LAPORAN PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA
JOBSHEET 14

NAMA = RAKAGALI RESDA KRISANDI PUTRA

KELAS = TI -1B / 19

NIM = 244107020136

Percobaan 1

Hasil percobaan

```
Daftar semua mahasiswa (in order traversal):
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85

Pencarian data mahasiswa:
Cari mahasiswa dengan ipk: 3.54 : Ditemukan
Cari mahasiswa dengan ipk: 3.22 : Tidak ditemukan

Daftar semua mahasiswa setelah penambahan 3 mahasiswa:
InOrder Traversal:
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160311 Nama: Dewi Kelas: A IPK: 3.72
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85

PreOrder Traversal:
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160311 Nama: Dewi Kelas: A IPK: 3.72

PostOrder Traversal:
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160311 Nama: Dewi Kelas: A IPK: 3.72
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57

Penghapusan data mahasiswa:
jika 2 anak, current =
NIM: 244160311 Nama: Dewi Kelas: A IPK: 3.72

Daftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160311 Nama: Dewi Kelas: A IPK: 3.72
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85
```

Peertanyaan

14.2.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding

binary tree biasa?

pencarian dapat dilakukan secara **terarah** (hanya ke kiri atau kanan), **tanpa harus memeriksa semua node**, seperti pada binary tree biasa.
Ini membuat operasi pencarian pada BST rata-rata memiliki kompleksitas **O(log n)** (lebih cepat), sedangkan pada binary tree biasa tetap **O(n)** karena harus traversal semua node.

pencarian data bisa dilakukan dengan **membuang setengah kemungkinan** pada setiap langkah (mirip binary search pada array).
Pada binary tree biasa, pencarian harus mengecek semua node (linear), sedangkan di BST bisa langsung ke kiri atau kanan, sehingga **waktu pencarian rata-rata lebih cepat**, biasanya O(log n).

2. Untuk apakah di class Node, kegunaan dari atribut left dan right?

- left: anak kiri dari node saat ini (berisi data yang lebih kecil).
- right: anak kanan dari node saat ini (berisi data yang lebih besar).

Keduanya membentuk **struktur pohon** yang memungkinkan traversal, pencarian, dan manipulasi tree secara efisien.

 left: Menyimpan referensi ke anak kiri (node yang nilainya lebih kecil).

 right: Menyimpan referensi ke anak kanan (node yang nilainya lebih besar).

3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?

root adalah referensi ke node **paling atas** dalam tree, yaitu tempat semua operasi (tambah, cari, hapus) dimulai. Tanpa root, tidak ada titik awal untuk menjelajahi pohon.

b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?

Ketika objek BinaryTree pertama kali dibuat, maka root di-set ke null karena pohon masih kosong.

4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?

 Node baru akan **langsung dijadikan root**.

 Tidak ada perbandingan yang dilakukan karena belum ada node lain.

5. Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah ini. Jelaskan

secara detil untuk apa baris program tersebut?

- parent = current;
  Menyimpan node sekarang sebagai **induk (parent)** untuk referensi saat menambah node baru.
- if (mahasiswa.ipk < current.mahasiswa.ipk)
  Menentukan arah traversal: jika IPK mahasiswa baru **lebih kecil**, maka pindah ke **subtree kiri**.

- current = current.left;
  Melanjutkan traversal ke anak kiri.
- if (current == null)
  Menandakan bahwa posisi kosong ditemukan, dan node baru bisa ditambahkan di situ.
- parent.left = newNode; return;
  Menambahkan newNode sebagai anak kiri dari parent, lalu keluar dari loop.
- Blok else melakukan hal yang **sama tetapi ke subtree kanan**, jika IPK lebih besar atau sama.

6. Jelaskan langkah-langkah pada method delete() saat menghapus sebuah node yang memiliki dua anak. Bagaimana method getSuccessor() membantu dalam proses ini?

Langkah-langkah saat menghapus node dengan dua anak:

1. Temukan node yang akan dihapus (current) dan node induknya (parent).
2. Cek bahwa current.left != null && current.right != null.
3. Panggil method getSuccessor(current) untuk mencari **pengganti yang cocok**, yaitu node paling kecil dari subtree kanan (current.right).
4. **Salin data dari successor** ke node yang akan dihapus (current).
5. Hapus node successor dari posisi lamanya.

Fungsi getSuccessor() membantu dengan:

- Menemukan node pengganti yang **menjaga aturan BST tetap valid**.
- Menjamin bahwa posisi baru pengganti akan membuat subtree tetap urut.
- Menghindari inkonsistensi saat node memiliki 2 anak, karena successor pasti hanya punya **maksimal satu anak kanan**, sehingga aman untuk digantikan.

Percobaan 2

Verifikasi hasil percobaan

```
InOrder Traversal Mahasiswa:
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.35
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.41
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.48
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.61
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.75
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.86
PS C:\Users\USER\kuliah\PRAKTEK ASD\JOBSHEET 14>
```

Pertanyaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?

- **data**: Merupakan array yang menyimpan data (objek Mahasiswa) untuk seluruh node pada tree.

- **idxLast**: Menyimpan indeks **terakhir** dari elemen (node) yang terisi pada array, sehingga kita tahu sampai elemen ke berapa array sudah berisi data tree.

2. Apakah kegunaan dari method populateData()?

- **Mengisi array data** dengan kumpulan data (misal Mahasiswa) dari luar class.

- Menetapkan nilai `idxLast` sebagai batas akhir data yang ada (jumlah data tree).

3. Apakah kegunaan dari method traverseInOrder()?

- Melakukan **traversal in-order** (kiri-root-kanan) pada tree yang disimpan di array.

- Menampilkan data mahasiswa sesuai urutan in-order, yang mencerminkan urutan naik pada binary search tree.

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masing-masing

- **Left child:** indeks = `2 * 2 + 1 = 5`

- **Right child:** indeks = `2 * 2 + 2 = 6`

5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?

- Memberitahu program bahwa **data Mahasiswa yang valid hanya sampai indeks ke-6** (yaitu, 7 elemen, dari indeks 0 sampai 6).

- Dengan `idxLast = 6`, traversal tidak akan membaca elemen setelah indeks ke-6 yang kemungkinan masih `null` (belum diisi data).

6. Mengapa indeks 2*idxStart+1 dan 2*idxStart+2 digunakan dalam pemanggilan rekursif, dan apa kaitannya dengan struktur pohon biner yang disusun dalam array?
   Dalam **representasi binary tree pada array**:

- Jika node parent berada di indeks i, maka:
  - **Left child:** indeks = 2*i + 1
  - **Right child:** indeks = 2*i + 2

  Oleh karena itu, pemanggilan rekursif:
- traverseInOrder(2*idxStart+1) akan menuju anak kiri.
- traverseInOrder(2*idxStart+2) akan menuju anak kanan.

  Ini sesuai dengan **struktur pohon biner** yang diimplementasikan menggunakan array, sehingga hubungan parent-child tetap terjaga tanpa pointer seperti di linked list.