(30)

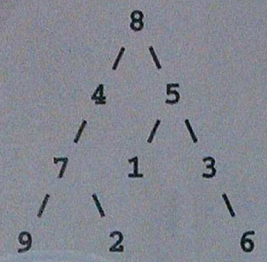
CSL201 Data Structures Minor 1 30th Aug, 2013

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

(2)

---

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

(5)

```
node * duplicate( node *A) {
```

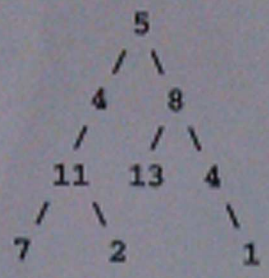
part 1

```
node * p = A;
if (p==NULL)
  { cout << "\n 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==NULL) break;
    }
  if (A->data == word1)
    {
      node * h = A;
      A = A->next;
      delete(h);
      if (A==NULL) return;
```

part 2

```
p=A;
if (p->next==NULL) break;
while(p->data==p->next->data)
{
  node * h = p;
  int word2 = p->data;
  p = p->next;
  delete(h);
  if (p==NULL) break;
}
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.

bool funct (node* p, int n) {

if p->data does not divides n then return false;

else return funct (p->left, n/p->data)

|| funct (p->right, n/p->data).

}

no checking of whether root is null or not.
no terminating cond^n

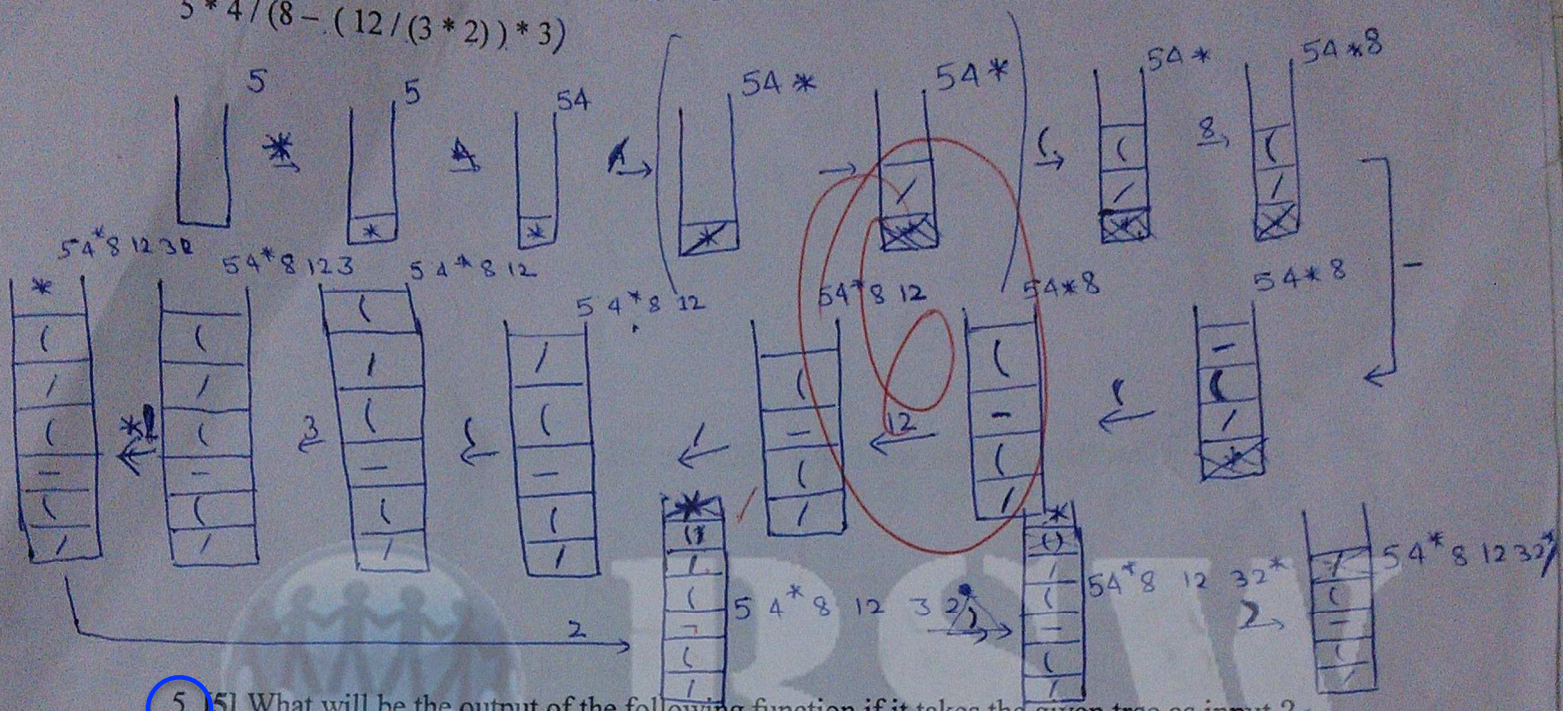
no correct logic

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.

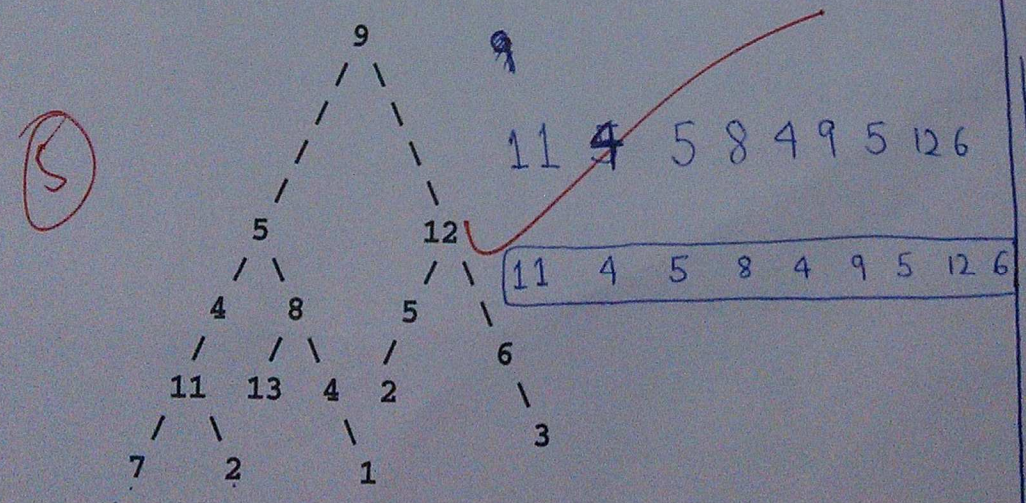
5 * 4 / (8 – (12 / (3 * 2)) * 3)

The character on the arrow is the character read from expression



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);}
```



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 which is 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;   → error: first = q;
}                    should be there
```

(5)

Output:

first → 78 - 84 - 62 - 40 - 92 - 55 -

13.5

Name: Sankalp Entry No: 2014PHY10822

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

238

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

Q1. (Time Complexity and 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(i);
           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.

assuming each comparison takes 1 unit of time

time units of $f(n) = 1 + [i+4]n + 3\log_2 n$

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

Time complexity $O(n^2)$

1.5

Name: _____ Entry No: _____

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.i.f

time units taken by f(n)

$$= 1 + 4n + \sum_{i=1}^{n} \log i + \text{(2+i) do } 2\log_2 n + \sum_{i=1}^{\lfloor \log_2 n \rfloor} i$$

$$= 1 + 4n + \log(n!) + 2\log_2 n + \frac{\lfloor \log_2 n \rfloor (\lfloor \log_2 n \rfloor + 1)}{2}$$

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

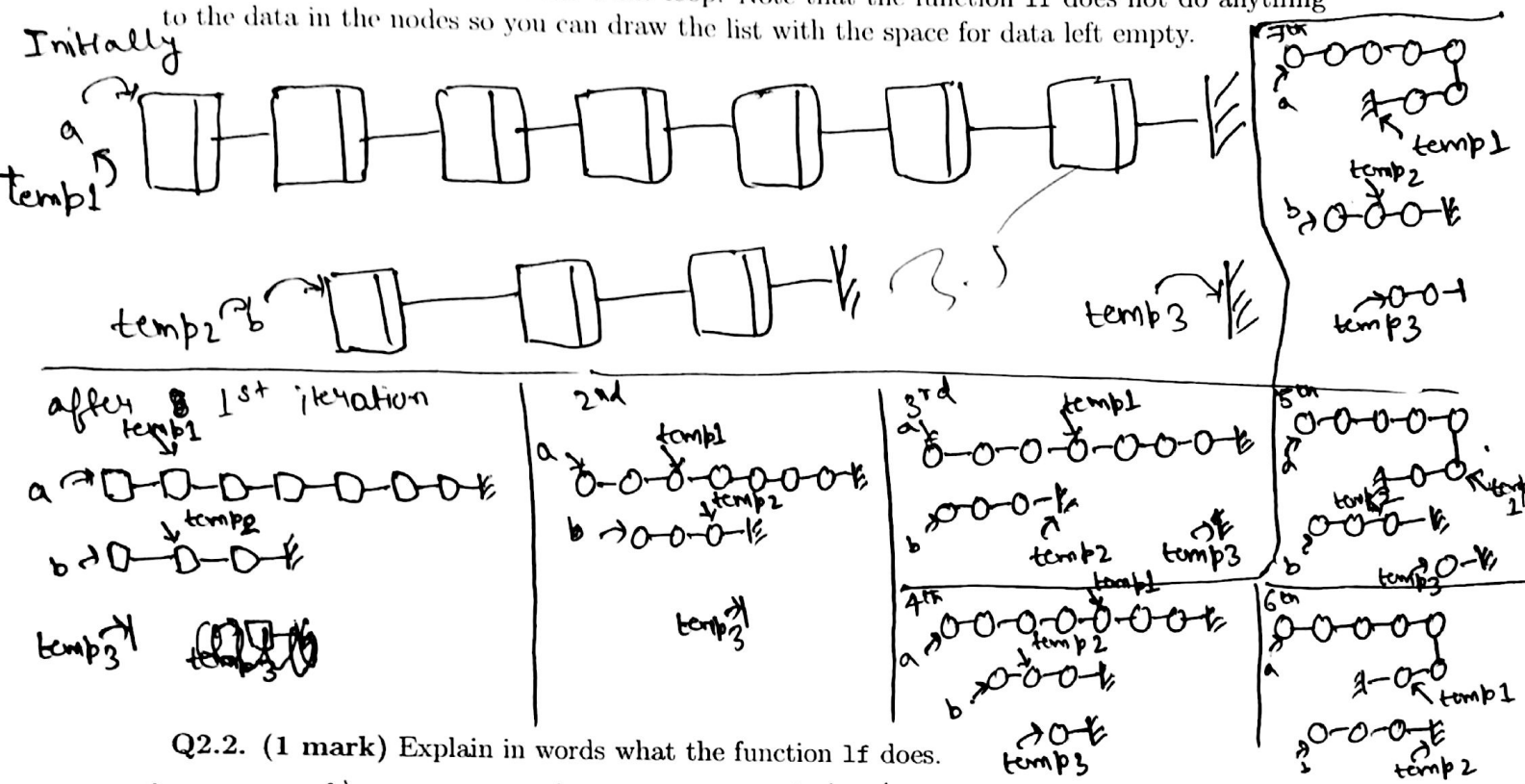
Marks: 5

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: 2014PH10822

Q2.1. (3.5 marks) Suppose we call `lf` 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 `lf` does not do anything to the data in the nodes so you can draw the list with the space for data left empty.



Q2.2. (1 mark) Explain in words what the function `lf` does.

The function lf returns a linked list with no. of nodes equal to ~~no. of~~ ~~least inter~~ 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 `lf` 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.

time taken by I $= m + m + [m/n]$

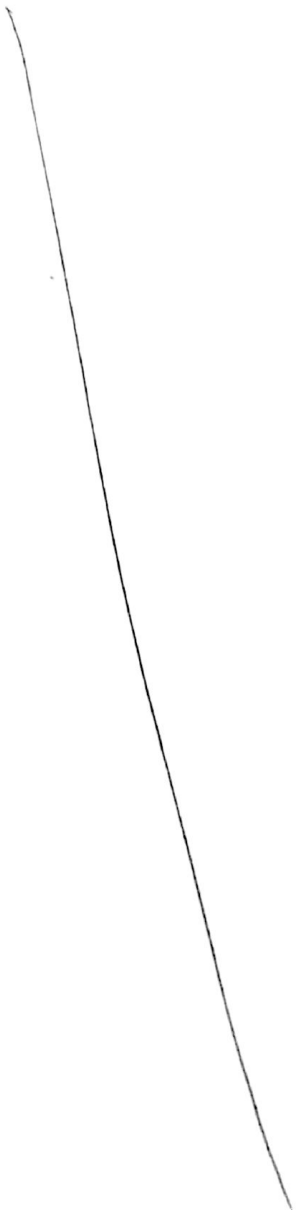
$= 2m + [m/n]$

time ~~complete~~ 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: Samkalp Entry No: 2014PH10822

Q4. (Trees. Total marks = 5) Marks: 4

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 ith child, returning `null` if there are less than i children.
- `public Tree subtree(int i)`: This returns the `Tree` rooted at the ith 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. (as each and every node/subtree 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

COL106: Data Structures, I semester, 2016-17
Minor I
1 PM to 2 PM, 29th August 2016.

Attendance Sheet Serial Number: 186

Question	1	2	3	4	Total
	(5 marks)	(5+2 marks)	(5 marks)	(5 marks)	(20+2 marks)
Marks	4	0 + 0	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 ith child, returning `null` if there are less than i children.
- `public Tree subtree(int i)`: This returns the `Tree` rooted at the ith 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 i = 0;
        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 f1(T) outputs? Your answer must be as precise as possible.

f1(T) outputs the maximum positive data stored in all nodes of T

f1(T) returns 0 if all nodes data is negative

(1)

Q1.2. (3 marks): Given a tree T explain in words what the function f2(T) outputs? Your answer must be as precise as possible.

f2(T) outputs 1 if value in each node is greater than value of parent node, else it returns 0.

(3)

Q2. (Amortised analysis. Total marks = 5 + 2 extra credit). Marks: 0 + 0

A *n-counter* is a 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 $i+1 \mod n$. 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 $0 \mod n$.

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 flips? On the basis of this answer, what is the worst case running time of the the algorithm applied n times starting from 0 and counting up $n-1$ and then back to 0?

O(n) is worst case time for ~~worst case~~ of bit flips

~~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 emulating 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 – 5) [illegible]

Suppose we have some already defined functions [illegible] the following [illegible] 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(i);
          c = 2*c;
       }
  }
}
```

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-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 $O(n^3)$. 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.

No. of units $= 1 + 2n + n + \sum_{i=1}^{n} \log i + n + \sum_{j=1}^{\lfloor \log n \rfloor} 2^j + n\lfloor \log n \rfloor + \lfloor \log n \rfloor$

(assignment, for, print, g(i), (i == c), h(i), print, print, multiplication)

Since, $\sum_{i=1}^{n} \log i$ is $O(n \log n)$ and $\sum_{j=1}^{\lfloor \log n \rfloor} 2^j$ is $O(n)$,

(2) 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 send 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 $1/i$ and h(i) costs Rs i. In configuration C_2, g(i) costs Rs 1 per execution and h(i) also costs Rs 1 per execution. *Asymptotically* which of the two configurations is cheaper in rupee terms? Argue your answer. If you just give the answer without any *mathematical* argument, you will receive a 0.

~~Let~~ For C_1

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

$O(n) \quad \log n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots \quad O(n^3)$

$g(n)$ = Cost 2 = $n + ~~(1+2 \ldots + \lfloor \log n \rfloor)~~ + n\lfloor \log n \rfloor$ ($O(n)$)

$= O(n^2)$

(1/2)

1

$f(n) > g(n)$

$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n^2}{n^3} \quad \log_{n\to\infty} \frac{g(n)}{f(n)} < \log \frac{n^2}{n^3}$ [Using definition of little o, $g(n) \in o(f(n))$

Hence, ~~cost for C_2~~ C_2 is cheaper

Scanned by CamScanner

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) ~~Dequeue~~ ~~Avail~~

~~If front~~ · Push() front() to S_1

Entry

COL 106 Autumn 2017 Minor 1

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

Before starting the exam, close your eyes and take three deep breath the exam is not an accurate reflection of your understanding of the n 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 upto second argument ~~greater~~ smaller 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 $2^k * m > n \Rightarrow k > \log_2(n/m)$

~~Each~~ $T(n,m) = c + T(n,2m)$

$= c + c + T(n, 2^2 m) = \underbrace{c + c + \dots c}_{k} + T(n, 2^k m)$

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

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 $r = n \quad q = 0$

$r + qs = n + 0 = n$

$\therefore$ initially invariant condition is true.

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

$\therefore \quad r_i + q_i * s = n$

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

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

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

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

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

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

$= r_i + q_i \times s = n$ by our assumption that invariant is true at i^{th} iteration.

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

Termination : Loop terminates when $r < s$.

By, loop invariant,

$n = qs + r \qquad r \in [0, s)$

According to Euclid's division lemma

$a = qb + c \quad c \in [0, s)$ 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 *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.

9.5+0.5

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)

10

```
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          // if still empty means both stacks are empty and there are no elements to remove.
else
    q.size--
    return s2.pop()
```

5

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

∴ Ammortized time for n operations of push = $\frac{\overbrace{T(1)+T(1)\ldots T(1)}^{n \text{ times}}}{n}$

= O(1) 0.5

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 uptil c^{th} call to dequeue. ∴ Time to remove first "c" elements = $c\,T(1)$

1.5 Now, for $(c+1)^{th}$ call, we need to transfer all $(n-c)$ elements to s2.

This will take $(n-c)$ steps + 1 step to pop again from s2.

+ ∴ Time = $2T(n-c)+T(1)$

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

0.5 ∴ Total time for n dequeue operations = $\underbrace{n}_{\text{n pops from s2}} + \underbrace{2(n-c)}_{\text{transfer operations}}$

+0.5 You need to specify where for a sequence a queue gets empty multiple times.

$\leq 3n$

Ammortised time $\leq \frac{3n}{n} = 3$ ✓

Ammortised time for dequeu = O(1)

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

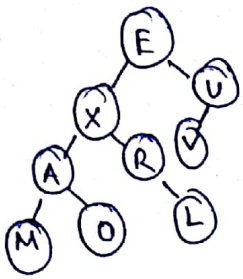
Deque Method	Realization with Vector Methods
`size()`	size()
`isEmpty()`	isEmpty()
`insertFirst(e)`	insertAtRank(0, e)
`insertLast(e)`	insertAtRank(this.size(), e)
`removeFirst()`	removeAtRank(0)
`removeLast()`	removeAtRank(this.size()-1)
`first()`	elemAtRank(0)
`last()`	elemAtRank(this.size()-1)

(9)

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.

2

(a) What is the asymptotic complexity of determining whether an element e appears in the list? $O(n)$ $\Theta(n)$

(b) What is the asymptotic complexity of finding the median of the list of numbers? $O(n)$ $\Theta(n)$

(c) What is the asymptotic complexity of finding the smallest element in the list? $O(1)$ $\Theta(1)$

(d) What is the asymptotic complexity of finding the largest element in the list? $O(1)$ $\Theta(1)$

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

(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)

(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

CSL 201 Data-Structures Minor Test 2 7th Oct, 2013

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.

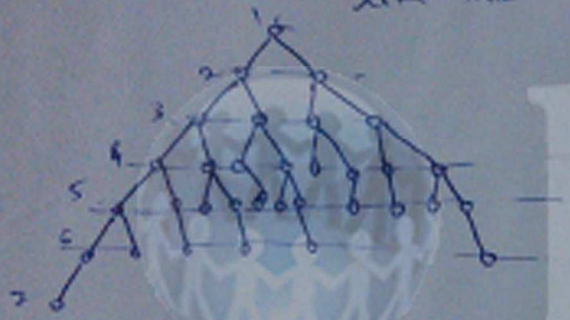
1. [5] What is the maximum and the minimum number of nodes in an AVL tree with 7 levels ?

Maximum number of nodes in AVL tree with 7 levels

$= 2^7 - 1 = 127$ nodes ✓

Minimum no of nodes: level 1 → 1

The tree will be (exception) like this



2 → 2

3 → $2^2 = 4$

4 → $2^3 = 8$

5 → 11

6 → 6

7 → 1

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 4th customer in order of placing orders with the company. Indicate the most efficient way of picking up the winner from the array.

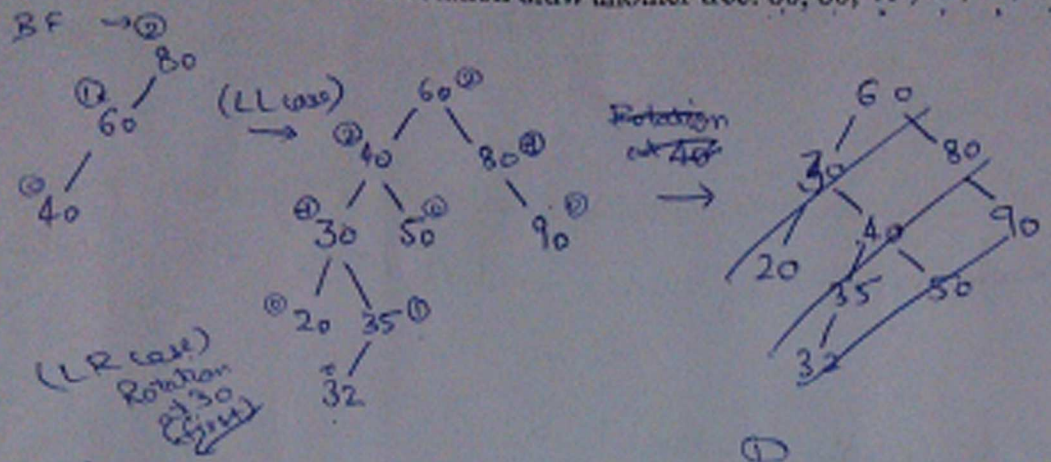
Assuming that order of placing orders is same as order of time stamp.

We pick first four customers, create ~~max~~ heap acc to time st if next element is less that max element, we replace & heapify else go one to next element. At last the ~~head~~ is change to min ~~heap~~ & the ~~last~~ first ele

5

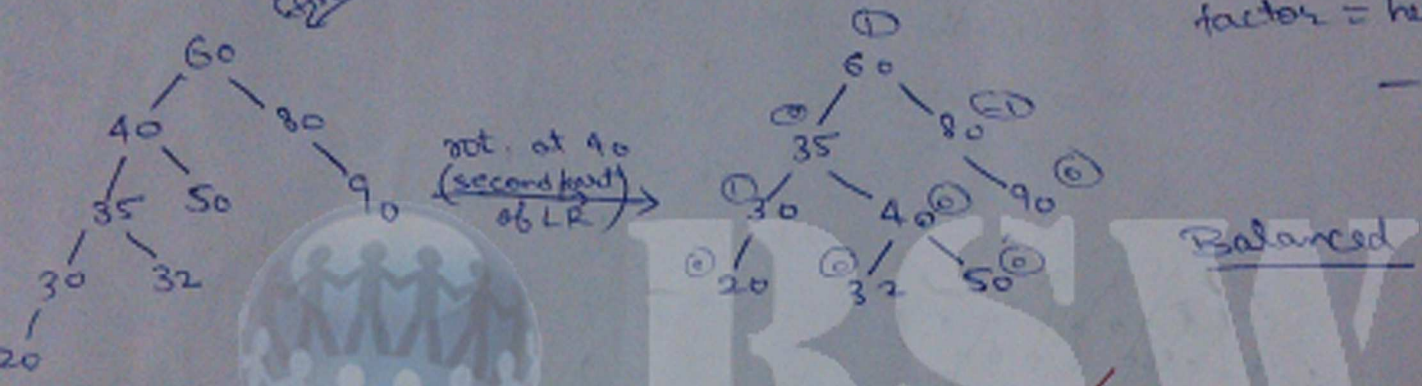
3. [6] Draw an AVL tree by inserting the following elements in the given order. Whenever the structure of the tree needs a rotation draw another tree: 80, 60, 40 , 30, 90, 50, 20, 35, 32.

(6)



(LR case) Rotation at 30 (first)

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



Balanced

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]

↘ Replacing 45 as pivot

[45 3 76 32 12 92 34 9 56 60 20]
i→ i j → j
20 9 92

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

[34 30 20 32 12 09 45 92 56 60 20]

↓

~~[9 3 20 32 12 34 45 30 56 60 92]~~

~~[3 9 12 20 32 34]~~

3

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

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

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

⇒ [[3 9 12 20 32 34] [45] [56 60 76 92]]

⇒ [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(n/5) + n, T(1)=1.

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

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

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

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

$\Rightarrow T(n) = 4\left[2T\left(\frac{n}{5^3}\right) + \frac{n}{5^2}\right] + \left(\frac{2n}{5} + n\right) = 2^3 T\left(\frac{n}{5^3}\right) + \left(\frac{2^2}{5^2} + \frac{2}{5} + 1\right)n$

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

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

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

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

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

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

∵ no of elements in GP = $\log_5 n$

for $= a\frac{(1-r^n)}{(1-r)}$, $r < 1$

6

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] >= A[2i] && A[i] >= A[2i+1])
    { k = A[i]; A[i] = A[2i]; A[2i] = k;
      percolate_down(A, 2i);
    }
    else if ( A[i] < A[2i+2] && A[2i+1] > A[2i])
    {
      k = A[i]
      A[i] = A[2i+1];
      A[2i+1] = k;
      percolate_down(A, 2i+1);
    }
  }
}
void percolatedown (int A[], int k)
{ int l; if (2k >= N) return;
  else if (A[k] < A[2k] && A[k] >= A[2k+1])
  { l = A[k];
    A[k] = A[2k]; A[2k] = l;
    percolatedown(A, 2k);
  }
  else if (A[k] < A[2k+1] && A[2k+1] > A[2k])
  { l = A[k]; A[k] = A[2k+1]; A[2k+1] = l; percolate_down(A,
}
```

Check if children exist or not? right child?

Name: Kamalnath. Pclakam 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.

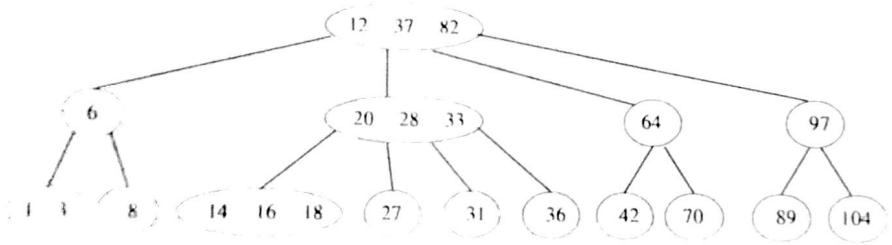
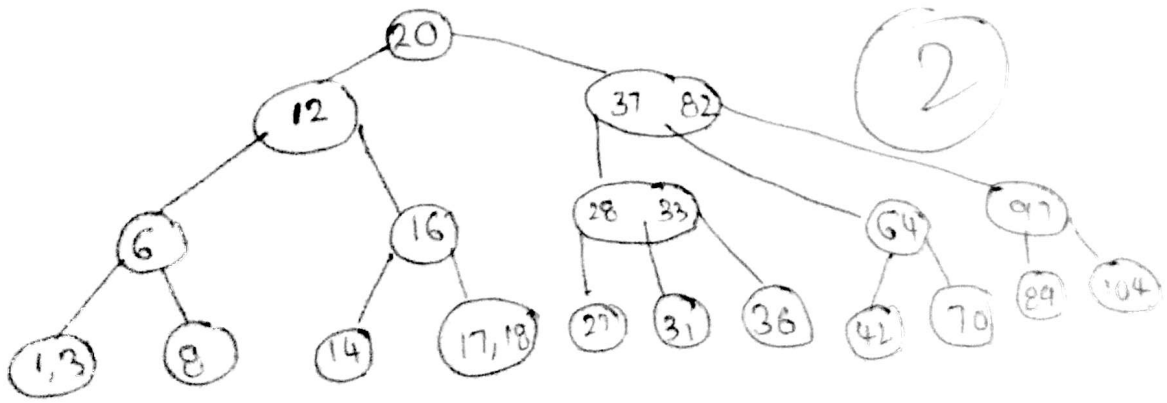
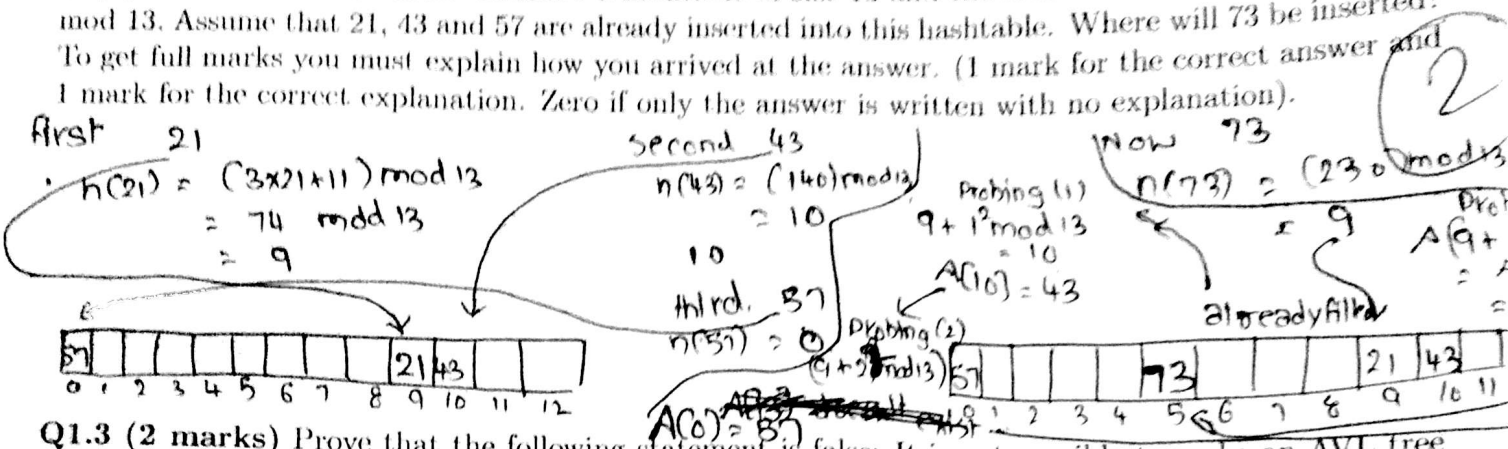


Figure 1: A 2-4 tree

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 $A[h(x) \mod n]$ is occupied, we try $A[h(x) + 1^2 \mod n]$, $A[h(x) + 2^2 \mod n]$, $A[h(x) + 3^2 \mod n]$, etc in sequence till we find an empty slot and insert x there. Consider a hashtable of size 13 and the hash function $h(x) = (3x + 11) \mod 13$. 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).



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 $x_1, x_2, \ldots, x_n$ 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) ≥ height(x_j) and
$\forall$ k ∈ (i, j) (k lies between i and
height(x_k) ≤ height(x_i

probability calculation :- 2 p(h(x_i) > h(x_j), ∀ k ∈ (i,j)
h(x_i) > h(x

h(x_t) = ht
∀ t ∈ [i+1, j
max(h_{i+1}, h_{i+2}, ..., h_j) + 1 2

(x_t) = ht
mean x_t was
ated ht time

2

(p)

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 $t_1, t_2, \ldots, t_k$ (each t_i is either L or R) and let $x_1, x_2, \ldots, x_k$ be the 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) ~~Let~~ ~~the~~ ~~new~~ node ~~be~~ ~~X~~ ~~has~~ ~~key~~ ~~X~~ as

As during traversal it is given both t_i and t_j are both left turns, The Node we are searching for is less than both the given N

case 1) If $i > j$. That means the node with key x_j is a child of x_j (since BST traversal path goes in this way) as t_j is left . Node(x_j) is in the left subtree of N

$\therefore x_j < x_i$, so for $x_j > x_i$ $i > j$ (In all other ca Node(x_j) is right c Node(x_j) so it s $x_j < x_i$

Marks: 8

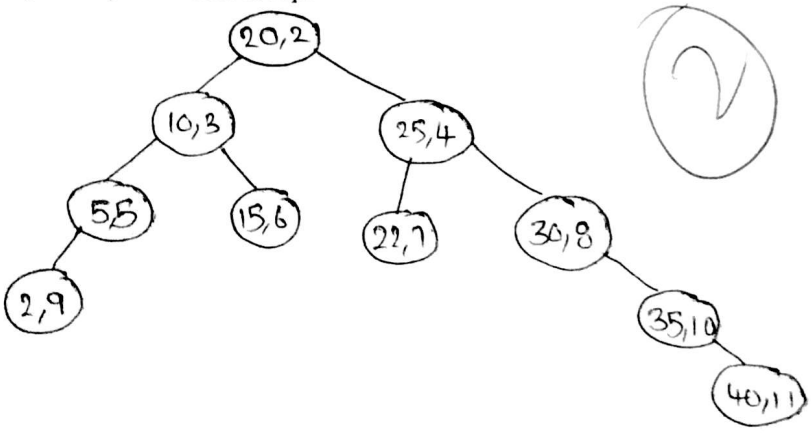
Q2. (Tree + heap = Treap. Total marks = 9).
A *Treap* is a special kind of binary search tree which also has the property of a heap. Each item in a treap is a tuple of two values i.e. $x_i = (k_i, p_i)$ where k_i is a key and p_i is a priority. A treap is a binary search tree on the keys of the items 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

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

8 2 9 3 8 9 4 6 2 5

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

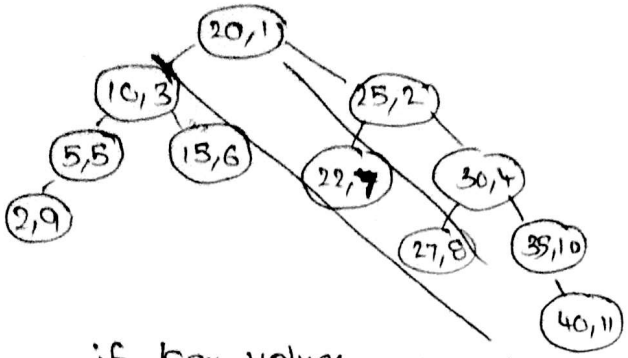


2

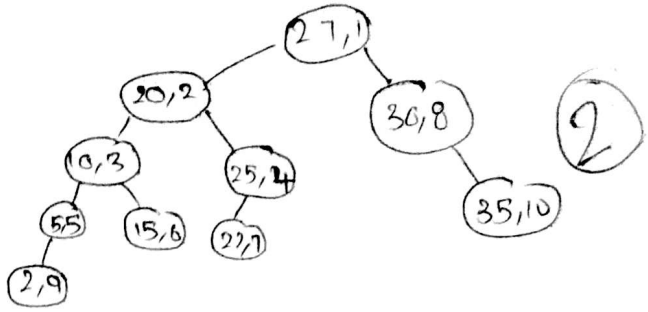
So time (complexity is O

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).

5A (Continued)) As it can be seen through rotations every sub BS can be successfully converted to new sub treap with N' as it new root. The rotations take constant time, B if the whole tree can't be corrected to a treap in 'phas then we must continue to 'Next phase'. This is recursive i nature and should be continued until the whole tree becomes treap. In the 'N' becomes the new root so whole tree height is th time



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.)

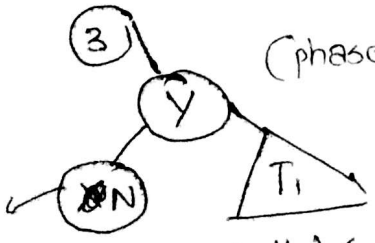
A) Treap insertion algorithm.

1) Begin with root if root(key) < NewNode(key) go to the right subtree of the root and repeat the same algorithm, if root(key) > NewNode(key) go to the left subtree of the root and repeat the same algorithm. Repeat until you reach the perfect node ~~a leaf and~~ attach the NewNode there as a child. (This is the insertion)

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

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

N is new node

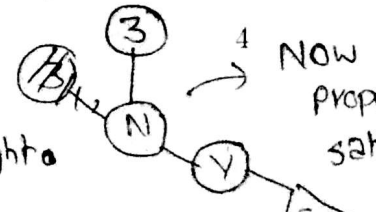


(phase 1)

N(priority) < y(priority)

phase (1). We must rotate this subtree such that BST properties are still valid.

Note: ~~3(key)~~



Now Treap property is satisfied.

if y was right child

Before. 3(key) ≤ N(key) ≤ y(key) ≤ Root(T1)(key)

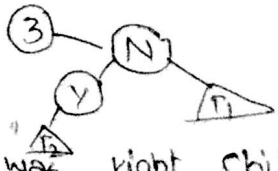
Now. 3(key) ≤ N(key) ≤ y(key) ≤ Root(T1)(key).

similar if ~~case~~ y was left child).

(Next phase)

N(priority) < y(priority)

Phase 1 already happend so now the subtrees T2 and 'N' are treaps but y is just a BST.



if y was right child.

Before: 3(key) ≤ N(key) ≤ y(key)

Now: 3(key) ≤ N(key) ≤ y(key)

COL 106 Autumn 2017 Minor 2

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

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

Question Number	Maximum Marks
1	14
2	06
3	07
4	05
5	08

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

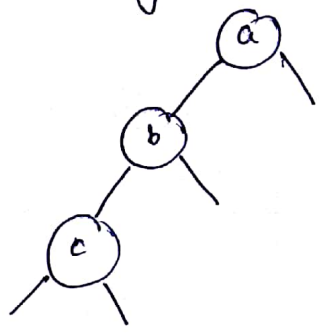
10.5

(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 ← a.getleftChild()

c ← b.getleftChild()

P ← a.getParent()

a.setLeftChild(b.getRightChild()) } transferred b's right child to a's left

~~b.setRightChi~~

if p.getleftchild() = a

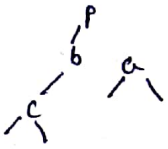
p.setleftchild(b)

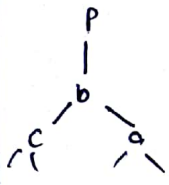
else p.setRightchild(b) } 1.5 promoted b

b.setparent(p)

b.setRightchild(a) } demoted a

a.setparent(b)





balance not updated

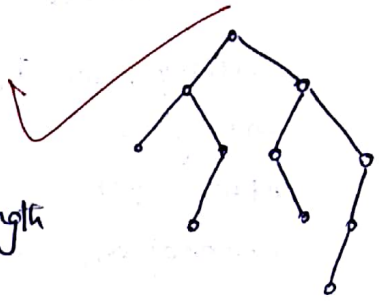
(b) [2.5 points] True or False: The AVL invariant implies that a tree's shortest and longest paths (from root to any leaf) differ in length by at most 1. Explain.

2.5

False.

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.



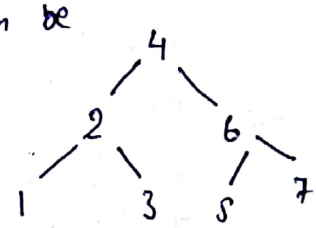
For eg. Tree drawn alongside is an AVL tree, though its shortest path is 2 units and longest 4 units.

(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.

2.5

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 ~~know~~ narrow down to half the combinations i.e. 60.

After IInd comperison, to 30.

Similarly after IIIrd, 15

IVth, 8 at max.

Vth, 4

VIth, 2

VIIth, 1

Thus, in seven ~~step~~ comperisons, 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 ~~have~~ 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, depthitely 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.
}
```

(5)

(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 $T(n)$.

$$T(n) = T(\tfrac{2n}{3}) + T(\tfrac{2n}{3}) + T(\tfrac{2n}{3}) + O(1)$$

$$= 3\,T(\tfrac{2n}{3}) + \Theta(1)$$

(1.5)

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

$$T(n) = 3\left(T(\tfrac{2n}{3})\right) + \Theta(1)$$

$$= 3\left(3\,T(\tfrac{2}{3}\cdot\tfrac{2}{3}n) + \Theta(1)\right) + \Theta(1)$$

$$= 9\,T\left((\tfrac{2}{3})^2 n\right) + (3+1)\,\Theta(1)$$

$$= 27\left(T((\tfrac{2}{3})^3 n)\right) + (9+3+1)\,\Theta(1)$$

$$= 3^r\left(T((\tfrac{2}{3})^r n)\right) + \sum_{i=0}^{r} 3^i \cdot \Theta(1)$$

$$= 3^{\log_{3/2}(n/2)}\,\Theta(1) + \sum_{i=0}^{r} 3^i\,\Theta(1) \quad \text{where } r \geq \log_{3/2}(n/2)$$

$$= (3^{r+1}-1)\,\Theta(1)$$

$$= \left(3^{\log_{3/2}(n/2)+1}-1\right)\Theta(1) = \Theta(n^2)$$ incorrect

(2.5)

we know at $n=2$, $T(n) = O(1)$

$(\tfrac{2}{3})^r n \leq 2$

$\Rightarrow n \leq 2\cdot(\tfrac{3}{2})^r$

$\log_{3/2} n \leq \log_{3/2} 2 + r$

$r \geq \log_{3/2}(n/2)$

(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.

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

bsearch (0, 2^m - 1, key) ← # 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.

(1)

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.

COSLS

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

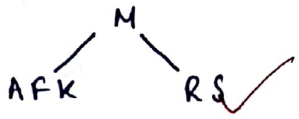
3. F M S

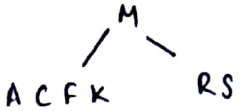
4. F M R S

5. A F M R S ✓

6. ~~A F K M R S~~ overflow

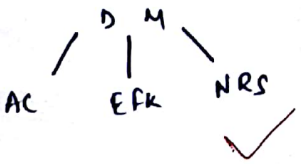
A 6 node seen hence a split followed by insertion of k



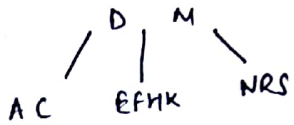
7. 

8. 

9. 

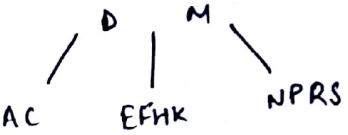
10. 

✓

11. 

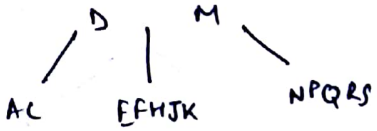
t-1 = 2

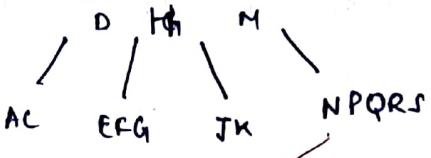
2t-1 = 5

12. 

✓

13. 

14 

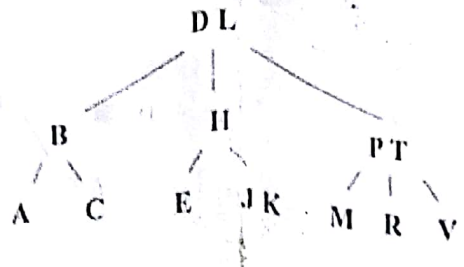
15. 

✓

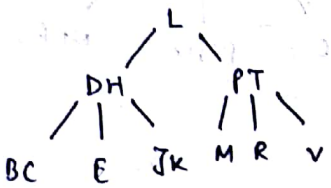
(5)

(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.

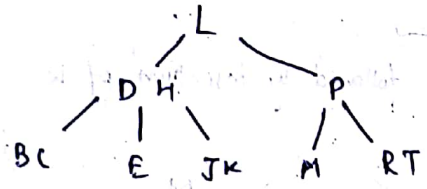
(3)

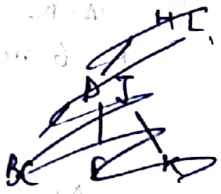


1. Deletion of A

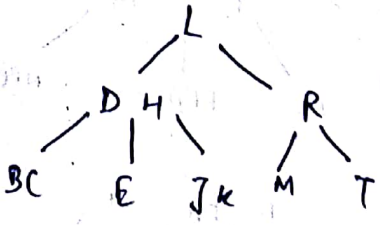


2. Deletion of V.





3. Deletion of P



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.)

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

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

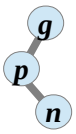
Insert:873	Insert:9734	Insert:280	Delete:9734	Search 143	Insert:14	Rough space

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 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 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.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEEING
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 funetion for the STACK. Briefly describe how you would ehange it if you were to implement a queue. (3+2)

e. Write the **insert(dataType D)** member function for the STACK. Briefly deseribe how you would ehange 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 ean for
   i. **getnext()** for your QUEUE class,
   ii. **insert()** for your QUEUE class

Assume N elements in the queue. Bricfly explain your answer. (2+2)

Q3. (13)

a. Given the following Adjaceney Matrix, draw unweighted direeted graph labelling nodes numerieally starting at 1. (4)

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

b. Perform a depth first seareh on the above graph. Clearly show the tree edges. Start the seareh at node 1 and always seleet successors in numerieally increasing order. What is the maximum size of the staek during this DFS run? (4+2)

e. What is the maximum degree of any vertex in the graph? Does the graph have any directed cyeles? (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, 4[illegible]

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 eonstrueted in (b). (2)

d. Suppose we want to use an array to implement a stack. Howevcr, we do not know in advance the size to which the stack can grow. Indeed there may not bc any apriori limit to thc 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 exeecded, all we need is a variable "top" that points to top of stack (so A[top] is the next free eell). 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 alloeate 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 ean "amortize" the cost over the previous eheap operations that got us to this point. (4)

   Prove that with the above implementation of a staek, the cost of an arbitrary sequence of 'n' stack[push/pop] operations to the staek 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:

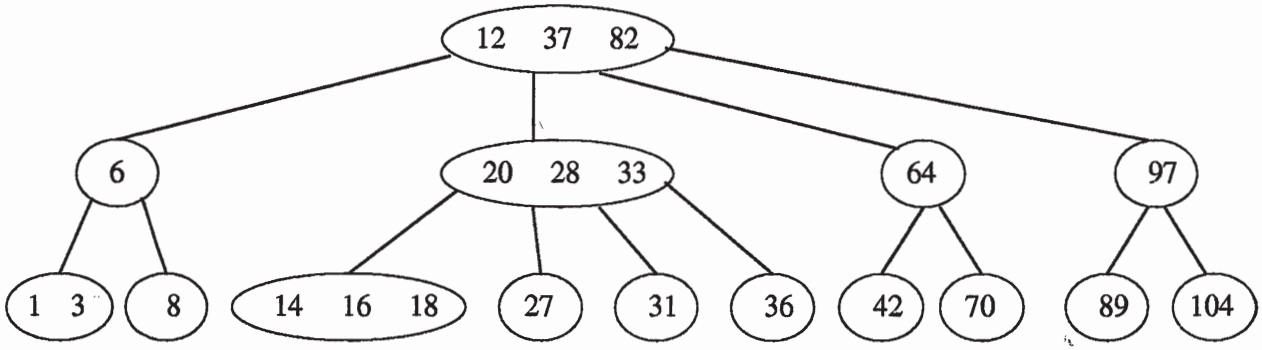


Figure 1: A 2-4 tree

   (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.

3. 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 (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 grwoth 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 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?

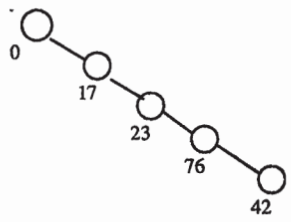


Figure 2:

(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) $O(1)$

(c) $\theta(\log n)$

(d) $\theta(1)$

(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 $a_1, a_2, \ldots, a_k$ in the list such that the a_1.next $= a_2, a_2$.next $= a_3, \ldots, a_{k-1}$.next $= a_k$ and a_k.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 + level(v) + 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.

Q1We 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 $h = \min\{h(x), h(y)\} \geqslant 1$ 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)**

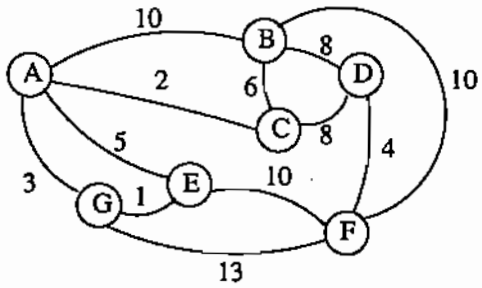
Name:_____ Entry No:_____

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 eounter 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 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)**