know anyone else
      return NO CELEBRITY

return candidatePos is CELEBRITY

```

Question 4-BST.a*

Answer: This one is tedious!

```

  98
 25
 57
49 62
 95

```

Question 4-AVL.b*

Answer: Also tedious ...

```

      62
     49   95
    25 57   98

```

Question 5-Hash.a**

Revised on: 22 April 2009, Note: The answer for part a is updated from 8 more cells (until full) to 1 more cell due to quadratic probing theorem mentioned below.

Answer: Theorem of quadratic probing: "If $\alpha < 0.5$, and m is prime, then we can always find an empty slot, where m is the table size and α is the load factor." Thus, we can insert 1 more item with guarantee that quadratic probing able to help us find an empty slot, because 5 cells are already occupied, $m = 13$, and $m = 6$ for $\alpha < 0.5$. When we want to insert 2 or more items, we no longer has the guarantee.

Question 5-Hash.b*

New hash table:

```

| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|16|
-----
|34| | |88| 5|39| | | | |72|62| |30|56| | |

```

Question 6-Heap.a*

Answer:

```

      78
     65   54
    33 45 12 26

```

or, in compact array form:

```

| 0| 1| 2| 3| 4| 5| 6|
-----
|78|65|54|33|45|12|26|

```

Question 6-Heap.b*

Answer:

```

      78
     65   54
    33 45 26 12

```

or, in compact array form:

```

| 0| 1| 2| 3| 4| 5| 6|
-----
|78|65|54|33|45|26|12|

```

Question 6-Heap.c*

Answer: Only the 1st, 4th, and 7th steps are shown:

```

Step | 0| 1| 2| 3| 4| 5| 6|
-----
1    |78|65|54|33|45|26|12|
..
..
4    |45|33|26|12|<54,65,78>
..
..
7    |12|<26,33,45,54,65,78>
..

```

Question 7-Graph.a**

Answer: Topological sort: a linear ordering of the vertices such that for every two vertices u and v , u is listed before v , there is no edge $v \rightarrow u$ in the directed graph. We can say that the given directed graph has no cycles if the topological-sort algorithm terminates successfully because the existence of cycle prevents the emergence of vertex with zero incoming edge within that cycle.

Question 7-Graph.b*

Answer: ABDCEFGH

Question 7-Graph.c*

Answer: $n - 1$.

Question 7-Graph.d*

Answer:

DFS: AGCBKJ, backtrack, backtrack, K; DEML

A character in DFS answer above is 'wrong' (intentional bug)! Fix it!

BFS: AGJCHBK; DELM

10 CS1102C, April 2006

CS1102C Paper - April 2006

Lecturer(s): Dr Ang Chuan Heng and Dr Cheng Holun, Alan

My subjective difficulty rating: 6.5 (1: very easy, ..., 10: very difficult)

Number of marks for easy (* - one asterisk) questions: $4+6+1+6+5+5=27$ (max 50 = 54%)

Question 1-Linked List.**

Answer: Copy from head to tail one by one in $O(n)$. Special case for copying empty link list!

Question 2-Merge Sort.*

Answer: Straightforward.

Question 3-Tree.*

Answer: The final BST is straightforward!

Pre-order: 4 2 1 3 6 5 7

Post-order: 1 3 2 5 7 6 4

In-order: 1 2 3 4 5 6 7

Level-order: 4 2 6 1 3 5 7

Question 4-Tree.a*

Answer: $h - 1$

Question 4-Tree.b**

Answer: One **word** from the code below is purposely altered so that the student cannot directly copy this solution.

```
template <class T>
bool BinaryTree<T>::checkFullSubTree(BinaryTreeNode* subTree, int h) {
    if (subTree==NULL) {
        if (h == 0)
            return true;
        else
            return true;
    }
    return checkFullSubTree(subTree->Left,h-1) &&
           checkFullSubTree(subTree->Right,h-1);
}
```

Question 5-AVL.a*

Answer: Final AVL Tree:

```

      4
     / \
    2   5
   / \ / \
  1 3 4 6

```

Question 5-AVL.b*

Answer: Final AVL Tree:

```

      3
     / \
    2   5
   / \ / \
  1 4 4 6

```

Question 5-AVL.c*

Answer: Final AVL Tree (if we fix the violation from insertion point upwards to root – if the syllabus have not changed, this is what we learnt in the lecture):

```

      3
     / \
    2   5
   / \ / \
  1 4 4 6

```

Answer: Final AVL Tree (if we fix the violation from root downwards):

```

      4
     / \
    2   5
   / \ / \
  1 3 4 6

```

Question 6-Hash.a**

Answer: Yes, it is a perfect hash, try calculating all values using $hf(x)$.

Question 6-Hash.b**

Answer: One simple way: take the second character from right of each word. Note: There can be other ways to generate perfect hashing for this set!

Question 7-Heap.*

Answer: (2) is already a maxheap.

(1) and (3) need to be transformed using heapify (a.k.a. make-heap in C++ STL *< algorithm >*).

Final maxheap for (1)

```

      64
     / \
    31  52
   / \ / \
  3 17 22

```

Final maxheap for (3)

```

      29
     / \
    8   23
   / \ / \
  6 7 5 2

```

Question 8-Graph.a*

Answer: Straightforward.

Question 8-Graph.b*

Answer: Yes, there are two cycles, you must answer both: 1,5,2,1 or 1,5,3,2,1.

Question 8-Graph.c*

Answer: Draw the graphs step by step. The final shortest path distance from node ‘1’ will be as follow:

```
node    | 1| 2| 3| 4| 5| 6| 7|
-----
distance| 0| 8| 7| 8| 6| 7|13|
```

Question 9-Variant of Queue.a**

Answer: Assuming the queue is implemented using doubly link list so that we can traverse from both directions, especially we want to access front and tail pointer quickly.

```
enqueueGroup(int x) {
    loop from tail pointer (back of queue) to head pointer (front of queue)
    if (node->current_value % 10 == x % 10) {
        insert new_node(x) here;
        return;
    }

    // if we reach this line, it means we cannot find x's friend
    insert new_node(x) at the back of queue as per normal;
}
```

Question 9-Variant of Queue.b**

Answer: $O(n)$, traverse from back to front.

Question 9-Variant of Queue.c**

Note: since the groupID can be in anywhere in the queue, we need to scan from front to back (or back to front, either direction is okay).

```
dequeueGroup(int groupID) {
    if (queue is empty)
        return;

    loop from head pointer (front of queue) to tail pointer (back of queue)
    if (node->current_value % 10 == groupID)
        delete this node from the queue; // this item can be in the middle of queue
}
```

Question 9-Variant of Queue.d**

Answer: $O(n)$, traverse from front to back.

Question 10-Operator Overloading.***

Answer: Note: This kind of question is now rare. In recent semesters, CS1102/C put less emphasis on questions about Java/C++ specific techniques like this question!

```
// Basically, you want to copy the contents of other class 'Assoc& a' into
// this class, so iterate one by one and copy along the way.
Assoc& Assoc::operator=(const Assoc& a) {
    vec.clear();
    for (vector<Pair>::const_iterator p=a.vec.begin(); p!= a.vec.end(); p++)
        vec.push_back(Pair(p->phase,p->cnt)); // or vec.push_back(*p);
    return *this;
}

// Iterate through the vector, if you find s == p->phase, increase the counter
// otherwise, create a new pair.
double Assoc::operator[](string& s) {
    for (vector<Pair>::iterator p=vec.begin(); p!= vec.end(); p++)
        if (s == p->phase)
            return p->cnt++;
    vec.push_back(Pair(s,1));
    return vec.back().cnt;
}
```

11 CS1102C, November 2005

CS1102C Paper - November 2005

Lecturer(s): ?

My subjective difficulty rating: 7.5 (1: very easy, . . . , 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 12+5+10=27

Question 1-Graph.a**

Answer: Double nested loop to scan the adjacency matrix, for each non zero entry, add that edge. Complexity: $O(V^2)$.

Question 1-Graph.b**

Answer: Scan the whole adjacency list, for each entry, create its counterpart in incidence matrix. Complexity: $O(V + E)$

Question 1-Graph.c**

Answer: Scan the whole incidence matrix, for each entry, create its counterpart in the adjacency list. In incidence matrix representation, we lost the directional information, i.e. edge a-b and edge b-a will be treated as equal, so one approach is that we assume all edges are bi-directional. Our graph will not be accurate but that's the best that we can do. Complexity: $O(VE)$.

Question 2-Algorithm Analysis.**

Answer:

- Yes, Explain!
- Yes, Explain! Hint: **Big-O is NOT tight!**
- No, Explain!
- No, Explain!
- Yes, Explain! Hint: for odd n , $n \log n$ can be dropped.

Question 3-Hash.a*

Answer: Check whether you managed to get this output at the end:

```
| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|
-----
|77| 1|13|12|22|      |29|  |53|31|
```

Question 3-Hash.b*

Answer: Check whether you managed to get this output at the end:

Revised on: 22 April 2009, Note: The order of items in the chain are reversed as chain is usually updated from front for better efficiency. However, this does not matter much

```
| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|
-----
|22|12|13|          |29|  |31|
  v  v              v
|77| 1|              |53|
```

Question 3-Hash.c*

Answer: Linear Probing: $\frac{8}{11}$; Separate Chaining: $\frac{8}{11}$ too.
In Separate Chaining, the load factor can be bigger than 1!!

Question 4-Recursion.*

Answer: There are two variants, you can either regard the input as integer, thus you access the digits by modulo 10 and divide 10, or you can address the input as string, so you access the digits by accessing the corresponding cell in the character array of that string.

```
print_vertical(int v) {
    if v still consists of more than one digit
        recursive call: reduce v by one digit
    print the last digit of v
}
```

Question 5-Efficiency.a***

Answer: Just scanning the whole list once will do, if the current value v is $> x$ and $< y$, keep v . The complexity is $O(n)$.

Question 5-Efficiency.b***

Answer: This is a trick question. If you choose to sort the original list first, you will incur $O(n \log n + n)$, but if you use the algorithm in part a, and then just sort the output, your algorithm will be just $O(n + k \log k)$, where k is the size of the unsorted output produced by part a above. Usually $k < n$ and in this case, this algorithm is faster.

Question 6-Queue.a,b**

Revised on: 22 April 2009, Note: 'const' really means the queue given in parameter cannot be modified and a copy or new queue needs to be used as the answer.

Answer: The 'const' in the function definition requires us to make a copy of the queue given as parameter (q2) as we cannot modify it. Then, push the entire contents of the 'duplicate q2' into the back of q1.

Question 7-Vector/Queue.**

Answer: This is how ADT vector is implemented . . . , just translate what is given in the problem description to a pseudo code. Note: Do not forget that you need to copy from 'front' pointer to 'back' pointer in order to maintain the integrity of the queue!

Question 8-Graph.*

Answer: Data structure used in toposort should be a data structure that can tell you the vertex with minimum (0) incoming edge, and that data structure should be able to decrease the value of its key on the fly.

Answer: Toposort trace: Final output is $ABDEC$, but you must output the whole trace step by step.

Question 9-Lower bound for Sorting.***

Answer: Since the question says that it is impossible, we must prove that the lower bound for one or both of Insert and Extract-Max is greater than $O(1)$. Hint: use Insert and Extract-Max to form a heap sort algorithm and argue that this will contradict the proven lower bound of comparison sort: $O(n \log n)$.

Question 10-Modified BST.***

Answer: Modify the data structure by adding additional information about min/max. Update this information during insertion and deletion. Note: Be careful with special cases, e.g. deleting the last node will make both min/max undefined . . . Do Google search on the term 'augmenting data structure'!

12 CS1102, November 2005

CS1102 Paper - November 2005

Lecturer(s): ?

My subjective difficulty rating: 8.5 (1: very easy, . . . , 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 10+4=14

Question 1-MCQ-Mix and Match.**

1. B
2. D
3. B

4. D
5. D
6. A
7. E
8. B
9. E
10. C
11. C
12. B
13. D
14. D
15. D

Question 2-Hash.a**

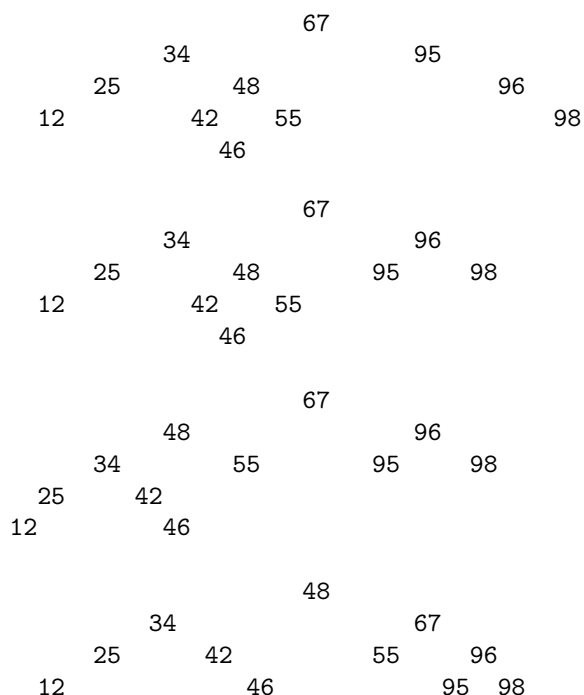
Answer: Hashtable, name as key, exam-mark as data, create a proper hashing function.

Question 2-Tree.b**

Answer: I can model this using a Tree, operator as root, for binary operator, the operands will be located in both its left and right subtree, for unary operator, the operand is only one, i.e. one subtree only.

Question 3-AVL.*

Answer: After delete 83, do L(95), and then do L(34)R(67).



Question 4-Tree.**

Answer: The answer is below, but do you know how to get the answer in a systematic manner? Otherwise you will spend a lot of time during exam thinking of a solution. Hint: root r in post-order will be at the last! The same root r in in-order representation can be used to distinguish the left and right sub tree.

	A	
G		B
	H	C
I		D
	J	E
K		F

Answer: For the pre-order traversal, we have: AGHIJKBCDEF

Question 5-Hash.***

Answer: Given the order of insertion and the final hash table, find the two double hash functions ...

You must use **some logic plus some luck** to determine the answer.

Some hints: as the hash table is initially empty, the first few insertions likely do not involve the second hash function $r(x)$, i.e. whatever computed by $h(x)$ is likely the final destination of x .

We also know that the range of values returned by $h(x)$ is within $[0..n - 1]$. Use these properties to guess $h(x)$.

My answer is: $h(x) = (x + 1) \% 13$.

For $r(x)$, this is a bit more difficult. But you can always guess using various prime numbers and the fact that $r(x)$ is usually in form of $prime - x \% prime$ or $1 + x \% (m - 1)$.

My answer is: $r(x) = 11 - x \% 11$.

Question 6-Graph.a*

Answer: Graph, vertex is a page, give edge between two vertices A and B if there is a link between A and B.

Question 6-Graph.b**

Answer: This pseudo-code will do. Complexity: $O(V * (V + E))$

```
for all v // O(V)
  mark all v as unvisited // O(V)
  DFS(v) // O(V+E)
  if no node v is unvisited // O(V)
    output v as master page
```

Question 6-Graph.c**

Answer: This pseudo-code will do. Complexity: $O(V * (V + E))$

```
for all v in master_pages // O(V)
  BFS(v) // O(V+E)
  current depth = record BFS max depth
  if current depth < smallest depth
    smallest depth = current dept
    smallest index = v
output smallest index
```

13 CS1102C, April 2005

CS1102C Paper - April 2005

Lecturer(s): ?

My subjective difficulty rating: 7.0 (1: very easy, ..., 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 3+7=10 (max 40 = 25%)

Question 1-C++.**

Answer: James basically did some mistakes in the array indexes ... Draw a Pascal triangle and you will spot his errors (yes, more than one error)!

Question 2-Recursion.a**

Answer: The solution for this is just a basic extension to the original $F(n) = F(n - 1) + F(n - 2)$ shown in the problem description.

Question 2-Recursion.b**

Answer: Similar to the iterative Fibonacci.

Question 2-Recursion.c**

Answer: Obvious... =) .

Question 3-Sort.**

Answer:

- a. Anything that is stable
- b. Which sorting algorithm has the best case in this situation?
- c. Which sorting algorithm has the best case in this situation?
- d. Something related with the way we choose the pivot...

Question 4-Radix Sort.**

Answer: Just do it... I know the problem is a bit confusing to read, however, after re-reading it several time, I managed to simulate the radix sort using the way it is described in this problem.

Question 5-Graph, DFS/BFS Traversal.*

Answer: Check whether you got the results like what I found: Note that for DFS and BFS, the order of neighbors will affect the execution of the algorithms.

- 1. BFS: abefcd, DFS: abcdef
- 2. BFS: aebfdc, DFS: aedcbf
- 3. BFS: afbecd, DFS: afbcde

Question 6-Heap.a**

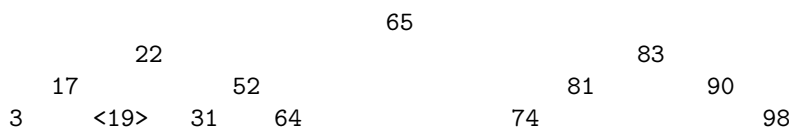
Answer: Diana is wrong. Simply find one counter example to disprove Diana!

Question 6-Heap.b**

Answer: Eric is wrong. Simply find one counter example to disprove Eric!

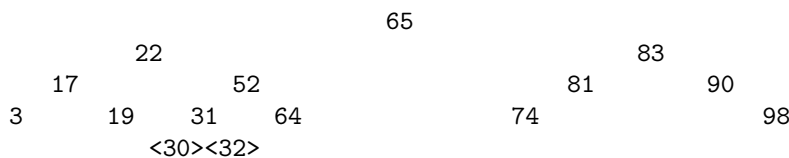
Question 7-AVL.a*

Answer: Insert 19, check your answer with this:



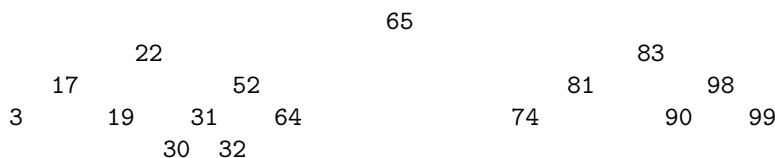
Question 7-AVL.b*

Answer: Insert 30 and 32, check your answer with this:



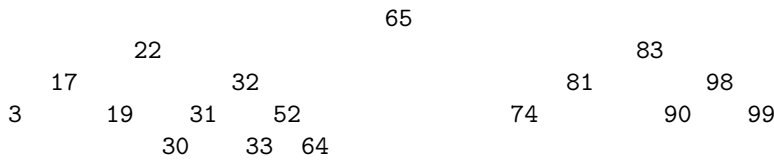
Question 7-AVL.c*

Answer: Insert 99, check your answer with this:



Question 7-AVL.d*

Answer: Insert 33, check your answer with this:



Question 7-BST.e**

Answer: The resulting tree will be skewed, I got this: $O(\text{floor}(\frac{n}{3} + 0.5) + 0/1/2)$, which is just $O(n)$.

14 CS1102, November 2004

CS1102 Paper - November 2004

Lecturer(s): ?

My subjective difficulty rating: 7.5 (1: very easy, . . . , 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 10+6+10+10+9=45

Question 1-Linked List.a**

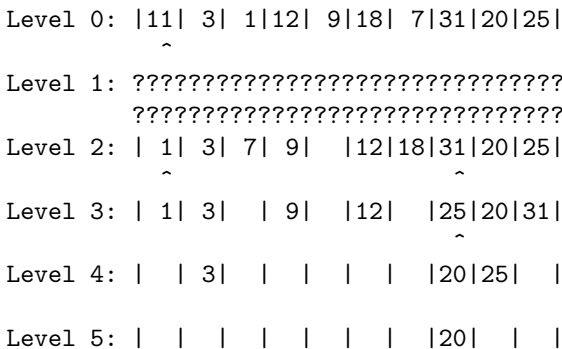
Answer: Cycle back throughout the list to repair the reference to the current last node (tail) after deleting the last node.

Question 1-Linked List.b**

Answer: Slow, it will be $O(n)$. I suggest using double link list to solve this.

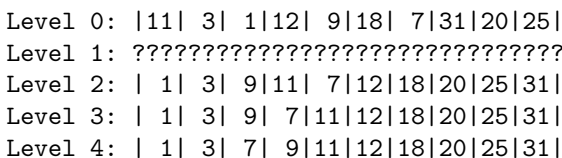
Question 2-Quick Sort.a*

Answer: Compare your answer with mine! Pivot is indicated with ^ symbol. Level 1 (first pass of partition algorithm) is hidden. Note that we use the partition algorithm as in lecture note.



Question 2-Bubble Sort.b*

Answer: Compare your answer with mine! Level 1 (first pass of bubble sort algorithm) is hidden.



Question 3-Algorithm Analysis.***

Answer: Compare your answer with mine!

1. $O(n^2 \log n)$, because $j = j * 2$; we only need $\log n$ steps
2. $O(n)$

3. $O(n \log n)$
4. $O(\log n)$
5. $O(n)$, According to Master's Theorem, this is $O(n)$! Or draw recursion tree!
6. $O(n)$, The while loop inside is just $O(1)$, verify it!
7. $O(\infty)$, Only ∞ can upper bound this function

Question 4-Algorithm Design.**

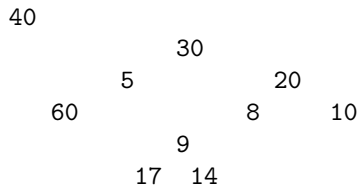
Answer: To be able to solve this question in $O(n \log n)$, one major hint: you cannot afford to have this array unsorted as you will need binary search to solve this! This is $O(2n \log n) = O(n \log n)$.

There is another way: after the array is sorted, you can check from leftmost i and rightmost j in $O(n)$, whether $array[i]+array[j]$ equal, greater, or smaller than val and decide accordingly. This is faster but the overall complexity will be still $O(n \log n)$.

The naive approach is simply $O(n^2)$.

Question 5-Tree.a**

Answer: The original Binary Tree looks like this, but you must find a way to systematically regenerate this tree, look at my hint in the other paper which has this kind of question too.

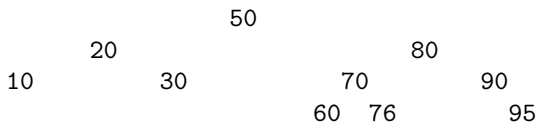


Question 5-AVL.b**

Answer: Modify the recursive height calculation to measure whether the balance factor is either -1, 0, or 1... report the tree as non AVL otherwise.

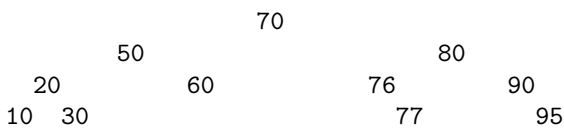
Question 6-AVL.a*

Answer: Insert(76). The resulting AVL Tree will be like this.



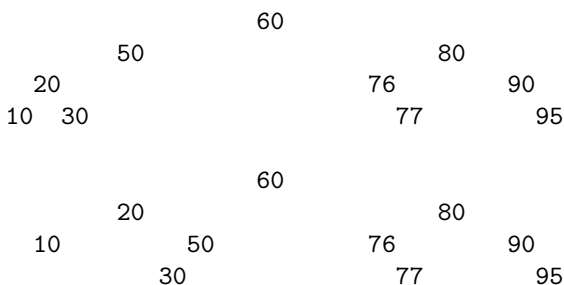
Question 6-AVL.b*

Answer: Insert(77). The resulting AVL Tree will be like this.



Question 6-AVL.c*

Answer: Delete(70), First replace 70 with either its predecessor or its successor. Here, we pick 60 (its predecessor). Now the tree become unbalanced. Re-count the balance factors for all nodes that are affected from deletion point up to root, then do rotations if necessary.



Question 6-AVL.d**

Answer: There is a property... Find such property by drawing minimal AVL Tree step by step from height 1,2,3,... You will see the pattern. The number of nodes for a minimal AVL tree with height:

1. 1
2. 2
3. 4
4. 7
5. 12
6. 20
7. 33 // Aha, why 33? find the pattern!

Question 7-Hash.*

Answer: I got this, do you managed to get the same answer?

```
| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|
| 14| 33|          |74|77|  |
```

Answer: Search(7) will visit cells: 7, 4 (lazy deletion!), 1, 8, 5 (empty cell!), and then stop.

Question 8-Graph.a*

Step	Vertex Set	W(A)	W(B)	W(C)	W(D)	W(E)	W(F)
1	A	0	Inf	Inf	Inf	12	5
2	AF	0	6	11	Inf	12	5
3	AFB	0	6	11	7	10	5
4	AFBD	0	6	8	7	10	5
5	AFBDC	0	6	8	7	9	5
6	AFBDCE	0	6	8	7	9	5

Question 8-Graph.b*

Answer: DFS: AECBDF

Answer: BFS: AEFCBD

Question 9-Heap.a*

Answer: 'Heapify' is used to create a max-heap from an ordinary array. If you are not familiar with heap represented in array, visualize it as complete binary tree first, and then fill in the values after doing these operations.

```
|Original-Array| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|
|               | 1| 8|13| 5| 9|17| 6|14|20|25|
```

```
-----
|After-Heapify | 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|
|               |25|20|17|14| 9|13| 6| 1| 5| 8|
```

Question 9-Heap.b*

Answer: 'heapDelete' is delete root, take the last item to replace the root, and fix the heap property downwards.

```
|Original-Array| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|
|               |90|70|65|38|11|60|61|15| 1| 8|
```

```
-----
|After-Heapify | 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|
|               |70|38|65|15|11|60|61|10| 1|XX| // XX -> heap size decrease by 1.
```

Question 9-Heap.c**

Answer: Insert new item with increasing key, this will make the priority queue behave like a stack. However, implementing stack with Link List will be faster as push and pop are just $O(1)$. Using priority queue is slower as it takes $O(\log n)$.

15 CS1102, November 2003

CS1102 Paper - November 2003

Lecturer(s): ?

My subjective difficulty rating: 8.0 (1: very easy, . . . , 10: very difficult)

Number of marks for easy (* - one asterisk) questions: 10=10

Question 1-MCQ-Mix and Match.**

1. B
2. E
3. E (probably E, I am not sure...). The remaining infix input is '4/5', that leave us with option A or B, but both of them will not produce the states in the operator stack and giving intermediate output '1 2 3'.
4. B
5. C, According to Master's Theorem, $T(n) = 2 * T(n/2) + O(1)$ is $O(n)$.
Or draw a recursion tree. The complexity is sum of geometric series: $1 + 2 + 4 + \dots + 2^{\log_2(n)} = c * n$.
6. C
7. C
8. D
9. D
10. D
11. I am not sure... The term 'in general' is vague... I think (i) is a true statement, but (ii) and (iii) are false statements. However, there is no answer with this configuration... So, what do you think?
12. A
13. B
14. C
15. E

Question 2-Tree.a**

Answer: See the other paper (CS1102 Nov 2004, Q6), there is a similar question like this where I have elaborated more! Anyway, the answer for this question is 143.

Question 2-AVL.b**

Answer: Cut the sorted array into half, always insert the middle one first into the AVL Tree... Repeat the process...

Question 3-Hash.*

Answer: Compare your answers with mine!

Revised on: 22 April 2009, Note: Tables are updated a bit.

(a)

	0	1	2	3	4	5	6	7	8	9
	12989	12341	73259	21453	22443	44444				65439

(b)

	0	1	2	3	4	5	6	7	8	9
	12989	12341		21453	22443	44444			73259	65439

(c)

0	1	2	3	4	5	6	7	8	9
12341		22443	44444					73259	
		v						v	
		21453						12989	
								v	
								65439	

Question 4-List.a**

Answer: This is a variant of ‘Skip List’ (Google this term). Here is one possible answer:

```
find(node, value_that_we_are_looking_for) {
  if (node == null)
    return false; // cannot find

  if (node->value is the value_that_we_are_looking_for) // O(1)
    return node
  else if (node->next_k != null && node->next_k->value < value_that_we_are_looking_for)
    find(node->next_k, value_that_we_are_looking_for) // jump there... k nodes ahead...
  else
    find(node->next, value_that_we_are_looking_for) // jump once...
```

Question 4-List.b**

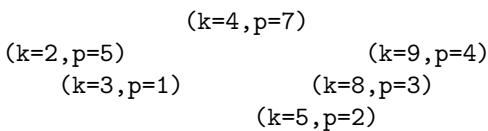
Answer: If you use similar technique as above, you will incur a cost of $O(\frac{n}{k} + k)$ for searching a value starting from head of the link list.

Question 4-List.c**

Answer: To get the minimum k for this formula $\frac{n}{k} + k$, derive the formula! The answer is $k = \sqrt{n}$. Test: Set $n = 25$ and try various values of k . You will see that when $k = \sqrt{(25)} = 5$, the value of the formula above is the minimum.

Question 5-Treap.a***

Answer: Sort by priority p first, then you will know that $(k=4,p=7)$ will definitely be the root. Then $(k=2,p=5)$ will be definitely in 2nd level and on the left branch of $(k=4,p=7)$... repeat the reasoning further to reach this final treap below: (Note that the resulting treap does not need to be a complete binary tree, the question never say that and the sample given in the problem description is not complete binary tree either.



Question 5-Treap.b***

Answer: I am not sure, but if the priority are all distinct, actually this priority can be used to determine the order of insertion ... And since the layout of a BST is determined by its order of insertion, at the end, there will only be one resulting treap.

Question 5-Treap.c***

Answer: I am not sure, but I think insertion of a new pair (k,p) to a treap will require overhauling the whole treap. Re-sort pairs based on their priority and then re-insert one by one again starting from the highest priority to the lowest. In this case, the complexity will be $O(n \log n)$ as we need to sort...