MODUL III

SORTING AND SEARCHING

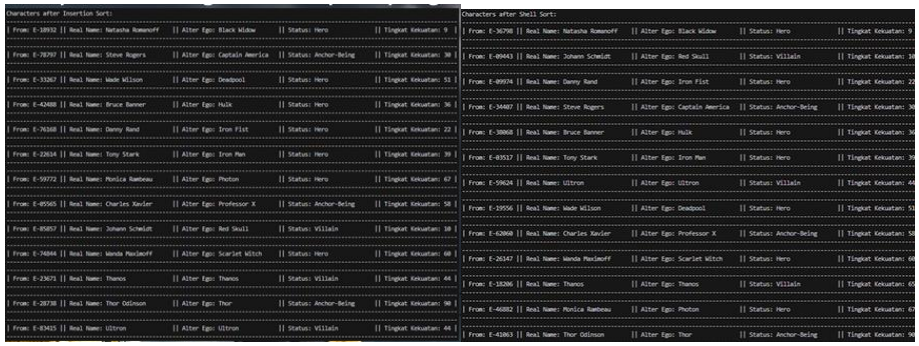
3.1 PERMASALAHAN

3.1.1 Sortpool & Searchvrine

Disuatu tempat bernama the Void tempat yanti dan yanto terbunuh.... entitas bernama SortPool dan Searcvrine 2 entitas yang merupakan pemburu para entitas lain di multiverse yang sangat kejam, membuat mereka dibuang ke the Void oleh TVA sebuah organisasi penjaga TimeLine Agar tidak rusak atau bercabang karena ulah SortPool Searcvrine. dan Namun dengan alat yang di bawa oleh yanti dan yanto bernama TemPad SortPool Searchvrine dan akhirnya keluar dari the Void dan mulai kembali memburu para entitas di Multiverse. SortPool dan Searchvrine mulai mencari entitas mana yang akan diburu dengan memanfaatkan TemPad. Saat mulai Tempad menampilkan data seperti list diatas. Tanpa pikir panjang meraka langsung menuju universe pertama yaitu Earth – 79963/ E-79963 dan membunuh entitas di universe tersebut, dan ternyata di sana SortPool dan Searchvrine bertemu dengan varian SortPool yang bernama InsertionPool, mereka akhirnya merekrut InsertionPool ke anggota untuk membantu membantai para entitas di universe lain. Mereka kembali mencari entitas baru di TemPad namun setiap mereka menjalankan ulang TemPad maka setiap nama universe atau Earth akan berubah dan tidak akan sama seperti sebelumnya.

Karena mereka bingung akhirnya insertionPool menawarkan untuk mengurutkan setiap entitas dengan kemampuan pengurutan InsertionSort miliknya agar dapat mencari nama bukan dilihat dari Earth-nya. InsertionPool akhirnya mengurutkanya berdasarkan nama sebutan ALTER EGO dengan 2 cara yaitu dengan pertukaran node secara langsung dan hanya pertukaran value. Dengan begitu mereka bertiga akhirnya mengunjungi setiap universe dengan urutan yang sudah di buatkan oleh InsertionPool. Setelah beberapa universe mereka kewalahan karena kekuatan dari setiap entitas ada yang melebihi kekuatan mereka dan sulit di taklukan mereka akhirnya mencari anggota baru di multiverse dan menemukan varian lain dari SortPool bernama ShellPool dan Varian Searchvrine bernama Binaryvrine. SortPool meminta ShellPool untuk membantu menganalisis TINGKAT KEKUATAN entitas tiap universe ShellPool pun melakukan pengurutan sesuai perintah menggunakan shell sort dengan 2 cara yang sama seperti yang insertion pool lakukan. Lalu Searchvrine menyarankan agar mereka menargetkan saja siapa entitas ya ng akan dilawan agar lebih efektif mereka pun akhirnya melakukan pencarian dan meminta Binaryvrine untuk melakukan tugas tersebut. Binaryvrine akhirnya melakukan pencarian dengan Binary Search pada sort hasil Insertion Sort dan Shell Short. Setelah melakukan pencarian ternyata mereka dapat

mengetahui lebih lengkap informasi dari setiap entitas baik yang sudah mati maupun yang hidup dan jenis level kekuatan mereka sehingga meningkatkan kesempatan untuk menyusun strategi dan tidak perlu mengunjungi universe yang entitasnya sudah mati.4.



```
Binaryvrine melakukan pencarian Alter Ego = Photon
--------------------------------------------------------------------------------
Alter Ego       : Photon          || Nama Asli      : Monica Rambeau
Level           : Cosmic          || Eksistensi     : Alive
Tingkat Kekuatan: 67              || Status         : Hero
--------------------------------------------------------------------------------
                                E-59772
--------------------------------------------------------------------------------
```

```
Binaryvrine melakukan pencarian tingkatKekuatan = 60
--------------------------------------------------------------------------------
Alter Ego       : Scarlet Witch   || Nama Asli      : Wanda Maximoff
Level           : Mutants         || Eksistensi     : Dead
Tingkat Kekuatan: 60              || Status         : Hero
--------------------------------------------------------------------------------
                                E-74844
--------------------------------------------------------------------------------
```

Setelah kembali menjelajah universe untuk kesekian kalinya mereka kembali bertemu varian lain dari Searchvrine yaitu Linearvrine dan langsung merekrutnya. Saat sedang mencari univers lain yang akan dikunjungi insretionPool & ShellPool memiliki ide untuk melakukan pengurutan entitas dengan cara yang berbeda mereka akan melakukan teknik InsertionPool Sort dan ShellPool Sort yang merupakan Algoritma InsertionSort dan ShellSort dengan 3 keyValue dan pertukaran terjadi pada NODE langsung bukan pada value. Karena InsertionPool dan ShellPool mempunyai kemampuan breakFourthWall dia meminta kamu untuk menganalisis 3 keyValue apa saja yang di pakai dalam teknik InsertionPool sort dan ShellPool Sort ini dan menampilkanya seperti gambar. Setelah itu Linearvrine menunjukan aksinya dia dapat memberikan pencarian Linear Search dengan 2 cara berbeda yaitu dengan pencarian melalui alter ego dan tingkat kekuatan, walaupun pencarian alter ego dicari dengan alter ego yang tidak lengkap, tetap bisa menampilkan entitas" yang memiliki nama mendekati input yang di cari seperti gambar, dan pencarian Linearvrine untuk level kekuatan akan menampilkan semua entitas yang di cari sesuai dengan tingkat kekuatan yang dicari


Characters after Insertion Pool:
From: E-28738 || Real Name: Thor Odinson || Alter Ego: Thor || Status: Anchor-Being || Tingkat Kekuatan: 90
From: E-05565 || Real Name: Charles Xavier || Alter Ego: Professor X || Status: Anchor-Being || Tingkat Kekuatan: 58
From: E-78797 || Real Name: Steve Rogers || Alter Ego: Captain America || Status: Anchor-Being || Tingkat Kekuatan: 30
From: E-59772 || Real Name: Monica Rambeau || Alter Ego: Photon || Status: Hero || Tingkat Kekuatan: 67
From: E-74844 || Real Name: Wanda Maximoff || Alter Ego: Scarlet Witch || Status: Hero || Tingkat Kekuatan: 60
From: E-33267 || Real Name: Wade Wilson || Alter Ego: Deadpool || Status: Hero || Tingkat Kekuatan: 51
From: E-22614 || Real Name: Tony Stark || Alter Ego: Iron Man || Status: Hero || Tingkat Kekuatan: 39
From: E-42488 || Real Name: Bruce Banner || Alter Ego: Hulk || Status: Hero || Tingkat Kekuatan: 36
From: E-76168 || Real Name: Danny Rand || Alter Ego: Iron Fist || Status: Hero || Tingkat Kekuatan: 22
From: E-18932 || Real Name: Natasha Romanoff || Alter Ego: Black Widow || Status: Hero || Tingkat Kekuatan: 9
From: E-23671 || Real Name: Thanos || Alter Ego: Thanos || Status: Villain || Tingkat Kekuatan: 44
From: E-83415 || Real Name: Ultron || Alter Ego: Ultron || Status: Villain || Tingkat Kekuatan: 44
From: E-85857 || Real Name: Johann Schmidt || Alter Ego: Red Skull || Status: Villain || Tingkat Kekuatan: 10
Characters after Shell Pool:
From: E-28738 || Real Name: Thor Odinson || Alter Ego: Thor || Status: Anchor-Being || Tingkat Kekuatan: 90
From: E-05565 || Real Name: Charles Xavier || Alter Ego: Professor X || Status: Anchor-Being || Tingkat Kekuatan: 58
From: E-78797 || Real Name: Steve Rogers || Alter Ego: Captain America || Status: Anchor-Being || Tingkat Kekuatan: 30
From: E-59772 || Real Name: Monica Rambeau || Alter Ego: Photon || Status: Hero || Tingkat Kekuatan: 67
From: E-74844 || Real Name: Wanda Maximoff || Alter Ego: Scarlet Witch || Status: Hero || Tingkat Kekuatan: 60
From: E-33267 || Real Name: Wade Wilson || Alter Ego: Deadpool || Status: Hero || Tingkat Kekuatan: 51
From: E-22614 || Real Name: Tony Stark || Alter Ego: Iron Man || Status: Hero || Tingkat Kekuatan: 39
From: E-42488 || Real Name: Bruce Banner || Alter Ego: Hulk || Status: Hero || Tingkat Kekuatan: 36
From: E-76168 || Real Name: Danny Rand || Alter Ego: Iron Fist || Status: Hero || Tingkat Kekuatan: 22
From: E-18932 || Real Name: Natasha Romanoff || Alter Ego: Black Widow || Status: Hero || Tingkat Kekuatan: 9
From: E-23671 || Real Name: Thanos || Alter Ego: Thanos || Status: Villain || Tingkat Kekuatan: 44
From: E-83415 || Real Name: Ultron || Alter Ego: Ultron || Status: Villain || Tingkat Kekuatan: 44
From: E-85857 || Real Name: Johann Schmidt || Alter Ego: Red Skull || Status: Villain || Tingkat Kekuatan: 10
Pencarian linearvrine : Ir
Alter Ego : Iron Man || Nama Asli : Tony Stark
Level : Tech || Eksistensi : Dead
Tingkat Kekuatan: 39 || Status : Hero
E-22614
Alter Ego : Iron Fist || Nama Asli : Danny Rand
Level : Mid-Tier || Eksistensi : Alive
Tingkat Kekuatan: 22 || Status : Hero
E-76168
Pencarian Linearvrine : 44
Alter Ego : Thanos || Nama Asli : Thanos
Level : Cosmic || Eksistensi : Dead
Tingkat Kekuatan: 44 || Status : Villain
E-23671
Alter Ego : Ultron || Nama Asli : Ultron
Level : Tech || Eksistensi : Alive
Tingkat Kekuatan: 44 || Status : Villain
E-83415

3.3 HASIL PERCOBAAN

3.3.1 Sortpool & Searchvrine

1. Algoritma
    a. Inisialisasi linked list ganda melingkar kosong.
    b. Tetapkan pointer awal daftar ke null.
    c. Tetapkan ukuran daftar menjadi nol.
    d. Untuk menambahkan node baru, buat node dengan data yang diberikan.
    e. Jika daftar kosong, set node baru sebagai awal dan hubungkan ke dirinya sendiri.
    f. Jika daftar tidak kosong, tambahkan node baru di akhir, hubungkan ke awal, dan perbarui pointer.
    g. Tingkatkan ukuran daftar.
    h. Untuk menampilkan daftar, mulai dari node awal dan tampilkan data hingga kembali ke awal.
    i. Untuk mengedit data dalam daftar, cari node dengan data lama yang sesuai.
    j. Jika ditemukan, perbarui data dengan data baru.
    k. Untuk menyortir daftar dengan insertion sort berdasarkan nama, pisahkan sementara dari struktur melingkar.
    l. Sisipkan setiap node di tempat yang sesuai dalam urutan dengan nama.
    m. Gabungkan ulang daftar menjadi melingkar setelah pengurutan.
    n. Untuk insertion sort berdasarkan status dan tingkat kekuatan, pisahkan sementara dari struktur melingkar.
    o. Urutkan berdasarkan status terlebih dahulu, lalu tingkat kekuatan jika status sama.
    p. Gabungkan ulang daftar menjadi melingkar setelah pengurutan.
    q. Untuk shell sort berdasarkan tingkat kekuatan, hitung jumlah node dan tetapkan interval awal.
    r. Lakukan shell sort dengan interval yang semakin kecil, pindahkan node berdasarkan tingkat kekuatan.
    s. Gabungkan ulang daftar menjadi melingkar setelah pengurutan.
    t. Untuk pencarian, telusuri daftar secara melingkar dan tampilkan informasi jika data ditemukan, atau tampilkan pesan tidak ditemukan jika tidak ada data yang cocok.
2. *Source Code*

```
package medium;

public class Node {
    String earth;
    String name;
```

```
        String alterEgo;
        String status;
        int powerLevel;
        Node next, prev;

        public Node(String earth, String name, String
    alterEgo, String status, int powerLevel) {
            this.earth = earth;
            this.name = name;
            this.alterEgo = alterEgo;
            this.status = status;
            this.powerLevel = powerLevel;
        }
    }
```

```
    package medium;

    public class LevelStatus {
        String level;
        String status;
        public LevelStatus(String level, String status) {
            this.level = level;
            this.status = status;
        }
    }
```

```
    package medium;

    public class SearchandSort {
        Node head, tail;
        int size;

        public SearchandSort() {
            head = null;
            size = 0;
        }

        public void addNode(String earth, String name, String
    alterEgo, String status, int powerLevel) {
            Node newNode = new Node(earth, name, alterEgo,
    status, powerLevel);
            if (head == null) {
                head = newNode;
                head.next = head;
                head.prev = head;
            } else {
                Node tail = head.prev;
                tail.next = newNode;
                newNode.prev = tail;
                newNode.next = head;
                head.prev = newNode;
            }
            size++;
        }

        public void display() {
            Node current = head;
            if (head != null) {
                do {
```

```
                System.out.println("+--------------------
----------------------------------------------------------
----------------------------------------------------------
+");
                System.out.printf("| From: %-10s  ||
Real Name: %-18s || Alter Ego: %-15s || Status : %-13s ||
Tingkat Kekuatan : %-3d |\n",
                                    current.earth,
current.name, current.alterEgo, current.status,
current.powerLevel);
                System.out.println("+--------------------
----------------------------------------------------------
----------------------------------------------------------
+");
                current = current.next;
            } while (current != head);
        }
    }

    public void editearth(String lama, String baru) {
        Node current = head;
        do {
            if (current.earth.equals(lama)) {
                current.earth = baru;
                return;
            }
            current = current.next;
        } while (current != head);
    }

    public void insertionNode() {
        if (head == null || head.next == head) return;

        Node urutan = null;
        Node current = head;
        head.prev.next = null;

        while (current != null) {
            Node next = current.next;
            if (urutan == null ||
urutan.alterEgo.compareToIgnoreCase(current.alterEgo) >=
0) {
                current.next = urutan;
                current.prev = null;
                if (urutan != null) urutan.prev = current;
                urutan = current;
            } else {
                Node temp = urutan;
                while (temp.next != null &&
temp.next.alterEgo.compareToIgnoreCase(current.alterEgo) <
0)
                    temp = temp.next;
                current.next = temp.next;
                if (temp.next != null) temp.next.prev =
current;
                temp.next = current;
                current.prev = temp;
            }
            current = next;
        }
```

```
            head = urutan;
            Node tail = head;
            while (tail.next != null) tail = tail.next;
            tail.next = head;
            head.prev = tail;
        }

        public void insertionvalue() {
            if (head == null || head.next == head) return;

            Node current = head.next;
            do {
                String tempAlterEgo = current.alterEgo;
                String tempEarth = current.earth;
                String tempName = current.name;
                String tempStatus = current.status;
                int tempPowerLevel = current.powerLevel;

                Node prev = current.prev;
                while (prev != head.prev &&
    prev.alterEgo.compareToIgnoreCase(tempAlterEgo) > 0) {
                    prev.next.earth = prev.earth;
                    prev.next.name = prev.name;
                    prev.next.alterEgo = prev.alterEgo;
                    prev.next.status = prev.status;
                    prev.next.powerLevel = prev.powerLevel;

                    prev = prev.prev;
                }

                prev.next.earth = tempEarth;
                prev.next.name = tempName;
                prev.next.alterEgo = tempAlterEgo;
                prev.next.status = tempStatus;
                prev.next.powerLevel = tempPowerLevel;

                current = current.next;
            } while (current != head);
        }

        public void shellNode() {
            if (head == null || head.next == head) return;
            head.prev.next = null;
            Node current = head;

            int jumlahNode = 0;
            while (current != null) {
                jumlahNode++;
                current = current.next;
            }

            for (int gap = jumlahNode / 2; gap > 0; gap /= 2)
    {
                for (int i = gap; i < jumlahNode; i++) {
                    Node now = head;
                    for (int k = 0; k < i; k++) now =
    now.next;

                    for (int j = i; j >= gap; j -= gap) {
                        Node banding = head;
```

```
                        for (int k = 0; k < j - gap; k++)
        banding = banding.next;

                        if (banding.powerLevel <=
        now.powerLevel) break;

                        int tempPower = banding.powerLevel;
                        String tempEarth = banding.earth;
                        String tempName = banding.name;
                        String tempAlter = banding.alterEgo;
                        String tempStatus = banding.status;

                        banding.powerLevel = now.powerLevel;
                        banding.earth = now.earth;
                        banding.name = now.name;
                        banding.alterEgo = now.alterEgo;
                        banding.status = now.status;

                        now.powerLevel = tempPower;
                        now.earth = tempEarth;
                        now.name = tempName;
                        now.alterEgo = tempAlter;
                        now.status = tempStatus;

                        now = banding;
                    }
                }
            }

            current = head;
            while (current.next != null) current =
        current.next;
            current.next = head;
            head.prev = current;
        }

        public void shellValue() {
            if (head == null || head.next == head) return;

            int jumlahNode = 0;
            Node temp = head;
            do {
                jumlahNode++;
                temp = temp.next;
            } while (temp != head);

            for (int gap = jumlahNode / 2; gap > 0; gap /= 2)
        {
                for (int i = gap; i < jumlahNode; i++) {
                    Node current = head;
                    for (int k = 0; k < i; k++) {
                        current = current.next;
                    }

                    int tempPower = current.powerLevel;

                    int j = i;
                    Node prevNode = current;
                    while (j >= gap) {
                        Node gapNode = head;
                        for (int k = 0; k < j - gap; k++) {
```

```
                            gapNode = gapNode.next;
                        }

                        if (gapNode.powerLevel <= tempPower)
break;

                        prevNode.powerLevel =
gapNode.powerLevel;
                        prevNode = gapNode;
                        j -= gap;
                    }
                    prevNode.powerLevel = tempPower;
                }
            }
        }

        public void search1(String temp) {
            Node current = head;
            boolean found = false;

            System.out.println("Binaryvrine melakukan
pencarian Alter ego: " + temp);

            do {
                if (current.alterEgo.equalsIgnoreCase(temp)) {
                    found = true;
                    LevelStatus levelStatus =
getLevel(current);
                    displayCharacterInfo(current,
levelStatus);
                    break;
                }
                current = current.next;
            } while (current != head);

            if (!found) {
                System.out.println("+-------------------------
----DATA TIDAK DITEMUKAN----------------------------------
+");
            }
        }

        public void search2(int temp) {
            Node current = head;
            boolean found = false;

            System.out.println("Binaryvrine melakukan
pencarian power level: " + temp);

            do {
                if (current.powerLevel == temp) {
                    found = true;
                    LevelStatus levelStatus =
getLevel(current);
                    displayCharacterInfo(current,
levelStatus);
                    break;
                }
                current = current.next;
            } while (current != head);
```

```
            if (!found) {
                System.out.println("+-------------------------
----DATA TIDAK DITEMUKAN----------------------------------
+");
            }
        }

        public void displayCharacterInfo(Node node,
LevelStatus levelStatus) {
            System.out.println("+-----------------------------
-------------------------------------------------------------
------------+");
            System.out.printf("| Alter Ego              : %-
25s  ||   Nama Asli         : %-20s |\n", node.alterEgo,
node.name);
            System.out.printf("| Level                  : %-
25s  ||   Eksistensi        : %-20s |\n",
levelStatus.level, levelStatus.status);
            System.out.printf("| Tingkat Kekuatan       : %-
25s  ||   Status            : %-20s |\n", node.powerLevel,
node.status);
            System.out.println("+-----------------------------
-------------------------------------------------------------
------------+");
            System.out.println("|
" + node.earth + "
|");
            System.out.println("+-----------------------------
-------------------------------------------------------------
------------+");
        }

        public void insertionSortByStatusAndPowerLevel() {
            if (head == null || head.next == head) return;

            Node urutan = null;
            Node current = head;
            head.prev.next = null;

            while (current != null) {
                Node next = current.next;

                if (urutan == null ||
urutan.status.compareToIgnoreCase(current.status) > 0 ||

(urutan.status.equalsIgnoreCase(current.status) &&
urutan.powerLevel < current.powerLevel)) {
                    current.next = urutan;
                    current.prev = null;
                    if (urutan != null) urutan.prev = current;
                    urutan = current;
                } else {
                    Node temp = urutan;
                    while (temp.next != null &&

(temp.next.status.compareToIgnoreCase(current.status) < 0
||

(temp.next.status.equalsIgnoreCase(current.status) &&
temp.next.powerLevel > current.powerLevel))) {
                        temp = temp.next;
```

```
                    }
                    current.next = temp.next;
                    if (temp.next != null) temp.next.prev =
current;
                    temp.next = current;
                    current.prev = temp;
                }
                current = next;
            }

            head = urutan;
            Node tail = head;
            while (tail.next != null) tail = tail.next;
            tail.next = head;
            head.prev = tail;
        }

        public void shellStatus() {
            if (head == null || head.next == head) return;

            head.prev.next = null;
            Node current = head;

            int nodeCount = 0;
            while (current != null) {
                nodeCount++;
                current = current.next;
            }

            for (int gap = nodeCount / 2; gap > 0; gap /= 2) {
                for (int i = gap; i < nodeCount; i++) {
                    Node now = head;
                    for (int k = 0; k < i; k++) now =
now.next;

                    for (int j = i; j >= gap; j -= gap) {
                        Node compareNode = head;
                        for (int k = 0; k < j - gap; k++)
compareNode = compareNode.next;

                        if
(compareNode.status.compareToIgnoreCase(now.status) < 0 ||

(compareNode.status.equalsIgnoreCase(now.status) &&
compareNode.powerLevel >= now.powerLevel)) {
                            break;
                        }

                        int tempPower =
compareNode.powerLevel;
                        String tempEarth = compareNode.earth;
                        String tempName = compareNode.name;
                        String tempAlterEgo =
compareNode.alterEgo;
                        String tempStatus =
compareNode.status;

                        compareNode.powerLevel =
now.powerLevel;
                        compareNode.earth = now.earth;
                        compareNode.name = now.name;
```

```
                        compareNode.alterEgo = now.alterEgo;
                        compareNode.status = now.status;

                        now.powerLevel = tempPower;
                        now.earth = tempEarth;
                        now.name = tempName;
                        now.alterEgo = tempAlterEgo;
                        now.status = tempStatus;

                        now = compareNode;
                    }
                }
            }

            current = head;
            while (current.next != null) current =
current.next;
            current.next = head;
            head.prev = current;
        }

        public void linearalterego(String alterEgo) {
            Node current = head;
            boolean found = false;

            System.out.println("Pencarian Linearvrine : " +
alterEgo);

            do {
                if
(current.alterEgo.toLowerCase().contains(alterEgo.toLowerC
ase())) {
                    found = true;
                    LevelStatus levelStatus =
getLevel(current);
                    displayCharacterInfo(current,
levelStatus);
                }
                current = current.next;
            } while (current != head);

            if (!found) {
                System.out.println("+-------------------------
----DATA TIDAK DITEMUKAN---------------------------------
+");
            }
        }

        public void linearpowerlevel(int powerLevel) {
            Node current = head;
            boolean found = false;

            System.out.println("Pencarian Linearvrine : " +
powerLevel);

            do {
                if (current.powerLevel == powerLevel) {
                    found = true;
                    LevelStatus levelStatus =
getLevel(current);
```

```
                    displayCharacterInfo(current,
levelStatus);
                }
                current = current.next;
            } while (current != head);

            if (!found) {
                System.out.println("+-------------------------
----DATA TIDAK DITEMUKAN----------------------------------
+");
            }
        }

        public LevelStatus getLevel(Node node) {
            switch (node.alterEgo) {
                case "Iron Man":
                    return new LevelStatus("Tech", "dead");
                case "Iron Fist":
                    return new LevelStatus("Mid-Tier",
"alive");
                case "Iron Patriot":
                    return new LevelStatus("Tech", "alive");
                case "Iron Spider":
                    return new LevelStatus("Tech", "alive");
                case "Thanos":
                    return new LevelStatus("Cosmic", "dead");
                case "Ultron":
                    return new LevelStatus("Tech", "alive");
                case "Photon":
                    return new LevelStatus("Cosmic", "alive");
                case "Scarlet Witch":
                    return new LevelStatus("Mutants", "dead");
                case "Iron Man 2":
                    return new LevelStatus("Tech", "alive");
                case "Raven":
                    return new LevelStatus("Mutants",
"alive");
                case "Starfire":
                    return new LevelStatus("Mutants", "dead");
                case "Beast Boy":
                    return new LevelStatus("Cosmic", "alive");
                default:
                    return new LevelStatus("?", "?");
            }
        }
    }
```

```
    package medium;

    public class Main {
        public static void main(String[] args) {
            SearchandSort entities = new SearchandSort();

            entities.addNode("E-13423", "Monica Rambeau",
    "Photon", "Hero", 67);
            entities.addNode("E-95458", "Charles Xavier",
    "Professor X", "Anchor-Being", 58);
            entities.addNode("E-88538", "Wanda Maximoff",
    "Scarlet Witch", "Hero", 60);
```

```
            entities.addNode("E-72064", "Steve Rogers",
"Captain America", "Anchor-Being", 30);
            entities.addNode("E-99610", "Bruce Banner",
"Hulk", "Hero", 36);
            entities.addNode("E-69074", "Tony Stark", "Iron
Man", "Hero", 39);
            entities.addNode("E-32631", "Ultron", "Ultron",
"Villain", 44);
            entities.addNode("E-01751", "Danny Rand", "Iron
Fist", "Hero", 22);
            entities.addNode("E-79963", "Natasha Romanoff",
"Black Widow", "Hero", 9);
            entities.addNode("E-71802", "Johann Schmidt", "Red
Skull", "Villain", 10);
            entities.addNode("E-77521", "Thor Odinson",
"Thor", "Anchor-Being", 90);
            entities.addNode("E-19745", "Wade Wilson",
"Deadpool", "Hero", 51);
            entities.addNode("E-92667", "Thanos", "Thanos",
"Villain", 44);

            System.out.println("Character before sorting : ");
            entities.display();

            entities.editearth("E-13423", "E-95458");
            entities.editearth("E-95458", "E-88538");
            entities.editearth("E-88538", "E-72064");
            entities.editearth("E-72064", "E-99610");
            entities.editearth("E-99610", "E-69074");
            entities.editearth("E-69074", "E-32631");
            entities.editearth("E-32631", "E-01751");
            entities.editearth("E-01751", "E-79963");
            entities.editearth("E-79963", "E-71802");
            entities.editearth("E-71802", "E-77521");
            entities.editearth("E-77521", "E-19745");
            entities.editearth("E-19745", "E-92667");
            entities.editearth("E-92667", "E-11101");



            System.out.println("\nCharacter before sorting :
");
            entities.display();
            System.out.println("\nCharacter after Insertion
sort : ");
            entities.insertionNode();
            entities.insertionvalue();
            entities.display();

            System.out.println("\nCharacter after shell sort :
");
            entities.shellNode();
            entities.shellValue();
            entities.display();

            entities.search1("Photon");
            entities.search2(60);

            entities.insertionSortByStatusAndPowerLevel();
            entities.display();
```

```
                entities.shellStatus();
                entities.display();

                entities.linearalterego("Ir");
                entities.linearpowerlevel(44);
            }
        }
```

3.4 ANALISIS DATA

3.4.1 Sortpool & Searchvrine

```
package medium;

public class Node {
    String earth;
    String name;
    String alterEgo;
    String status;
    int powerLevel;
    Node next, prev;

    public Node(String earth, String name, String alterEgo, String
status, int powerLevel) {
        this.earth = earth;
        this.name = name;
        this.alterEgo = alterEgo;
        this.status = status;
        this.powerLevel = powerLevel;
    }

}
```

Kelas "`Node`" mendefinisikan node dalam linked list ganda melingkar, dengan atribut "`earth`", "`name`", "`alterEgo`", "`status`", dan "`powerLevel`" untuk menyimpan informasi karakter. Pointer "`next`" dan "`prev`" memungkinkan traversal di kedua arah. Konstruktornya menginisialisasi semua atribut saat node dibuat.

```
package medium;
public class LevelStatus {
String level;
String status;
public LevelStatus(String level, String status) {
this.level = level;
this.status = status;
}
}
```

Kelas "`LevelStatus`" merepresentasikan level dan status dari sebuah karakter. Kelas ini memiliki dua atribut, yaitu "`level`" dan "`status`", yang diinisialisasi melalui konstruktor untuk menyimpan informasi level dan status karakter terkait.

```
public class SearchandSort {
    Node head, tail;
    int size;

    public SearchandSort() {
        head = null;
```

```
        size = 0;
    }

    public void addNode(String earth, String name, String alterEgo,
String status, int powerLevel) {
        Node newNode = new Node(earth, name, alterEgo, status,
powerLevel);
        if (head == null) {
            head = newNode;
            head.next = head;
            head.prev = head;
        } else {
            Node tail = head.prev;
            tail.next = newNode;
            newNode.prev = tail;
            newNode.next = head;
            head.prev = newNode;
        }
        size++;
    }
```

Kelas "SearchandSort" mengelola linked list ganda melingkar dengan atribut "head", "tail", dan "size". Konstruktor kelas ini menginisialisasi "head" menjadi null dan "size" menjadi 0. Metode "addNode" digunakan untuk menambahkan node baru ke dalam linked list. Jika list kosong, node baru akan menjadi "head" dan saling terhubung dengan dirinya sendiri (mengarah ke "head" di "next" dan "prev"). Jika list tidak kosong, node baru ditambahkan di akhir dan saling terhubung dengan "tail" dan "head" untuk menjaga struktur linked list melingkar.

```
public void display() {
        Node current = head;
        if (head != null) {
            do {
                System.out.println("+----------------------------------
--------------------------------------------------------------------
-------------------------------+");
                System.out.printf("| From: %-10s  ||   Real Name: %-18s
|| Alter Ego: %-15s || Status : %-13s || Tingkat Kekuatan : %-3d |\n",
                                  current.earth,          current.name,
current.alterEgo, current.status, current.powerLevel);
                System.out.println("+----------------------------------
--------------------------------------------------------------------
-------------------------------+");
                current = current.next;
            } while (current != head);
        }
    }
```

Metode "display" digunakan untuk menampilkan informasi dari setiap node dalam linked list melingkar. Dimulai dari "head", metode ini menelusuri seluruh linked list menggunakan pointer "next" hingga kembali ke "head". Setiap node ditampilkan dengan format yang mencakup informasi "earth", "name", "alterEgo", "status", dan

"`powerLevel`". Jika linked list kosong ("`head == null`"), tidak ada yang ditampilkan. Hasil tampilan disusun dengan garis pemisah untuk membuatnya lebih rapi.

```
public void editearth(String lama, String baru) {
        Node current = head;
        do {
            if (current.earth.equals(lama)) {
                current.earth = baru;
                return;
            }
            current = current.next;
        } while (current != head);
    }
```

Metode "`editearth`" berfungsi untuk mengubah nilai "`earth`" pada node yang sesuai dengan nilai yang diberikan (parameter "`lama`"). Proses pencarian dimulai dari "`head`" dan terus menelusuri setiap node dengan menggunakan pointer "`next`". Jika ditemukan node dengan "earth" yang cocok dengan "`lama`", maka nilai "`earth`" pada node tersebut diperbarui dengan nilai baru ("`baru`"). Setelah itu, metode akan keluar tanpa melanjutkan pencarian. Jika node dengan "`earth`" yang sesuai tidak ditemukan, pencarian akan berhenti ketika kembali ke "`head`" karena linked list adalah melingkar.

```
public void insertionNode() {
        if (head == null || head.next == head) return;

        Node urutan = null;
        Node current = head;
        head.prev.next = null;

        while (current != null) {
            Node next = current.next;
            if           (urutan           ==           null           ||
urutan.alterEgo.compareToIgnoreCase(current.alterEgo) >= 0) {
                current.next = urutan;
                current.prev = null;
                if (urutan != null) urutan.prev = current;
                urutan = current;
            } else {
                Node temp = urutan;
                while        (temp.next        !=        null        &&
temp.next.alterEgo.compareToIgnoreCase(current.alterEgo) < 0)
                    temp = temp.next;
                current.next = temp.next;
                if (temp.next != null) temp.next.prev = current;
                temp.next = current;
                current.prev = temp;
            }
            current = next;
        }

        head = urutan;
        Node tail = head;
        while (tail.next != null) tail = tail.next;
        tail.next = head;
        head.prev = tail;
```

```
    }
```

Metode "insertionNode" mengurutkan linked list berdasarkan "alterEgo" menggunakan algoritma insertion sort. Node disisipkan pada posisi yang sesuai, baik di depan atau di tengah urutan yang sudah ada. Setelah diurutkan, "head" dan "tail" diperbarui untuk mempertahankan struktur linked list melingkar.

```
public void insertionvalue() {
        if (head == null || head.next == head) return;

        Node current = head.next;
        do {
            String tempAlterEgo = current.alterEgo;
            String tempEarth = current.earth;
            String tempName = current.name;
            String tempStatus = current.status;
            int tempPowerLevel = current.powerLevel;

            Node prev = current.prev;
            while (prev != head.prev &&
prev.alterEgo.compareToIgnoreCase(tempAlterEgo) > 0) {
                prev.next.earth = prev.earth;
                prev.next.name = prev.name;
                prev.next.alterEgo = prev.alterEgo;
                prev.next.status = prev.status;
                prev.next.powerLevel = prev.powerLevel;

                prev = prev.prev;
            }

            prev.next.earth = tempEarth;
            prev.next.name = tempName;
            prev.next.alterEgo = tempAlterEgo;
            prev.next.status = tempStatus;
            prev.next.powerLevel = tempPowerLevel;

            current = current.next;
        } while (current != head);
    }
```

Metode insertionvalue ini mengurutkan node dalam linked list berdasarkan nama alter ego secara alfabetis (dengan memperhatikan huruf kapital). Dimulai dari node kedua, data pada node yang sedang diproses dibandingkan dengan node sebelumnya. Jika alter ego dari node yang sedang diproses lebih kecil daripada alter ego node sebelumnya (dibandingkan secara huruf besar-kecil), data node tersebut dipindahkan ke posisi yang lebih awal. Proses ini dilakukan secara berulang hingga semua node terurut sesuai dengan urutan alter ego mereka.

```
public void shellNode() {
        if (head == null || head.next == head) return;
        head.prev.next = null;
        Node current = head;

        int jumlahNode = 0;
```

```
        while (current != null) {
            jumlahNode++;
            current = current.next;
        }

        for (int gap = jumlahNode / 2; gap > 0; gap /= 2) {
            for (int i = gap; i < jumlahNode; i++) {
                Node now = head;
                for (int k = 0; k < i; k++) now = now.next;

                for (int j = i; j >= gap; j -= gap) {
                    Node banding = head;
                    for (int k = 0; k < j - gap; k++) banding =
banding.next;

                    if (banding.powerLevel <= now.powerLevel) break;

                    int tempPower = banding.powerLevel;
                    String tempEarth = banding.earth;
                    String tempName = banding.name;
                    String tempAlter = banding.alterEgo;
                    String tempStatus = banding.status;

                    banding.powerLevel = now.powerLevel;
                    banding.earth = now.earth;
                    banding.name = now.name;
                    banding.alterEgo = now.alterEgo;
                    banding.status = now.status;

                    now.powerLevel = tempPower;
                    now.earth = tempEarth;
                    now.name = tempName;
                    now.alterEgo = tempAlter;
                    now.status = tempStatus;

                    now = banding;
                }
            }
        }

        current = head;
        while (current.next != null) current = current.next;
        current.next = head;
        head.prev = current;
    }
```

Metode shellNode ini mengimplementasikan algoritma pengurutan Shell Sort pada linked list berdasarkan tingkat kekuatan (power level). Pertama, kode menghitung jumlah node dalam list. Kemudian, algoritma Shell Sort diterapkan dengan menggunakan "`gap`" yang terus dibagi dua hingga mencapai 1. Untuk setiap gap, elemen yang lebih kecil (berdasarkan power level) akan dipindahkan ke posisi yang lebih tinggi, secara bertahap menyusun list. Setelah proses sorting selesai, linked list akan terurut berdasarkan tingkat kekuatan secara menaik.

```
public void shellValue() {
        if (head == null || head.next == head) return;
```

```
        int jumlahNode = 0;
        Node temp = head;
        do {
            jumlahNode++;
            temp = temp.next;
        } while (temp != head);

        for (int gap = jumlahNode / 2; gap > 0; gap /= 2) {
            for (int i = gap; i < jumlahNode; i++) {
                Node current = head;
                for (int k = 0; k < i; k++) {
                    current = current.next;
                }

                int tempPower = current.powerLevel;

                int j = i;
                Node prevNode = current;
                while (j >= gap) {
                    Node gapNode = head;
                    for (int k = 0; k < j - gap; k++) {
                        gapNode = gapNode.next;
                    }

                    if (gapNode.powerLevel <= tempPower) break;

                    prevNode.powerLevel = gapNode.powerLevel;
                    prevNode = gapNode;
                    j -= gap;
                }
                prevNode.powerLevel = tempPower;
            }
        }
    }
```

Metode shellValue ini mengimplementasikan algoritma "`Shell Sort`" yang mengurutkan linked list berdasarkan nilai "`powerLevel`". Pertama, kode menghitung jumlah node dalam list, lalu algoritma ini mulai dengan gap yang lebih besar dan secara bertahap mengurangi gap tersebut hingga mencapai 1. Pada setiap iterasi gap, elemen-elemen dalam linked list yang memiliki nilai power level lebih besar daripada elemen lainnya dipindahkan ke posisi yang lebih tinggi. Proses ini berulang sampai seluruh linked list terurut berdasarkan power level secara menaik.

```
public void search1(String temp) {
        Node current = head;
        boolean found = false;

        System.out.println("Binaryvrine melakukan pencarian Alter ego: "
+ temp);

        do {
            if (current.alterEgo.equalsIgnoreCase(temp)) {
                found = true;
                LevelStatus levelStatus = getLevel(current);
                displayCharacterInfo(current, levelStatus);
```

```
                break;
            }
            current = current.next;
        } while (current != head);

        if (!found) {
            System.out.println("+-----------------------------DATA
TIDAK DITEMUKAN-----------------------------------+");
        }
    }
```

Metode search1 digunakan untuk mencari node berdasarkan "`alterEgo`" dalam linked list. Fungsi ini menerima parameter "`temp`" (alter ego yang dicari) dan mengiterasi setiap node dalam list. Jika ditemukan node yang memiliki "`alterEgo`" yang sesuai, pencarian dihentikan dan informasi level serta status karakter ditampilkan. Jika tidak ditemukan, pesan "`Data tidak ditemukan`" akan ditampilkan. Pencarian dilakukan dengan membandingkan "`alterEgo`" dalam setiap node dan menggunakan pencarian yang tidak sensitif terhadap huruf besar/kecil.

```
    public void displayCharacterInfo(Node node, LevelStatus levelStatus)
{
        System.out.println("+-----------------------------------------
-----------------------------------------------------+");
        System.out.printf("| Alter Ego               : %-25s  ||   Nama
Asli          : %-20s |\n", node.alterEgo, node.name);
        System.out.printf("| Level                  : %-25s  ||  Eksistensi
: %-20s |\n", levelStatus.level, levelStatus.status);
        System.out.printf("| Tingkat Kekuatan       : %-25s  ||   Status
: %-20s |\n", node.powerLevel, node.status);
        System.out.println("+-----------------------------------------
-----------------------------------------------------+");
        System.out.println("|                                                "
+ node.earth + "                                      |");
        System.out.println("+-----------------------------------------
-----------------------------------------------------+");
    }
```

Metode displayCharacterInfo digunakan untuk menampilkan informasi lengkap tentang karakter berdasarkan data yang ada dalam objek "`Node`" dan "`LevelStatus`". Fungsi ini menerima dua parameter: "`node`" yang berisi data karakter (seperti alter ego, nama asli, kekuatan, status, dan bumi asal), serta "`levelStatus`" yang berisi level dan status eksistensi karakter. Informasi tersebut ditampilkan dengan format yang terstruktur rapi, termasuk nama asli, alter ego, tingkat kekuatan, dan status eksistensi karakter. Hasilnya dicetak dalam tabel yang terformat.

```
public void insertionSortByStatusAndPowerLevel() {
        if (head == null || head.next == head) return;

        Node urutan = null;
        Node current = head;
        head.prev.next = null;
```

```
        while (current != null) {
            Node next = current.next;

            if          (urutan          ==          null          ||
urutan.status.compareToIgnoreCase(current.status) > 0 ||
                (urutan.status.equalsIgnoreCase(current.status)      &&
urutan.powerLevel < current.powerLevel)) {
                current.next = urutan;
                current.prev = null;
                if (urutan != null) urutan.prev = current;
                urutan = current;
            } else {
                Node temp = urutan;
                while (temp.next != null &&

(temp.next.status.compareToIgnoreCase(current.status) < 0 ||

(temp.next.status.equalsIgnoreCase(current.status)                    &&
temp.next.powerLevel > current.powerLevel))) {
                    temp = temp.next;
                }
                current.next = temp.next;
                if (temp.next != null) temp.next.prev = current;
                temp.next = current;
                current.prev = temp;
            }
            current = next;
        }

        head = urutan;
        Node tail = head;
        while (tail.next != null) tail = tail.next;
        tail.next = head;
        head.prev = tail;
    }
```

Metode insertionSortByStatusAndPowerLevel mengimplementasikan algoritma penyortiran "`insertion sort`" untuk mengurutkan karakter berdasarkan dua kriteria: status eksistensi terlebih dahulu, dan jika statusnya sama, berdasarkan tingkat kekuatan (power level) secara menurun. Proses dimulai dengan memeriksa apakah linked list kosong atau hanya memiliki satu elemen. Kemudian, untuk setiap node, karakter akan dipindahkan ke posisi yang sesuai dengan kriteria yang ditentukan, dengan membandingkan status dan power level. Setelah penyortiran selesai, linked list diatur ulang dengan "`head`" yang baru dan "`tail`" yang di-update agar tetap membentuk list melingkar.

```
public void shellStatus() {
        if (head == null || head.next == head) return;

        head.prev.next = null;
        Node current = head;

        int nodeCount = 0;
        while (current != null) {
            nodeCount++;
```

```
            current = current.next;
        }

        for (int gap = nodeCount / 2; gap > 0; gap /= 2) {
            for (int i = gap; i < nodeCount; i++) {
                Node now = head;
                for (int k = 0; k < i; k++) now = now.next;

                for (int j = i; j >= gap; j -= gap) {
                    Node compareNode = head;
                    for (int k = 0; k < j - gap; k++) compareNode =
compareNode.next;

                    if
(compareNode.status.compareToIgnoreCase(now.status) < 0 ||

(compareNode.status.equalsIgnoreCase(now.status)                          &&
compareNode.powerLevel >= now.powerLevel)) {
                        break;
                    }

                    int tempPower = compareNode.powerLevel;
                    String tempEarth = compareNode.earth;
                    String tempName = compareNode.name;
                    String tempAlterEgo = compareNode.alterEgo;
                    String tempStatus = compareNode.status;

                    compareNode.powerLevel = now.powerLevel;
                    compareNode.earth = now.earth;
                    compareNode.name = now.name;
                    compareNode.alterEgo = now.alterEgo;
                    compareNode.status = now.status;

                    now.powerLevel = tempPower;
                    now.earth = tempEarth;
                    now.name = tempName;
                    now.alterEgo = tempAlterEgo;
                    now.status = tempStatus;

                    now = compareNode;
                }
            }
        }

        current = head;
        while (current.next != null) current = current.next;
        current.next = head;
        head.prev = current;
    }
```

Metode shellStatus mengimplementasikan algoritma "`Shell Sort`" untuk mengurutkan karakter berdasarkan status eksistensi terlebih dahulu, dan jika statusnya sama, mengurutkan berdasarkan tingkat kekuatan (power level). Proses dimulai dengan menghitung jumlah node dalam list. Kemudian, algoritma melakukan iterasi menggunakan gap yang semakin mengecil, mulai dari setengah jumlah node dan membagi gap hingga mencapai 1. Selama iterasi, karakter akan dibandingkan berdasarkan status dan power level. Jika karakter

sebelumnya memiliki status yang lebih rendah atau status yang sama namun power level lebih rendah, maka elemen-elemen tersebut akan dipertukarkan. Setelah penyortiran selesai, list diatur ulang untuk memastikan hubungan antara node-head dan tail tetap benar pada list melingkar.

```
public void linearpowerlevel(int powerLevel) {
        Node current = head;
        boolean found = false;

        System.out.println("Pencarian Linearvrine : " + powerLevel);

        do {
            if (current.powerLevel == powerLevel) {
                found = true;
                LevelStatus levelStatus = getLevel(current);
                displayCharacterInfo(current, levelStatus);
            }
            current = current.next;
        } while (current != head);

        if (!found) {
            System.out.println("+----------------------------DATA
TIDAK DITEMUKAN---------------------------------+");
        }
    }
```

Metode linearpowerlevel adalah implementasi dari pencarian linear pada linked list untuk menemukan karakter berdasarkan nilai "`powerLevel`". Pencarian dimulai dari node "`head`" dan terus melangkah ke node berikutnya hingga mencapai node pertama lagi (karena list ini bersifat melingkar). Jika ditemukan node yang memiliki "`powerLevel`" yang sesuai dengan nilai yang dicari, informasi karakter akan ditampilkan dengan menggunakan metode displayCharacterInfo. Jika tidak ada node yang cocok, maka sistem akan menampilkan pesan bahwa data tidak ditemukan.

```
public LevelStatus getLevel(Node node) {
        switch (node.alterEgo) {
            case "Iron Man":
                return new LevelStatus("Tech", "dead");
            case "Iron Fist":
                return new LevelStatus("Mid-Tier", "alive");
            case "Iron Patriot":
                return new LevelStatus("Tech", "alive");
            case "Iron Spider":
                return new LevelStatus("Tech", "alive");
            case "Thanos":
                return new LevelStatus("Cosmic", "dead");
            case "Ultron":
                return new LevelStatus("Tech", "alive");
            case "Photon":
                return new LevelStatus("Cosmic", "alive");
            case "Scarlet Witch":
                return new LevelStatus("Mutants", "dead");
            case "Iron Man 2":
```

```
                return new LevelStatus("Tech", "alive");
            case "Raven":
                return new LevelStatus("Mutants", "alive");
            case "Starfire":
                return new LevelStatus("Mutants", "dead");
            case "Beast Boy":
                return new LevelStatus("Cosmic", "alive");
            default:
                return new LevelStatus("?", "?");
        }
    }
```

Metode getLevel berfungsi untuk menentukan level dan status karakter berdasarkan nama "`alterEgo`" yang diberikan. Dalam metode ini, sebuah "switch-`case`" digunakan untuk memeriksa nilai "`alterEgo`" dari node yang diberikan. Berdasarkan "`alterEgo`", metode ini mengembalikan objek "`LevelStatus`" yang berisi informasi level (seperti "`Tech`", "`Cosmic`", "`Mutants`") dan status (seperti "`alive`", "`dead`"). Jika nama "`alterEgo`" tidak ditemukan dalam daftar yang telah ditentukan, maka metode ini akan mengembalikan "`LevelStatus`" dengan nilai "`?`" untuk kedua atribut (level dan status).

```
        SearchandSort entities = new SearchandSort();

        entities.addNode("E-13423", "Monica Rambeau", "Photon", "Hero",
67);
        entities.addNode("E-95458", "Charles Xavier", "Professor X",
"Anchor-Being", 58);
```

Kode ini menjelasakan entities dibuat dari kelas SearchandSort untuk mengelola data menggunakan struktur linked list. Dua node ditambahkan ke dalamnya menggunakan metode addNode, yang menyimpan informasi seperti identifier, nama, alter ego, status, dan tingkat kekuatan. Setiap node berisi data yang berhubungan dengan karakter, yang kemudian disusun dalam linked list yang memungkinkan pencarian, penyaringan, dan pengurutan berdasarkan kriteria tertentu.

```
System.out.println("\nCharacter before sorting : ");
        entities.display();
        System.out.println("\nCharacter after Insertion sort : ");
        entities.insertionNode();
        entities.insertionvalue();
        entities.display();

        System.out.println("\nCharacter after shell sort : ");
        entities.shellNode();
        entities.shellValue();
        entities.display();

        entities.search1("Photon");
        entities.search2(60);


        entities.insertionSortByStatusAndPowerLevel();
        entities.display();
```

```
        entities.shellStatus();
        entities.display();

        entities.linearalterego("Ir");
        entities.linearpowerlevel(44);
```

Pada kode di atas, pertama-tama, daftar karakter ditampilkan sebelum pengurutan menggunakan metode display. Kemudian, karakter diurutkan menggunakan insertionNode dan insertionvalue untuk menyusun node berdasarkan urutan tertentu, lalu hasilnya ditampilkan kembali. Setelah itu, pengurutan menggunakan shellNode dan shellValue dilakukan, diikuti dengan menampilkan hasilnya. Selanjutnya, pencarian karakter berdasarkan alter ego menggunakan search1 dan pencarian berdasarkan power level menggunakan search2 dilakukan untuk menemukan karakter tertentu. Pengurutan lebih lanjut berdasarkan status dan power level dilakukan dengan insertionSortByStatusAndPowerLevel dan hasilnya ditampilkan. Setelah itu, pengurutan berdasarkan status dilakukan dengan shellStatus, diikuti dengan tampilan hasil akhir. Terakhir, pencarian karakter berdasarkan alter ego dan power level dilakukan melalui linearalterego dan linearpowerlevel untuk mencari karakter dengan kriteria tertentu.

3.5.1 Tambahan

3.5.2 Analisi data

```
    public void addNodeDesc(String earth, String name, String
alterEgo, String status, int powerLevel) {
        Node newNode = new Node(earth, name, alterEgo, status,
powerLevel);
        if (head == null) {
            head = newNode;
            head.next = head;
            head.prev = head;
        } else {
            Node tail = head.prev;
            newNode.next = head;
            newNode.prev = tail;
            head.prev = newNode;
            tail.next = newNode;
            head = newNode;
        }
        size++;
    }
```

Metode addNodeDesc menambahkan node baru ke dalam circular doubly linked list dengan menjadikannya node pertama (head). Jika list kosong, node baru menunjuk dirinya sendiri sebagai `next` dan `prev`. Jika list berisi node lain, `newNode` disisipkan sebelum `head`, dihubungkan dengan node terakhir (tail) untuk menjaga circularity, lalu `head`

diperbarui menjadi `newNode`. Ukuran list (`size`) bertambah satu setiap kali node ditambahkan.

```
    public void addNodeRandom(String earth, String name, String alterEgo,
    String status, int powerLevel) {
              Node newNode = new Node(earth, name, alterEgo, status,
    powerLevel);

            if (head == null) {
                head = newNode;
                head.next = head;
                head.prev = head;
            } else {
                int position = (int) (Math.random() * (size + 1));

                if (position == 0) {
                    Node tail = head.prev;

                    newNode.next = head;
                    newNode.prev = tail;
                    head.prev = newNode;
                    tail.next = newNode;
                    head = newNode;
                } else if (position == size) {
                    // Insert at the end
                    Node tail = head.prev;

                    tail.next = newNode;
                    newNode.prev = tail;
                    newNode.next = head;
                    head.prev = newNode;
                } else {
                    Node current = head;
                    for (int i = 0; i < position; i++) {
                        current = current.next;
                    }
                    Node previous = current.prev;
                    previous.next = newNode;
                    newNode.prev = previous;
                    newNode.next = current;
                    current.prev = newNode;
                }
            }
            size++;
        }
```

Metode addNodeRandom menambahkan node baru ke dalam circular doubly linked list pada posisi acak. Jika list kosong, node baru menjadi head dan menunjuk dirinya sendiri untuk next dan prev. Jika tidak, posisi acak dipilih antara 0 dan size. Jika posisi adalah 0, node baru ditempatkan di awal (sebagai head); jika pada posisi terakhir (size), node baru ditempatkan di akhir; jika di tengah, node baru disisipkan di antara node-node yang sudah ada. Ukuran list (size) kemudian diperbarui setiap kali node baru ditambahkan.

1. AddNodeAscen

```
Character after Insertion sort :
Time taken insertion Sort: 990600 ns

Character after shell sort :
Time taken shell Sort: 1045400 ns

Time taken insertion status and level Sort: 153900 ns

Time taken shell status and level Sort: 543500 ns
```

Pada data yang sudah terurut secara menaik, **Insertion Sort** menunjukkan waktu yang lebih cepat dibandingkan **Shell Sort** untuk pengurutan dasar. Insertion Sort umumnya lebih efisien pada data yang sudah hampir terurut karena hanya perlu melakukan sedikit pergeseran posisi elemen, sehingga waktu eksekusi lebih singkat. Untuk pengurutan dengan tambahan status dan level, Insertion Sort tetap lebih unggul dibandingkan Shell Sort.

2. AddNodeDescen

```
Character after Insertion sort :
Time taken insertion Sort: 1414600 ns

Character after shell sort :
Time taken shell Sort: 1664600 ns

Time taken insertion status and level Sort: 235700 ns

Time taken shell status and level Sort: 481700 ns
```

Pada data yang terurut secara menurun, Insertion Sort juga tetap lebih cepat dibandingkan Shell Sort. Meskipun Insertion Sort membutuhkan lebih banyak operasi pergeseran elemen untuk data yang berlawanan arah (descending menjadi ascending), tetap saja ia menunjukkan efisiensi yang lebih baik dari Shell Sort. Pengurutan dengan tambahan status dan level juga memperlihatkan bahwa Insertion Sort lebih unggul.

3. AddNodeRandom

```
Character after Insertion sort :
Time taken insertion Sort: 1426700 ns

Character after shell sort :
Time taken shell Sort: 1367800 ns

Time taken insertion status and level Sort: 146800 ns

Time taken shell status and level Sort: 462900 ns
```

Untuk data yang acak, Shell Sort sedikit lebih cepat dibandingkan Insertion Sort dalam pengurutan dasar, menunjukkan bahwa Shell Sort lebih efisien dalam mengatasi data yang tidak terstruktur karena interval pergeseran yang lebih besar. Namun, dalam pengurutan dengan status dan level, Insertion Sort kembali lebih cepat dibandingkan Shell Sort.

Kesimpulan :

1. Insertion Sort lebih efisien untuk data yang sudah hampir terurut (ascending) atau memiliki tambahan kondisi (status dan level) dalam pengurutan.
2. Shell Sort lebih efisien untuk data acak dalam pengurutan dasar, tetapi kalah efisien dalam pengurutan dengan kondisi tambahan.
3. Merge Sort memiliki kompleksitas waktu yang stabil (O(n log n)) dan bekerja dengan baik pada data acak maupun terstruktur.
4. Quick Sort sering lebih cepat pada data besar dengan distribusi acak, tetapi kinerjanya bisa menurun pada data yang sudah terurut tanpa modifikasi.