

BACHELOR OF COMPUTER SCIENCE
FACULTY/SCHOOL OF COMPUTER SCIENCE
BINA NUSANTARA UNIVERSITY
JAKARTA

ASSESSMENT FORM

Course: COMP6879051 - Data Structure

Method of Assessment: Case Study

Semester/Academic Year : 2/2022-2023

Name of Lecturer : ANANG PRASETYO, S.Kom, M.Kom.

Date : 6 Juni 2023

Class : LC95

Topic : Review II

| | |
|--------------|--------------------------------|
| Name: | Michael Geraldin Wijaya |
|--------------|--------------------------------|

Student Outcomes:

SO 2 - Mampu merancang, mengimplementasikan, dan mengevaluasi solusi berbasis komputasi untuk memenuhi serangkaian persyaratan komputasi dalam konteks ilmu computer

Able to design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of computer science

LObj 2.2 - Mampu mengimplementasikan solusi berbasis komputasi untuk memenuhi serangkaian persyaratan komputasi tertentu dalam konteks ilmu computer

Able to implement a computing-based solution to meet a given set of computing requirements in the context of computer science

| No | Assessment criteria | Weight | Excellent (85 - 100) | Good (75-84) | Average (65-74) | Poor (0 - 64) | Score | (Score x Weight) |
|---|---|-------------|--|---|--|--|-------|------------------|
| 1 | Ability to identify the problems to find the solution | 10 % | Able to identify both the input and output in fully detail for the problem | Able to identify both the input and output but in less detail for the problem | Able to identify both the input and output but not in clear detail for the problem | Able to identify only the input or output for the problem | | |
| 2 | Ability to design an algorithm for the problem | 40 % | Able to design an algorithm for the problem in full detail | Able to design an algorithm but not have full detail | Able to design an algorithm for the problem but not have clear process | Able to design an algorithm but cannot be implemented in the problem | | |
| 3 | Ability to solve the problem | 50 % | Able to solve 76-100% the problem with fully functional feature | Able to solve 51-75% the problem and lack some features | Able to solve 26-50% of the problem and lack some features | Able to solve less than 25% of the problem | | |
| Total Score: $\sum(\text{Score} \times \text{Weight})$ | | | | | | | | |

Remarks:

ASSESSMENT METHOD

Instructions

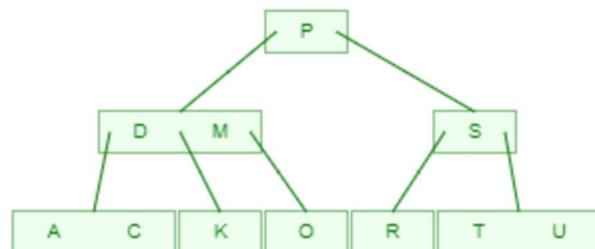
- This case study is individual with 1 week processing
- Design an algorithm in pseudocode/code to solve the problem and write down the algorithm
- If in the case study does not have specific instruction about the algorithm that must be used, it means that students can determine the best algorithm to solve the given problem
- Report will be submitted in pdf format to binusmaya

Note for Lecturers:

- The lecture notifies this case study to the student from Week 9.
- Deadline for the case study is one week after the lecture post it on binusmaya
- The student should submit the report to binusmaya no later than deadline
- If the student do plagiarism, their score for this case study will be zero

1. B-TREE (ORDER 3)

Given a 2-3 Tree below



- (5%) Proceed **insertion** on that tree above with these data: E,I,L,G,X
- (10%) Proceed **deletion** after doing the insertion on point a with these order: O,T,D,P,X,S,I,C,M,K,E

2. RED BLACK TREE

- (10%) Create a Red Black Tree using the following sequence: 41,22,5,51,48,29,18,21,45,3
- (15%) Try implementing (a) Red Black Tree insertion in C Program and print those datas with InOrder Traversal

```
Inorder Traversal of Created Tree
3 5 18 21 22 29 41 45 48 51
-----
Process exited after 0.1592 seconds with return value 0
Press any key to continue . . .
```

3. AVL TREE

- a. Into empty AVL Tree:
 - i. (10%) Insert the following values: 6, 27, 19, 11, 36, 14, 81, 63, 75
 - ii. (10%) Delete the following values: 14, 75, 36, 19, 11
- b. (30%) Write a program to insert, delete and print datas from AVL Tree insertion in C Program

```
1. Insertion
2. Deletion
3. Traversal
4. Exit
Choose:
```

- i. Insertion

In this menu, the program will asked the value that the user want to insert into AVL tree

```
1. Insertion
2. Deletion
3. Traversal
4. Exit
Choose: 1
Insert: 6
```

- ii. Deletion

In this menu, the program will asked the value that the user want to delete from AVL tree. If the value is found in the tree, than it will be deleted otherwise the program gives message 'data not found'

```
1. Insertion
2. Deletion
3. Traversal
4. Exit
Choose: 2
Delete: 14
Data Found
Value 14 was deleted
```

```
1. Insertion
2. Deletion
3. Traversal
4. Exit
Choose: 2
Delete: 7
Data not found
```

iii. Traversal

In this menu, the program will print all datas from AVL tree in preorder, inorder and postorder

```
1. Insertion
2. Deletion
3. Traversal
4. Exit
Choose: 3
Preorder: 19 11 6 14 36 27 75 63 81
Inorder: 11 6 14 19 36 27 75 63 81
Postorder: 11 6 14 36 27 75 63 81 19
```

iv. Exit

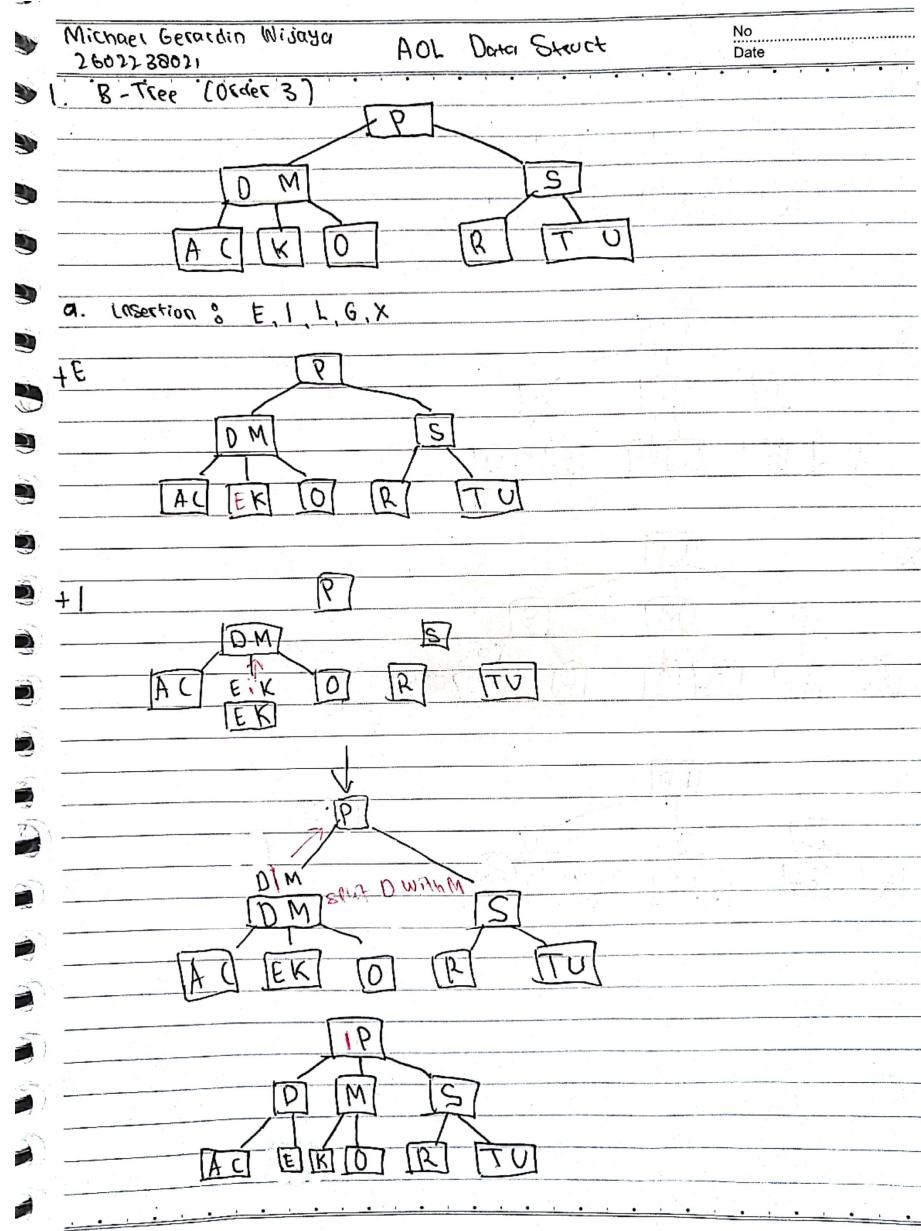
```
1. Insertion
2. Deletion
3. Traversal
4. Exit
Choose: 4
Thank you
```

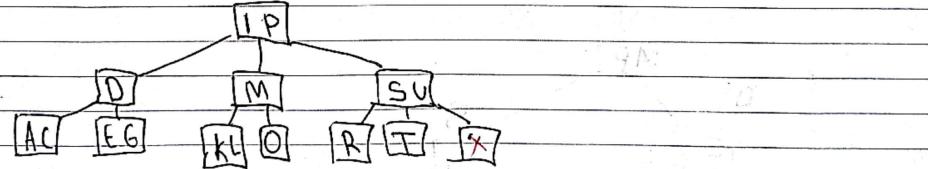
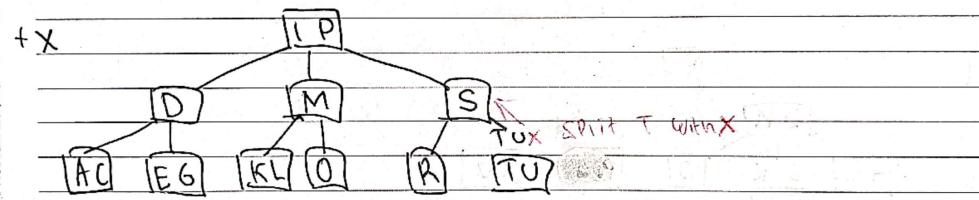
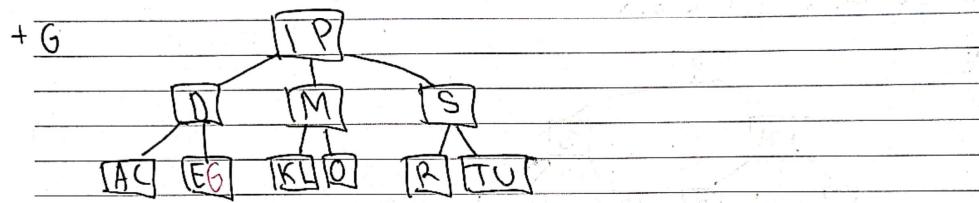
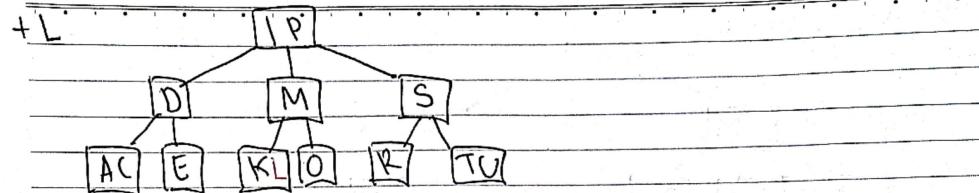
4.

- a. (5%) What are the difference between binary tree and b-tree?
- b. (5%) What are the difference between AVL Tree and Red Black Tree? In what case would you want to use a red black tree over an AVL tree and vice versa?

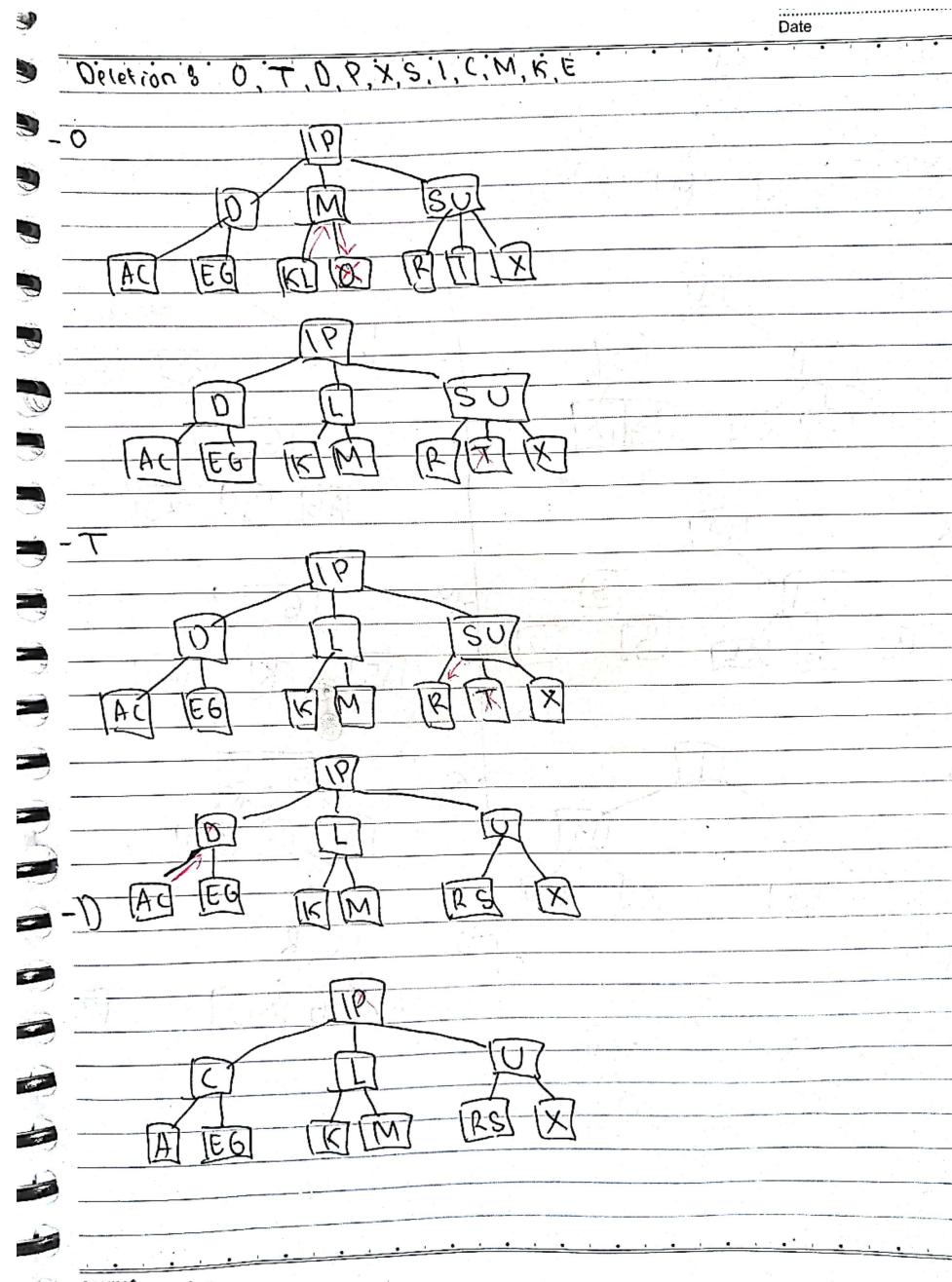
Answer

1. A) Insertion B-Tree

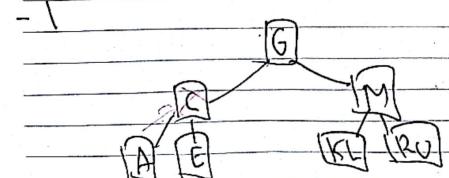
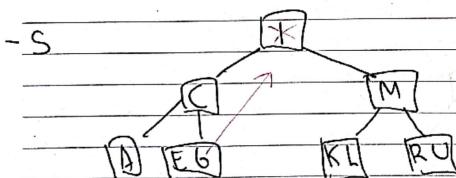
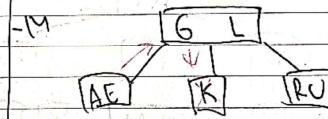
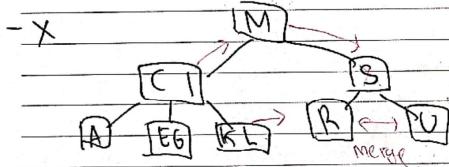
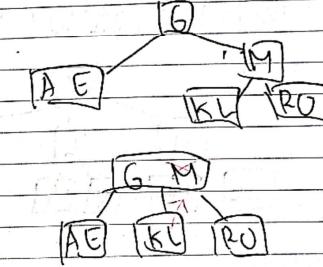
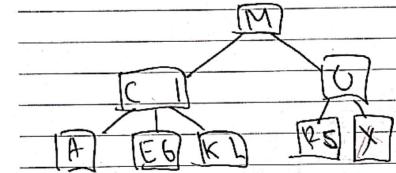
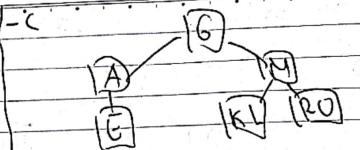
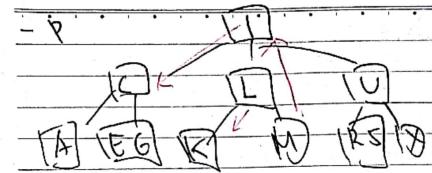




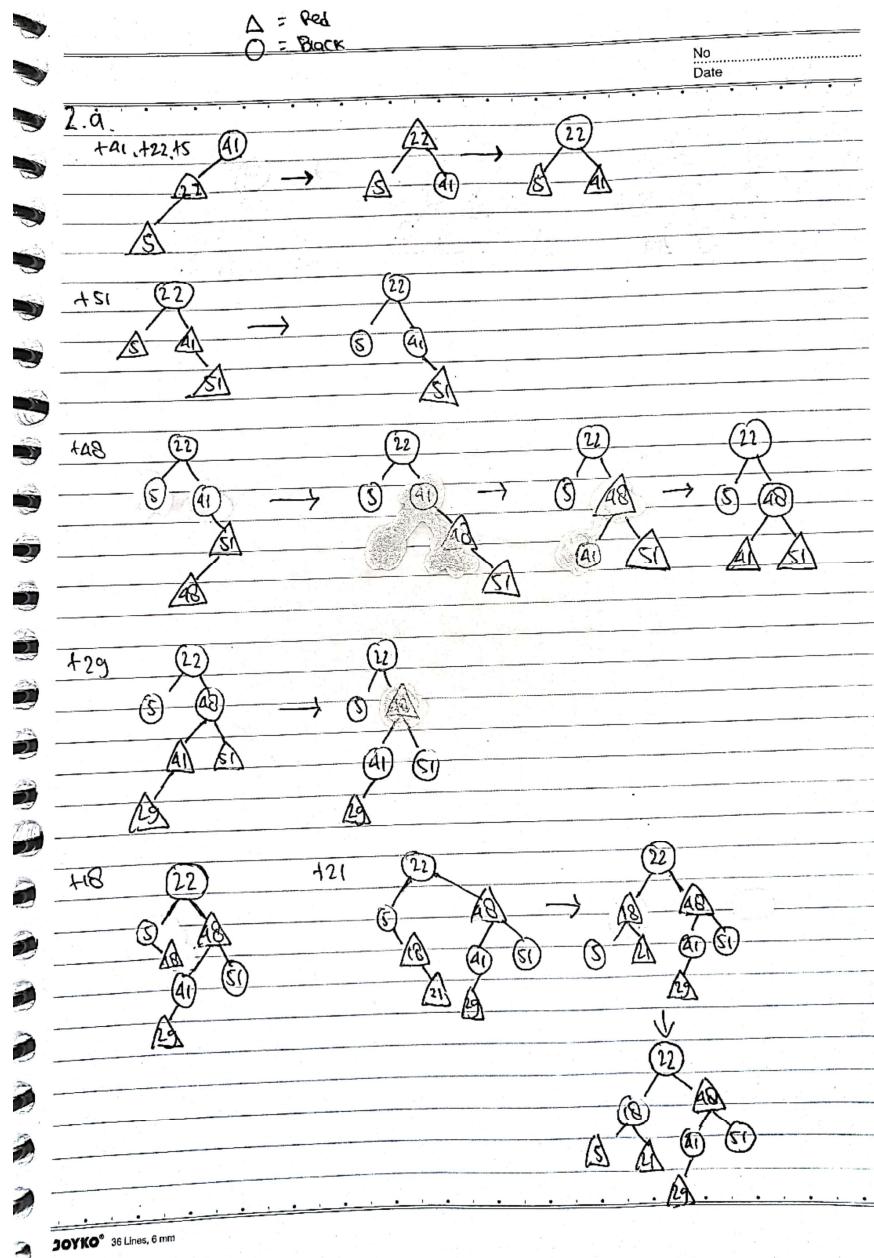
B) Deletion B-Tree



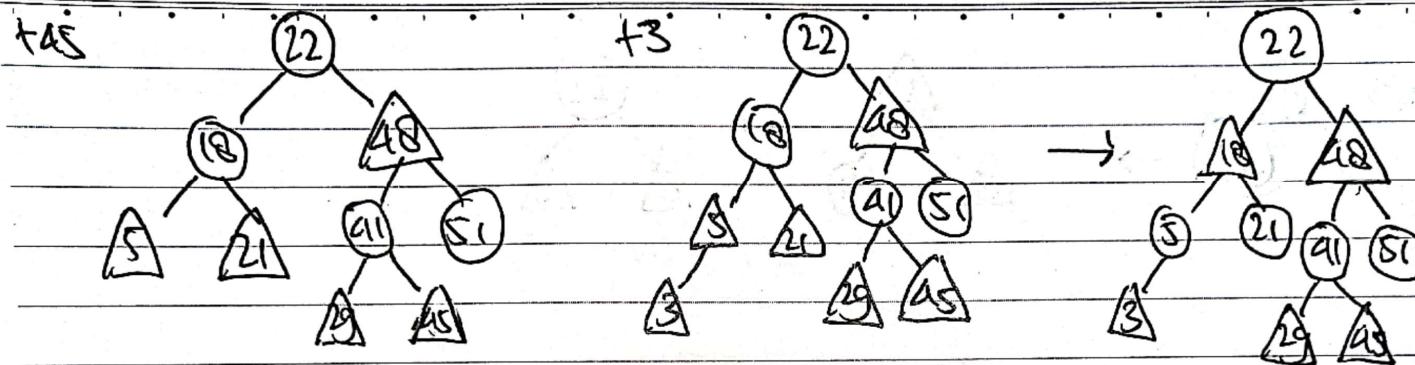
No
Date



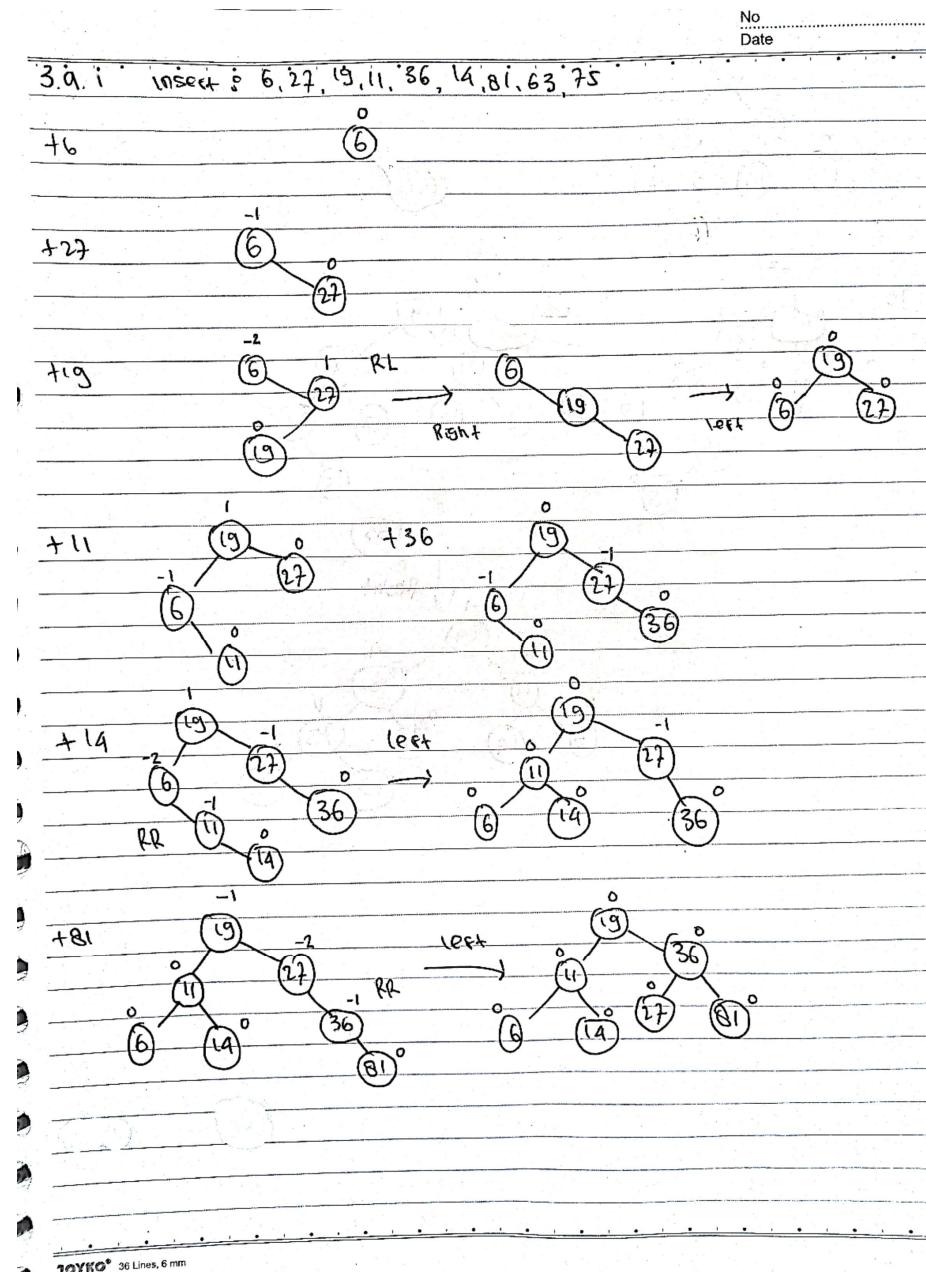
2. A) Insertion Red Black Tree



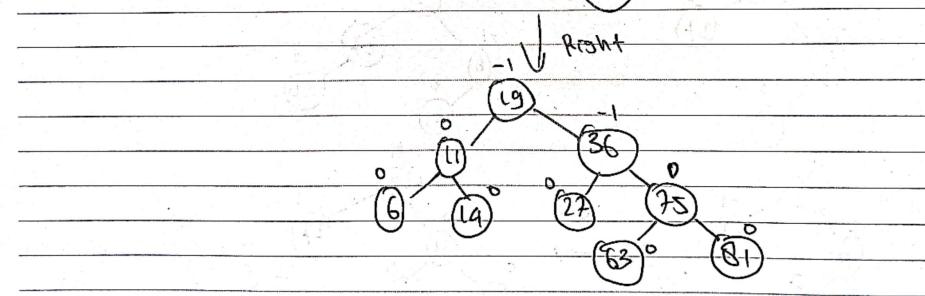
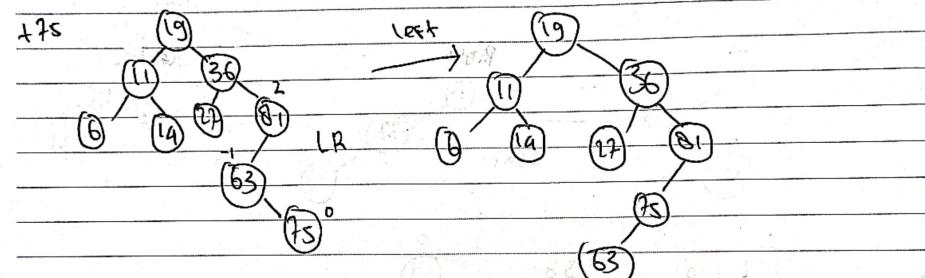
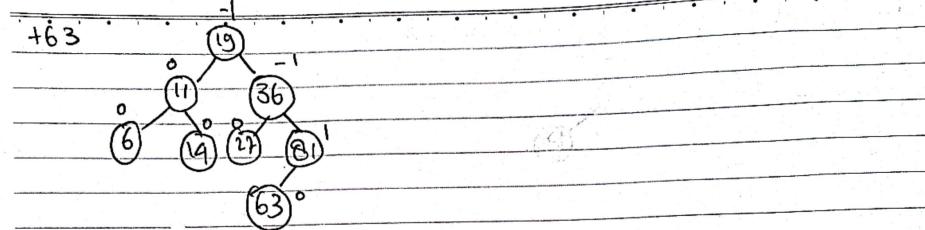
No.....
Date



3. A) Insertion (i) and Deletion (ii) AVL Tree

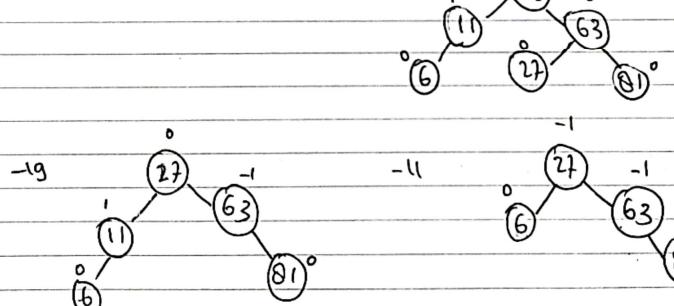
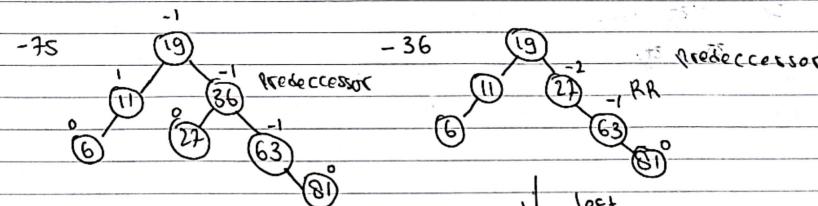
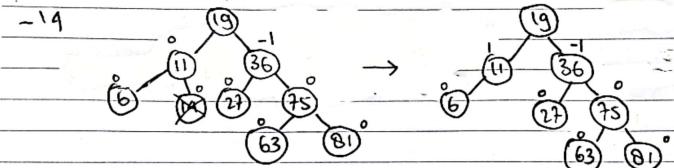


No
Date



No
Date

ii) Delete 81, 4, 75, 36, 19, 11



Annotations:
- 19: A label above the node 19.
- 11: A label above the node 11.
- 63: A label above the node 63.
- 81: A label above the node 81.

4. A) Perbedaan antara binary tree dan B-tree:

Binary Tree:

- Binary tree adalah struktur data hierarkis di mana setiap simpul memiliki maksimal dua anak simpul, yaitu anak kiri dan anak kanan. Jumlah anak pada setiap simpul dibatasi maksimal dua.
- Binary tree lebih sederhana dalam strukturnya dibandingkan dengan B-tree. Karena hanya memiliki dua anak pada setiap simpul, binary tree memiliki lebih sedikit kompleksitas dalam implementasinya.
- Binary tree umumnya digunakan untuk menyimpan data dalam memori utama, seperti RAM, dan cocok untuk jumlah data yang relatif kecil.
- Ketidakseimbangan dapat terjadi dalam binary tree. Jika operasi penyisipan atau penghapusan dilakukan tanpa menjaga keseimbangan, maka binary tree bisa menjadi tidak seimbang (unbalanced), yang dapat mengakibatkan kinerja yang buruk dalam operasi pencarian.

B-tree:

- B-tree adalah struktur data yang dapat mengandung lebih dari dua anak simpul. Setiap simpul dalam B-tree dapat memiliki banyak anak, tergantung pada batasan derajat minimum dan maksimum yang ditentukan.
- B-tree dirancang untuk menyimpan dan mengakses data dalam media penyimpanan sekunder, seperti hard disk. B-tree dirancang khusus untuk mengoptimalkan operasi I/O dengan meminimalkan jumlah akses ke media penyimpanan.
- B-tree memiliki struktur yang lebih kompleks dibandingkan dengan binary tree. Karena memiliki beberapa anak pada setiap simpul, B-tree memerlukan pengaturan dan algoritma yang lebih rumit dalam implementasinya.
- B-tree dirancang untuk tetap seimbang secara otomatis. Setiap simpul dalam B-tree harus memenuhi persyaratan derajat minimum dan maksimum yang menjaga keseimbangan secara efisien. Ini memungkinkan B-tree memiliki tingkat keefisienan dan kinerja yang tinggi dalam operasi pencarian dan pengurutan pada jumlah data yang besar.

B) Perbedaan antara AVL Tree dan Red Black Tree:

AVL Tree:

- AVL Tree adalah jenis pohon biner yang seimbang secara sempurna. Setiap simpul dalam AVL Tree memiliki perbedaan tinggi maksimal dua antara anak kiri dan anak kanannya.
- AVL Tree mempertahankan keseimbangannya melalui rotasi dan penyisipan atau penghapusan node. Jika setelah operasi tersebut, terjadi ketidakseimbangan pada AVL Tree, maka dilakukan rotasi untuk mengembalikan keseimbangan.
- Operasi dasar pada AVL Tree adalah rotasi, baik rotasi kiri maupun rotasi kanan. Rotasi digunakan untuk mempertahankan keseimbangan saat melakukan operasi penyisipan atau penghapusan simpul.

- AVL Tree memiliki kinerja yang baik dalam operasi pencarian karena keseimbangannya yang ketat. Tinggi pohon AVL Tree selalu berada dalam kisaran logaritmik, sehingga meminimalkan waktu yang dibutuhkan untuk mencari simpul tertentu.
- AVL Tree membutuhkan lebih sedikit ruang penyimpanan daripada Red Black Tree karena tidak ada informasi tambahan yang perlu disimpan. AVL Tree hanya menyimpan informasi mengenai simpul dan nilai yang terkait.

Red Black Tree:

- Red Black Tree adalah jenis pohon biner yang tidak seimbang secara sempurna, tetapi mematuhi beberapa aturan atau properti tertentu. Properti utama Red Black Tree adalah memastikan bahwa setiap jalur dari akar ke daun mengandung jumlah simpul hitam yang sama.
- Red Black Tree menggunakan rotasi dan pewarnaan simpul sebagai operasi dasar untuk mempertahankan propertinya. Selain rotasi untuk menjaga keseimbangan, Red Black Tree memberikan atribut warna (merah atau hitam) pada setiap simpul untuk memastikan bahwa properti jumlah simpul hitam terpenuhi.
- Operasi pencarian pada Red Black Tree tidak secepat AVL Tree karena tidak memiliki keseimbangan yang ketat. Red Black Tree lebih memprioritaskan keseluruhan struktur dan menjaga properti-propertinya, yang mengorbankan sedikit waktu operasi pencarian.
- Red Black Tree memiliki kinerja yang baik dalam operasi penyisipan dan penghapusan. Karena tidak terlalu fokus pada keseimbangan yang ketat, Red Black Tree lebih fleksibel dalam mempertahankan struktur dan propertinya saat melakukan operasi penyisipan dan penghapusan simpul.
- Red Black Tree memerlukan sedikit ruang penyimpanan tambahan untuk menyimpan atribut warna pada setiap simpul. Meskipun demikian, penambahan ruang penyimpanan ini relatif kecil dan tidak signifikan dalam skala yang besar.

Dalam pemilihan AVL Tree atau Red Black Tree, pertimbangkan:

- Jika memerlukan operasi pencarian yang sering dan operasi penyisipan atau penghapusan yang jarang, AVL Tree dapat menjadi pilihan yang baik karena operasi pencarinya lebih cepat.
- Jika memiliki persyaratan penyisipan atau penghapusan yang lebih sering daripada operasi pencarian, Red Black Tree dapat menjadi pilihan yang lebih baik karena kinerjanya dapat lebih baik dalam situasi tersebut.
- Jika efisiensi ruang penyimpanan adalah faktor yang penting, Red Black Tree cenderung lebih efisien karena memerlukan lebih sedikit ruang penyimpanan untuk menyimpan informasi tambahan seperti warna simpul.

NOTE: Untuk Program(.cpp) nomor 2 (B) dan 3 (B) itu berada pada zip yang dilampirkan.