# COL106 Major

ARPIT SAXENA

TOTAL POINTS

**85.5 / 119**

QUESTION 1

## Q1 6 pts

### 1.1 Q1-a 2 / 2

✓ **+ 0.5 pts** first output line: 0

✓ **+ 0.5 pts** second output line: Child. 10

✓ **+ 0.5 pts** third output line: Parent

✓ **+ 0.5 pts** Correct order of all output lines

**+ 0.25 pts** Partially correct order

**+ 0 pts** Incorrect

### 1.2 Q1-b 3 / 4

**+ 1 pts** Output pattern: 0 1 2.. 2.. 1 2.. 2..

✓ **+ 3 pts** Preorder traversal and each node prints its depth. if n is the number of nodes at a depth d then d is printed n times

**+ 0 pts** Incorrect

QUESTION 2

## 2 Q2 2 / 3

**+ 1 pts** Class C ( line 1) : Class B is not public

**+ 1 pts** Class B ( line 2) : returns pa which is never initialized

✓ **+ 1 pts** Class C( line 3) : identifier for new object required

✓ **+ 1 pts** Class C ( line 4 ): Attempt to access protected method of class A( set() )

**+ 0 pts** Incorrect

QUESTION 3

## 3 Q3 1.5 / 4

**+ 0 pts** Inorrect

**+ 2 pts** 'Suppose the pivot is the first element' is a flaw since it does not guarantee worst case partitioning.

**+ 2 pts** Partition takes = kn+c is a flaw since there is no support to it in the proof (may not take theta(n) always)

✓ **+ 1.5 pts** Partial marking

QUESTION 4

## 4 Q4 4 / 4

**+ 0 pts** Incorrect

✓ **+ 2 pts** Mentioning that 'An algorithm must search through all k+1 elements' is a flaw.

✓ **+ 2 pts** Reasoning for above flaw is that to find the element which is going to be obtained in the (k+1)th comparison, we need not search through all the k+1 elements again.

QUESTION 5

## 5 Q5 2 / 3

**+ 3 pts** Wrong assumption of taking "n" to be a largest number

**+ 2 pts** mentioning n^2 <= n is true only for n=1

**+ 3 pts** Starting with "n" not equal to 1 , and arrive at n=1, so contradiction

**+ 0 pts** Incorrect

**+ 3 pts** Correct approach

**+ 2** Point adjustment

right approach but didn't pin point mistake of wrong assumption

QUESTION 6

## 6 Q6 4 / 5

**+ 0 pts** Incorrect/Unattempted/Irrelevant

✓ **+ 2 pts** Recognized that there can be no comparison-based sorting that does better that 'n logn' in general case

✓ **+ 2 pts** Recognized that worst case of any

**comparison based sorting cannot be better than 'n logn'**

+ **1 pts** Definition of omega (specifying equivalence relation (not equality) of bounds)

QUESTION 7

7 Q7 **7 / 12**

+ **5 pts** Part 1 - argument is correct

+ **2 pts** Part 1 - calculations are correct

✓ **+ 2 pts** **If the final answer of part 1 is correct but the logic is inherently buggy/ incomplete**

✓ **+ 5 pts** **Part 2 - correct approach and argument**

+ **2 pts** Extra credit: for exact expression in part 2

+ **0 pts** Incorrect or not attempted

QUESTION 8

8 Q8 **8 / 8**

✓ **+ 4 pts** **Fastest algorithm (log2 N + 1 ) by splitting into n/2 parts and doing 1 comparison at every level.**

✓ **+ 1 pts** **Time complexity (only if algorithm is the fastest)**

+ **3 pts** Algorithm Partial Credit: Algorithm with O(logn) asymptotic complexity but not the fastest

+ **1 pts** BONUS MARK: Identifying that in the first iteration, the n/3 split is better than n/2 split with same number of comparisons(2).

✓ **+ 3 pts** **Argument that shows no other algorithm can be faster than the above algorithm**

+ **1.5 pts** Argument partial credit : If asymptotic argument is provided or argument of optimality of 2 splits is shown.

+ **0 pts** Incorrect/Unattempted

QUESTION 9

9 Q9 **3 / 8**

+ **0 pts** Incorrect or Unattempted

+ **5 pts** Correct Algorithm

+ **3 pts** Correct Analysis (O(logn) solution)

- **2 pts** Inefficient Algorithm (Linear Time)

No marks to be given for analysis for O(n) part

Max 3 marks for correctness

**+ 3 Point adjustment**

QUESTION 10

10 Q10 **3 / 8**

+ **8 pts** Correct ( 5+3)

+ **3 pts** Correct Data Structure which give O(n) complexity

+ **2 pts** Correct Algorithm for the given data structure

+ **2 pts** Correct Analysis of Result. That is expected complexity is O(n)

+ **1 pts** Worst Case analysis

✓ **+ 2 pts** **For mentioning other data structure and algorithm.**

✓ **+ 1 pts** **Correct analysis though O(n) is not guaranteed.**

+ **0 pts** Incorrect/Unattempted

QUESTION 11

11 Q11 **9 / 16**

+ **0 pts** Incorrect / Unattempted

✓ **+ 2 pts** **Part A: State assumptions about "insert_into_output"**

+ **1 pts** Part A: Partially correct (e.g. analysed special case of m=n or m=1 and analysis wrong even in that special case)

✓ **+ 3 pts** **Part A: Partially correct (e.g. correctly analysed a special case, say m=n or m=0/1, but not worst case for general m and n, or analysed general case incorrectly)**

+ **5 pts** Part A: Partially correct (e.g. analysed the worst case for general m and n but not much progress beyond that)

+ **6 pts** Part A: Partially correct (e.g. analysed the worst case for general m and n but did not count the paths outside the range after the first node in the range)

+ **7 pts** Part A: O(mlog(n)) with correct explanation

+ **10 pts** Part A: Best Approach O(m+log(n)) with correct explanation

✓ **+ 2 pts** **Part B: Mention Balanced BST or Heap or Skip List**

+ **1 pts** Part B: Mention Array or Linked List with in-place sorting, or unbalanced BST

+ **1 pts** Part B: Balanced BST or Heap or Skip list - Explanation correct except incorrect complexity

+ **0.5 pts** Part B: Insert into Array or Linked List without sorting

✓ **+ 2 pts** **Part B: Balanced BST or Heap or Skip List correct Explanation**

+ **1 pts** Part B: In-place sorting or unbalanced BST correct explanation

+ **0.5 pts** Part B: Sort array after inserting

+ **1 pts** Part B: Bonus for Skip List

In part a, why are you analysing a modified algorithm?

QUESTION 12

## Q12 14 pts

### 12.1 Q12-a 7 / 10

+ **3 pts** A correct explanation is provided

✓ **+ 3.5 pts** **One of the insertion (either left or right) is correct**

✓ **+ 3.5 pts** **Other insertion is also correct**

+ **1 pts** Another insertion is partially correct (and previous one is completely correct)

+ **2 pts** Partial Marks ( Idea is somewhat correct and/or issues in code) or algorithm is just a BST

+ **1 pts** Partial Marks ( Idea is in-correct and/or major issues in code)

+ **0 pts** Not Attempted or Incorrect

### 12.2 Q12-b 2 / 4

✓ **+ 2 pts** **Explicitly saying it will help in the finding.**

+ **2 pts** Explicitly saying it will help in the insertion

+ **2 pts** If skip-list is used in Question 11.

+ **1 pts** Partial marks

+ **0.5 pts** For a single word "YES", or equivalent

+ **0 pts** Not Attempted or Incorrect

QUESTION 13

## 13 Q13 8 / 8

✓ **+ 8 pts** **Correct**

+ **0 pts** Incorrect or not attempted

+ **2 pts** Out of place but O(n)

+ **4 pts** In place algorithm

+ **2 pts** Algorithm is O(n)

+ **1 pts** Complexity have been argued

+ **1 pts** Correctness has been argued

+ **0 pts** Click here to replace this description.

not complete

QUESTION 14

## 14 Q14 5 / 5

✓ **+ 1.25 pts** **Pass 1 Completely Correct**

✓ **+ 1.25 pts** **Pass 2 Completely Correct**

✓ **+ 1.25 pts** **Pass 3 Completely Correct**

✓ **+ 1.25 pts** **Pass 4 Completely Correct**

+ **0 pts** Incorrect/Not answered

QUESTION 15

## 15 Q15 5 / 5

+ **0 pts** Incorrect/Not Attempted

+ **3.5 pts** Correct without compressed trie and sorting

+ **4 pts** Correct, sorted but not compressed trie

+ **4.5 pts** Correct with minor errors

✓ **+ 5 pts** **Fully Correct**

QUESTION 16

## 16 Q16 5 / 5

✓ **+ 5 pts** **Correct**

+ **1 pts** Only written Theta(n)

+ **0 pts** Incorrect

QUESTION 17

## 17 Q17 5 / 5

+ **0 pts** Incorrect

✓ **+ 2 pts** **Correct Counter Example**

✓ **+ 3 pts** **Proper Explanation to the given counter example**

+ **5 pts** No counter example given but Proper argument made with justification

Please write your entry number and name on each sheet.

**Entry Number:** 2018MT10742 **Name:** ARPIT SAXENA

1 mark for writing the entry number and name neatly on all sheets

# COL 106 MAJOR EXAM

SEMESTER I 2019-2020

2 hours

**B**

***Please do not allow any bag, phone or other electronic device near you. Keep your ID card next to you on the desk. Max marks for questions are listed in []. Write answers in the provided space. Please justify your answers. There are marks for efficiency and analysis of algorithms.***

1. [2+4] What should be printed in the following pieces of codes. Write within the boxes.

(a)
```
// List precise output
class GrandParent {
  void print() { System.out.println("Grand Parent");}
}
class Parent extends GrandParent {
  Parent(int p) { System.out.println(p); }
  Parent() { this(0); }
  void print() { System.out.println("Parent");}
}
class Child extends Parent {
  Child(int p) { System.out.println("Child."+p); }
  public static void main(String arg[]) {
    GrandParent someone = new Child(10);
    someone.print();
  }
}
```

10
Child.10
Parent

(b)
```
// Characterize the output: what's printed in what order
class Node { Node le, ri; }
class Tree { Node root; }

class main {
  private static void traverse(Node n, int arg) {
    if(null == n) return;
    System.out.println(arg);
    traverse(n.le, arg+1); traverse(n.ri, arg+1);
  }
  public static void main(String arg[]) {
    Tree tree = new Tree();
    build(tree); // Assume that it populates tree
    traverse(tree.root, 0);
  }
}
```

This a pre-order traversal. It prints the height of the node whenever that node is conserved.

2. [3] Find the error(s) in the following code. Write within the box.

```
package AB;
class A {
  private    int   i;
  protected void set(int j) { i = j; }
}
class B extends A {
  private A pa;
  public  A pa()  { return pa; }
}
------
Import AB.*; // This is not in package AB
class C {
  B b1 = new B();
  B b2 = b1;
  b1.pa() = new A();   ← b1.pa() is not an Lvalue and can't be assigned to
  b2.pa().set(0);      ← class A's set is protected, so it is not available to non-child classes in another package.
}
```

*Proofs in Qn 3-5 are flawed. Point out the flaw(s) in each proof.*

3. [4] **Theorem:** The worst case complexity of quick sort with a randomly selected pivot is $O(n^2)$.

**Proof:** Suppose the pivot is the first element. This partitions the array into two parts: the pivot and the rest. The rest must be sorted next. Partition of $n$ items must take time = $kn+c$ with constant k,c for $n > n_0$

$\Rightarrow$ Thus the recurrence relation is $T(n) = kn + T(n-1)$, whose solution is $\Theta(n^2)$

because, $T(n)$ expands to $T(n) = kn + k(n-1) \ldots = \Theta(n(n-1)/2) = \Theta(n^2)$

The pivot partitions the array into 3 parts: ~~a subarray less t~~ elements less than the pivot, the pivot, and elements greater than the pivot. We need to ~~process~~ sort these independently. ~~Gee~~ Since 50% of the pivots are good pivots, at every other level, we'll have $T(n) = T(n/4) + T(3n/4) + kn \leq 2T(3n/4) + kn$

This gives $T(n) = O(n \log n)$

4. [4] **Theorem**: To search in an array of $n$ items, Binary search performs $n$ comparisons in the worst case.
**Proof** by induction. Denote the number of comparisons by C.
*Base case*: $C(1) = 1$. Binary search must make one comparison with the single item.
*Inductive hypothesis*: Assume $C(k) = k$.
*Inductive step: Prove* $C(k+1) = k+1$.
An algorithm must search through all $k+1$ elements. Since searching through any $k$ of them takes time $C(k)$, and additionally the remaining item must also be compared (lest it is what we are looking for), $C(k+1) = k+1$.

Binary search doesn't need to look through all k of the elements in a k+1 sorted array to find what it is looking for. The inductive step disregards how the binary search algorithm works.
It will compare the middle of the k+1-elements and then a $\lfloor \frac{k+1}{2} \rfloor$ length array would remain to be searched through.
So, the recurrence is, $C(k+1) = C\left(\lfloor \frac{k+1}{2} \rfloor\right) + 1$

This not prove the inductive hypothesis. It solves to $C(n) = O(\log n)$

5. [3] **Theorem**: 1 is the largest natural number.
**Proof**: If 1 is not the largest natural number, let it be $n$.
Naturally, $n^2$ is also a natural number AND $n^2 \leq n$, since $n$ is the largest one
$\Rightarrow n^2 - n \leq 0$ and hence $n(n-1) \leq 0$,
$\Rightarrow (n-1) \leq 0$, since $n$ being natural is strictly greater than 0.
$\Rightarrow n = 1$.

The proof says $n^2 \leq n$, since $n$ is the largest natural number. While true under the assumption, we also know that for all natural numbers greater than 1, $n^2 > n$.
So, we have a wrong statement there, which shows that the proof is wrong.

6. [5] The *lower bound* on the time complexity of comparison-based sorting is $\Omega(n \log n)$. Explain the meaning of this statement. Why does it use $\Omega$?

The statement means that given a comparison-based sorting, it runs asymptotically worse or the same as $n \log n$, i.e., $O(n \log n)$ is the best worst-case time it can have.

$f(n) = \Omega(g(n))$
$\Rightarrow \exists k > 0, \exists n_0 \in \mathbb{N}, f(n) \geq k g(n) \quad \forall n \geq 0$

So, $\Omega$ denotes the asymptotic lower bound.

Please write your entry number and name on each sheet.
**Entry Number:** 2018MT10742 **Name:** ARPIT SAXENA

7. [12] Prove that the **average** number of comparisons required to merge two sorted arrays of $n$ items each is less than $1.5n+c$, c is a constant. Next, show that the average number of comparisons required to Merge-sort $n$ items is $n \log n + \Theta(n)$ on average. Please provide detailed analysis. Extra credit for a tighter bound.

When merging two sorted arrays, one needs to keep comparing till one of the arrays becomes empty. Then the remaining part of the non-empty array can be carried over without any comparison.

Since 1 comparison reduces length of one array by 1,
if the number of items remaining in the non-empty array is $k$,
$2n-k$ comparisons have been made.

$k$ ranges from 1 (not 0, since 1 item will remain) to $n$ (if the other array is all smaller than the first element of this one)

So, average no. of comparisons $= \frac{1}{n}\sum_{k=1}^{n}(2n-k) = \frac{1}{n}\left\{2n^2 - \frac{n(n+1)}{2}\right\}$

$$= 2n - \frac{n}{2} - \frac{1}{2}$$

$$= 1.5n - 0.5$$

$1.5n \geq$ Average number of comparisons $\leq 1.5n$
$\Rightarrow$ Avg. comparisons $= \Theta(n)$

Suppose average comparisons required to merge sort $n$ items is $T(n)$.

Then $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil)$

Assuming $\lfloor \frac{n}{2} \rfloor$ same as $\lceil \frac{n}{2} \rceil$ everywhere,

$$T(n) = 2T\left(\frac{n}{2}\right) + 1.5 \times \frac{n}{2} + c$$

$$= 4T\left(\frac{n}{4}\right) + 2 \times 1.5 \times \frac{n}{4} + 1.5 \times \frac{n}{2} + 3c$$

$$= \cdots = 2^k T\left(\frac{n}{2^k}\right) + 1.5 \times \frac{n}{2} \times k + (2^k - 1)c$$

When $k$ is smallest number s.t. $2^k \geq n$

i.e. $2^k \geq n$ but $2^{k-1} \not\geq n \Rightarrow 2^k = \Theta(n) \Rightarrow k$

$\Rightarrow k \geq \log n$ and $k < \log n + 1$

Then $T(n) = 1.5 \times \frac{n}{2} k + (2^k - 1)$

$$= 1.5 \times \frac{n}{2} \times \log n + (2^k - 1)$$

$$= 0.75\, n \log n + \Theta(n)$$

8. [8] Given a box of look-alike nails, find the one that is lighter than the rest (there is only one). The weight of the nails are not given, but you have an electronic scale, which can give the weight of any number of nails at a time. It charges only ₹1 per weighing. Provide the least costly algorithm to complete your task (in the worst case). Be precise. Argue why no one could do better (partial credit if it is asymptotically best). You may count nails for free.

Take two nails. If same weight, then we know weight of the majority of nails, (say w) else found lighter nail.

The algorithm ~~Given~~ (n nails) :

If n is even, divide nails into two equal parts and ~~weigh~~.
weight first one. If it's weight is less than weight of $n/2$ * w
weight we found, search there. Else search in second part.

If n is odd, ~~put 1 nail aside and make~~ two parts of ~~$\frac{n-1}{2}$~~ $\frac{n-1}{2}$ and $\frac{n+1}{2}$ nails.

If weight of part ~~with~~ with $\frac{n+1}{2}$ nails is less than $w \times \frac{n-1}{2}$, then search there. Else search in other part.

This takes O(log n) time.

~~To~~ To prove this is optimal, we can consider a decision tree for this algorithm. Each of the ~~decision~~ n nails must appear in one of the leaves.
So, height of tree is O(log n) and we can't do better than that.
Note the decision tree of our algo. has each nail only once in the leaves, ~~as once discarded, we never pick~~. So, the number of leaves is maximum and hence ~~this~~ ~~no~~ one can't do better than this.

9. [8] Comparable elements in a given array are initially in increasing order starting at index 0, but after some index $k$ they are in decreasing order until index $n-1$, i.e., value[$k$+1] < value[$k$]. Find $k$. Analyze your algorithm.

```
int k=0
for (int i=0; i<n-1; i++)
    if (value[i].compareTo(value[i+1]) > 0)
        k=i;
        break;
if (i==n-1) k=n-1;
```

The algorithm terminates as i increases by 1 in each step from 0 to n-2.
When it terminates, either k=n-1 or value[k] > value[k+1].
The loop maintains the invariant that value[0..i] is in increasing order. (assuming no repeated elements)

So, if k=n-1, value[0..n-1] is in ascending order and value[k..n-1] is in descending order

Otherwise, value[0..k] is in ascending order and value[k] > value[k+1] and so, ~~all~~ by info. given. values decrease from there.

This runs in O(n) time, since the loop runs at most n times, doing O(1) work in each iteration.

Please write your entry number and name on each sheet.
**Entry Number:** 2018MT10742 **Name:** ARPIT SAXENA

10. [8] Intersection is a critical operation for sets. Propose a general data structure for sets that allows efficient computation of intersection, given two sets. Note that two elements of a sets are not necessarily comparable. Analyze your algorithm's worst case time complexity. Full marks for reaching O($n$) – expected or guaranteed – for two sets of size $n$ each.

We can store the elements of the set in a sorted array. To find the intersection, assume the two sorted arrays are a and b and we need to put intersection in res

```
int i=0,j=0;
while(i<n && j<n) :
    if (a[i] == b[j])
        res.push(a[i]);
        i++; j++;
    elif (a[i] < b[j])
        i++;
    else:
        j++;
```

Since a and b represent sets, they contain distinct elements. We maintain two pointers, similar to merge, and when a[i] is same as b[j] we push the element to res.

If a[i] < b[j], then a[i] < b[j+1, ..] and we have already seen elements indices less than j, i.e. b[:j-1] < a[i].

Similarly, if a[i] > b[j].

For time complexity, we note that atleast one of i and j increases by 1 in each iteration.

So, $2n-(i+j)$ decreases by 1 in each iteration and loop exits before it reaches zero. So, O(n) steps are taken.

11. **(a)** [12] Analyze the following algorithm to find each key $k$ in an AVL tree, s.t. $a <= k <= b$, given $a$ and $b$. The output must be in an increasing order. Provide the worst-case asymptotic time complexity in terms of $n$, the number of keys in the BST and $m$, the actual number of keys in the range $[a:b]$ in the tree. State any assumptions.

```
find(node, k):
if node.key in range [a:b]:
    insert_into_output(node.key)
    find(node.left, k)
    find(node.right, k)
elseif node.key < a:
    find(node.right, k)
else:
    find(node.left, k)
```

~~Correctness:~~
~~The algorithm seems to be incorrect as it is. If we swap insert into output and find(node.left, k) statements, then it works fine. Then we can assume insert into output simply appends in O(1) time.~~ Else, insert into output would had to insert into the sorted array taking O(n) time. (assuming for now, output is as a sorted array)

So, with the modified algo:

If node.key is in range [a:b], then we have to search both its left and right subtrees. find on left adds all keys in range [a:b] in sorted order into output (can be formally taken by induction), then node.key is inserted and find is called for right subtree. If key < a, then potential keys in [a:b] are in right subtree. Similarly for right.

~~Time complexity of the modified algo is O(m+log n), linear in m as it takes inserts all m keys into the output and log n for traversal of the tree.~~

For the algorithm as given in question, it will essentially have to perform an insertion sort on the m keys found, and thus takes $O(m^2 + \log n)$ time.

{ this is assuming output is maintained as a sorted array. See next ans for a better output structure. }

**(b)** [4] What data structure would be a good choice for the output in the previous question? Explain.

output can be maintained as an AVL tree as inserting a key into it when it already has n keys would take O(log n) time. Then we can simply perform an in-order traversal on the tree to get items in sorted order. The time for previous algo. then becomes O(m log m + log n)

12. **(a)** [10] A threaded BST (This is not about concurrency.) is a binary search tree that stores in each node without a right child, a reference to its in-order successor in place of the right *null* reference. Similarly, if the left reference would be *null* in a node, it stores a reference to the in-order predecessor instead. Two boolean flags at each node indicate for each reference if it is a tree references or a thread reference. For simplicity, you may assume separate references, *leftthread* and *rightthread*, are employed when *left* and *right* are null, respectively. Provide an algorithm to insert a key in a threaded BST. (You do not need to balance it.)

```
def insert(node, k):        # insert k into tree rooted at node?
    if node is null, return
    if (k < node.key):
        if (node.left == null):
            node.left = new node(k); node.leftthread = no
            node.left.rightthread = node; node.left.leftthread = node.leftthread;
            node.leftthread = null
        else:
            insert(node.left, k)
    else:
        if node.right == null:
            node.right = new node(k);
            node.right.leftthread = node; node.right.rightthread = node.rightthr
            node.rightthread = null;
        else:
            insert(node.right, k)
```

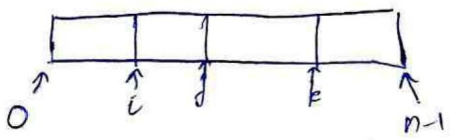
**(b)** [4] Is there any advantage of using *threaded* AVL tree in the problem in Qn 11? Explain.

There is an advantage of using threaded AVL tree in Q11's output. key output which is maintained in AVL tree. So, given a node in the output, we will be able to find the next node to it in lesser time.

Please write your entry number and name on each sheet.
**Entry Number:** 2018MT10742 **Name:** ARPIT SAXENA

13. [8] Explain how you may perform in-place bucket sort (stability not required) of $n$ items in an array in $O(n)$ time into 3 buckets. Detailed pseudo-code not required, but explanation needs to be clear.

This is similar to partitioning into 3 partitions




invariant
elements in 0..i-1 denote elements in first bucket,
i..j-1 denote elements in second bucket
k+1..n-1 denote elements in third bucket
and we haven't processed elements in
j..k

Initially, i=0, j=0, k=n-1
we check at $j^{th}$ place each item, say the element is a
if it belongs to the first bucket,
swap i and j, increment both i and j
if it belongs to second bucket,
increment j
else:
swap j and k; decrement k.
{Terminate when j>k}

The algo is correct as it maintains invariant throughout and when j=k+1, we are done. k-j+1 is initially n and decreases by 1 in each iteration, algo stops when it hits zero. So, it takes O(n) time

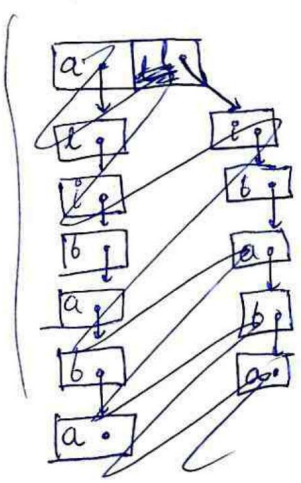
14. [5] Give the steps of radix-sorting the following list of integers, one digit per step, i.e., show the state of the list after each bucket-sort.

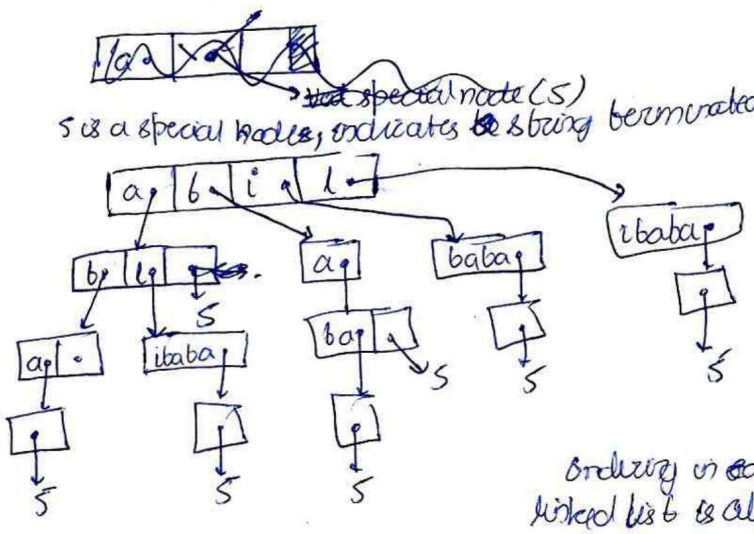
| | | | | |
|---|---|---|---|---|
| 3241 | 3241 | 2219 | 3122 | 0134 |
| 3241 | 3241 | 1321 | 0134 | 1321 |
| 2341 | 2341 | 3122 | 2219 | 1441 |
| 2219 | 1321 | 0134 | 3241 | 2219 |
| 3122 | 1441 | 3241 | 3241 | 2341 |
| 1321 | 0134 | 3241 | 1321 | 3122 |
| 1441 | 3122 | 2341 | 2341 | 3241 |
| 134 | 2219 | 1441 | 1441 | 3241 |

15. [5] A suffix tree is useful in pattern matching. It is a compressed trie that stores all suffixes of a given string. For example, the suffix trie of **alibaba** stores *alibaba, libaba, ibaba, baba, aba, ba,* and *a*. Draw this suffix trie. Represent each node as a sorted linked list (of values present at that node). Mention your ordering scheme.

Rough
alibaba
libaba
ibaba
baba
aba
ba
a

a
aba
alibaba
ba
baba
ibaba
libaba






Ordering in each node's linked list is alphabetical.

16. [5] What is the **average** time complexity (as a function of $n$) of breadth-first search in a graph representing a proper mesh without boundaries comprising $n$ triangles. (Such a mesh has adjacent triangles on each side of every triangle.) Each node in this graph represents a triangle and an edge represents that those triangles are adjacent.

since this is a proper mesh, each triangle has exactly three neighbours.

$\Rightarrow deg(v) = 3 \quad \forall v$

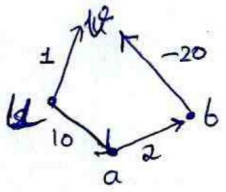
since $\sum deg(v) = 2m$, where $m$ is number of edges in the graph

$3n = 2m \Rightarrow m = 1.5n$ or $m = O(n)$

Breadth-first search in a graph with $n$ nodes and $m$ edges takes $O(m+n)$ time.

since $m = O(n)$, here, breadth-first search has on average time complexity $O(n)$.

17. [5] Consider computing the shortest path from node $u$ to $v$ in a directed graph using Dijkstra's algorithm. Assume that the graph has no (directed) cycle. Show that Dijkstra's algorithm fails to work if negative weight edges exist, because its greedy step does not guarantee that once a node is brought into the cluster, its shortest path from $u$ is found.




In the example graph, when we begin from $u$, we first add $v$ to the cluster that initially only contained $u$. Thus, the algorithm claims its distance is 1, however we see there's a path $u \to a \to b \to v$ with weight $-8$, which is clearly the shortest.

Dijkstra's algorithm ensures that whenever a node $v$ is brought into the cluster, for all nodes $w$ connected to cluster, $d(u,v) \leq d(u,w)$

If there were only non-negative edge weights, a path from any other node $w$ just outside the cluster would have a path length shorter than one we've found. (as $d(w,v) \geq 0$)

However, if -ve edge weights existed, this can't be guaranteed as $d(w,v)$ might be less than a

Please write your entry number and name on each sheet.

**Entry Number:** 2018MT10742 **Name:** ARPIT SAXENA

$$T(m) = \begin{cases} T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(1) \\ T(n/2) + O(1) \end{cases}$$

~~$O(m+n)$~~

$O(\log n + m)$