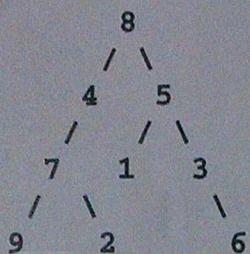


Name \_\_\_\_\_ Entry Number \_\_\_\_\_ Gp No. \_\_\_\_\_

Your Lab day \_\_\_\_\_ Your TA \_\_\_\_\_

Answer all the questions in the space provided for each question. You may use the answer script for rough work.

1. [2] Write the preorder traversal of the following tree:



8 - 4 - 7 - 9 - 2 - 5 - 1 - 3 - 6

Q

2. [9] Complete the following function which accepts a sorted linked list in ascending order, with each node containing an integer, and returns the list after removing all nodes which appear more than once. Thus given the input

34 - 40 - 55 - 55 - 62 - 78 , it should return 34 - 40 - 62 - 78

S

node \* duplicate( node \*A) {

part 1

    node     node \* p = A;

~~part 2~~  
    cont.     if (p==NULL)

        { cout << "In Empty list.";  
          return; }

    while (p->next != NULL) & A->next != NULL)

    {     while (A->data == (A->next->data))

        {     node \* h = A;

~~int~~ word1 = A->data;

          A = A->next;

          delete(h);     if (A->next == NULL) break;

        }

    if (A->data == word1)

        {     node \* n = A;

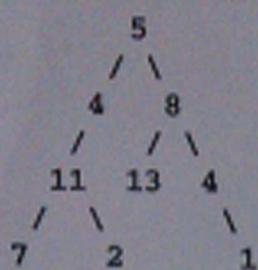
          A = A->next;

          delete(n);

        if (A == NULL) return;

    if (p->data == word2)  
    {     node \* h = p;  
          word2 = p->data;  
          p = p->next;  
          delete(h);  
          if (p == NULL) return;  
    }  
    if (p->data == word2)  
    {     node \* h = p;  
          p = p->next;  
          delete(h);  
          if (p == NULL) return;

3. [10] Given a tree and an integer, write a function which returns true if there is a path from the root down to a leaf, such that product of all the values along the path equals the given integer, otherwise return false. For example given the tree



Root-to-leaf paths for the above tree :

path 1: 5 4 11 7  
path 2: 5 4 11 2  
path 3: 5 8 13  
path 4: 5 8 4 1

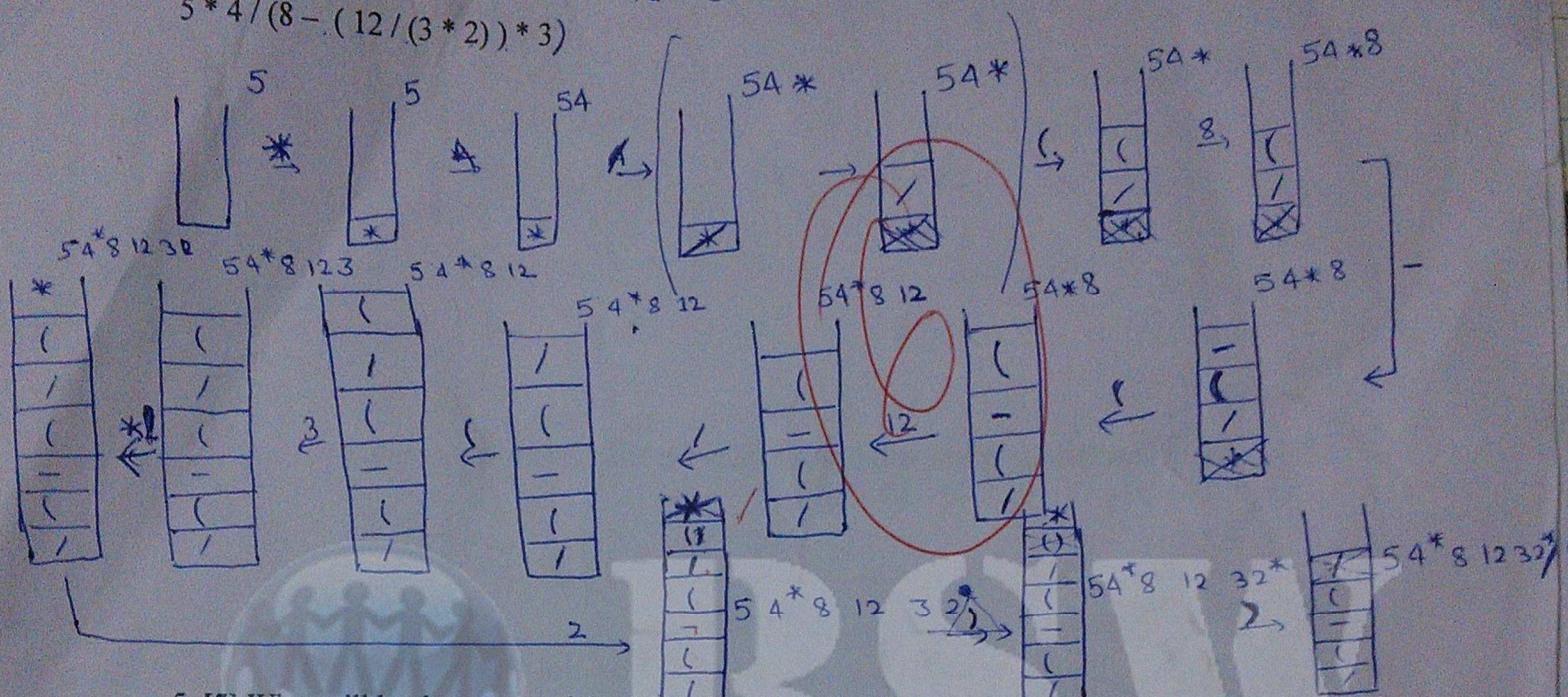
For this problem, we will be concerned with the product of the values of such a path -- for example, the product of the values on the 5-4-11-7 path is  $5 * 4 * 11 * 7 = 1540$ .

✓ base function (node \* A, int n) =  
if p->data does not divide n then return false;  
else return root function (p->left, n/p->data)  
|| function (p->right, n/p->data);  
no checking of whether root is null or not.  
no terminating condition  
no connect logic

④

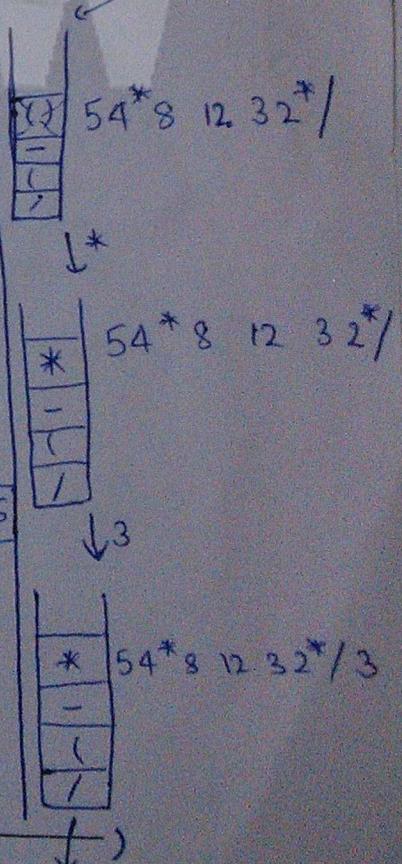
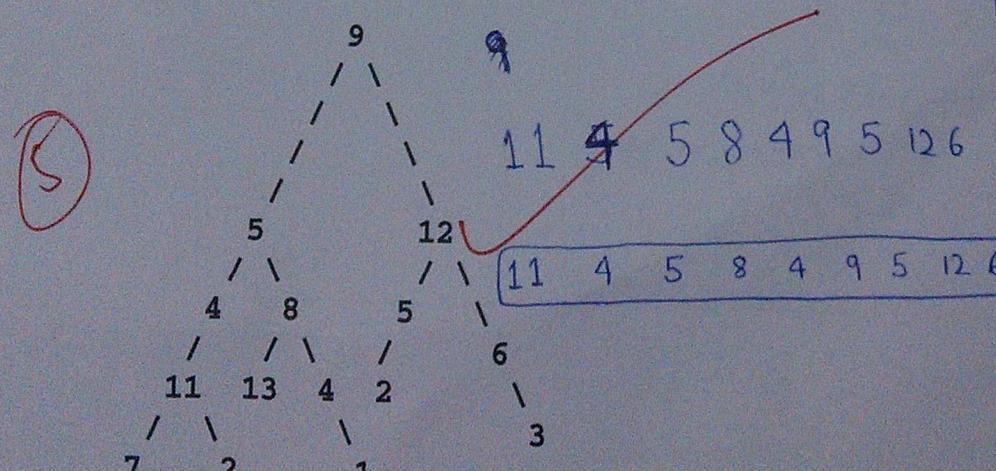
4. [6] Convert the following expression into postfix notation using a stack. Show the entries in the stack every time a character is read from the expression.

$$5 * 4 / (8 - (12 / (3 * 2)) * 3)$$

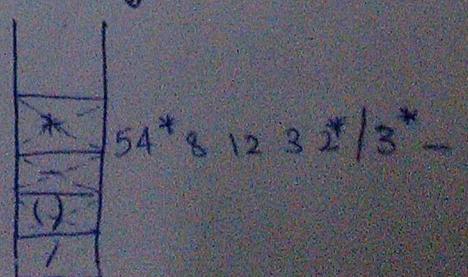


5. [5] What will be the output of the following function if it takes the given tree as input?

```
void wonder ( node *p ) {
    wonder ( p -> left );
    if ( p -> right != NULL || p -> left != NULL ){
        cout << p -> data << " ";
        wonder ( p -> right );
    }
}
```



$54 * 8 12 32 */ 3 * - / \leftarrow$



6. [3] What is the maximum number of nodes in a binary tree if it contains 120 leaf nodes? Justify your answer.

Max nodes = 240 (correct answer is infinite)

(3)

As we go down, no of nodes(max) is doubled  
In 8th level we have 128 nodes(max). But we need 120. So we remove 8 nodes horiz. 4 nodes from above level, 2 from above that & 1 from above that is removed. Total nodes =  $(1+2+4+8+16+32+64+128) - (1+2+4+8) = 16+15$

7. [5] Indicate how the list below will change when we call the following routine  
first-> 57 - 55 - 92 - 40 - 62 - 84 - 78

void magic(node \*first)  
{  
 struct node \*p = first,  
 \*q = NULL,  
 \*r;  
 while (p != NULL)  
 {  
 :  
 r = q;  
 q = p;  
 p = p->next;  
 q->next = r;  
 }  
 q = first;  $\rightarrow$  error: first=q;  
}

Output:

first-> 78 - 84 - 62 - 40 - 92 - 55

should be there

Name \_\_\_\_\_

Entry Number \_\_\_\_\_

Gp No. \_\_\_\_\_

Your Lab day \_\_\_\_\_

Your TA \_\_\_\_\_

Answer all the questions in the space provided for each question. You can use the last page for rough work.

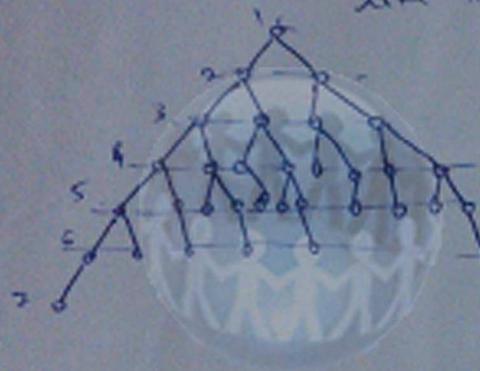
1. [5] What is the maximum and the minimum number of nodes in an AVL tree with 7 levels?

Maximum number of nodes in AVL tree with 7 levels

$$= 2^7 - 1 = 127$$

Minimum no of nodes: level 1  $\rightarrow 1$

The tree will be (complete) like this



$$2 \rightarrow 2$$

$$3 \rightarrow 2^2 = 4$$

$$4 \rightarrow 2^3 = 8$$

$$5 \rightarrow 2^4 = 16$$

$$6 \rightarrow 2^5 = 32$$

$$7 \rightarrow 2^6 = 64$$

Minimum no = 33

(5)

2. [5] A company maintains records of its customers in an array. The record of each customer contains the name, time stamp indicating the date when he became a customer, and the order placed by the customer. The entries in the array are stored in alphabetical order of the names of the customers. Currently there are around 80,000 customers. The company wants to give a special gift to its 4<sup>th</sup> customer in order of placing orders with the company. Indicate the most efficient way of picking up the winner from the array.

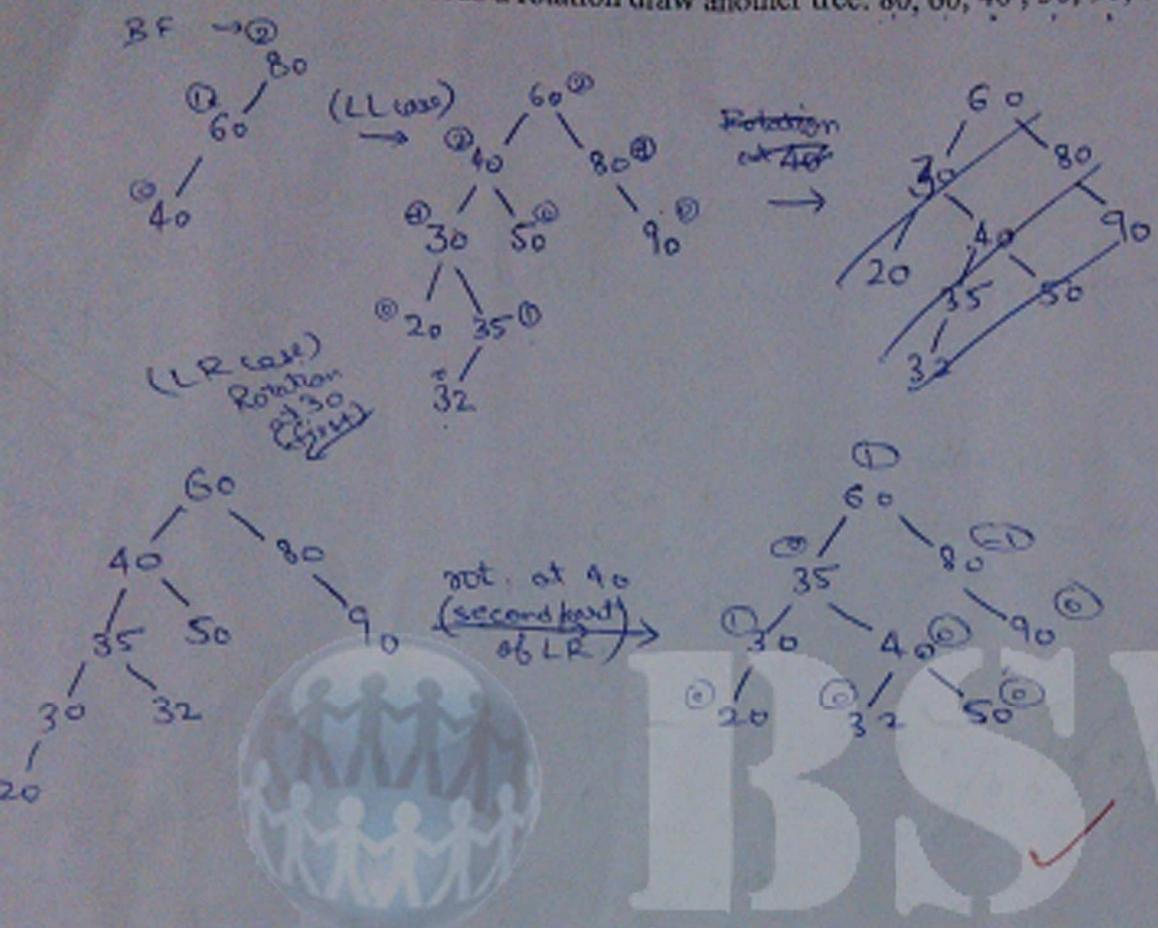
Assuming that order of placing orders is same as order of time stamp.

We pick first four customers, create max heap acc to time stamp. If next element is less than max element, we replace & heapify else go one to next element. At last the heap is change to min heap & take the first element.

(5)

3. [6] Draw an AVL tree by inserting the following elements in the given order. Whenever the structure of the tree needs a rotation draw another tree: 80, 60, 40, 30, 90, 50, 20, 35, 32.

(6)



The number in circle is Balance factor = height(left) - height(right)

4. [4] We want to put 30,000,000 records in a B-tree of order 230. What is the maximum height of the B-tree?

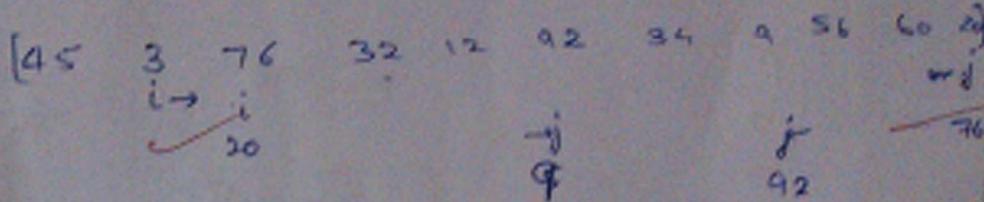
The ~~maximum~~ height of a B-tree with of order 230 with 30,000,000 records is

$$h = \lceil \log_{230} (30,000,000) \rceil = \lceil 3.165 \rceil = 4$$

$\lceil \frac{m}{2} \rceil$  0.5

5. [6] Sort the following array using the quick algorithm that we did in the class. You should use the first element as the pivot. Show the arrays that are formed during the algorithm:  
 [ 45 3 76 32 12 92 34 9 56 60 20 ]

→ Respecting 45 as pivot



(as 45 will be swapped with 34, the following array is formed)

$$[ \underline{34} \ 3 \ 20 \ 32 \ 12 \ 99 ] [ 45, \underline{92} \ 56 \ 60 \ 20 ]$$

↓

$$[ \underline{9} ; \underline{3} \ 20 \ 32 \ 12 \ 99 ] [ \underline{34} ; \underline{45} ] [ \underline{20} \ 56 \ 60 ; \underline{92} ]$$

$$\textcircled{3} \quad [ \underline{3} ; \underline{9} \ 12 ; \underline{20} ; \underline{32} \ 34 ]$$

$$[ [ \underline{9} \ 3 \ 20 \ 32 \ 12 ] [ 34 ] ] [ 45 ] [ \underline{26} \ 56 \ 60 ] [ 92 ] ]$$

$$[ [ \underline{3} ; \underline{9} ; \underline{20} \ 32 \ 12 ] ] [ 34 ] [ 45 ] [ \underline{60} \ 56 ] [ 26 ] [ 92 ] ]$$

$$[ [ \underline{3} ; \underline{9} ; [ \underline{12} ; \underline{20} ] [ 32 ] ] ] [ 34 ] [ 45 ] [ 56 ; [ 60 ; 26 ] ] [ 92 ] ]$$

$$\Rightarrow [ 3 \ 9 \ 12 \ 20 \ 32 \ 34 ] [ 45 ] [ 56 \ 60 \ 76 \ 92 ]$$

$$\Rightarrow [ 3 \ 9 \ 12 \ 20 \ 32 \ 34 \ 45 \ 56 \ 60 \ 76 \ 92 ]$$

6. [6] Solve the following recurrence relation. Continue to next page if necessary.

$$T(n) = 2 T\left(\frac{n}{5}\right) + n$$

$$T\left(\frac{n}{5}\right) = 2 T\left(\frac{n}{25}\right) + \frac{n}{5}$$

$$\Rightarrow T(n) = 2 \left[ 2 T\left(\frac{n}{25}\right) + \frac{n}{5} \right] + n = 4 T\left(\frac{n}{25}\right) + \left( \frac{2n}{5} + n \right)$$

$$T\left(\frac{n}{25}\right) = 2 T\left(\frac{n}{5^3}\right) + \frac{n}{25}$$

$$\Rightarrow T(n) = 4 \left[ 2 T\left(\frac{n}{5^3}\right) + \frac{n}{5^2} \right] + \left( \frac{2n}{5} + n \right) = 2^3 T\left(\frac{n}{5^3}\right) + \left( \frac{2^2 n}{5^2} + \frac{2n}{5} + n \right)$$

$$\Rightarrow T(n) = 2^k \cdot T\left(\frac{n}{5^k}\right) + \left( 1 + \frac{2}{5} + \left(\frac{2}{5}\right)^2 + \dots + \left(\frac{2}{5}\right)^{k-1} \right) n$$

$$\text{For } \frac{n}{5^k} = 1 \Rightarrow 5^k = n \Rightarrow k = \log_5 n$$

$$\Rightarrow T(n) = 2^{\log_5 n} \cdot T(1) + \left[ 1 + \left(\frac{2}{5}\right) + \dots + \left(\frac{2}{5}\right)^{\log_5 n - 1} \right] \cdot n$$

$$\text{as } T(1)=1 \Rightarrow T(n) = 2^{\log_5 n} + \frac{1 \cdot \left[ 1 - \left(\frac{2}{5}\right)^{\log_5 n} \right]}{\left(1 - \frac{2}{5}\right)} \cdot n$$

... no of elements  
 $\ln C = P = \log_5 n$   
 $C = a \cdot \frac{(1-\lambda^n)}{(1-\lambda)}$   
 $\lambda < 1$

$$= 2^{\log_5 n} + \frac{5}{3} \left( 1 - \frac{2^{\log_5 n}}{n} \right) \cdot n = 2^{\log_5 n} + \frac{5}{3} \left( n - 2^{\log_5 n} \right)$$

$T(n) = \frac{5}{3}n - \frac{2}{3} \cdot 2^{\log_5 n} = \frac{1}{3} [5n - 2^{1+\log_5 n}]$

7. [8] Max-heap is a heap where the value at a node is larger than the values at the children.

Given an array representation of a max-heap, complete the following code for **heapify** operation.

void heapify(int A[], int m) { // m is the number of elements in A

int i, n=m/2; int k;

for (i=n; i>0; i--) {

if ( $A[i] \geq A[2i] \text{ & } A[i] \geq A[2i+1]$ )

{ k = A[i]; k = A[i];

A[i] = A[2i]; A[2i] = k;

percolate\_down(A, 2i);

}

else if ( $A[i] < A[2i+2] \text{ & } A[2i+1] > A[2i]$ )

{

k = A[i];

A[i] = A[2i+1];

A[2i+1] = k;

percolate\_down(A, 2i+1);

}

percolate\_down

int (int n(), int k(), int l(), int r(), int 2k(), int 2k+1());

{ int r(); if (2k > N) return;

else if ( $A[k] < A[2k] \text{ & } A[k] \geq A[2k+1]$ )

{ k = A[k];

A[k] = A[2k]; A[2k] = k;

percolate\_down(A, 2k);

else if ( $A[k] < A[2k+1] \text{ & } A[2k+1] > A[2k]$ )

{ k = A[k]; A[k] = A[2k+1]; A[2k+1] = k; percolate\_down(A,

Check  
children  
exist  
not ?

right child  
not left ?

Name \_\_\_\_\_ Entry Number \_\_\_\_\_ Gp No. \_\_\_\_\_

Your Lab day \_\_\_\_\_

*Answer all the questions in the space provided for each question.*

**Q1.[15 marks]** Answer the following questions with proper justification (no marks will be awarded if the reasoning is incorrect.)

a. What is the running time complexity of the following code ? You can use order notation. Justify your answer.

```
sum = 0;
for (k=1; k<n; k++)
    for (i=1; i<k*k; i++)
        if (i % k == 0)
            for (j=0; j<i; j++)
                sum++;

```

$$\begin{aligned}
 & \sum_{k=1}^n [0 + k + 2k + \dots + k \cdot k] \\
 &= \sum_{k=1}^n k [1 + 2 + \dots + k] \\
 &= \sum_{k=1}^n k^2 \frac{(k+1)}{2} = O(n^4).
 \end{aligned}$$

b. Is it true that  $\log 1 + \log 2 + \dots + \log n$  is  $\Theta(n \log n)$ ? Explain your answer.

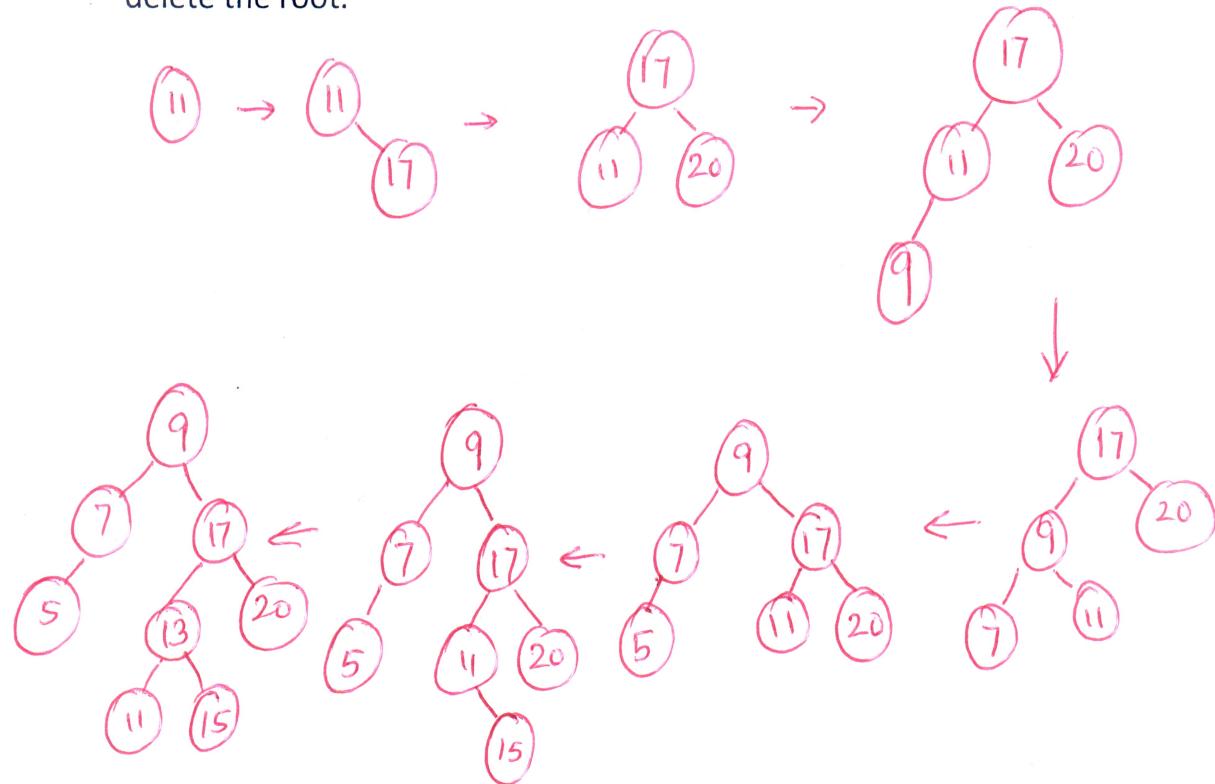
Yes. Two parts :

$$\textcircled{1} \quad \log 1 + \dots + \log n \leq \log n + \dots + \log n = n \log n \Rightarrow \text{LHS is } O(n \log n).$$

$$\begin{aligned}
 \textcircled{2} \quad \log 1 + \dots + \log n &\geq \log \frac{n}{2} + \dots + \log \frac{n}{2} \geq \log \frac{n}{2} + \dots + \log \frac{n}{2} = \frac{n}{2} \log \frac{n}{2} \\
 &= \Omega(n \log n).
 \end{aligned}$$

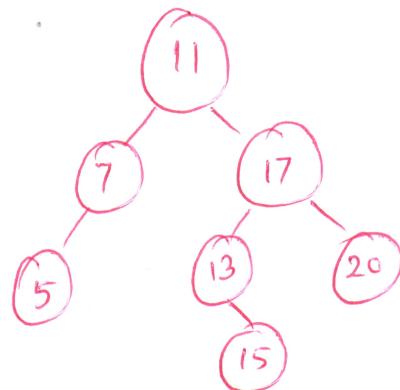
∴ LHS is both  $O(n \log n)$  and  $\Omega(n \log n) \Rightarrow \Theta(n \log n)$ .

**Q2.[10 marks]** Starting from an empty AVL tree, the following elements are inserted (in this order) in the AVL tree: 11, 17, 20, 9, 7, 5, 15, 13. You should use the rotation/balancing procedures described in the class. Show the AVL tree after each insert operation. After inserting the above numbers in the AVL tree, delete the root.



Delete 9

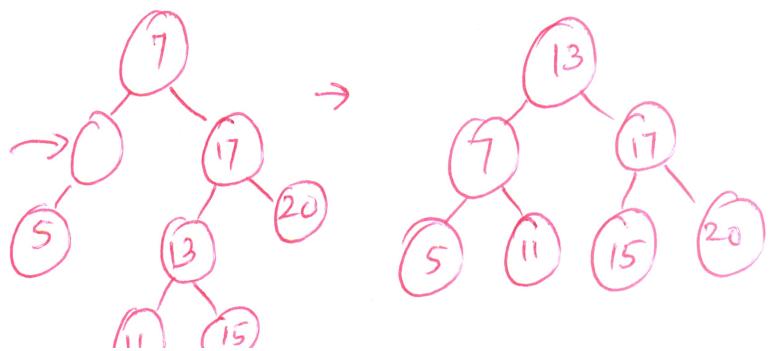
i) Delete successor node of 9:



OR

ii) Delete predecessor node:

Delete this



**Q3.[15 marks]** Given a node A in a binary search tree, write an efficient procedure to find its inorder predecessor. Fill in the details of the procedure below. Do not declare any other function.

Node\* Predecessor ( Node\* A)

```
{
    if (A->right is not a leaf) {
        p = A->right;
        while (p->left is not a leaf)
            p = p->left;
        return p;
    }
    else {
        p = A;
        q = A->parent;
        while (p is the right child of q) {
            p = q;
            q = p->parent;
        }
        return q;
    }
}
```

7 marks

8 marks

**Q4.[20 marks]** Given a stack, and a sequence of distinct numbers in an array A = [a<sub>1</sub>, .., a<sub>n</sub>] you can perform the following operations starting from the first number in A: either push the next number in the sequence on the stack, or pop from the stack.

Given a permutation B =[b<sub>1</sub>, .., b<sub>n</sub>] of A=[a<sub>1</sub>, ..., a<sub>n</sub>], we would like to know if it can be obtained from a<sub>1</sub>...a<sub>n</sub> by such operations.

For example, let A =[1 2 3 4 5] and B = [2 3 5 4 1]. Then B can be obtained from A by the following operations:

Push 1. Push 2. Pop 2. Push and Pop 3. Push 4, push 5. Pop 5. Pop 4. Pop 1.

(i) Let A = [1 2 3 4 5], B = [ 4 3 1 5 2]. Can B be obtained from A ? Why?

No. First we need to push 1,2,3,4 and then pop, pop, Now, 1 cannot } 3.  
be printed before 2.

(ii) Write a **linear time** procedure which given A and B of length n each, outputs whether B can be obtained from A using the method above. If yes, the procedure should output the sequence of operations. Note that you can access the stack using **push**, **pop**, **is\_empty** and **top** functions only. You should not modify the arrays A and B, and should use only a constant amount of extra space (besides using the stack and reading from the two arrays A and B).

$i = 0$ ; ( $i$  scans A)

$j = 0$ ; ( $j$  scans B)

while ( $j < n$ ) {

    if (stack S is not empty and  $S.\text{top}() == B[j]$ ) {

        7 marks

$S.\text{pop}();$   
         $j++;$   
    }

    else {

        7 marks.

$S.\text{push}(A[i]);$   
         $i++;$  ← if  $i$  exceeds  $n$ , "ERROR"  
    }

3 marks

}

If quadratic time algorithm, max. 5 marks.

Worse than quadratic time: NO MARKS.

Name: Sankalp

Entry No: 2014PH10822

COL106: Data Structures, I semester, 2015-16.

## **Minor I**

2:30 PM to 3:30 PM, 1st September 2015.

Question	1 (4 marks)	2 (6 marks)	3 (6 marks)	4 (4 marks)	Total (20 marks)
Marks	4	5.5	0	4	13.5

Write your answers on the printed question paper in the space provided. ROUGH SHEETS WILL NOT BE COLLECTED.

---

**Q1. (Time Complexity and  $\mathcal{O}$ -notation. Total marks = 4)**

Marks: 4

Suppose we have some already defined functions  $g(n)$  and  $h(n)$ , consider the following function definition of the function  $f(n)$ .

```

void f (int n)
{
    c = 1;
    for i = 1 to n
    {
        print g(i);
        if (i = c)
            then
            {
                print h();
                c = 2*c;
            }
    }
}

```

In the following we will assume that each assignment statement (e.g.  $c = 1$ ) takes 1 unit of time, each time the for statement is run it takes 2 units of time, each a print statement is executed it takes 1 unit of time, and each multiplication takes 1 unit of time.

In each of the following questions you have to calculate the exact number of time units it takes for  $f(n)$  to run, and also write the time complexity in simplified big-Oh notation, e.g., if the number of time units turns out to be  $3n^3 + 2\log n$ , you must give the final answer as  $O(n^3)$ .

**Q1.1. (1.5 marks)** Calculate the number of time units and time complexity of  $f(n)$ , if  $g(i)$  takes  $i$  units of time and  $h(i)$  takes 1 unit of time.

time units

assuming each comparison takes 1 unit  
of time

$$f(n) = \cancel{+} [i+4]n + 3 \log_2 n$$

$$1 + \cancel{4n} + \frac{n(n+1)}{2} + 3\log_2 n = \frac{1}{2}n^2 + \frac{9}{2}n + 3\log_2 n + 1$$

## Time complexity

$O(n^2)$

1.5 1

Name: \_\_\_\_\_

Entry No: \_\_\_\_\_

Name: \_\_\_\_\_

Q1.2. (2.5 marks) Calculate the number of time units and time complexity of  $f(n)$ , if  $g(i)$  takes  $\log i$  units of time and  $h(i)$  takes  $i$  units of time.

$$[ \cdot ] \rightarrow g \cdot h$$

time units taken by  $f(n)$

$$= 1 + 4n + \sum_{i=1}^n \log_i + \cancel{\sum_{i=1}^{(2+i)} i} + 2 \log_2 n + \sum_{i=1}^{\lceil \log_2 n \rceil} i$$

$$= 1 + 4n + \log(n!) + 2 \log_2 n + \cancel{\sum_{i=1}^{\lceil \log_2 n \rceil} i} + \frac{\lceil \log_2 n \rceil (\lceil \log_2 n \rceil + 1)}{2}$$

Time complexity  $O(\log(n!))$

Marks: 5/5

Q2. (Linked Lists. Total marks = 6).

Consider the following function that takes two linked lists as input:

```

list lf (list a, list b)
{
    temp1 = a;

    if b = null
        then
            throw exception Ex;
        else
            temp2 = b;

    temp3 = null;

    while (temp1 not equal to null)
    {
        if (temp2 = null)
            then
            {
                x = new node;
                x.next = temp3;
                temp3 = x;
                temp2 = b;
            }
        temp2 = temp2.next;
        temp1 = temp1.next;
    }

    return temp3;
}

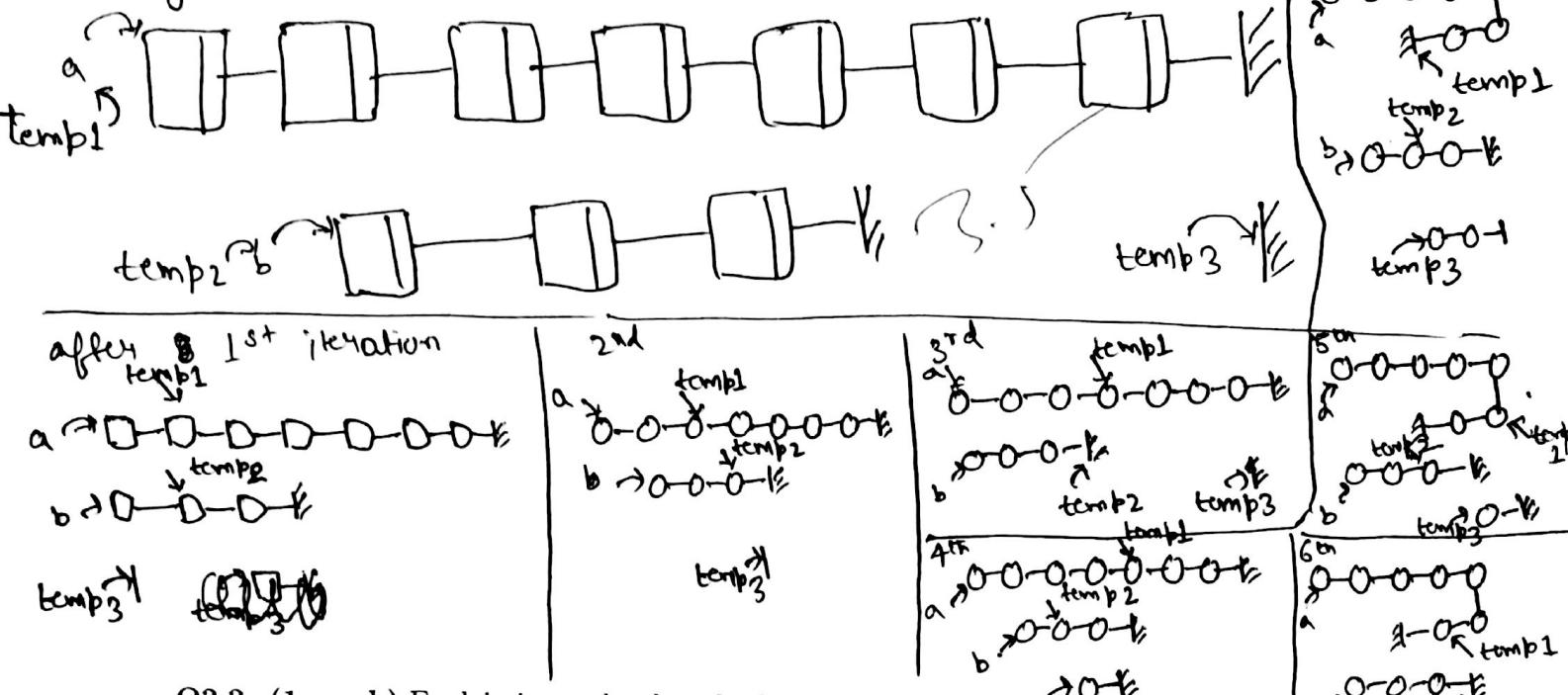
```

Name: Sankalp

Entry No: 2014PH10B22

**Q2.1. (3.5 marks)** Suppose we call 1f with the list a having 7 nodes and b having 3 nodes, show the state of the three lists (a, b, temp3) and the positions of the pointers temp1, temp2 at the end of each iteration of the while loop. Note that the function 1f does not do anything to the data in the nodes so you can draw the list with the space for data left empty.

Initially



**Q2.2. (1 mark)** Explain in words what the function 1f does.

The function 1f returns a linked list with no. of nodes equal to ~~no. of~~ ~~at least~~ integer greatest integer before the value of no. of nodes of a divided by no. of nodes of b.

**Q2.3. (1.5 marks)** Assume that the only operation that takes 1 unit of time is the .next operation (e.g. temp1.next) and all other operations take 0 units of time. Calculate the amount of time taken by 1f if the size of a is m and the size of b is n. Your answer must be in terms of m and n. Calculate the exact number of steps and also give the answer in terms of big-oh notation.

$$\begin{aligned} \text{time taken} &= m + m + [ \frac{m}{n} ] \\ \text{by I} &= 2m + [ \frac{m}{n} ] \end{aligned}$$

1.5

$$\text{time complexity} = O(m)$$

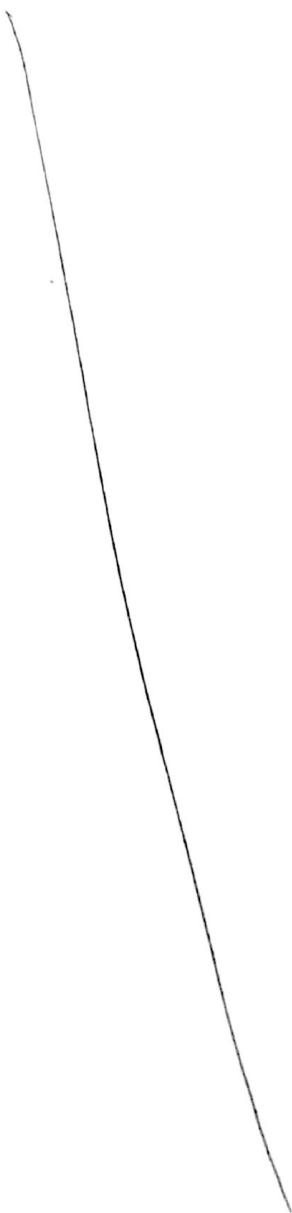
Name: \_\_\_\_\_

Entry No: \_\_\_\_\_

**Q3. (Queues. 6 marks).**

Marks: 0

A queue contains three types of characters R (red), W (white) and B (blue) in a jumbled up (assorted) fashion. The total number of characters is say  $n$  and  $n$  is known to you. Write an algorithm to arrange these characters in the queue such that all the R's come first, the W's come next, and B's come last in the queue. You may use one additional queue and one or two constant space variables if required. Your algorithm is **not** allowed to create or destroy any character of the input, it can only rearrange them. Please write your algorithm in plain English giving exact steps. *No code or pseudocode. If you write code or pseudocode you automatically get 0.*



Name: Sankalp

Entry No: 2014PH10822

Marks: 4

**Q4. (Trees. Total marks = 5)**

Suppose we are given a java implementation which has a `Tree` class and a `Node` class. The `Tree` class has the following methods available:

- `public Node root()`: This returns the root of the tree.
- `public Boolean isEmpty()`: This returns 1 if the tree is empty, 0 otherwise.

The `Node` class has the following methods available:

- `public int data()`: This returns the integer data value stored in the node.
- `public int no_children()`: This returns the number of children of the node.
- `public Node child(int i)`: This returns the `Node` of the *i*th child, returning `null` if there are less than *i* children.
- `public Tree subtree(int i)`: This returns the `Tree` rooted at the *i*th child, returning an empty tree if there are less than *i* children.

Consider the following functions:

```
public int f1 (Tree T) {  
    if (T.isEmpty()) {  
        return 0;  
    }  
    else {  
        Node curr = T.root();  
        int temp = curr.data();  
        if (temp < 0) { temp = 0; }  
        int i = 0;  
  
        while (i < curr.no_children()) {  
            temp = temp + f1(curr.subtree(i));  
            i++;  
        }  
        return temp;  
    }  
}  
  
public int f2 (Tree T) {  
    if (T.isEmpty()) {  
        return 1;  
    }  
    else {  
        Node curr = T.root();  
        if (curr.no_children() > 2) {
```

Name: \_\_\_\_\_

Entry No: \_\_\_\_\_

```
        return 0;
    }
else {
    int temp = 1;
    int i = 0;

    while (i < curr.no_children()) {
        if (temp > f2(curr.subtree(i))) {
            temp = 0;
        }
        else
            i++;
    }
    return temp;
}
```

Q4.1. (1.5 marks): Given a tree T explain in words what the function  $f1(T)$  outputs?

$f1(T)$  gives the sum of all the non -ve values stored in that tree. (or each and every node/n<sup>subtree</sup> is considered and the value is added to a counter if it is non -ve finally the counter is returned) 1.5

Q4.2. (2.5 marks): Given a tree T explain in words what the function  $f2(T)$  outputs?

returns zero if not a binary tree / subtree  
For  
binary tree / subtree returns 1 2.5

Name: Kamalnath Polakam

Entry No: 2018CS10244

COL106: Data Structures. I semester, 2016-17.

Minor II

1 PM to 2 PM, 9th October 2016.

Attendance Sheet Serial Number: 67			
Question	1 (11 marks)	2 (9 marks)	Total (20 marks)
Marks	8	8	16

0. The Dean has instructed us to give **no clarifications during the course of the exam** to prevent students from being disturbed. If you have any doubts, please state your assumptions and answer the question to the best of your ability.

1. Write your answers on the **printed question paper** in the space provided. **ROUGH SHEETS WILL NOT BE COLLECTED.**

2. Please write your name on every page and enter your serial number in the box above. **If you miss out any of these: -1 and no rechecking.**

3. **No pseudocode** means: For every loop you use, you must explain what it will do to the input. You cannot write things like " $x = x + y$ ", you must explain the significance of every step in words. You cannot write "for  $i = 1$  to ...". or "while <condition>..."., you must *explain* what the loop achieves. In summary: if we need to interpret how your description will treat a particular input then it is pseudocode.

**Q1. Short answer questions (Total marks = 11)**

Marks: 8

**Q1.1 (2 marks)** Consider the 2-4 tree given in Figure 1.

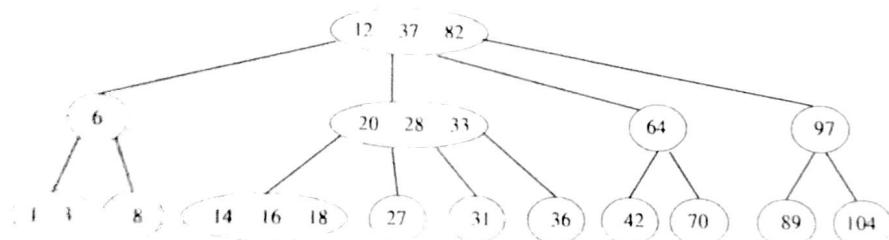
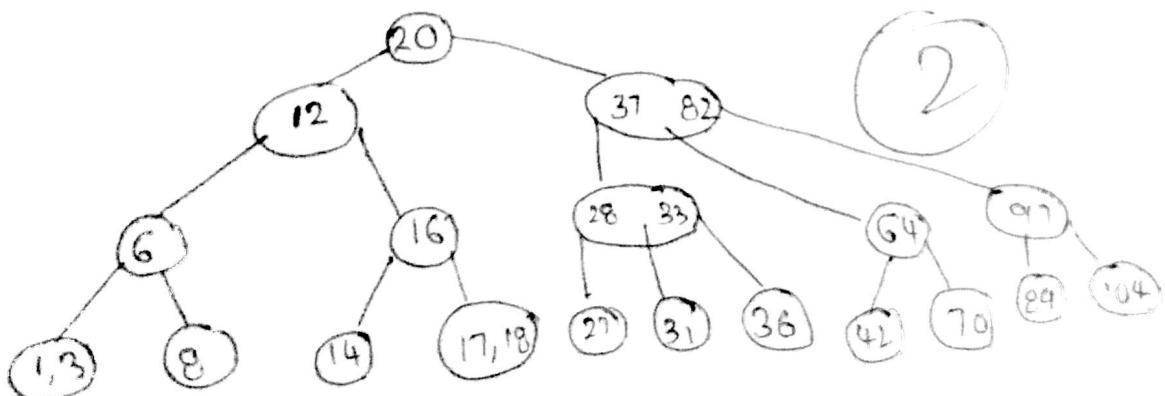


Figure 1: A 2-4 tree

Draw the state of the tree after inserting 17. You *do not* have to show the intermediate steps, only the final answer.



**Q1.2 (2 marks)** Consider open addressing with *quadratic probing* i.e. if the hashtable  $A$  has size  $n$ , and the hash function is  $h(\cdot)$ , then if we have to insert  $x$  and if the location  $A[h(x) \bmod n]$  is occupied, we try  $A[h(x) + 1^2 \bmod n]$ ,  $A[h(x) + 2^2 \bmod n]$ ,  $A[h(x) + 3^2 \bmod n]$ , etc in sequence till we find an empty slot and insert  $x$  there. Consider a hashtable of size 13 and the hash function  $h(x) = (3x + 11) \bmod 13$ . Assume that 21, 43 and 57 are already inserted into this hashtable. Where will 73 be inserted? To get full marks you must explain how you arrived at the answer. (1 mark for the correct answer and 1 mark for the correct explanation. Zero if only the answer is written with no explanation).

First 21

$$\begin{aligned} h(21) &= (3 \times 21 + 11) \bmod 13 \\ &= 74 \bmod 13 \\ &= 9 \end{aligned}$$

57							21	43			
0	1	2	3	4	5	6	7	8	9	10	11

Second 43

$$h(43) = (3 \times 43 + 11) \bmod 13$$

10

Third 57

$$h(57) = (3 \times 57 + 11) \bmod 13$$

$$AC_0 = 57$$

Now 73	$n(73) = (3 \times 73 + 11) \bmod 13$	$\Rightarrow 230 \bmod 13$	Prob 1	$9 + 1^2 \bmod 13 = 10$	$AC(10) = 43$	Prob 2	$(1+2)^2 \bmod 13 = 5$	$AC(5) = 57$	already filled	21 43

**Q1.3 (2 marks)** Prove that the following statement is false: It is not possible to make an AVL tree on 10 or more nodes such that the left subtree of the root contains less than one third of the nodes that the right subtree contains.

**Q1.4 (3 marks)** Consider a probabilistic skip list with parameter  $p$  (i.e. a key in  $S_i$  is selected for  $S_{i+1}$  with probability  $p$ ) containing the elements  $x_1, x_2, \dots, x_n$  numbered in ascending order (i.e. the order in which they appear in the base list  $S_0$ ). Consider two elements  $x_i$  and  $x_j$  such that  $i < j$ . Under what condition (on the heights of the various elements in the skip list) is it true that the search path to  $x_j$  will go through  $x_i$ ? What is the probability that the search path to  $x_j$  goes through  $x_i$ ? (2 marks for the condition and 1 mark for the probability calculation).

b Condition :-  $height(x_i) \geq height(x_j)$  and  $\forall k \in (i, j) \quad height(x_k) \leq height(x_j)$

$$height(x_k) \leq height(x_j)$$

Probability  
calculation



$$P(h(x_i) > h(x_j), \forall k \in (i, j) \quad h(x_k) \leq h(x_j))$$

$$h(x_t) = ht$$

2

$\Phi(P)$

$$\max(h_{i+1}, h_{i+2}, \dots, h_j) + 1 \geq ht$$

$$\forall t \in [i+1, j]$$

**Q1.5 (2 marks)** Any find path in a binary search tree can be written as a sequence of turns, some left turns and some right turns, e.g., if the root has key 12 and we are searching for 26, the first turn is Right and so on. Let this sequence of turns be  $t_1, t_2, \dots, t_k$  (each  $t_i$  is either L or R) and let  $x_1, x_2, \dots, x_k$  be the nodes at which the turns were taken (e.g.  $x_1$  is always the root). If  $t_i$  and  $t_j$  are both left turns, under what condition is  $x_j > x_i$ ?

A) ~~As we have seen, when node by node has key as~~

As during traversal it is given that  $t_i$  and  $t_j$  are both left turns, the node we are searching for is less than both  $x_i$  and  $x_j$ .  
**Case 1:** If  $i > j$ , that means the node with key  $x_i$  is a child of  $x_j$  (since BST traversal path goes in this way). As  $x_j > x_i$ , so for  $x_j > x_i$ .

(In all other cases, Node( $x_j$ ) is right of Node( $x_i$ ), so it is  $x_j < x_i$ )

Marks: 8

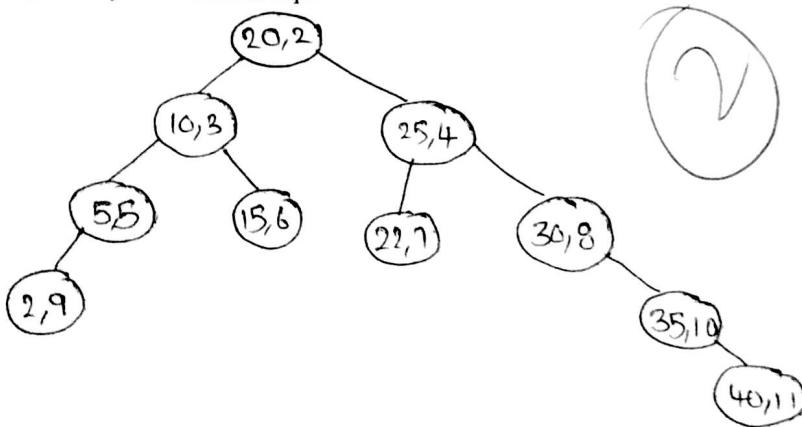
**Q2. (Tree + heap = Treap. Total marks = 9).**

A Treap is a special kind of binary search tree which also has the property of a heap. Each item in a treap is a tuple of two values i.e.  $x_i = (k_i, p_i)$  where  $k_i$  is a key and  $p_i$  is a priority. A treap is a binary search tree on the keys of the items and maintains the *min-heap order* property on the priorities i.e. the priority value of a node is less than the priority value of any children it may have. Note that the Treap *does not* have to have the structural property of a heap i.e., it may not be a complete binary tree but must have the order property of a heap.

**Q2.1 (2 marks).** Construct a Treap on the set

$$S = \{(22, 7), (20, 2), (2, 9), (10, 3), (35, 10), (40, 11), (15, 6), (30, 8), (25, 4), (5, 5)\}.$$

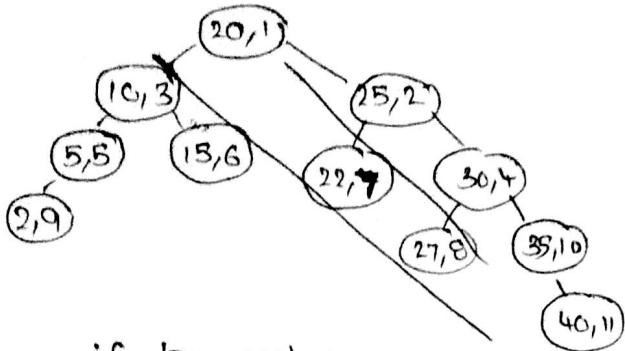
You do not need to show the steps, only the final treap.



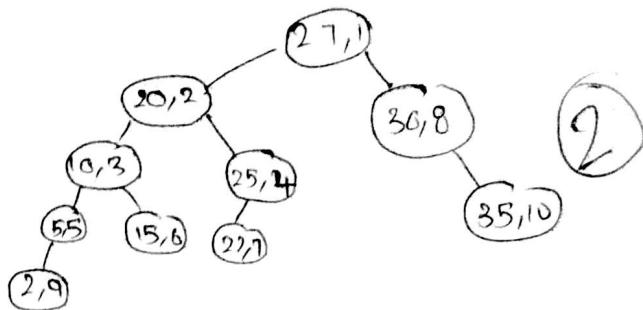
so time O  
complexity is

**Q2.2 (2 marks).** In the Treap constructed in the previous part insert (27, 1). The insertion algorithm proceeds by first inserting the node in its appropriate place as a binary search tree node and then adjusting the key correctly. Draw the heap after the insertion has completed. (Use the space given overleaf on page 4).

SA (Continued)) As it can be seen through rotations every sub BST can be successfully converted to new sub treap with  $N'$  as its new root. The rotations take constant time, B if the whole tree can't be converted to a treap in 'phase' then we must continue to 'Next phase'. This is recursive in nature and should be continued until the whole tree becomes treap. In this case the  $N'$  becomes the new root so whole tree height is  $\downarrow$ .



if key values can be swapped.



if key values can't be swapped.

**Q2.3 (5 marks)** Explain the Treap insertion algorithm. As mentioned before you must begin by doing a BST insertion of the new (key, priority) pair and then adjust the priorities. Explain how this adjustment phase proceeds. Note: (1) No pseudocode. (2) You must justify your steps, i.e. argue the for the correctness of your algorithm. Without an argument of correctness you will get a maximum of 2 marks even for a perfectly correct answer. (3) Your insertion mechanism should work in  $O(h)$  time where  $h$  is the height of the Treap. (Hint: If the priorities and keys are all different, as they are in this case, then any set of keys gives a unique Treap, so in the previous part you can find the correct answer simply by constructing a Treap on  $S \cup \{(27, 1)\}$  from scratch. The purpose of the previous part is to help you figure out the adjustment phase of the insertion algorithm. Since you can tell the final answer by constructing the final Treap from scratch, you can check using this example if you are doing the adjustment phase correctly.)

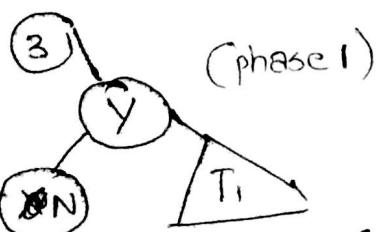
#### A) Treap insertion algorithm.

1) Begin with root if  $\text{root}(\text{key}) < \text{NewNode}(\text{key})$  go to right subtree of the root and repeat the same algorithm, if  $\text{root}(\text{key}) > \text{NewNode}(\text{key})$  go to the left subtree of the root and repeat until you reach a child, (This is ! insertion).

You also need to justify that the rotation maintains the heap order property

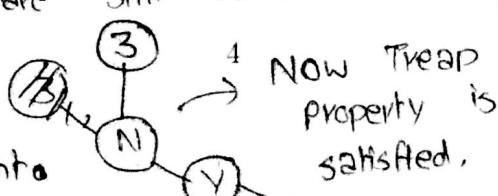
2) Now adjustment phase, if they newly created tree follows treap properties then well and good. otherwise we get cases.

$N$  is new node



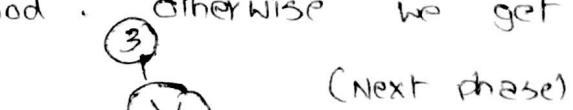
phase 1, we must rotate this subtree such that BST properties are still valid.

Note:- ~~3~~



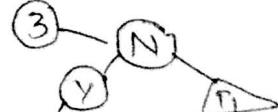
if Y was right child

Before,  $3(\text{key}) \leq N(\text{key}) \leq Y(\text{key}) \leq \text{Root}(T_1)(\text{key})$   
Now,  $3(\text{key}) \leq N(\text{key}) \leq Y(\text{key}) \leq \text{Root}(T_1)(\text{key})$ .



(next phase)

Phase 1 already happened so now the subtree's T and 'N' are treaps but Y is just a BST.



if Y was right child.  
Before,  $3(\text{key}) \leq N(\text{key}) \leq Y(\text{key})$   
Now,  $3(\text{key}) \leq N(\text{key}) \leq Y(\text{key})$ .

Attendance Sheet Serial Number: 186					Total
Question	1 (5 marks)	2 (5+2 marks)	3 (5 marks)	4 (5 marks)	
Marks	4	0.50	2.5	0	6.5

1. Write your answers on the printed question paper in the space provided. **ROUGH SHEETS WILL NOT BE COLLECTED.**

2. Please write your name on every page and enter your serial number in the box above. **If you miss out any of these: -1 and no rechecking.**

3. **No pseudocode** means: For every loop you use, you must explain what it will do to the input. You cannot write things like " $x = x + y$ ", you must explain the significance of every step in words. You cannot write "for  $i = 1$  to ..." or "while <condition>...", you must *explain* what the loop achieves. In summary: if we need to interpret how your description will treat a particular input, then it is pseudocode.

### Q1. (Trees. marks = 5)

Marks: 4

Suppose we are given a java implementation which has a **Tree** class and a **Node** class. The **Tree** class stores integer values as data at the nodes and has the following methods available:

- **public Node root()**: This returns the root of the tree.
- **public Boolean isEmpty()**: This returns 1 if the tree is empty, 0 otherwise.

The **Node** class has the following methods available:

- **public int data()**: This returns the integer data value stored in the node.
- **public int no\_children()**: This returns the number of children of the node.
- **public Node child(int i)**: This returns the **Node** of the *i*th child, returning **null** if there are less than *i* children.
- **public Tree subtree(int i)**: This returns the **Tree** rooted at the *i*th child, returning an empty tree if there are less than *i* children.

Consider the following function:

```
public int f1 (Tree T) {
    if (T.isEmpty()) {return 0;}
    else {
        Node curr = T.root();
        int temp = curr.data();
        if (temp < 0) { temp = 0;}
        int i = 0;
        temp2 = 0;
        while (i < curr.no_children()) {
            temp2 = f1(curr.subtree(i));
            if temp2 > temp then {temp = temp2;}
            i++;
        }
        return temp;
    }
}

public int f2 (Tree T) {
    if (T.isEmpty()) { return 1;}
    else {
        Node curr = T.root();
```

```

int f1(T)
{
    int flag = 1;
    while (i < curr.no_children() && (flag > 0)) {
        temp = curr.subtree(i);
        if (curr.data() > temp.root().data()) {flag = 0;}
        else {flag = flag*f2(temp);}
        i++;
    }
    return flag;
}

```

**Q1.1. (2 marks):** Given a tree T explain in words what the function  $f_1(T)$  outputs? Your answer must be as precise as possible.

(1)  $f_1(T)$  outputs the maximum <sup>positive</sup> data stored in all nodes of T

$f_1(T)$  returns 0 if all nodes data is negative

**Q1.2. (3 marks):** Given a tree T explain in words what the function  $f_2(T)$  outputs? Your answer must be as precise as possible.

(3)  $f_2(T)$  outputs 1 if value in each node is greater than value of parent node, else it returns 0.

**Q2. (Amortised analysis. Total marks = 5 + 2 extra credit).**

Marks: 0 + 0

A  $n$ -counter is an array of  $\log n$  bits that can store the binary representation of any number from 0 to  $n - 1$ . Typically such  $n$ -counters are used to store values from 0 to  $n - 1$  in sequence, i.e., they are initialised with all 0s and at every step the value stored is incremented by 1, i.e., if at any time the value stored is  $i$ , it is incremented by 1 to  $i + 1 \bmod n$ . To increment a binary number stored in an  $n$ -counter,  $A$  which is an array with indexes 0 to  $\log n - 1$ , we perform the following algorithm:

- Set  $i = \log n - 1$
- While ( $A[i]$  is not 0) and ( $i \geq 0$ )
  - Set  $A[i]$  to 0
  - $i = i - 1$
- if  $i \geq 0$  then set  $A[i]$  to 1.

This algorithm starts from the last entry and moves left and flipping all the 1s it encounters until it encounters the first 0, which it changes to 1. If it reaches all the way to the beginning of the array without encountering a 0 (which means that the counter contains all 1s) it does nothing, which is correct since when we increment  $n - 1$  we get  $n$  which is  $0 \bmod n$ .

We will now analyse the running time of this algorithm. We will neglect all operations apart from array write operations, i.e. we will count only the number of times the algorithm flips a bit.

**Q2.1. (1 mark)** What is the worst case time of any increment operation in terms of bit flip? On the basis of this answer, what is the worst case running time of the algorithm applied *n* times? Counting from 0 and counting up  $n - 1$  and then back to 0?

~~$O(n)$  is worst case time for increment operation  
by p 3~~

~~$O(n)$  is worst case time~~

**Q2.2. (4 marks)** Prove that the total time taken to count from 0 back to 0 (i.e. 0 to  $n - 1$  and then, with one increment, back to 0) is  $O(n)$ . You may use an argument similar to that for either enqueueing a queue using two stacks (the argument uses chips) or for growable stacks (averaging argument).

**Q2.3\* Extra credit. Only attempt if you have time left (2 marks)** An alternative way to prove the result of Q2.2 is to consider the time taken to increment a random number and then average it over all possible numbers. A way to generate a random binary number between 0 and  $n - 1$  is to set each bit to 0 with probability 1/2 and 1 with probability 1/2. Use this method of generating a random binary number to prove that the time taken to count from 0 back to 0 is  $O(n)$  on average.

### Q3. Time Complexity and O notation. (Total marks = 3)

In this section we have some already defined functions  $g(i)$  and  $h(i)$ . Calculate the time complexity of the function  $f(n)$ .

```

int f(int n)
{
    if (n == 0)
        return 1;
    for (i = 1 to n)
        print(g(i));
        if (i == c)
            then {
                print(h(i));
                i = 2 * n;
            }
    }
}

```

**Q3.1. (2 marks)** Calculate the *exact* number of time units and time complexity of  $f(n)$ , if  $g(i)$  takes  $\log i$  units of time and  $h(i)$  takes  $i$  units of time. Write the time complexity in simplified big-O notation, e.g. if the number of time units turns out to be  $3n^3 + 2\log n$ , you must give the final answer as  $O(n^3)$ . Assume that each assignment statement (e.g.  $c = 1$ ) takes 1 unit of time, each time the **for** statement is executed it takes 2 units of time, each time a **print** statement is executed it takes 1 unit of time, each multiplication takes 1 unit of time, and each comparison operation ( $i == c$ ) takes 1 unit of time.

$$\text{No. of units} = 1 + 2n + n + \sum_{i=1}^n \log i + n + \sum_{i=1}^n 2^i + n + \text{[log } n] + [\log n]$$

Assignment  $\downarrow$        $\downarrow$        $\downarrow$        $\downarrow$        $\downarrow$        $\downarrow$        $\downarrow$   
 for print       $\downarrow$        $\downarrow$        $\downarrow$        $\downarrow$        $\downarrow$        $\downarrow$   
 $g(i)$        $i == c$        $h(i)$       print      print      multiplication  
 since,  $\sum_{i=1}^n \log i$  is  $O(n \log n)$  and  $\sum_{i=1}^n 2^i$  is  $O(n)$ ,  
 therefore the above expression is  $O(n \log n)$

**Q3.2. (3 marks)** Now consider the case where we run this program locally on a machine where all local operations are free but have to wait  $h(i)$  and  $g(i)$  to be computed on a machine that charges money to compute them. The machine is available in two configurations. In configuration  $C_1$ , execution of  $g(i)$  costs Rs 1/r and  $h(i)$  costs Rs r. In configuration  $C_2$ ,  $g(i)$  costs Rs 1 per execution and  $h(i)$  also costs Rs 1 per execution. Argue formally which of the two configurations is cheaper in rupee terms? Answer your answer. If you just give the answer without any mathematical argument, you will receive 0.

~~for all for C,~~

$$f(n) = \text{Cost 1} = \sum_{i=1}^n \frac{1}{r} + \frac{1}{r} \sum_{j=1}^{\lfloor \log n \rfloor} 2^j + \frac{1}{r} \sum_{i=1}^n \frac{1}{r} = O(n^3)$$

$O(n)$   $\log n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{r}$   $\approx O(n^3)$

$$g(n) = \text{Cost 2} = n + (1 + 2 + \dots + \lfloor \log n \rfloor) + n \lfloor \log n \rfloor$$

$$= O(n^2)$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{n^3}{n^2} = \frac{n^3}{n^2} = n < \log \frac{O(n^2)}{n^3} = \log \frac{O(n^2)}{n^3} = O(1)$$

Using definition of little O,  
 $f(n) \in O(g(n))$

Hence, ~~cost for C~~ cost for  $C_2$  on  $C_1$  is cheaper

**Q4. (Sorting with stacks) (5 marks)** You are given two stacks  $S_1$  and  $S_2$  and a queue  $Q$ . There are  $n$  numbers stored in the queue. You have to sort the numbers in ascending order and put them back in  $Q$  in sorted order. You can use  $S_1$  and  $S_2$  for this purpose. You may use no other extra storage but you can use operations like *top* to look at the top of a stack or *front* to look at the front of the queue. Describe in English how to sort the elements of  $Q$  in ascending order using  $S_1$  and  $S_2$ . No pseudocode. Write your answer without using any variable names except  $S_1$ ,  $S_2$  and  $Q$ . If you write pseudocode you get an automatic 0 without your answer being read at all.

Marks 0

1) ~~Sequence~~ ~~....~~

~~If front . Push( ) front( ) to  $S_1$~~

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**  
**MAJOR : CSL201**  
**(Data Structures)**

**Max. Time – 2 hrs.**

**Max. Marks 70**

**Date: 28/Nov/06**

**Note: Answers of all the questions with all parts in the sequence.**

**Q1. (24)**

- a. Write an algorithm to reverse the order of items in a queue Q using stack S. Use standard functions of queue & stack. (4)
- b. Suppose that you have to implement MIN-STACK which supports the stack operations PUSH and POP and a third operation **FindSum** which returns the sum of all elements in the stack, all in O(1) worst case time. Specify the data structures used along with implementation of all these three operations. (1+4)
- c. Design an efficient algorithm to find the middle node of a singly linked list. [Hint use multiple pointers and a single loop]. (4)
- d. Given a list of 'n' integers along with count field with each key containing the count (number of keys < the key), write an efficient algorithm to sort this list without using additional array. State the complexity of your algorithm. (4+1)
- e. You are given two AVL trees T1 and T2. Write an algorithm to merge T1 and T2 to get a height balanced tree. (6)

**Q2. (18)** Please read parts (a to d) of this question carefully and answer all of them.

Rather than writing a stack and queue implementation from scratch, you decide to use a priority queue that is already implemented. You realize that by assigning the right priorities to data items when they are inserted, you can make data come out in either a LIFO or FIFO order. Your priority queue class has the following prototype:

```
class Pqueue
{
public:
    void insert(int priority, dataType D); // Insert D with priority
    dataType getmax(); // Get the max element and remove it
    int empty(); // Return 1 if Pqueue is empty, 0 otherwise
    Pqueue(); // Create an empty priority queue
};
```

and the insert definition says:

```
void Pqueue::insert(int priority, dataType D)
{ // pre: priority is any integer, positive or negative...
```

Both your stack and your queue should have member functions

```
void insert(dataType D); // Insert data into the stack/queue
dataType getnext(); // Get & remove the next element (in LIFO/FIFO order)
int empty(); // Return 1 if stack/queue is empty, 0 otherwise
```

- a. Write the class prototype for STACK and QUEUE which includes private data fields, and use a comment to describe each entry of the class: (2+2)

- b. Write the `getnext()` member function for the STACK. Briefly describe how you would change it if you were to implement a queue. (3+2)
- e. Write the `insert(DataType D)` member function for the STACK. Briefly describe how you would change it if you were to implement a QUEUE. (3+2)
- d. If Pqueue class is implemented with a HEAP, then give the tightest big-O bounds you can for
- `getnext()` for your QUEUE class,
  - `insert()` for your QUEUE class
- Assume N elements in the queue. Briefly explain your answer. (2+2)

Q3. (13)

- a. Given the following Adjacency Matrix, draw unweighted directed graph labelling nodes numerically starting at 1. (4)

	1	1	1
		1	1
1	1		1
			1
1	1	1	
	1	1	

- b. Perform a depth first search on the above graph. Clearly show the tree edges. Start the search at node 1 and always select successors in numerically increasing order. What is the maximum size of the stack during this DFS run? (4+2)
- e. What is the maximum degree of any vertex in the graph? Does the graph have any directed cycles? (2+1)

Q4. (15)

- a. Starting with an empty B-Tree of order 4, insert elements into the B-tree with the following keys, showing the insertion at each step and each split operation: (5)  
 55, 100, 90, 75, 10, 5, 15, 40, 65, 60, 50, 20, 30, ~~41~~
- b. Show the result of inserting 2,1,4,5,8,3,6,7 into an empty splay tree.[Show the tree at the end of each insertion] (4)
- c. Show the result of deleting node 8 from the splay tree constructed in (b). (2)
- d. Suppose we want to use an array to implement a stack. However, we do not know in advance the size to which the stack can grow. Indeed there may not be any apriori limit to the size of the stack. Suppose we start with a 'small' array A of size say 10. Now as long as our array bounds are not exceeded, all we need is a variable "top" that points to top of stack (so A[top] is the next free cell). Each of PUSH and POP operations are O(1). However, when the array is full and we need to push a new element on, we have a problem! In that case we can allocate a new larger array of double the size [say], copy the old one over and then go on from there. This is going to be an expensive operation, so a push that requires us to do this is going to cost a lot. But maybe we can "amortize" the cost over the previous cheap operations that got us to this point. (4)

Prove that with the above implementation of a stack, the cost of an arbitrary sequence of 'n' stack[push/pop] operations to the stack costs at most O(n) and uses at most O(n) memory.

**CSL201 II Sem 2006-07, Major Exam**

3:30PM to 5:30PM, 8th May 2007

**Note.** There are twenty multiple choice questions in this exam. Questions may have more than one correct answer. **Marking all the correct answers of a given question gets you 1.5 marks.** Marking even a single wrong answer gets you **- 0.5 marks.** Marking some correct answers (and no wrong answers) gets you **0.5 marks.**

All answers are to be marked in the response sheet attached to the question paper. Please detach this response sheet from the question paper. There are 6 different question papers with codes from 1 to 6. **Please ensure that the code on top of your response sheet matches the code on the question paper.**

1. You are given a single stack and a sequence of six numbers 123456 that appear as input one at a time. As each number appears it can either be pushed on the stack or sent to output. When a number on the stack is popped it has to be sent to the output (there is no additional storage available.) Given this system, which of the following permutations of this sequence can be output?

- (a) 164235
- (b) 654123
- (c) 465321
- (d) 132546

2. Consider the 2-4 tree given in Figure 2. Which of the following statements are true:

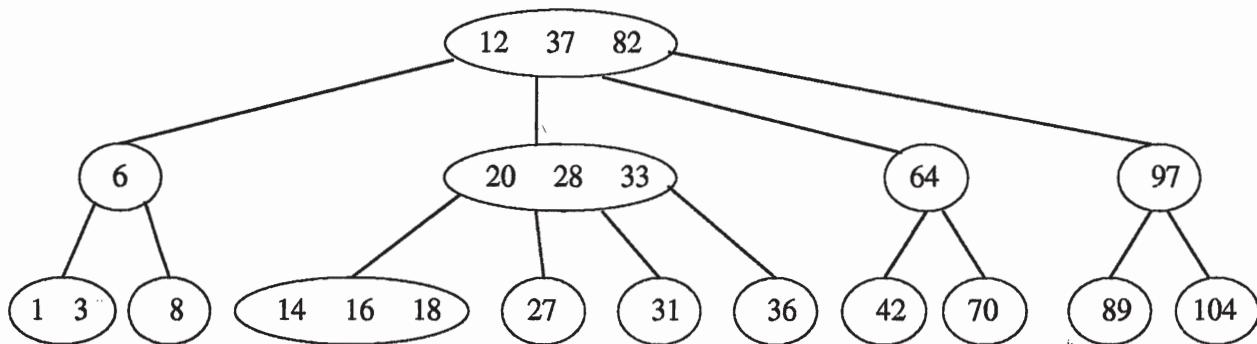


Figure 1: A 2-4 tree

- (a) Inserting 17 increases the height of the tree.
- (b) The tree could have 28 in its root node after 17 is inserted.
- (c) Inserting 90, 91, 92, 93, 94, 95, 96 increases the height of the tree.
- (d) Inserting 29, 30, 32, 33 decreases the height of the tree.

Suppose we are given a minimum spanning tree  $T$  for a graph  $G$ . Let  $e = (u, v)$  be an edge of  $G$  which is not in  $T$ . Suppose we decrease the length of edge  $e$  from  $l_e$  to  $l'_e$ . In which of the following cases, we can be sure that  $T$  will not remain a minimum spanning tree..

- (a)  $l'_e$  is less than the lengths of all other edges which have an end-point in  $u$  or  $v$ .
- (b) The path from  $u$  to  $v$  in  $T$  has an edge of length larger than  $l'_e$ .

- (c) There is a cycle in  $G$  which contains  $e$  such that  $l'_e$  is the length of the smallest edge in this cycle.  
 (d) Every cycle containing  $e$  has at least one edge of length larger than  $l'_e$ .
4. A  $k$ -heap is a heap where each non-leaf node has exactly  $k$  children. It has the same properties as those of a binary heap. What is the worst case running time of **insert** and **delete-min** operations (let  $n$  denote the number of elements in the heap).
- (a)  $O(\log_k n)$  and  $O(k \log_k n)$ .  
 (b)  $O(k \log_k n)$  and  $O(\log_k n)$ .  
 (c)  $O(k \log_k n)$  and  $O(k \log_k n)$ .  
 (d)  $O(\log_k n)$  and  $O(\log_k n)$ .
5. The inorder traversal for a binary tree is DEACBFGIKJOLPHMNRQS. The preorder and postorder traversals are:
- (a) IGFEDCABHJKLOPMNQRS and DABCEFGKOPLJRSQNMHI  
 (b) IGFEDCBAHJKLOPMNQRS and DBACEFGKOPLJRSQNMHI  
 (c) KIGFEDCABHJLPOMNQRS and DABCEFGIPOLJRSQNMHK  
 (d) KIGFEDCABHJLOPMNQRS and DABCEFGIOPLJRSQNMHK
6. Suppose that while your computer is sorting an array of objects, its memory is struck by a cosmic ray that changes exactly one of the keys to something completely different. For which of these sorting algorithms will the final array have just one or two keys out of place in the worst-case scenario?
- (a) Quick sort  
 (b) Merge sort  
 (c) Radix sort  
 (d) Heap sort
7. Suppose that we are using double hashing and have selected the table size  $m$  to be 1200. When we do an insertion, the first hash function determines the point at which we initially probe the table, and the second hash function determines the amount by which we jump  $(\text{mod } 1200)$  as we search for an empty table position. Which of the values below for the second hash function would guarantee that we will find an empty position if there is one in the table?
- (a) 95  
 (b) 74  
 (c) 53  
 (d) 39  
 (e) 27
8. Arrange the following functions in order of their asymptotic growth rates, i.e., if  $f(n)$  appears to the left of  $g(n)$ , then  $f(n)$  is  $O(g(n))$ :
- $2^n, n!, n^n, n^{\log^3 n}$
- (a)  $n^{\log^3 n}, 2^n, n^n, n!$   
 (b)  $n^{\log^3 n}, 2^n, n!, n^n$

- (c)  $2^n, n^{\log^3 n}, n^n, n!$   
 (d)  $2^n, n^{\log^3 n}, n!, n^n$

9. Suppose we start with an AVL tree  $T$ . We do an insertion of a new key in the tree just using the binary search tree algorithm without yet doing any rotations; the result is a tree  $T'$ . It turns out that the deepest node in the tree with a balance outside of  $\{1, 0, -1\}$  is the root, so we do an appropriate rotation at the root to produce the final AVL tree  $T''$ . If the height of  $T''$  is 10, what were the heights of  $T$  and  $T'$  respectively?

- (a) 10 and 11  
 (b) 11 and 10  
 (c) 11 and 11  
 (d) 10 and 10  
 (e) We do not have enough information to determine the answer.

~~10.~~ Which of the following sorting algorithms cannot be implemented as a stable sorting algorithm.

- (a) Quick sort  
(b) Merge sort  
(c) Radix sort  
(d) Heap sort

11. Consider an undirected connected graph  $G$  with edge lengths. Assume all edge lengths are positive integers. We would like to find the length of shortest path from  $s$  to every other vertex in  $G$ . We initialize an array  $D[\cdot]$  to  $D[s] = 0$ ,  $D[v] = \infty$  if  $v \neq s$ . Consider the following algorithm. Here  $l(e)$  denotes the length of edge  $e$ .

```
repeat
    find an edge  $e = (u, v)$  such that  $D[u] > D[v] + l(e)$ 
    update  $D[u]$  to  $D[v] + l(e)$ .
until no such edge can be found
```

Which of the following statements are true about this algorithm.

- (a) The algorithm may not terminate on some graphs.  
(b) The algorithm always terminates. But there exist graphs for which the final values  $D[v]$  are not equal to the length of shortest path from  $s$  to  $v$  for some vertices.  
(c) The algorithm always terminates. Further, on termination the values  $D[v]$  are equal to the length of the shortest  $s$  to  $v$  path for every vertex  $v$ .  
(d) The values  $D[v]$  are always at least the length of the shortest  $s$  to  $v$  path.

~~12.~~ Suppose we want to find shortest path from a source vertex  $s$  to a vertex  $t$  in an undirected graphs. However some edges in the graph may have negative lengths. In which of the following cases will the Dijkstra's algorithm find the desired shortest path?

- (a) No shortest path from  $s$  to  $t$  has an edge of negative length.  
(b) There is at most one edge of negative length

- (c) There does not exist a cycle in which the sum of the lengths of the edges in it is negative.  
 (d) None of the above.
13. Suppose that we have built up a compressed trie for a list of strings. The rightmost path of this trie is as shown in Figure 2, where the numbers to the left of each node specify the total length of the labels on the path from the root to that node. Let  $v$  be the lexicographically largest of the strings we have inserted. We now add a new string  $w$  which is lexicographically larger than  $v$ , and for which the length of the longest common prefix of  $v$  and  $w$  is 17. In the new compressed trie, how many edges are there on the rightmost path?

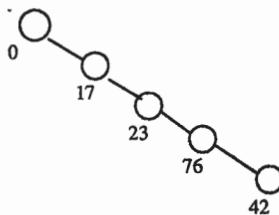


Figure 2:

- (a) 5  
 (b) 4  
 (c) 3  
 (d) 2  
 (e) 1
14. Consider again the 2-4 tree of height 2 given in Figure 2. Which of the following statements are true:
- (a) After 27 is deleted the final tree has 20 a leaf node.  
 (b) After 37 is deleted the final tree has 42 in the root node.  
 (c) 7 is the min number of keys in a 2-4 tree of height 2.  
 (d) At least 15 keys have to be deleted from the tree in Figure 2 to decrease its height.
15. Suppose we run the DFS and the BFS algorithms on a connected undirected graph  $G$ . It so happens that the DFS tree and the BFS tree obtained from these algorithms are identical. What can we claim about the graph ?
- (a) The graph does not have a cycle  
 (b) The graph can have one cycle  
 (c) The graph can have several cycles but no two of these cycles share a common vertex.  
 (d) No such conclusion can be drawn.
16. In a skip list each node is promoted to the next level with probability  $1 - \frac{1}{\log n}$ . The expected height of a node in such a skip list is
- (a)  $O(1)$

- ( $\log n$ )  
 $\theta(1)$   
(d)  $O(\log \log n)$

17. Consider a connected undirected graph  $G$ . Each edge  $e$  has two lengths, namely  $l(e)$  and  $w(e)$ . Let  $s$  be a starting vertex in  $G$ . Consider the following modified Dijkstra's algorithm. Here  $D[\cdot]$  and  $F[\cdot]$  are two arrays initialized to  $D[s] = F[s] = 0$  and  $D[v] = F[v] = \infty$  if  $v \neq s$ . Let  $n$  denote the number of vertices in  $G$ .

```

Initialize a set M to emptyset
While M has less than n elements do
    pick a vertex v which is not in M and for which D[v] is minimum
    add v to M
    for every neighbour u of v such that u is not in M do {
        if D[u] > D[v] + l((u,v)) {
            update D[u] = D[v] + l((u,v))
            F[u] = F[v] + w((u,v))
            update parent[u] = v
        }
        else if D[u] == D[v] + l((u,v)) AND F[u] < F[v] + w((u,v)) {
            update F[u] = F[v] + w((u,v))
            update parent[u] = v
        }
    }
}

```

What does this algorithm do ? For a path  $P$ , and a function  $f(e)$  on edges of the graph, define  $f(P)$  as the sum of  $f(e)$  for all edges  $e$  in  $P$ .

- (a) For every vertex  $v$ , it finds a path  $P$  from  $s$  to  $v$  for which  $f(P)$  is minimum, where  $f(e) = l(e) + w(e)$ .
  - (b) For every vertex  $v$ , it finds a path  $P$  from  $s$  to  $v$  for which  $f(P)$  is minimum, where  $f(e) = \min(l(e), w(e))$ .
  - (c) For every vertex  $v$ , it finds a path from  $s$  to  $v$  for which  $l(P)$  is minimum. Further if there are several shortest paths from  $s$  to  $v$  in terms of length  $l(e)$ , then it outputs the path among these for which  $w(P)$  is minimum.
  - (d) For every vertex  $v$ , it finds a path from  $s$  to  $v$  for which  $w(P)$  is minimum. Further if there are several shortest paths from  $s$  to  $v$  in terms of length  $w(e)$ , then it outputs the path among these for which  $l(P)$  is minimum.
18. Consider a linked list where each node has a `next` pointer. We say that the list has a loop if there are nodes  $a_1, a_2, \dots, a_k$  in the list such that the  $a_1.\text{next} = a_2, a_2.\text{next} = a_3, \dots, a_{k-1}.\text{next} = a_k$  and  $a_k.\text{next} = a_1$ . Consider the following code to find out if a list  $L$  has a loop or not. Here  $p$  and  $q$  are pointers. What is the worst case running time of this code ? Let  $n$  denote the number of nodes.

```

p = L
q = p.next
loop_exists = FALSE

```

```

while (p!=NULL && loop_exists = FALSE) {
    if (p == q)
        loop_exists = TRUE
    else
        { p = p.next
          q = (q.next).next
        }
}
return loop_exists

```

- (a)  $O(n^2)$
- (b)  $O(\log n)$
- (c)  $O(n \log n)$
- (d)  $O(n)$

19. Suppose an undirected graph  $G$  has 100 vertices and 28 edges. What are the maximum and the minimum number of possible connected components in the graph ?

- (a) 93 and 86
- (b) 93 and 72
- (c) 86 and 72
- (d) 93 and 86

20. Define the length of a cycle as the number of edges (or vertices) in the cycle. The girth of a graph is the smallest length of a cycle in the graph. Consider the following algorithm find the girth of a connected undirected graph.

Pick a vertex  $u$  in  $G$ .

Starting from  $u$ , run the BFS algorithm on  $G$ .

Let  $T$  be the BFS tree obtained

Define the level of a vertex  $v$  as its level in this tree (level of root is 0)

For every edge  $e=(v,w)$  which is not in  $T$

compute a quantity  $Q(e) = 1 + \text{level}(v) + \text{level}(w)$ .

Output the smallest  $Q(e)$ .

Which of the following are true about this algorithm ?

- (a) The algorithm outputs the girth of the graph.
- (b) The algorithm outputs the girth if the starting vertex  $u$  lies in a cycle which has length equal to the girth of the graph. Otherwise the graph may give a wrong result.
- (c) The algorithm will always give a wrong result if the starting vertex  $u$  does not lie on a cycle whose length is equal to the girth of the graph.
- (d) There are graphs for which the algorithm will give a wrong answer no matter how we pick the starting vertex  $u$ .

Name: \_\_\_\_\_

Entry No: \_\_\_\_\_

CSL201: Data Structures. II semester, 2007-08.

Major Exam

3:30 PM to 5:30 PM, 3rd April 2008

Question	1	2	3	4	5	6	Total
Marks							

Note. Explain your solutions in words. No pseudocode. Write concisely and clearly. If necessary solve the problems on the rough sheet first and then copy clearly onto this sheet.

Q1 We consider a *deterministic* version of skip lists in this question (i.e. without the use of probability) called *1-2-3 skip lists*. Before we describe this structure, recall that each node in a skip list has a *height* associated with it (denoted  $h(\cdot)$ ) which corresponds to the highest list it belongs to i.e. if a node is only in the base list it has height 0, if it was promoted to the next level but not further up it has height 1 and so forth. Now, we describe 1-2-3 skip lists as the skip list dictionary structure on a set  $S$  which maintains the following invariant:

Given 2 keys  $x \leq y \in S$ , if  $h = \min\{h(x), h(y)\} \geq 1$  there must be at least one and at most 3 keys  $z \in S$  such that  $x < z < y$  and  $h(z) = h - 1$ . Sentinel nodes (at the two extremes) have the height of *current.max + 1*

Q1.1. Starting from an empty 1-2-3 skip list, insert the following items (in the order given): 3, 1, 6, 2, 9, 8, 4, 5, 14, 10, 11, 7, 12, 13, 15. You do not have to show all steps, just show the final skip list. (6 marks)

Name: \_\_\_\_\_

Entry No: \_\_\_\_\_

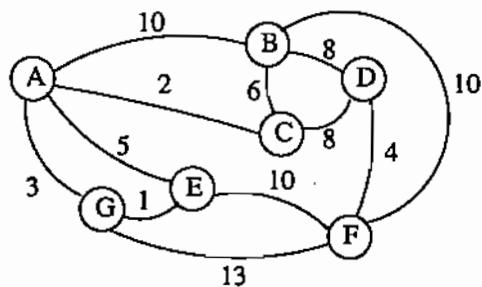
Q1.2. Give the algorithm for inserting an item into a 1-2-3 skip list as used in Q1.1. (5 marks)

Q1.3. Is the order of insertion important? Do you get the same 1-2-3 skip list no matter which order the keys are inserted in? If yes, prove your answer. If no, give a counter example. (5 marks)

Name: \_\_\_\_\_

Entry No: \_\_\_\_\_

Q1.4. Show that the maximum height of any 1-2-3 skip list is  $\theta(\log n)$ . (6 marks)

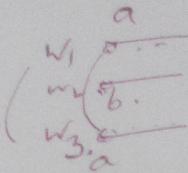


Q2. Run Dijkstra's algorithm on the network given in the figure starting from vertex  $s = A$ . Show the steps in running the algorithm, you do not need to draw the graph repeatedly, just write which is the next vertex picked and which labels get updated from what value to what value at each step. (10 marks)

Name: \_\_\_\_\_

Entry No: \_\_\_\_\_

Q6. Given a sequence of  $n$  integers, a pair of elements  $x$  and  $y$  are called an *inversion* in this sequence if  $x > y$  but  $x$  occurs before  $y$  in the sequence. Give an  $O(n \log n)$  algorithm for finding the number of inversions in a sequence. [Hint: Modify merge sort]. (12 marks)



Name \_\_\_\_\_ Entry Number \_\_\_\_\_ Gp No. \_\_\_\_\_

*Answer all the questions in the space provided for each question.*

1. For each of the problems below, describe how you will solve this problem and how much time it will take (in the size of the input).

a. [8 marks] Suppose you are given a dictionary in a foreign language (where the words appear in alphabetical order) and are asked to find the order of characters that appear in the language. For English, say our dictionary has "accord, apple, canary, corner, dice" in that order. Then your output must be "a, c, d, e, i, l, n, o, p, r, y" (for example, the fact that "accord" appears before "canary" means that "a" appears before "c", and so on).

*Build a directed graph G, where the vertices are the letters. For every pair of words,  $w_1, w_2$ , ( $w_1$  appears before  $w_2$ ), find the first position where they differ, put a directed edge between the corresponding letters.*

*Run topological sort.*

3 marks for identifying the graph.

3 marks for edges and topological sort

2 marks:

{ of the order }  $n$  words,  $m$  letters

$$n^2 L + m + n^2$$

$$= O(m + n^2 L), \quad L: \text{max. length of a word.}$$

- b. [8 marks] Snake and ladder problem: You are given an  $n$ -by- $n$  snake and ladder board. The squares in the board are labeled with numbers from 1 to  $n^2$  in a continuous manner. At some of the squares, you may find a ladder which goes to a square labeled with a higher number, whereas a snake brings

Modeling:  $\Theta^2$  pts  
(graphical vertices)

Edges: 3  $\Theta$  pts

BFS: 1 pt

Build a graph, vertices are the squares. For every square which is not a start point of ladder/snake, 6 edges to the next 6 squares. For a square which is a starting point, an edge to the end of the ladder/snake.

you down to a square with a smaller number. Assume that you have a dice which can show a number between 1 and 6. Whenever you throw the dice, you move ahead by these many steps - if you end-up at a square which has a ladder or a snake, you get transferred to the corresponding square. You are asked to find the minimum possible number of dice throws required to reach the last cell from the first cell.

Running time :

Run BFS on this graph.

2 marks      Running time :  $O(\# \text{ squares})^{\frac{1}{2}}$  : degree of every vertex is  $\leq 6$   
 $= O(n^2)$ .  
 $\Rightarrow \# \text{ edges} = O(\# \text{ vertices})$

(if they say Dijkstra, ok with the  
 correct running time of  $O(n^2 \log n)$ )

$$|a - b| \leq 1$$

2. [2 marks] Consider two vertices x and y that are simultaneously on the queue during execution of BFS from vertex s in an undirected graph. Which of the following is/are true?

- I. The number of edges on the shortest path from s to x is at most one more than the number of edges on the shortest path from s to y.
  - II. The number of edges on the shortest path from s to x is at least one less than the number of edges on the shortest path from s to y.
  - III. There is a path from x to y.
- (a) Only one of these statements is true. (b) Items I. and II. are true.  
 ✓ Items I. and III. are true. (d) Items II. and III. are true.  
 (e) All three statements are true.

X  
 0 for  
 all else

(C)  $\rightarrow$  2

3. [12 marks] You are given a directed acyclic graph (DAG), represented using an adjacency list, with non-negative edge weights and a fixed source vertex  $s$ . Give a  $O(|V| + |E|)$  time algorithm to find the shortest path distances from source vertex to all other vertices.

4 marks

Topological sort. can assume  
 $D[s] = 0, D[v] = \infty$  ( $s$  has indegree 0)

for  $i = 1$  to  $n$  do

    for each nbr  $j$  of  $i$

        update  $D[j] \leftarrow D[i] + d(i,j)$  if  
 $D[j] > D[i] + d(i,j)$ .

Running :  $O(|V| + |E|)$ .

---

Dijkstra: 4 if correctly written.

4. [12 marks] A bipartite graph is an undirected graph whose vertex set can be partitioned into two sets  $A$  and  $B$  such that each edge in the graph goes between a vertex in  $A$  and a vertex in  $B$  (no edges exist between vertices in the same set). Give an  $O(|V| + |E|)$  time algorithm that takes an input graph represented as an adjacency list, and decides if the graph is bipartite.

Run BFS.     All cross-edges should be between different levels

6 marks.           6 marks.

if the number starts from 0  
 $i$  children  $d(i+1)$   
 $i$  parents  $\left\lfloor \frac{i}{d} \right\rfloor - 1$

5. [10 marks] A d-ary heap is like a binary heap but instead of two children, nodes have  $d$  children.

4

(a) How would you represent a d-ary heap in an array?  
 Number the vertices starting 1 and level-wise  
 child of a vertex labeled  $i$  :  $d(i+1)$   
 parent of a vertex  $i$  :  $\left\lfloor \frac{i-1}{d} \right\rfloor + 1$

2

(b) What is the height of a d-ary heap of  $n$  elements in terms of  $n$  and  $d$ ?  
 $O(\log_d n)$

(c) How much time will deleteMin and insert operations take?

$\lg n$

$O(d \log_d n)$ ,  $O(\log_d n)$

(2+2)

6. [6 marks] There is a bag containing 8 white and 8 black balls. You repeat the following experiment till you see a white ball : draw two balls uniformly at random out of the bag. If both are white, stop. Otherwise, put both back in the bag. What is the expected number of times you will need to draw two balls from the bag ?

3

$$\text{pr. of } = \frac{8C_2}{16C_2} = \frac{8 \cdot 7}{16 \cdot 15} = \frac{7}{30}.$$

$$1 \text{ (Expected #times} = \frac{30}{7}$$

$$(17/28)$$

$$2 \left[ = \sum_i i \cdot \left(1 - \frac{7}{30}\right)^{i-1} \frac{7}{30} \right]$$

7. [2 marks] Suppose we have finished the first iteration of quick sort. What were the possible pivot elements, if the array looks like the following after the first

iteration:

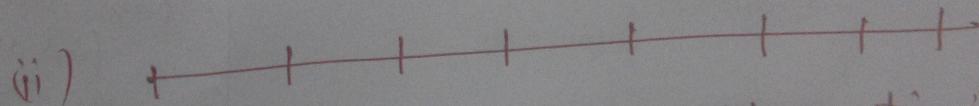
13, 10, 12, 14, 15, 18, 17, 16, 19, 24

14, 15, 18, 19, 24

( $\frac{1}{2}$  pt. for each correct)  
-  $\frac{1}{2}$  pt. for each incorrect)

8. [12 marks] You are given an array of  $n$  elements with the guarantee that each element is at most  $K$  positions from its target position. Devise an algorithm that sorts it in  $O(n \log K)$  time.

(i) Heapify  $K$  elements - 6  
 $6 \rightarrow$  repeat [each step: 1 deletion, insert the next element]

(ii)   
 $6 \rightarrow$  Sort each block in  $k \log K$  time:  $\frac{N}{K} \times k \log K = N \log K$

$6 \rightarrow$  Merge with the next block (starting from the first block)

9. [4 marks] Recall that in the Rabin Karp algorithm, the text subsequence  $t[i .. i+M-1]$  is hashed to the number  $x(i) = t[i] b^{M-1} + t[i+1]b^{M-2} + \dots + t[i+M-1]$ . Suppose we replace the hash function by

$$x(i) = t[i] R_{M-1} + t[i+1]R_{M-2} + \dots + t[i+M-1] \text{ for some fixed numbers } R_{M-1} \dots R_0.$$

Is this a good choice for the hash function? Justify your answer.

No. Cannot reuse computation of previous hash fn. for the next one.

3.

10. [4 marks] Consider the following program for finding a number  $x$  in an array  $A$  of size  $n$ . Assuming that the number  $x$  appears exactly twice in the array  $A$ , what is the probability that the program will output true ?

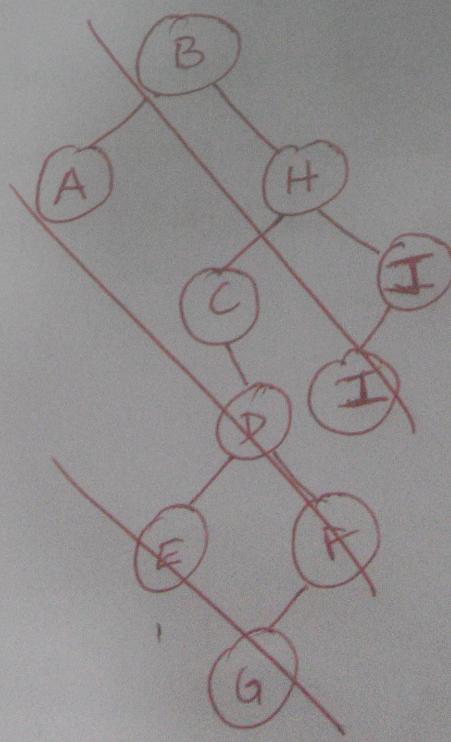
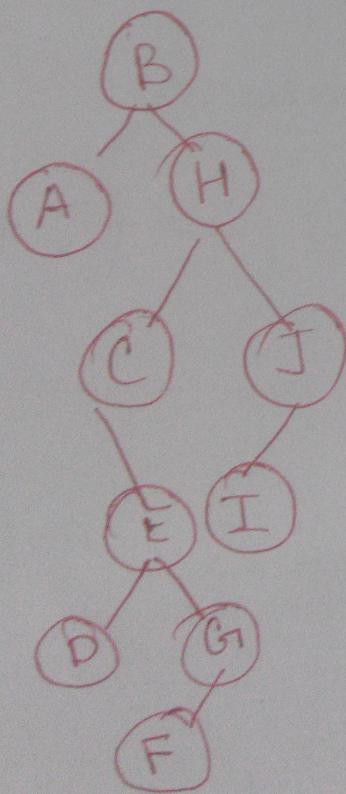
```
function find(A, x) {
    for (i=1; i <= n; i++) {
        pick a random index j between 0 and n-1.
        If (A[i]==x)
            output true;
    }
    output false;
}
```

prob. that we  
find in one  
iteration =  $\frac{2}{n}$

prob. that we never  
find it =  $(1 - \frac{2}{n})^n$

answ =  $1 - (1 - \frac{2}{n})^n$ .

11. [4 marks] Suppose you have a binary tree whose data fields are single characters. The inorder traversal of the tree gives ABCDEFGHIL and the preorder traversal gives BAHCEDGFIJL. Draw the binary tree showing the data in each node and the pointers between nodes.



pre  
in.

**12. [6 marks]** You are given the following points in 2-dimensional space :

(100,100), (2,1), (2,2), (3,5), (5,4), (-2, -3), (-5,5)

Draw the corresponding quad-tree and K-D tree to represent the above data.

K-D Tree — 3 marks  
Quad Tree — 3 marks.

COL106: Data Structures. I semester, 2021-22. Re-Major Exam.  
08 January 2022. Maximum Marks: 40 marks

**Problem 1 (8 marks)** Given an  $n \times n$  integer matrix  $A$  with at most  $k$  non-zero entries, store the matrix in an appropriate data structure such that the following queries can be answered in  $O(n + k)$  time:

`MatrixVectorMult( $x$ )` : takes as input an  $n$ -length integer vector  $x$ , and outputs  $A.x$ .

You must argue for the correctness of your data structure by presenting an analysis of how the time complexity requirement above is met.

**Problem 2 (8 marks)** Let  $T$  be a binary search tree with  $n$  nodes. Every node  $x$  in  $T$  has a key (denoted by  $x.key$ ), a left-child (denoted by  $x.left$ ) and a right-child (denoted by  $x.right$ ). The left/right children can be null. If the left (resp. right) child is non-null, then  $x.left.key \leq x.key$  (resp.  $x.right.key \geq x.key$ ).

Someone has swapped the left and right children of certain nodes, resulting in a binary tree  $T'$ . You are given the root of  $T'$ . Explain in words (**no pseudocode**) how to recover the original tree  $T$  given  $T'$ , using only  $o(n)$  additional space. No partial credit for using  $\Omega(n)$  additional space. You must include an argument for the correctness of your method and an analysis of the space used.

**Problem 3 (8 marks)** Given an array  $A$  with  $n$  integer entries, describe an  $o(n^3)$  algorithm (in words, **no pseudocode**) to find the number of triplets  $(a_i, a_j, a_k)$  such that  $i < j < k$  and  $a_i + a_j^2 = a_k^3$ . Analyse the running time of your algorithm and present arguments for its correctness.

**Problem 4 (8 marks)** Let  $N = 4^k - 1$ , and let  $A$  be an array of size  $N$ , where each element is an integer in the range  $[0, N^3]$ . You can assume all  $N$  integers are distinct. Give an  $O(N)$  time algorithm to construct a (2,4) tree with  $N$  keys, where each key is an element of  $A$ . You must argue for the correctness of your algorithm and present an analysis of the running time. You will get no credit for any algorithm whose running time is  $\omega(N)$ .

**Problem 5 (8 marks)** Given an weighted, undirected graph  $G = (V, E)$  in adjacency-list representation, a source vertex  $s \in V$  and a destination vertex  $t \in V$ , construct a data structure such that the following queries can be answered in  $O(1)$  time:

- `Query(edge  $e$ )` : the query takes as input an edge  $e \in E$ , and outputs 1 if there exists a **shortest path** from  $s$  to  $t$  that uses edge  $e$ , else outputs 0.

Explain in words (**no pseudocode**) (a) how to construct the above data structure, (b) the time required to construct the data structure, (c) the time required to answer queries.

# **COL 106 Autumn 2017 Minor 1**

Welcome to minor 1. The exam is for 1 hour 10 minutes. Please use a pen for answering questions. Do not use a pencil.

Before starting the exam, close your eyes and take three deep breaths. You will find that the exam is not an accurate reflection of your understanding of the material if you are not relaxed. If you are relaxed, you will likely perform better.

Question Number	Maximum Marks
1	05
2	06
3	05
4	11
5	04
6	03
7	02
8	04

1. [5 points] Analyze the following pseudo code:

```
void fun(int n, int m) {  
    if (m <= 0) return;  
    if (m > n) return;  
  
    print m;  
    fun(n, 2*m);  
    print m;  
}
```

(a) [2 points] What sequence will be printed if we call `fun(2000, 8)`?

8 16 32 64 128 ... 1028 1028 ... 128 64 32 16 8

... includes  
powers of 2  
in between.

(b) [3 points] What will be the time complexity (expressed in terms of  $m$  and  $n$ ) for this procedure? Explain your answer.

The procedure runs until second argument ~~greater~~ than first one.

If `fun` is called inside the first call to `fun`  $k$  times, then second argument becomes  $2^k * m$ .

∴ The recursion terminates when  $2^k * m > n \Rightarrow k > \log_2(m/n)$

$$\begin{aligned} T(n, m) &= C + T(n, 2m) \\ &= C + C + T(n, 2^2m) = \underbrace{C + C + \dots + C}_{k \text{ terms}} + T(n, 2^k m) \end{aligned}$$

∴ Time complexity is  $O(\log(m/n))$  assuming ~~→ m > n~~

2. [6 points] Read the following pseudo-code

```
int divide(int n, int s) {  
    q = 0;  
    r ← n;  
  
    while (r >= s) {  
        q ← q + 1;  
        r ← r - s;  
    }  
  
    return q;  
}
```

Define a loop invariant for the `while` loop (invariant should be true at the beginning of each loop). Prove the loop invariant using methods discussed in class. Using this loop invariant prove that the function `divide` (for positive  $n$  and  $s$ ) returns the quotient from the division of  $n$  by  $s$ .

Loop invariant: At the beginning of each iteration  $r + q \times s = n$ .

Initial check: Initially  $r=n$   $q=0$   
 $\therefore$  initially  $r + qs = n + 0 = n$   
 $\therefore$  initially invariant condition is true.

Maintenance: Let invariant be true at  $i^{\text{th}}$  iteration.

$$\therefore r_i + q_i \times s = n$$

$r_i$  is value of  $r$  at beginning of  $i^{\text{th}}$  iteration

At  $(i+1)^{\text{th}}$  iteration,

⑥

$$r_{i+1} = r_i - s$$

$$q_{i+1} = q_i + 1$$

$q_i$  is value of  $q$  at beginning of  $i^{\text{th}}$  iteration.

$$r_{i+1} + q_{i+1} \times s = r_i - s + (q_i + 1) \times s$$

$$= r_i + q_i \times s = n$$

by our assumption  
that invariant is  
true at  $i^{\text{th}}$  iteration.

Thus truth at  $i^{\text{th}}$  iteration  $\Rightarrow$  truth at  $(i+1)^{\text{th}}$  iteration.

Termination: Loop terminates when  $r < s$ .

By loop invariant,

$$n = qs + r$$

$$r \in [0, s)$$

According to Euclid's division lemma

$$a = qb + r$$

$c \in [0, s)$  then  $q$  is quotient  
of  $a/b$   
and  $r$  is remainder.

Thus, using this we know that  $q$  at termination is  
quotient for  $n/s$ .

Thus, the divide function returns quotient of  $n/s$ .  
(if it returns).

In case of  $s=0$ , the function never returns any value and  
goes in an endless loop.

3. [5 points] Consider functions  $f(n)$  and  $g(n)$  as given below. Use the "most precise" asymptotic notation to show how function  $f$  is related to function  $g$  in each case (i.e.,  $f \in ?(g)$ ). For example, if you were given the pair of functions  $f(n) = n$  and  $g(n) = 2n + 1$  then the correct answer would be:  $f \in \Theta(g)$ . To avoid any ambiguity between  $O(g)$  and  $o(g)$  notations due to writing, use  $\text{Big-}O(g)$  instead of  $O(g)$ .

$f(n)$	$g(n)$	Relation $f \in ?(g)$
$2^n$	$3^n$	$f \in o(g)$ ✓
$2^n$	$n^{1000}$	$f \in \omega(g)$ ✓
$0.5^n$	1	$f \in O(g)$ ✓
$n^3 + 2n + 1$	$\frac{1}{100}n^3 + n \log n$	$f \in \Theta(g)$ ✓
$\log_2 n$	$\log_3 n$	$f \in \Theta(g)$ ✓

5

4. [11 points] Recall that queue is a FIFO data structure with two main operations: *enqueue* and *dequeue*. Similarly, stack is a LIFO data structure with two main operations: *push* and *pop*, and additional operations like *isEmpty()*. Your goal is to implement a queue with two stacks so that the amortized time complexity of a sequence of *enqueue* and *dequeue* operations is constant (in the number of stack operation calls). Provide the pseudo-code for the *enqueue* and *dequeue* methods. Also, provide a proof for the amortized time complexity.

q ← The queue to be implemented  
 s1 ← stack 1 (we will be using it for enqueue operations)  
 s2 ← stack 2 (we will be using it for dequeue operations)

algorithm enqueue (object o)  
 s1.push(o);  
 q.size++; ✓

algorithm dequeue ()  
 if s2.isEmpty() then  
 while !(s1.isEmpty()) do  
 s2.push(s1.pop()); ✓

```

if s2.isEmpty() then
    throw EmptyDequeException
else
    q.size--
    return s2.pop()

```

// if still empty means both stacks are empty and there are no elements to remove.

The enqueue operation returns passed irrespective of the size of queue after exactly 2 step for any object.

$$\therefore \text{Amortized time for } n \text{ operations of push} = \frac{\overbrace{T(1) + T(1) \dots T(1)}^{n \text{ times}}}{n}$$

$$= O(1) \quad O.S$$

For deque operation, suppose our queue has  $n$  elements and we need to remove all.

Initially  $s2$  contains  $c$  elements ( $c \geq 0$ ) and  $s1$  contains  $(n-c)$  elements.

First dequeue call will return in  $O(1)$ . Similarly second, third upto  $c^{\text{th}}$  call to dequeue.  $\therefore$  Time to remove first  $c$  elements =  $cT(1)$

**1.5** Now, for  $(c+1)^{\text{th}}$  call, we need to transfer all  $(n-c)$  elements to  $s2$ . This will take  $(n-c)$  steps + 1 step to pop again from  $s2$ .

$$\therefore \text{Time} = 2T(n-c) + T(1)$$

+ After this all dequeue calls will be returned in  $O(1)$ .

**O.S to O.S** Total time for  $n$  dequeue operations =  $\underbrace{n}_{\text{"pops from } s2} + 2\underbrace{(n-c)}_{\text{transfer op.}}$

You need to clearly where for a sequence

$$\text{Amortised time} \leq \frac{3n}{n} = 3$$

for a queue gets multiple empty times Amortised time for dequeu =  $O(1)$

Since both enqueue and dequeue run in amortised time  $O(1)$ , their combination will run too in  $O(1)$  amortised time.

5. [4 points] You are given a Vector ADT with the following interface:

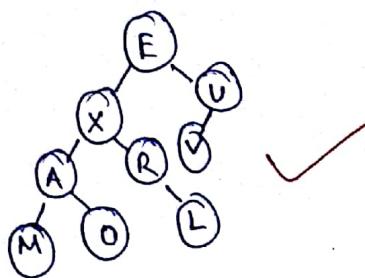
```
public interface Vector {  
    /** returns the number of elements in the vector */  
    public int size();  
  
    /** returns whether the vector is empty */  
    public Boolean isEmpty();  
  
    /** returns the element stored at the given rank (rank <= 0..size()-1) */  
    public Object elemAtRank(int r) throws OutOfBoundException;  
  
    /** replaces the element stored at the given rank (rank <= 0..size()-1) */  
    public Object replaceAtRank(int r, Object e) throws OutOfBoundException;  
  
    /** inserts an element at the given rank (rank <= 0..size()) */  
    public void insertAtRank(int r, Object e) throws OutOfBoundException;  
  
    /** removes the element stored at the given rank (rank <= 0..size()-1) */  
    public Object removeAtRank(int r) throws OutOfBoundException;  
}
```

Our goal is to define an adaptor for the Deque ADT using the Vector ADT. For each Deque method give the corresponding call to the Vector method that realizes the same functionality.

Deque Method	Realization with Vector Methods
size()	size()
isEmpty()	isEmpty()
insertFirst(e)	insertAtRank(0, e)
insertLast(e)	insertAtRank(this.size(), e)
removeFirst()	removeAtRank(0)
removeLast()	removeAtRank(this.size() - 1)
first()	elemAtRank(0)
last()	elemAtRank(this.size() - 1)

6. [3 points] Draw a single binary tree T such that each internal node of T stores a single character and

- a preorder traversal of T yields EXAMORLUV
- an inorder traversal of T yields MAOXRLEVU



3

7. [2 points] Consider a sorted circular doubly linked list of numbers where the head element points to the smallest element in the list.

- (a) What is the asymptotic complexity of determining whether an element  $e$  appears in the list?  $O(n)$  ✓ (2)
- (b) What is the asymptotic complexity of finding the median of the list of numbers?  $O(n)$  ✓ (2)
- (c) What is the asymptotic complexity of finding the smallest element in the list?  $O(1)$  ✓ (2)
- (d) What is the asymptotic complexity of finding the largest element in the list?  $O(1)$  ✓ (2)

8. [4 points] True/False

- (a) If  $f(n)$  is  $O(g(n))$  and  $g(n)$  is  $O(h(n))$  then  $h(n)$  is always  $\Omega(f(n))$ . True ✓ (4)
- (b) The minimum number of nodes in a binary tree of height  $d$  is  $d+1$ . True ✓ (4)
- (c) The minimum height of tree containing  $n$  nodes is  $\lceil \log_2 n \rceil$ . False ✓ (only true for binary trees) (4)
- (d) If class A implements interface I, class B extends A, class C extends B, and interface J extends interface I, then C always implements J. False ✓ (4)

---

**Started on** Sunday, 19 September 2021, 9:15 AM

**State** Finished

---

**Completed on** Sunday, 19 September 2021, 9:46 AM

**Time taken** 30 mins 44 secs

---

**Marks** 5.00/5.00

---

**Grade** **10.00** out of 10.00 (**100%**)

**Question 1**

Correct

Mark 1.00 out of  
1.00Given  $g(n)$  and  $h(n)$ , define  $f(n)$  as follows.

```
1: void f (int n){  
2:   c = 1;  
3:   j = 3;  
4:   for i = 1 to n {  
5:     print g(i);  
6:     if (i == c)  
7:       then {  
8:         print h(i);  
9:         c = c+j;  
10:        j=j+2;}}}
```

Calculate the time complexity of  $f(n)$  in big-theta notation. It is given that the time taken by  $g(i)$  is  $\sqrt{i}$  units and that by  $h(i)$  is  $i^2$  units. Assume that: assignments, **for** statements, **print** statements, mathematical operations and comparisons take 1 unit of time.

Select one or more:

- a.  $\Theta(n * \log(n))$
  - b.  $\Theta(n^{5/4})$
  - c.  $\Theta(\sqrt{n})$
  - d.  $\Theta(n^{3/2} + n^2)$
  - e.  $\Theta(\sqrt{n} * \log(n))$
  - f.  $\Theta(n^{5/2})$
- ✓
- g.  $\Theta(n + \log(n))$
  - h.  $\Theta(n^2)$
  - i.  $\Theta(n * \log(n) + \sqrt{n})$

- j.  $\Theta(n^{3/2})$
- k.  $\Theta(n)$
- l.  $\Theta(n^{5/2} + n^{3/2})$
- m.  $\Theta(n^{3/2} * \log(n))$



The correct answers are:  $\Theta(n^{5/2})$

,  $\Theta(n^{5/2} + n^{3/2})$

### Question 2

Correct

Mark 1.00 out of  
1.00

What will be the final state of a min-heap (implemented using an array) if the following 7 elements are inserted in order: 7, 0, 8, 5, 13, 17, 14. Assume that the heap was empty before inserting these elements.

Select one:

- a. [0, 8, 5, 17, 14, 7, 13]
- b. [0, 7, 5, 17, 8, 14, 13]
- c. [0, 8, 5, 13, 14, 17, 7]
- d. [0, 5, 8, 17, 7, 14, 13]
- e. [0, 5, 8, 7, 13, 17, 14] ✓
- f. [0, 8, 5, 14, 13, 7, 17]
- g. [0, 5, 13, 8, 7, 17, 14]
- h. [0, 7, 5, 17, 13, 14, 8]
- i. [0, 5, 7, 8, 14, 13, 17]
- j. [0, 7, 5, 17, 14, 8, 13]

The correct answer is: [0, 5, 8, 7, 13, 17, 14]

**Question 3**

Correct

Mark 1.00 out of  
1.00

Consider a Merkle tree with 8 integer documents. Assume that the hash function used to calculate the digest at each node N is  $f(a,b) = a*3 + b \text{ (mod. 17)}$  where a and b are the digests (or values for the leaves) of the left and right children of N respectively. Calculate the digest at the root node if the documents (leaves) in order are [15, 12, 6, 18, 7, 9, 13, 17].

Select one:

- a. 0
- b. 8
- c. 14
- d. 16
- e. 11
- f. 12
- g. 9
- h. 10
- i. 2 ✓
- j. 1
- k. 6
- l. 4
- m. 13
- n. 15
- o. 3
- p. 7
- q. 5

The correct answer is: 2

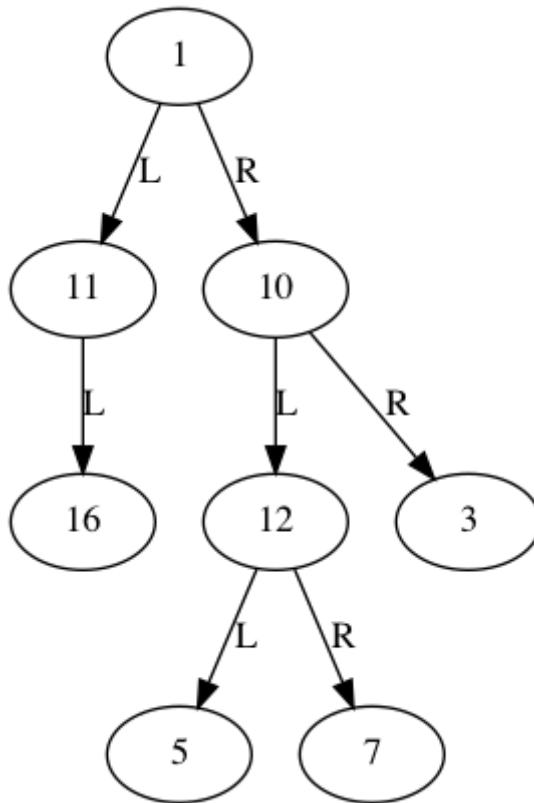
**Question 4**

Correct

Mark 1.00 out of

1.00

For the given binary tree, choose the options corresponding to the correct pre-order, in-order and post-order traversals. L and R denote the left and right child of a node respectively.



Select one or more:

- a. In-order traversal: 16, 11, 1, 5, 12, 7, 10, 3 ✓
- b. In-order traversal: 5, 10, 11, 3, 12, 7, 16, 1
- c. Post-order traversal: 5, 7, 10, 11, 12, 16, 3, 1
- d. Post-order traversal: 16, 12, 3, 11, 1, 7, 5, 10
- e. In-order traversal: 16, 12, 10, 5, 1, 3, 7, 11
- f. Pre-order traversal: 1, 11, 16, 10, 12, 5, 7, 3 ✓
- g. Pre-order traversal: 12, 5, 1, 16, 3, 10, 7, 11
- h. Pre-order traversal: 12, 5, 1, 16, 10, 7, 11, 3
- i. Pre-order traversal: 1, 11, 5, 3, 16, 10, 7, 12
- j. Post-order traversal: 1, 7, 10, 5, 12, 3, 16, 11
- k. In-order traversal: 16, 7, 5, 1, 3, 10, 11, 12
- l. Post-order traversal: 16, 11, 5, 7, 12, 3, 10, 1 ✓
- m. Post-order traversal: 16, 7, 1, 3, 11, 12, 5, 10
- n. In-order traversal: 16, 12, 3, 10, 11, 1, 5, 7
- o. Pre-order traversal: 12, 1, 16, 10, 3, 5, 11, 7

The correct answers are: Pre-order traversal: 1, 11, 16, 10, 12, 5, 7, 3, In-order traversal: 16, 11, 1, 5, 12, 7, 10, 3, Post-order traversal: 16, 11, 5, 7, 12, 3, 10, 1

**Question 5**

Correct

Mark 1.00 out of  
1.00

In this question we look at an open addressing scheme called cuckoo hashing. The set of keys we are trying to store is a subset of the natural numbers and we have a hash table T of size n. We are given two hash functions h1 and h2 which map all natural numbers to the set {0, 1, ..., n-1}. In order to place a newly inserted key x into the table we do the following:

1. Compute  $I_1 = h_1(x)$ .
2. If  $T[I_1]$  is free then store x in  $T[I_1]$  and exit, else
3. Compute  $I_2 = h_2(x)$ .
4. If  $T[I_2]$  is free then store x in  $T[I_2]$  and exit else if y is currently stored in  $T[I_2]$ . Remove y from  $T[I_2]$  and store x there.
5. Now we need to find an alternate position for y. If  $I_2$  was  $h_1(y)$  then compute  $I_3 = h_2(y)$  and go to step 4 else if  $I_2$  was  $h_2(y)$  then compute  $I_3 = h_1(y)$  and go to step 4. The fifth time we have to go to step 4 we assume a cycle has occurred and an exception is thrown declaring the insertion unsuccessful.

Now, consider a hash table with 5 elements already inserted:

[49, 69, 89, \_\_, \_\_, \_\_, \_\_, \_\_, \_\_, 58, 18]

What is the state of the table after you insert keys 55 and 56 if  $h_1(x) = (5x + 8) \bmod 11$ , and  $h_2(x) = (1x + 6) \bmod 11$ ?

In case you detect a cycle (i.e. you throw an exception and abort the insertion) please show the state of the hashtable at the 5th time the "go to" statement is executed.

Select one or more:

- a. State after inserting 55: [18, 55, 69, \_\_, \_\_, \_\_, \_\_, \_\_, 58, 49, 89]
- b. State after inserting 56: [18, 55, 69, \_\_, \_\_, \_\_, \_\_, 56, \_\_, 58, 49, 89]
- c. State after inserting 55: [49, 69, 89, \_\_, \_\_, \_\_, \_\_, 55, \_\_, 58, 18]
- d. State after inserting 56: [69, 89, 18, \_\_, \_\_, \_\_, \_\_, 55, 49, 58, 56]
- e. State after inserting 55: [49, 69, 89, \_\_, \_\_, \_\_, \_\_, \_\_, 55, 58, 18] ✓
- f. State after inserting 56: [49, 69, 89, \_\_, \_\_, \_\_, \_\_, 55, 56, 58, 18]
- g. State after inserting 56: [49, 69, 89, 56, \_\_, \_\_, \_\_, \_\_, 55, 58, 18]
- h. State after inserting 56: [49, 69, 89, \_\_, \_\_, \_\_, \_\_, 56, 55, 58, 18] ✓
- i. State after inserting 55: [49, 69, 89, 55, \_\_, \_\_, \_\_, \_\_, \_\_, 58, 18]
- j. State after inserting 55: [49, 69, 89, \_\_, \_\_, \_\_, \_\_, 55, \_\_, \_\_, 58, 18]

The correct answers are: State after inserting 55: [49, 69, 89, \_\_, \_\_, \_\_, \_\_, \_\_, 55, 58, 18], State after inserting 56: [49, 69, 89, \_\_, \_\_, \_\_, \_\_, 56, 55, 58, 18]

◀ Lab Submission 3 Friday 17.9.2021

Jump to...

Dummy Minor Part 1 ►

Name: \_\_\_\_\_

Entry number: \_\_\_\_\_

There are 4 questions for a total of 15 points.

## 1. Answer the following questions:

- (a) ( $\frac{1}{2}$  point) State true or false: In any undirected graph, the sum of degrees of odd-degree vertices is always an even. (*Degree of a vertex is the number of neighbouring vertices it has.*)

(a) \_\_\_\_\_ **True** \_\_\_\_\_

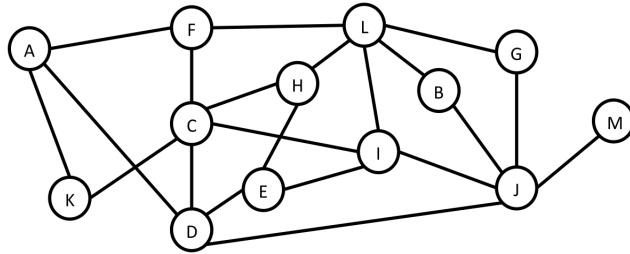
- (b) ( $\frac{1}{2}$  point) State true or false: In any strongly connected undirected graph with at least two vertices, there are always two vertices that have the same degree.

(b) \_\_\_\_\_ **True** \_\_\_\_\_

- (c) ( $\frac{1}{2}$  point) State true or false: For any directed graph  $G$  with  $n \geq 2$  vertices and any two vertices  $s, t$  in  $G$ , the number of distinct paths from  $s$  to  $t$  is  $O(2^n)$

(c) \_\_\_\_\_ **False** \_\_\_\_\_

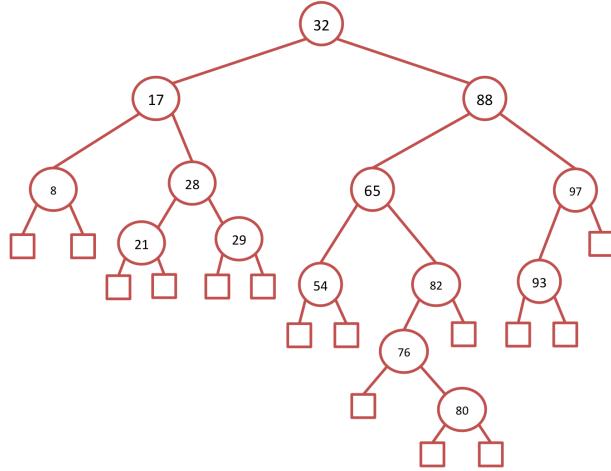
- (d) ( $\frac{1}{2}$  point) State true or false The graph given below is a bipartite graph.

(d) \_\_\_\_\_ **True** \_\_\_\_\_

- (e) ( $\frac{1}{2}$  point) State true or false Any strongly connected directed graph with at least two vertices contains a cycle.

(e) \_\_\_\_\_ **True** \_\_\_\_\_

(f) (½ point) Consider the binary search tree below. Draw the binary search tree obtained after performing `remove(88)`.



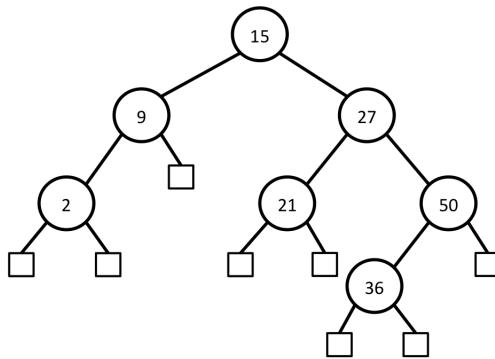
(g) (1 point) Consider the array  $A$  representing a min-heap below (only keys are shown) and answer the question that follow:

Array index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Key	3	10	5	13	17	6	11	15	16	21	18	9	8	23

Give the array after performing one `removeMin()` operation.

Array index	1	2	3	4	5	6	7	8	9	10	11	12	13
Key	5	10	6	13	17	8	11	15	16	21	18	9	23

(h) (1 point) Show the AVL tree that results after each of the integer keys 9, 27, 50, 15, 2, 21, and 36 are inserted, in that order, into an initially empty AVL tree. (*You may want to show steps for partial grading*)



2. (4 points) You are given an array  $A[1\dots n]$  representing a min-heap and an integer  $k$ . You have to design an algorithm to output all the keys in the array that are less than  $k$ .  
*(For example if  $k = 6$  and the array  $A$  is as shown below, then your algorithm should output the keys 3 and 5.)*

Array index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Key	3	10	5	13	17	6	11	15	16	21	18	9	8	23

Give pseudocode for your algorithm and discuss running time. Let  $m$  be the number of keys in  $A$  that are smaller than  $k$ . The running time of your algorithm should be  $O(m)$ .

**Solution:** Here is the pseudocode for the algorithm. The function call should be `SmallerThan(A, n, k, 1)`.

```

SmallerThan(A, n, k, i)
  - if(A[i] ≥ k OR i > n) return
  - Print(A[i])
  - SmallerThan(A, n, k, 2i)
  - SmallerThan(A, n, k, 2i + 1)

```

Running time: Let the number of elements smaller than  $k$  in the array  $A$  be  $m$ . The above algorithm is a recursive algorithm. There are constant number of operations performed in every recursive call before further recursive calls are made. So, the total running time of the algorithm is proportional to the total number of recursive calls that are made during the execution of the algorithm. For each element that are smaller than  $k$ , at most two recursive calls will be made. Note that no recursive calls are made with respect to elements that are at least  $k$ . So, the total number of recursive calls will be bounded by  $2m$ .

3. (3 points) Consider the following basic java implementation of Binary Search Tree that we have used in the lecture. Complete the body of the method `PrintKeys` below that is supposed to print all the keys in the tree in a sorted order (increasing order). Your algorithm should run in  $O(n)$  time where  $n$  is the number of keys in the binary search tree. (*The methods for insert, search, delete are not shown below*)

```
class Node{
    public int key;// This denotes the key stored at the node
    public String value;// This denotes the value stored in the node.
    public Node leftChild;// This is a reference to the left child
    public Node rightChild;// This is a reference to the right child
    public Node parent;// This is a reference to the parent of the node
    public Node(){parent = leftChild = rightChild = null;}
}

public class BinarySearchTree{
    public Node root;
    public int size;
    public BinarySearchTree(){size=0;root=null;}

// This method checks if the given node is a leaf
public boolean isLeaf(Node N){
    if(N.leftChild == null && N.rightChild == null)return(true);
    else return(false);
}
... other methods

// This method should print the keys in sorted order
public void PrintKeys(){
    InOrderTraverse(root);
}

public void InOrderTraverse(Node N){
    if(isLeaf(N))return;
    InOrderTraverse(N.leftChild);
    System.out.println(N.key);
    InOrderTraverse(N.rightChild);
}
```

4. (3 points) You are given an unsorted array  $A = A[1..n]$  containing  $n$  distinct integers. Design an algorithm that outputs the smallest  $k$  elements in the array  $A$ . The running time of your algorithm should be  $O(n + k \log n)$ . Give pseudocode and discuss running time.

**Solution:** Here is the pseudocode for the algorithm.

```

SmallElements( $A, k, n$ )
  - BottomUpHeapConstruction( $A, n$ )
  - For  $i = 1$  to  $k$ 
    - Print( $A[1]$ )
    -  $A[1] \leftarrow A[n]$ 
    -  $n \leftarrow n - 1$ 
    - DownHeapBubble( $A, n, 1$ )

DownHeapBubble( $A, n, i$ )
  -  $c \leftarrow i$ 
  - if( $2i > n$  and  $2i + 1 > n$ )return;
  - elseif( $2i \leq n$  and  $2i + 1 > n$ ){if( $A[c] > A[2i]$ )  $c \leftarrow 2i$ }
  - else{
    if( $A[c] > A[2i]$ ) $c \leftarrow 2i$ 
    if( $A[c] > A[2i + 1]$ ) $c \leftarrow 2i + 1$ 
  }
  - if( $c = i$ )return;
  - else{swap( $A, i, c$ ); DownHeapBubble( $A, n, c$ )}

BottomUpHeapConstruction( $A, n$ )
  -  $h \leftarrow \lfloor \log n \rfloor$ 
  - For  $i = h - 1$  to 0
    - For  $j = 2^i$  to  $2^{i+1} - 1$ 
      - DownHeapBubble( $A, n, j$ )

```

We use the array based bottom-up heap construction discussed in the class to convert array  $A$  to one representing a min-heap. After this we perform the remove minimum operation on this heap to obtain the  $k$  smallest elements.

Running time: Constructing the heap will take time  $O(n)$  and performing  $k$  remove minimum operations costs  $O(k \log n)$  time. So, the total running time is  $O(n + k \log n)$ .

Name: \_\_\_\_\_

Entry number: \_\_\_\_\_

There are 4 questions for a total of 15 points.

1. Answer the following:

(a) (1/2 point) State true or false: Let  $f(n) = 5n2^n + 3^n$  and  $g(n) = n3^n$ . Then  $f(n) = O(g(n))$ .(a) \_\_\_\_\_ **True** \_\_\_\_\_(b) (1/2 point) State true or false: Let  $f(n) = 5n2^n + 3^n$  and  $g(n) = n3^n$ . Then  $g(n) = O(f(n))$ .(b) \_\_\_\_\_ **False** \_\_\_\_\_

(c) (1/2 point) Express the running time of the algorithm below in big-O notation.

```

Alg2(A, n)
- for i = 1 to n
  - for j = 2i to n
    - A[i] ← A[j] + 1

```

(c) \_\_\_\_\_  **$O(n^2)$**  \_\_\_\_\_(d) (1 1/2 points) Solve the following recurrence relation and write the exact value of  $T(n)$ . Show calculations.

$$T(n) = 2 \cdot T(n/2) + n^2; \quad T(1) = 1 \quad (\text{assume } n \text{ is a power of 2})$$

(d) \_\_\_\_\_  **$T(n) = 2n^2 - n$**  \_\_\_\_\_**Solution:**

$$\begin{aligned}
T(n) &= 2 \cdot T(n/2) + n^2 \\
&= 2 \cdot (2 \cdot T(n/2^2) + (n/2)^2) + n^2 \\
&= 2^2 \cdot T(n/2^2) + n^2 \cdot (1 + 1/2) \\
&\vdots \\
&= 2^i \cdot T(n/2^i) + n^2 \cdot (1 + 1/2 + \dots + 1/2^{i-1}) \\
&\vdots \\
&= 2^{\log n} \cdot T(n/2^{\log n}) + n^2 \cdot (1 + 1/2 + \dots + 1/2^{\log n-1}) \\
&= n \cdot T(1) + n^2 \cdot \frac{1 - 1/n}{1 - 1/2} \\
&= n + 2n(n-1) \\
&= 2n^2 - n
\end{aligned}$$

2. (4 points) Solve the following recurrence relation and write the exact value of  $T(n)$ .

$$T(n) = \begin{cases} T(n-1) & \text{if } n > 1 \text{ and } n \text{ is odd} \\ 2 \cdot T(n/2) & \text{if } n > 1 \text{ and } n \text{ is even} \\ 1 & \text{if } n = 1 \end{cases}$$

2. \_\_\_\_\_  $T(n) = 2^{\lfloor \log n \rfloor}$

(*Hint: Unrolling may be tricky. You may want to guess the bound and then use induction to confirm your guess.*)

**Solution:** The answer follows from the following claim.

Claim: For all  $k \geq 0$ , the following holds: For all  $2^k \leq n < 2^{k+1}$ ,  $T(n) = 2^k$ .

*Proof.* We show this by induction on  $k$ . Let  $P(k)$  denote the given proposition in the claim. We need to show that  $\forall k, P(k)$  is true.

Base step: Base case is trivially true since  $T(1) = 1$ .

Inductive step: Suppose  $P(1), P(2), \dots, P(i)$  are true. We will show that  $P(i+1)$  is true. Consider any  $2^{i+1} \leq n < 2^{i+2}$ . We need to consider the case when  $n$  is even and  $n$  is odd.

If  $n$  is odd, then  $T(n) = T(n-1) = 2 \cdot T(\frac{n-1}{2})$ . Note that  $2^{i+1} \leq n-1 < 2^{i+2}$ . So, we have  $2^i \leq (n-1)/2 < 2^{i+1}$ . Applying induction hypothesis, we get  $T(n) = 2^{i+1}$ .

If  $n$  is even, then  $T(n) = 2 \cdot T(n/2)$ . Since  $2^{i+1} \leq n < 2^{i+2}$ , we have  $2^i \leq n/2 < 2^{i+1}$ . Applying induction hypothesis, we get that  $T(n) = 2^{i+1}$ .  $\square$

3. (4 points) In the lecture, we saw an implementation of Dynamic Arrays using regular arrays involving doubling the size of the array on overflows. What is the amortized running time for insert operations for the implementation where we increase the size of the array by 42 on overflows. That is, if the size of the array before overflow was  $m$ , then the size of the array after overflow is  $m + 42$ . Also, you may assume that we start with an empty array of size 1.

**Solution:** Starting from an empty array of size 1, consider  $n$  insertions. We can break down the running time into time for copying when an overflow occurs and time for inserting into the array. Note that the total time for insertion is going to be  $n$  since  $n$  elements are being inserted. Let us now focus on the time for copies.

Every time the current array overflows, we need to copy. The time for this copying is equal to the size of the array when the overflow occurs. The first time the array overflows, size of the array is 1. The second time the array overflows, size of the array is  $(1 + 42)$ . The third time the array overflows, size of the array is  $(1 + 42 + 42)$ . So, the  $i^{th}$  time the array overflows, the size of the array is  $(1 + 42 \cdot (i - 1))$ . Suppose  $k$  is the number of times the array overflows on  $n$  insertions. Then we have

$$1 + 42(k - 1) \leq n < 1 + 42k$$

This gives  $k = \lfloor \frac{n-1}{42} \rfloor$ . So, the total cost of copying is

$$\sum_{i=1}^k 1 + 42(i - 1) = k + 42 \cdot \frac{k(k - 1)}{2} \leq \frac{n - 1}{42} + \frac{(n - 1) \cdot (n - 43)}{2 \cdot 42} = O(n^2).$$

So, the asymptotic running time for insert operation is  $O(n)$ .

4. (4 points) Consider the following basic java implementation of linked list that we have used in the lecture. Complete the body of the method *SortList* below that is supposed to sort the integers in the linked list in non-decreasing order. It will be sufficient to give an  $O(n^2)$ -time algorithm for this problem. You are NOT allowed to use arrays within this method.

```
class Node{
    public int value;
    public Node next;
}

class LinkedList{
    public Node head;
    public int size;
    public LinkedList(){size=0; head = null;}
    //other methods
    :
    //Implementation of Bubble Sort
    public void SortList(){
        Node N;
        for(int i=0;i<size-1;++i){
            N = head;
            for(int j=0;j<size - i - 1;++j){
                if(N.value > N.next.value){
                    int x = N.value;
                    N.value = N.next.value;
                    N.next.value = x;
                }
                N = N.next;
            }
        }
    }
}
```

Name \_\_\_\_\_ Entry Number \_\_\_\_\_ Gp No. \_\_\_\_\_

Your Lab day \_\_\_\_\_

*Answer all the questions in the space provided for each question.*

**Q1.[15 marks]** Answer the following questions with proper justification (no marks will be awarded if the reasoning is incorrect.)

a. What is the running time complexity of the following code ? You can use order notation. Justify your answer.

```
sum = 0;
for (k=1; k<n; k++)
    for (i=1; i<k*k; i++)
        if (i % k == 0)
            for (j=0; j<i; j++)
                sum++;

```

$$\begin{aligned}
 & \sum_{k=1}^n [0 + k + 2k + \dots + k \cdot k] \\
 &= \sum_{k=1}^n k [1 + 2 + \dots + k] \\
 &= \sum_{k=1}^n k^2 \frac{(k+1)}{2} = O(n^4).
 \end{aligned}$$

b. Is it true that  $\log 1 + \log 2 + \dots + \log n$  is  $\Theta(n \log n)$ ? Explain your answer.

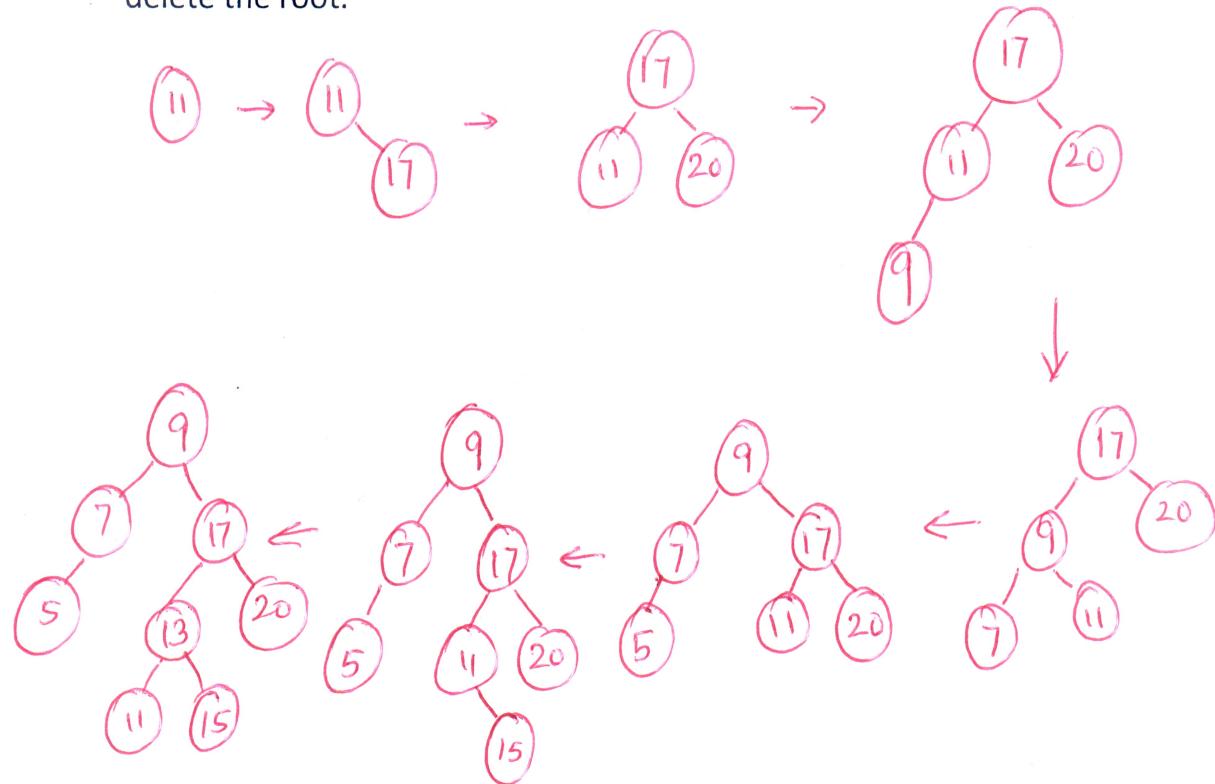
Yes. Two parts :

$$\textcircled{1} \quad \log 1 + \dots + \log n \leq \log n + \dots + \log n = n \log n \Rightarrow \text{LHS is } O(n \log n).$$

$$\begin{aligned}
 \textcircled{2} \quad \log 1 + \dots + \log n &\geq \log \frac{n}{2} + \dots + \log \frac{n}{2} \geq \log \frac{n}{2} + \dots + \log \frac{n}{2} = \frac{n}{2} \log \frac{n}{2} \\
 &= \Omega(n \log n).
 \end{aligned}$$

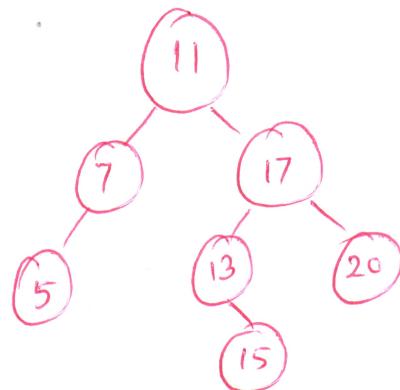
∴ LHS is both  $O(n \log n)$  and  $\Omega(n \log n) \Rightarrow \Theta(n \log n)$ .

**Q2.[10 marks]** Starting from an empty AVL tree, the following elements are inserted (in this order) in the AVL tree: 11, 17, 20, 9, 7, 5, 15, 13. You should use the rotation/balancing procedures described in the class. Show the AVL tree after each insert operation. After inserting the above numbers in the AVL tree, delete the root.



Delete 9

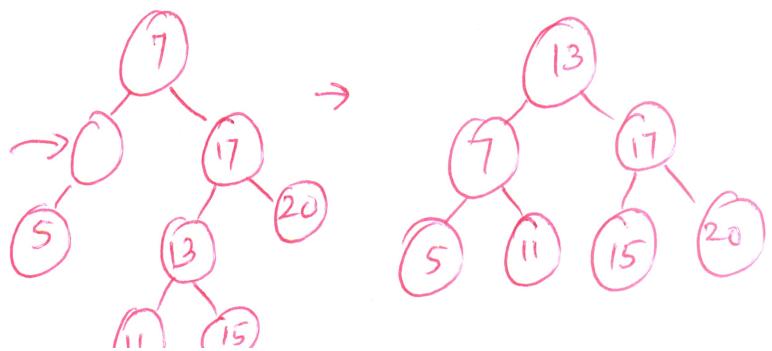
i) Delete successor node of 9:



OR

ii) Delete predecessor node:

Delete this



**Q3.[15 marks]** Given a node A in a binary search tree, write an efficient procedure to find its inorder predecessor. Fill in the details of the procedure below. Do not declare any other function.

Node\* Predecessor ( Node\* A)

```
{
    if (A->right is not a leaf) {
        p = A->right;
        while (p->left is not a leaf)
            p = p->left;
        return p;
    }
    else {
        p = A;
        q = A->parent;
        while (p is the right child of q) {
            p = q;
            q = p->parent;
        }
        return q;
    }
}
```

7 marks

8 marks

**Q4.[20 marks]** Given a stack, and a sequence of distinct numbers in an array A = [a<sub>1</sub>, .., a<sub>n</sub>] you can perform the following operations starting from the first number in A: either push the next number in the sequence on the stack, or pop from the stack.

Given a permutation B =[b<sub>1</sub>, .., b<sub>n</sub>] of A=[a<sub>1</sub>, ..., a<sub>n</sub>], we would like to know if it can be obtained from a<sub>1</sub>...a<sub>n</sub> by such operations.

For example, let A =[1 2 3 4 5] and B = [2 3 5 4 1]. Then B can be obtained from A by the following operations:

Push 1. Push 2. Pop 2. Push and Pop 3. Push 4, push 5. Pop 5. Pop 4. Pop 1.

(i) Let A = [1 2 3 4 5], B = [ 4 3 1 5 2]. Can B be obtained from A ? Why?

No. First we need to push 1,2,3,4 and then pop, pop, Now, 1 cannot } 3.  
be printed before 2.

(ii) Write a **linear time** procedure which given A and B of length n each, outputs whether B can be obtained from A using the method above. If yes, the procedure should output the sequence of operations. Note that you can access the stack using **push**, **pop**, **is\_empty** and **top** functions only. You should not modify the arrays A and B, and should use only a constant amount of extra space (besides using the stack and reading from the two arrays A and B).

$i = 0$ ; ( $i$  scans A)

$j = 0$ ; ( $j$  scans B)

while ( $j < n$ ) {

    if (stack S is not empty and  $S.\text{top}() == B[j]$ ) {

        7 marks

$S.\text{pop}();$   
         $j++;$

    }

    else {

        7 marks.

$S.\text{push}(A[i])$

$i++;$  ← if  $i$  exceeds  $n$ , "ERROR"

}

3 marks

}

If quadratic time algorithm, max. 5 marks.

Worse than quadratic time: NO MARKS.

COL106: Data Structures. I semester, 2021-22. Major Exam.  
 18 November 2021. Maximum Marks: 40 marks

Name	I	Ent. No.
------	---	----------

**Exam Instructions for offline students.**

1. There are 3 sheets of paper. Write your name and entry number in the given space on each of the two sheets.
2. Answer clearly *only in the space provided*. Do NOT write on this page. **No extra sheet will be given or collected.**
3. Rough sheets will **not** be collected.
4. If you need a new printed sheet, you can get one but you will have to transfer your entire exam onto the new sheet. We will collect only three sheets of paper from you.

**Problem 1 (8 marks)** Explain in words (**no pseudocode**) how to delete *any* element from the heap (not just the min). You can assume that (a) all elements in the heap are distinct, (b) the element to be deleted is actually present in the heap and (c) that you are given a pointer to the location of the element that you have to delete. You *must* argue for the correctness of your algorithm and tell us, with justification, its running time.

**Problem 2 (4+4 marks)** In an AVL tree for all nodes the difference between the heights of the left subtree and right subtree of  $x$  is at most 1. Suppose we relax this and allow the difference between the heights of the subtrees to be at most 2. We will call such BSTs *relaxed AVL trees*.

**Problem 2.1** Let  $\chi(h)$  denote the minimum number of nodes in a relaxed AVL tree of height  $h$ . Give a recurrence relation for computing  $\chi(h)$ , and compute  $\chi(6)$  (include all intermediate calculations). Note: You do *not* have to solve the recurrence. You just have to write the recurrence *with justification* and then use the recurrence to compute  $\chi(6)$ .

**Problem 2.2** Let  $\tau(h)$  denote the minimum depth of a leaf node in a relaxed AVL tree of height  $h$ . Give a recurrence relation for computing  $\tau(h)$ . Note that you have to justify why this recurrence is correct. Solve the recurrence to find the exact value of  $\tau(h)$  for general  $h$ .

**Problem 3 (5+3 marks)** In class, we saw that any comparison-based sorting algorithm must perform  $\Omega(n \log n)$  comparisons. In this problem, we will consider the problem of *almost-sorting* an array. Given an array  $A = [a_1, a_2, \dots, a_n]$ , we say that  $A$  is almost-sorted if for all  $i < n - \sqrt{n}$ ,  $a_i \leq a_j$  for all  $j > i$ . That is, the first  $n - \sqrt{n}$  elements are in sorted order, and the last  $\sqrt{n}$  elements are at least as large as the first  $n - \sqrt{n}$  elements. Example:  $[3, 5, 6, 7, 9, 11, 12, 14, 15, 17, 18, 20, 41, 33, 25, 38]$  is an almost-sorted array.

**Problem 3.1** Prove that there does not exist a comparison based algorithm that takes as input an array  $A$ , performs  $O(n)$  comparisons, and outputs an almost-sorted array. Note that you are expected to write a formal proof.

**Problem 3.2** Suppose the input array consists of **positive integers**, and you are given that at least  $n - \sqrt{n}$  entries are in the range  $[1, 100]$ . Give an  $O(n)$  time algorithm that outputs an almost-sorted array. You must argue for the correctness of your algorithm and its running time.

**Problem 4 (8 marks)** Let  $T$  be a 2-4 tree with  $n$  keys and height  $h = O(\log n)$ . Explain in words (**no pseudocode**) how to find the  $k^{\text{th}}$  largest key in  $T$  in  $O(h)$  operations. You can assume that all keys are distinct. You may need to store additional attributes at every node in order to solve this problem. Explain clearly how you will augment the 2-4 tree and then explain your algorithm.

**Problem 5 (8 marks)** Given an unweighted, undirected graph  $G = (V, E)$  in adjacency-list representation, construct a data structure  $\mathcal{D}_G$  of size  $O(n)$  such that the following queries can be answered in  $O(1)$  time:

- $\text{Query}(i, j)$  : outputs 1 if there exists a path from vertex  $i$  to vertex  $j$  in  $G$ , else outputs 0

You can assume that each vertex has a unique label in  $\{1, 2, \dots, n\}$ . Explain in words (**no pseudocode**) (a) how to construct the above data structure, (b) the time required to construct the data structure  $\mathcal{D}_G$ , (c) the time required to answer queries.

# **COL106: Minor Examination**

11/11/2020

Maximum Marks: 30

**Instructions:** Please read carefully before you start your examination

- Make sure that you keep your videos on for the duration of the examination
- Answer each question and its subpart on a separate piece of paper
- Write your name, entry number and page number on each page
- At the end of the examination, you have 10 minutes to send images of your answers to your TAs using WhatsApp/Telegram
- Once you have sent the pictures of your answers to your TAs, you need to convert them into a single PDF file and upload it on Gradescope by 3:00pm.
- Answers not uploaded on gradescope will not be evaluated
- In case you are unable to upload your answers on Gradescope by 3:00pm due to Internet or power issues, but you have sent the images of your answers to your TA by the deadline, please send your PDF to your TA as soon as possible.

### Question 1 [10 Marks]

- (a) Consider the two functions  $f: N \rightarrow N$  and  $g: N \rightarrow N$  defined as follows:

$$\begin{aligned} f(n) &= n && \text{when } x \text{ is not a prime number} \\ && n^3 && \text{when } x \text{ is a prime number; and} \\ g(n) &= n^2 && \text{for all } x \end{aligned}$$

Formally prove that neither  $f(n)$  is in  $O(g(n))$ , nor  $g(n)$  is in  $O(f(n))$ .

[7]

Let us call such a pair of functions as 'mutually non-dominating'.

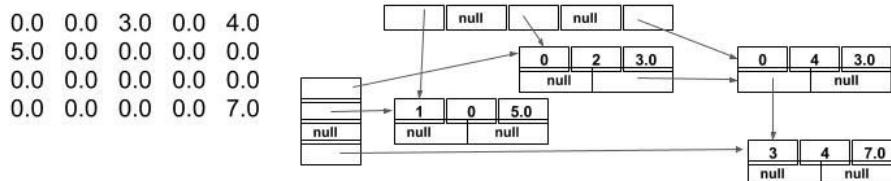
- (b) Consider the functions  $u: R^+ \rightarrow R$  and  $v: R^+ \rightarrow R$  defined as follows:

$$\begin{aligned} u(n) &= 1 + \sin(n); \text{ and} \\ v(n) &= 1 \end{aligned}$$

Do  $u$  and  $v$  also form a mutually non-dominating pair of functions? Why or why not? [3]

### Question 2 [10 Marks]

A matrix is a two-dimensional data object made of  $M$  rows and  $N$  columns, therefore having total  $M \times N$  values. If most of the elements of the matrix have **0 value**, it is called a sparse matrix. E.g. out of 20 possible values, only 4 are non-zero below. It can be represented using linked lists, with each node containing {int row, int column, float val, node nextincol, node nextinrow}. SparseMatrixClass will contain two arrays of header pointers, one for rows and one for columns.



Write pseudocode to multiply two sparse matrices  $A$  and  $B$ , using the above representation. Note that the two matrices can have 0 values at different indices, though number of columns in  $A$  can be assumed to be the same as the number of rows in  $B$ . [10]

### Question 3 [10 Marks]

A balanced binary min-heap is a binary tree with the following two properties:

1. The root of any sub-tree has a key value smaller than key values all the other nodes of the subtree
2. For any node in the tree, the number of nodes in its left and right subtree differ by at most 1.

- (a) Design a suitable data structure for implementing balanced binary min-heap [2]  
 (b) Write pseudo code to insert a new element in the balanced binary min-heap [4]  
 (c) Write pseudo code to delete the minimum element from a balanced binary min-heap [4]

Ensure that the above two properties of the balanced binary min-heap remain satisfied after the insert and delete-min operations. Also ensure that your code is efficient. Provide suitable proofs of correctness of your algorithm.

Please write your entry number and name on each sheet.

Entry Number:

Name:

A

---

**COL 106 MINOR EXAM II**  
**SEMESTER I 2019-2020**  
**1 hour**

---

**Please do not allow any bag, phone or other electronic device near you. Keep your ID card next to you on the desk. Maximum marks available for questions are listed in []. Write answers in the provided space. Justify all answers.**

1. (11)

a) [8] Given the Hash function  $h$  below, list the table slots touched/probed and show the status of the Hash table after each listed operation (in order from left to right, starting with an empty hash table). The table has 5 slots. Assume open addressing with  $h_i$  as given: (Deletion is by marking as deleted.)

$$h = h_0 = (\sum \text{ digits }) \% 5$$

$$h_i = (h_0 + 3*i) \% 5$$

Insert:873	Insert:9734	Insert:280	Delete:9734	Search 143	Insert:14	Rough space																									
<table border="1"><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table>						<table border="1"><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table>						<table border="1"><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table>						<table border="1"><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table>							<table border="1"><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table>						

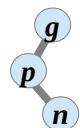
Slots probed:

|            |            |            |            |            |

b) [3] What is the termination condition for a search in such a Hash table?

2. [5] Write code (approximate syntax is OK) to restructure to balance the following configuration. Assume references left, right, parent and value are stored for each node. Assume  $p.parent.left == p$  and  $n.parent.right == n$ . (No references should be assumed to be null.) **Restructure(Node n):**

Node p = n.parent, g = p.parent;



3. (10) Consider a binary search tree that allows keys to be repeated in multiple nodes such that *keys equal to any node's key are always in that node's left subtree*. Write the pseudo-code for search and delete for this tree that each take time  $O(h)$  for a tree with height  $h$ . The search function must return all instances and the delete function must delete all instances of the given key.

[6] `Delete(root, key):`

[4] `Search(root, key):`

4. [5] You need to implement a *Red-Black tree* that supports multiple threads inserting, searching or deleting in parallel. (You may assume that each thread uses the standard algorithms to perform these operations.) What synchronization is required to ensure that the threads do not interfere with each other's operation?

5. [5] Consider a *skip-list* that supports multiple threads inserting, searching or deleting in parallel. (Assume each thread uses the standard algorithms to perform these operations.) What synchronization is required to ensure that the threads do not interfere with each other's operation?

**Please write your entry number and name on each sheet.**

**Entry Number:**

**Name:**

6. [4+3+3] Show that the height of a 2-4 tree with  $n$  keys is (a) no less than  $\text{floor}(0.5 \log_2 n)$  and (b) no more than  $\log_2 n$ .  
(c) Also show that the number of comparisons required to find a key is no more than  $2 \log_2 n$ . (5 extra marks to show the bound to be  $1.3 \log_2 n$ )

7. [6] You are given two heaps with  $2^h$  keys each, where  $h$  is an integer. Provide an  $O(h)$  algorithm to merge these two heap. (Assume all keys in the two heaps are comparable to each other.)

8. [6] In an AVL tree with 20 nodes having numbers 1 to 20, respectively, as keys, which numbers may not appear in the root?

**Please write your entry number and name on each sheet.**

**Entry Number:**

**Name:**

9. [6] Given the keys of a Red-black tree in the pre-order traversal order, provide an algorithm to re-construct the Red-black tree. (You do not have to determine the nodes' colors.)

10. [6] Provide *non-recursive* pseudo-code to compute the depth of a binary tree.



# **COL 106 Autumn 2017 Minor 2**

Welcome to minor 2. The exam is for 1 hour and 10 minutes. Please answer all questions. Do not use a pencil.

Before starting the exam, close your eyes and take three deep breaths. The exam is not an accurate reflection of your understanding of the material if you are relaxed, you will likely perform better.

<b>Question Number</b>	<b>Maximum Marks</b>
1	14
2	06
3	07
4	05
5	08

1. [14 points] Answer the following questions about AVL trees.

(a) [9 points] Recall that optimized implementations of AVL trees store *balance* (= height(left)-height(right)) in each node. They do not store the size of each subtree explicitly. Consider the case where a new *AVLNode*s has been inserted in the left subtree of left subtree of *AVLNode* a and balance values up to *a* have been updated in a bottom up pass. The algorithm finds *a* to be the first node in this pass where the updated balance is not between -1 and 1. Write the pseudo-code for the appropriate rotation to balance the AVL tree. You may assume these methods:

```
void setParent(AVLNode n);
void setLeftChild(AVLNode n);
void setRightChild(AVLNode n);
void setBalance(int b);
AVLNode getParent();
AVLNode getLeftChild();
AVLNode getRightChild();
int getBalance();
```

*5.5* Since, insertion has been made into left of left subtree it is an outside case, and hence can be resolved by a single rotation.

*b*  $\leftarrow$  *a*.getLeftChild()

*c*  $\leftarrow$  *b*.getLeftChild()

*p*  $\leftarrow$  *a*.getParent()

*a*.setLeftChild(*b*.getRightChild());  
~~*b*.setRightChild()~~

if *p*.getLeftChild() = *a*

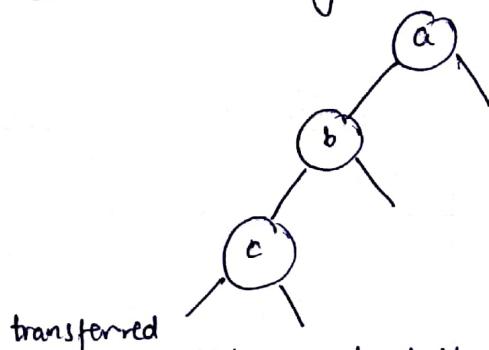
*p*.setLeftChild(*b*)

else *p*.setRightChild(*b*)

*b*.setParent(*p*)

*b*.setRightChild(*a*)

*a*.setParent(*b*)



promoted *b*

demoted *a*

balance not updated

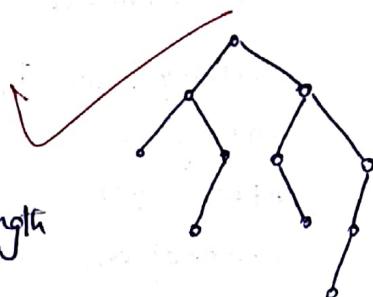
- (b) [2.5 points] True or False: The AVL invariant implies that a tree's shortest and longest paths (from root to any leaf) differ in length by at most 1. Explain.

False.

2.5

AVL invariant only implies that for every node height of left and right subtree differ by atmost 1.

This places a constraint on max. path length in right and left subtree and not on all path lengths in general.



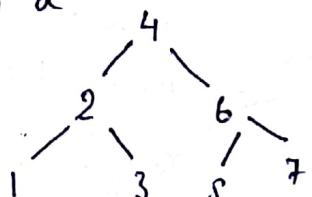
For eg. Tree drawn alongside is an AVL tree, though its shortest path is 2 units and longest 4 units.

2.9

- (c) [2.5 points] What order should we insert the elements {1, 2, 3, 4, 5, 6, 7} into an empty AVL tree so that we don't have to perform any rotations on it? Explain.

There can be many possible orders, one of them can be

4 2 6 1 3 5 7



The one thing to be kept in mind is to insert median of the array first.

Then median of left half-array and right half-array.

Follow this recursively.

2. [6 points] Prove or disprove: Five elements cannot be sorted with at most seven comparisons in the comparison model.

Five elements can have  $5!$  i.e. 120 possible combinations.

Initially, all the  $5!$  combinations could be potential sorted arrangement.

After 1st comparison, we ~~can~~ narrow down to half the combinations i.e. 60.

After 2nd comparison, to 30.

Similarly after 3rd, 15

4th, 8 at max.

5th, 4

6th, 2

7th, 1



Thus, in seven step comparisons, we can uniquely determine the permutation (in worst case) which is the sorted one.

Hence the given statement is false.

---

To get a feel and intuition for answer

The binary decision tree made by the working of algorithm

will ~~be~~ like an AVL tree and hence its height

must be bounded  $\log n! \sim O(n \log n)$ . For 5,  $n \log n$

$\approx 10$ . Hence, definitely the answer should be around 10.

3. [7 points] Answer questions about the procedure Stooge-sort

Input: array  $A[0..n-1]$  of  $n$  numbers  
 Output:  $A$  is sorted in increasing order.

If  $n = 2$  and  $A[0] > A[1]$ , then swap ( $A[0]$ ,  $A[1]$ )

If  $n > 2$  then {

Stooge-sort( $A[0..ceil(2n/3)]$ ) // sort first two-thirds.

Stooge-sort( $A[floor(n/3)..n]$ ) // sort last two-thirds.

Stooge-sort( $A[0..ceil(2n/3)]$ ) // sort first two-thirds again.

}

(a) Let  $T(n)$  denote the worst case number of comparisons ( $A[0] > A[1]$ ) made for an input array of  $n$  numbers. Give a recurrence relation for  $T(n)$ .

$$\begin{aligned} T(n) &= T\left(\frac{2n}{3}\right) + T\left(\frac{2n}{3}\right) + T\left(\frac{2n}{3}\right) + O(1) \\ &= 3 T\left(\frac{2n}{3}\right) + O(1) \end{aligned}$$

1.5

(b). Solve the recurrence – give a tight ( $\Theta$ ) asymptotic bound for  $T(n)$ . You are not allowed to use Master theorem (if you know it).

$$\begin{aligned} T(n) &= 3 (T\left(\frac{2n}{3}\right)) + O(1) \\ &= 3 (3 T\left(\frac{n}{3}, \frac{2n}{3}\right) + O(1)) + O(1) \\ &= 9 T\left(\left(\frac{2}{3}\right)^r n\right) + (3+1) O(1) \\ &= 27 (T\left(\left(\frac{2}{3}\right)^r n\right)) + (9+3+1) O(1) \\ &= 3^r (T\left(\left(\frac{2}{3}\right)^r n\right)) + \sum_{i=0}^{r-1} 3^i \cdot O(1) \\ &\stackrel{\log \frac{n}{2}}{=} 3^{\frac{\log n}{\log \frac{3}{2}}} + \sum_{i=0}^{\log \frac{n}{2}-1} 3^i \\ &= (3^{\log \frac{n}{2}} - 1) O(1) \\ &= (3^{\log \frac{n}{2}} - 1) O(1) = \Theta(n^{\frac{1}{2}}) \end{aligned}$$

incorrect

We know at  $n=2$   
 $T(n) = O(1)$   
 $\left(\frac{2}{3}\right)^r n \leq 2$   
 $n \leq 2 \cdot \left(\frac{2}{3}\right)^r$   
 $\log_2 n \leq \log_2 2 + \dots$   
 $r \geq \log_{\frac{3}{2}} \left(\frac{n}{2}\right)$

(c) Is Stooge-sort a correct sorting algorithm? (no explanation needed)

Yes

1

(d) Complexity-wise is Stooge-sort a better algorithm than insertion sort? Explain.

4. [5 points] We are given a sorted array  $A$  of size  $n=2^m-1$ . We are given one of the elements of  $A$  as the search input  $key$  and our goal is to find the index in the array at which  $key$  is present. Describe a recursive divide and conquer procedure for this problem (no pseudo-code necessary). Find its average case time complexity (in terms of number of comparisons). What are the various cases for computing the average complexity? Show your work. You may assume that the input will always be present in the array.

Since we know, that the array is sorted, we can employ binary search to find the index of element.

① ✓

```
def bsearch (low, high, key)
    mid <- (low + high) / 2
    if A[mid] = key
        return mid
    elif A[mid] < key
        return bsearch (mid+1, high, key)
    else
        return bsearch (low, mid-1, key)
```

bsearch (0,  $2^m-1$ , key)  $\leftarrow$  # init call

we pass the range of possible values of key.

It checks if middle value of range is equal to key. If yes then we are done. If no, and middle value is less than key, then key must be in right half and we call the right half of array.

Else, we call the search in left half.

Best case: When key is median

Worst case: When key is at 0 or  $2^m-1$  index.

① ✓ Avg. case can have uniformly distributed key across the array.

5. [8 points] Insertions and deletions in balanced binary search trees.

(a) Show the series of B-trees (with  $t=3$ ) when inserting M,S,F,R,A,K,C,D,E,N,H,P,J,Q,G (in that order) in an empty tree. Use top down insertion. You need to only draw the trees just before and after each split.

1. M

2. M S

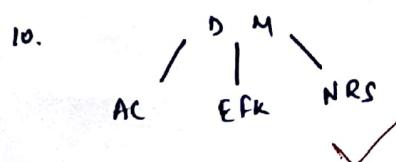
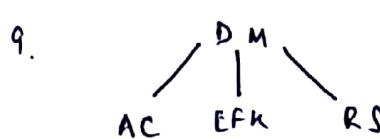
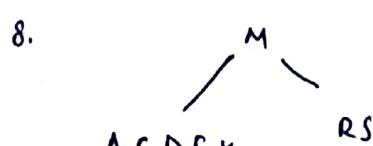
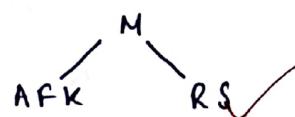
3. F M S

4. F M R S

5. A F M R S ✓

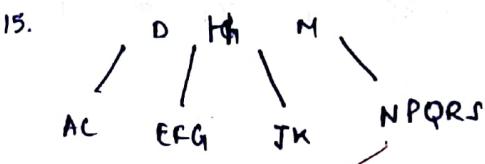
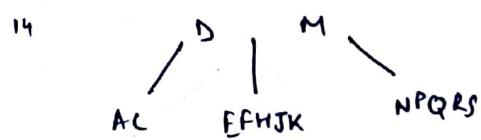
6. ~~WAN PRS~~ overflow

A 6 node seen hence a split followed by insertion of K



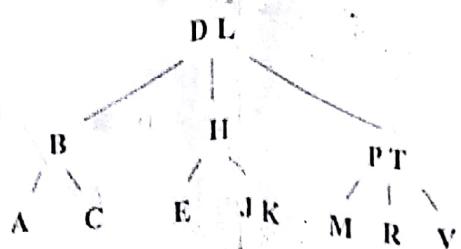
$$t-1 = 2$$

$$2t-1 = 5$$



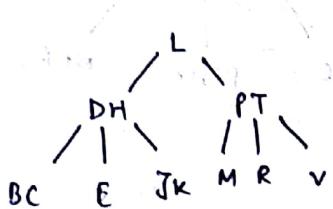
⑤

(b) Delete the keys A, V, and P from the following 2-4 tree using the top down deletion algorithm discussed in class. Show the result after each deletion.



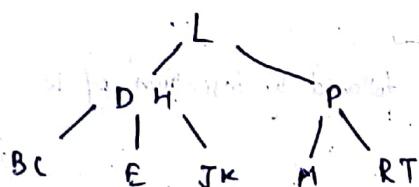
③

1.



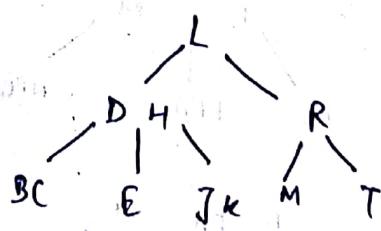
Deletion of A

2.



Deletion of V

3.



Deletion of P