

Laporan Praktikum

10S2102 Algoritma Dan Struktur Data

Sorting



Nama Kelompok:

11S20018 Ruth Aulya Silalahi

11S20043 Putri Ruth Berliana Siahaan

INSTITUT TEKNOLOGI DEL
FAKULTAS INFORMATIKA DAN TEKNIK
ELEKTRO

T.A. 2021/2022

Materi Praktikum

Anda sudah mempelajari materi topik Sorting yang sudah dipublish via cis dan ecouse. Berikut ini diberikan modul praktikum yang diambil dari buku Mark Allen Weiss, Data Structures and Problem Solving Using Java (4rd edition), Addison Wesley, 2010, Chapter 8, untuk menjawab soal-soal berikut. Silahkan mengerjakan modul praktikum ini secara berpasangan.

1. Kerjakan soal 8.1, lengkapi jawaban anda dengan penjelasan yang detail.

IN SHORT

- 8.1 Sort the sequence 8, 1, 4, 1, 5, 9, 2, 6, 5 by using
- a. Insertion sort
 - b. Shellsort for the increments {1, 3, 5}
 - c. Mergesort
 - d. Quicksort, with the middle element as pivot and no cutoff (show all steps)
 - e. Quicksort, with median-of-three pivot selection and a cutoff of 3

Jawab :


- a) Insertion Sort

sorted list	unsorted list							
8	1	4	1	5	9	2	6	5

Tahap 1:

Angka pertama dibandingkan dengan angka yang di sampingnya. Pada tahap ini, angka '8' dibandingkan dengan angka '1', yang mana nilai angka '1' lebih kecil sehingga terjadi pertukaran posisi.

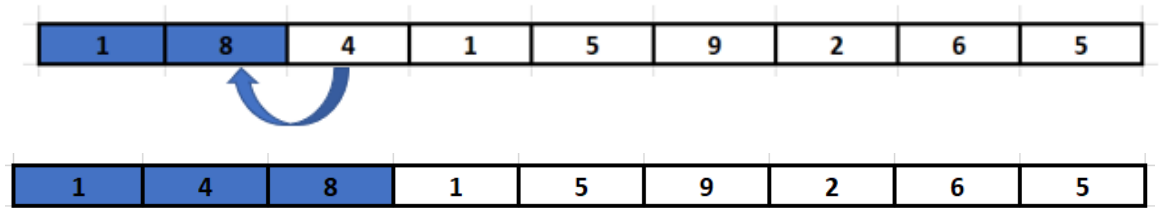
sorted list	unsorted list							
8	1	4	1	5	9	2	6	5



1	8	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

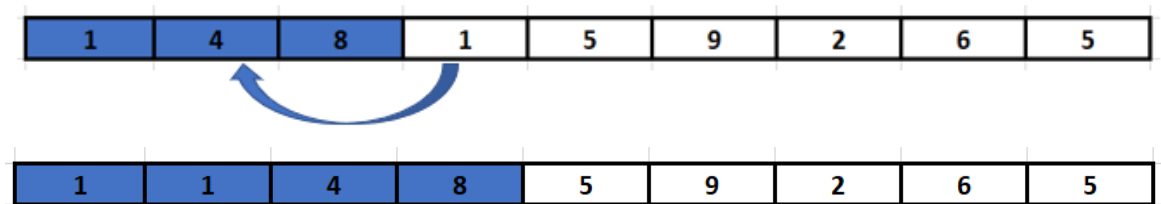
Tahap 2:

Angka pertama dalam unsorted list adalah '4'. Angka '4' dibandingkan dengan '8', yang mana angka '4' lebih kecil sehingga terjadi pertukaran posisi.



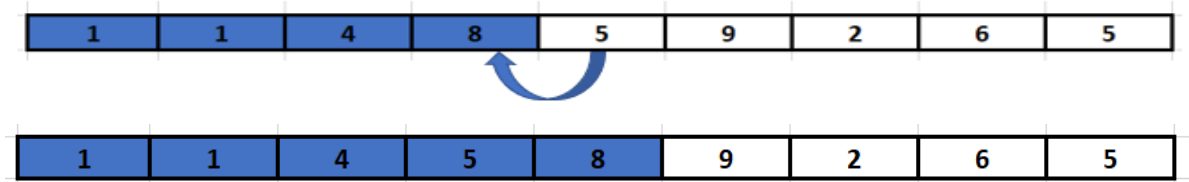
Tahap 3:

Angka pertama dalam unsorted list adalah '1'. Angka '1' dibandingkan dengan '4', yang mana angka '1' lebih kecil sehingga ditempatkan sesuai posisi yang benar.



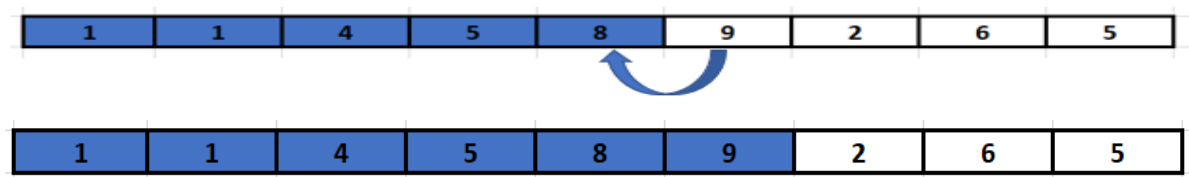
Tahap 4:

Angka pertama dalam unsorted list adalah '5'. Angka '5' dibandingkan dengan '8', yang mana angka '5' lebih kecil sehingga terjadi pertukaran posisi.



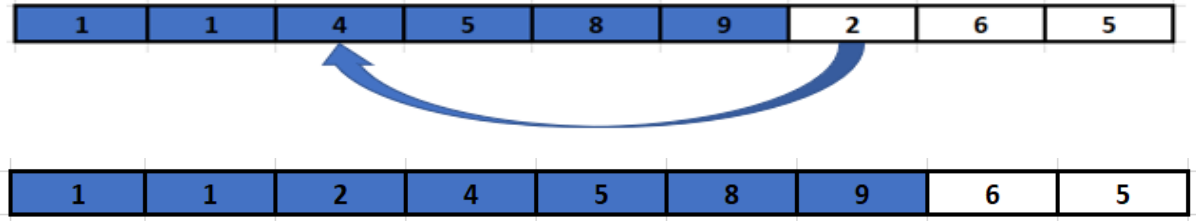
Tahap 5:

Angka pertama dalam unsorted list adalah '9'. Angka '9' dibandingkan dengan '8', yang mana angka '9' lebih besar sehingga masuk ke dalam sorted list.

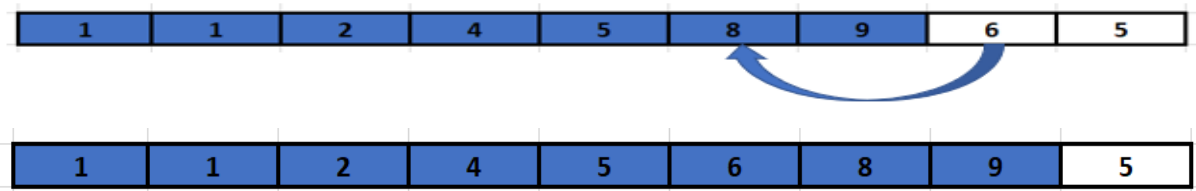


Tahap 6:

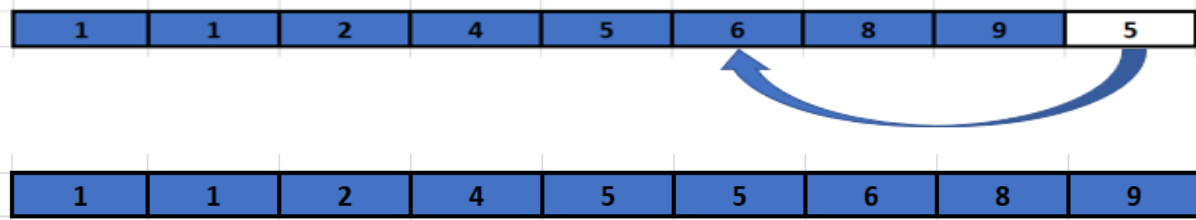
Angka pertama dalam unsorted list adalah '2'. Angka '2' dibandingkan dengan '4', yang mana angka '2' lebih kecil sehingga ditempatkan sesuai posisi yang benar.

**Tahap 7:**

Angka pertama dalam unsorted list adalah '6'. Angka '6' dibandingkan dengan '8', yang mana angka '6' lebih kecil sehingga ditempatkan sesuai posisi yang benar.

**Tahap 8:**

Angka pertama dalam unsorted list adalah '5'. Angka '5' dibandingkan dengan '6', yang mana angka '5' lebih kecil sehingga ditempatkan sesuai posisi yang benar. Pada tahap ini semua sudah masuk ke dalam sorted list.



b) Shell sort for the increment {1,3,5}

8	1	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

- Increment sequence for 5 sorted

8	1	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

Tahap 1: Angka '8' dibandingkan dengan '9'. Pada tahap ini tidak ada perubahan karena angka '8' lebih kecil.

8	1	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

Tahap 2: Angka '1' dibandingkan dengan '2'. Pada tahap ini tidak ada perubahan karena angka '1' lebih kecil.

8	1	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

Tahap 3: Angka '4' dibandingkan dengan '6'. Pada tahap ini tidak ada perubahan karena angka '4' lebih kecil.

8	1	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

Tahap 4: Angka '1' dibandingkan dengan '5'. Pada tahap ini tidak ada perubahan karena angka '1' lebih kecil.

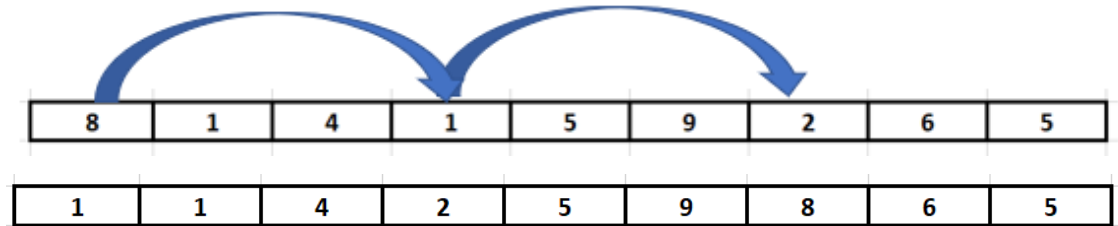
8	1	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

Hasil:

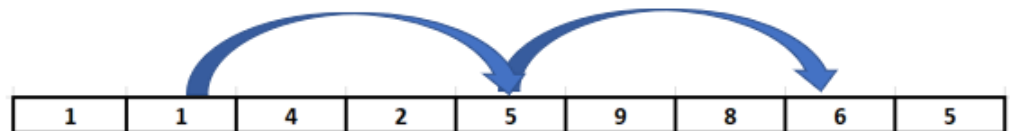
8	1	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

- Increment sequence for 3 sorted

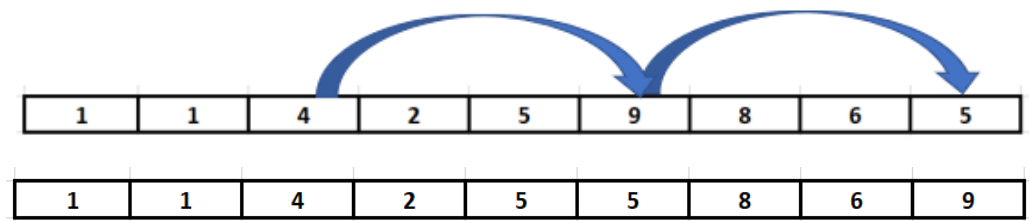
Tahap 1: Angka '8' dibandingkan dengan '1' dan '2'. Pada tahap ini terjadi perubahan urutan menjadi '1', '2', dan '8'.



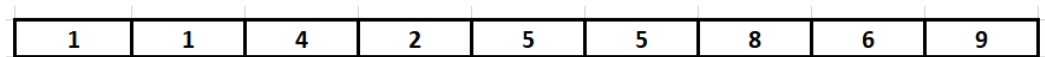
Tahap 2: Tahap 1: Angka '1' dibandingkan dengan '5' dan '6'. Pada tahap ini tidak terjadi perubahan urutan.



Tahap 3: Angka '4' dibandingkan dengan '9' dan '5'. Pada tahap ini terjadi perubahan urutan menjadi '4', '5', dan '9'.

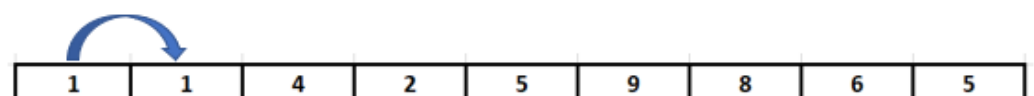


Hasil:

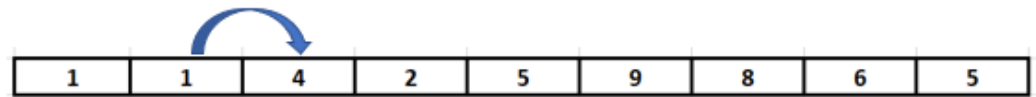


- Increment sequence for 1 sorted

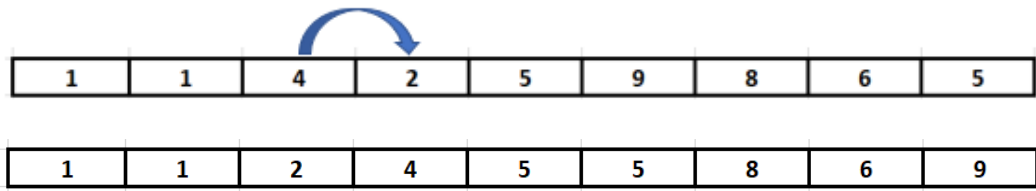
Tahap 1: Angka '1' dibandingkan dengan '1'. Tahap ini tidak mengalami perubahan urutan karena bernilai sama.



Tahap 2: Angka '1' dibandingkan dengan '4'. Tahap ini tidak terjadi perubahan karena nilai angka '1' lebih kecil.



Tahap 3: Angka '4' dibandingkan dengan '2'. Tahap ini terjadi perubahan posisi karena nilai '4' lebih besar.



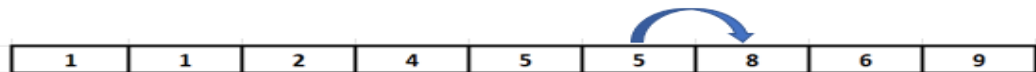
Tahap 4: Angka '4' dibandingkan dengan '5'. Tahap ini tidak terjadi perubahan karena nilai angka '4' lebih kecil.



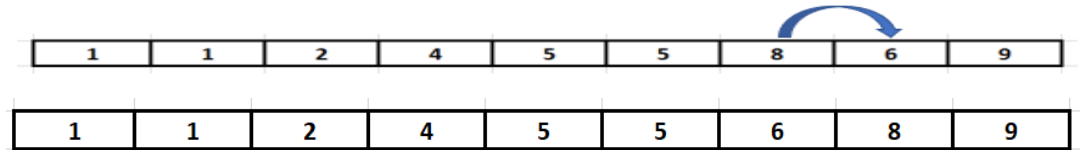
Tahap 5: Angka '5' dibandingkan dengan '5'. Tahap ini tidak mengalami perubahan urutan karena bernilai sama.



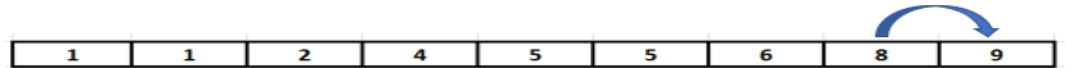
Tahap 6: Angka '5' dibandingkan dengan '8'. Tahap ini tidak terjadi perubahan karena nilai angka '5' lebih kecil.



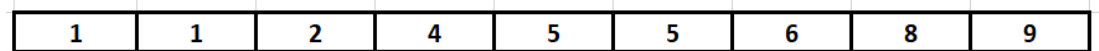
Tahap 7: Angka '8' dibandingkan dengan '6'. Tahap ini terjadi perubahan posisi karena nilai '8' lebih besar.



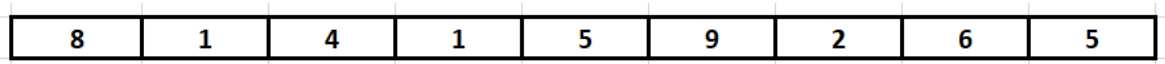
Tahap 8: Angka '8' dibandingkan dengan '9'. Tahap ini tidak terjadi perubahan karena nilai angka '8' lebih kecil.



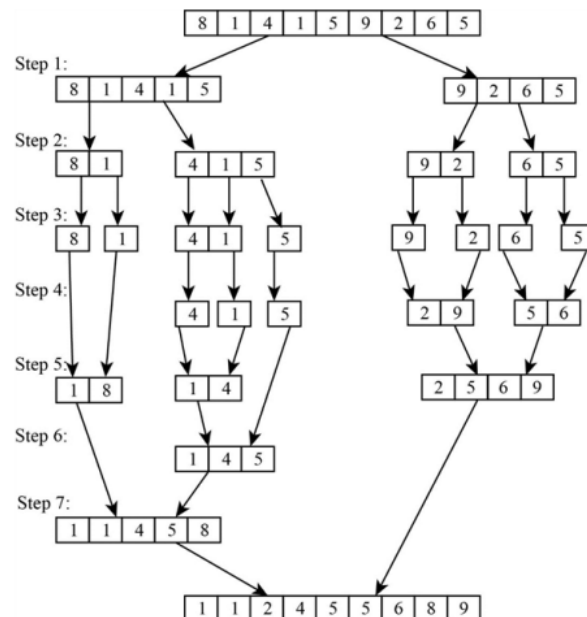
Hasil:



c) Merge sort



Merge sort memiliki ide dasar untuk membagi elemen menjadi dua sub arrays.



d) Quick sort with middle element as pivot with no cutoff

8	1	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

Pivot pada elemen adalah angka '5' karena berada di tengah elemen. Lalu, pivot diletakkan di urutan paling depan.

Tahap 1: Pointer 'i' bergeser ke kanan hingga menemukan nilai yang lebih besar dari pivot. Nilai yang lebih besar dari '5' adalah '8' sehingga 'i' berada di '8'.

5	1	4	1	8	9	2	6	5
pivot	i							j

Tahap 2: Pointer 'j' bergeser ke kiri hingga menemukan nilai yang lebih kecil dari pivot. Nilai yang lebih kecil dari '5' adalah '2' sehingga 'j' berada di '2'.

5	1	4	1	8	9	2	6	5
pivot				i				j

Tahap 3: Nilai pointer 'i' ditukar dengan 'j'.

5	1	4	1	8	9	2	6	5
pivot				i		j		

Tahap 4: Pointer 'i' bergeser ke kanan hingga menemukan nilai yang lebih besar dari pivot. Nilai yang lebih besar dari '5' adalah '9' sehingga 'i' berada di '9'.

5	1	4	1	2	9	8	6	5
pivot				i		j		

Tahap 5: Pointer 'j' bergeser ke kiri hingga menemukan nilai yang lebih kecil dari pivot. Nilai yang lebih kecil dari '5' adalah '2' sehingga 'j' berada di '2'.

5	1	4	1	2	9	8	6	5
pivot					i	j		

Tahap 6: Pointer 'i' dan 'j' saling bersilangan sehingga menukar elemen pivot dengan elemen pointer 'j'.

5	1	4	1	2	9	8	6	5
pivot				j	i			

Tahap 7: Pada tahap ini, sebelah kiri ‘pivot’ memiliki nilai lebih kecil dan sebelah kanan memiliki nilai lebih besar.

2	1	4	1	5	9	8	6	5
				pivot				

Array dibagi menjadi 2 set.

Set 1:

2	1	4	1
---	---	---	---

Set 2:

9	8	6	5
---	---	---	---

Set 1:

Elemen pertama unsorted list dibandingkan dengan list lain.

2	1	4	1
---	---	---	---

Elemen pertamanya adalah ‘1’ dibandingkan dengan ‘2’ sehingga terjadi pertukaran posisi.

1	2	4	1
---	---	---	---

Elemen pertamanya adalah ‘4’ dibandingkan dengan ‘1’ sehingga terjadi pertukaran posisi.

1	1	2	4
---	---	---	---

Set 2:

Elemen pertama unsorted list dibandingkan dengan list lain.

9	8	6	5
---	---	---	---

Elemen pertamanya adalah '9' dibandingkan dengan '8' sehingga terjadi pertukaran posisi.

8	9	6	5
---	---	---	---

Elemen pertamanya adalah '6' dibandingkan dengan '9' sehingga terjadi pertukaran posisi.

6	8	9	5
---	---	---	---

Elemen pertamanya adalah '5' dibandingkan dengan '9' sehingga terjadi pertukaran posisi.

5	6	8	9
---	---	---	---

Tahap 8: Pada tahap ini menggabungkan 2 set tadi dengan nilai elemennya adalah '5' (pivot).

1	1	2	4	5	5	6	8	9
---	---	---	---	---	---	---	---	---

e) Quick sort with median of three pivot selection and a cutoff of 3

8	1	4	1	5	9	2	6	5
---	---	---	---	---	---	---	---	---

Indeks [0] pada array ini adalah '8', elemen tengah adalah '5', dan elemen akhir adalah '5'. Elemen tadi diurutkan menjadi '5, 5, 8'. Elemen tengah (pivot) diletakkan di awal array.

Tahap 1: Pointer 'i' bergeser ke kanan hingga menemukan nilai yang lebih besar dari pivot. Nilai yang lebih besar dari '5' adalah '9' sehingga 'i' berada di '9'.

5	1	4	1	5	9	2	6	8
pivot	i							j

Tahap 2: Pointer 'j' bergeser ke kiri hingga menemukan nilai yang lebih kecil dari pivot. Nilai yang lebih kecil dari '5' adalah '2' sehingga 'j' berada di '2'.

5	1	4	1	5	9	2	6	8
pivot					i			j

Tahap 3: Nilai pointer 'i' ditukar dengan 'j'.

5	1	4	1	5	9	2	6	8
pivot					i	j		
5	1	4	1	5	2	9	6	8
pivot					j	i		

Tahap 4: Pada tahap ini, sebelah kiri 'pivot' memiliki nilai lebih kecil dan sebelah kanan memiliki nilai lebih besar.

2	1	4	1	5	5	9	6	8
					pivot			

Array dibagi menjadi 2 set.

Set 1:

2	1	4	1	5
---	---	---	---	---

Set 2:

9	6	8
---	---	---

Tahap 6: Set 1

Indeks [0] pada set 1 ini adalah '2', elemen tengah adalah '4', dan elemen akhir adalah '5'. Elemen tadi diurutkan menjadi '2, 4, 5'. Elemen tengah (pivot) diletakkan di awal array.

4	1	2	1	5
pivot	i			j

Pointer 'i' bergeser ke kanan hingga menemukan nilai yang lebih besar dari pivot. Nilai yang lebih besar dari '4' adalah '5' sehingga 'i' berada di '5'.

Tahap 7:

4	1	2	1	5
pivot				i,j

Pointer 'j' bergeser ke kiri hingga menemukan nilai yang lebih kecil dari pivot. Nilai yang lebih kecil dari '4' adalah '1' sehingga 'j' berada di '1'.

Tahap 8:

4	1	2	1	5
pivot			j	i

Kedua pointer "i" dan "j" saling bersilangan sehingga dapat menukar elemen "pivot" dengan elemen pointer "j".

Tahap 9:

1	1	2	4	5
			pivot	

Tahap 10: Set 2

Indeks [0] pada set 2 ini adalah '9', elemen tengah adalah '6', dan elemen akhir adalah '8'. Elemen tadi diurutkan menjadi '6, 8, 9'. Elemen tengah (pivot) diletakkan di awal array.

8	6	9
pivot	i	j

Pointer 'i' bergeser ke kanan hingga menemukan nilai yang lebih besar dari pivot.

Nilai yang lebih besar dari '8' adalah '9' sehingga 'i' berada di '9'.

Tahap 11:

8	6	9
pivot		i,j

Pointer 'j' bergeser ke kiri hingga menemukan nilai yang lebih kecil dari pivot.

Nilai yang lebih kecil dari '8' adalah '6' sehingga 'j' berada di '6'.

Tahap 12:

8	6	9
pivot	j	i

Kedua pointer "i" dan "j" saling bersilangan sehingga menukar elemen "pivot" dengan elemen pointer "j".

Tahap 13: Pada tahap ini menggabungkan kedua set menjadi seperti gambar di bawah ini.

1	1	2	4	5	5	6	8	9
---	---	---	---	---	---	---	---	---

2. Kerjakan soal 8.4, 8.5, dan 8.6, sertakan penjelasan pada laporan anda.

- 8.4** When all keys are equal, what is the running time of
- Insertion sort
 - Shellsort
 - Mergesort
 - Quicksort

Jawab :

a) Insertion sort

Pertimbangkan array 'a', jenis penyisipan akan mengurutkan array 'a[i]' dengan memilih elemen sebelumnya seperti 'a[i-1]'. Kemudian memeriksa apakah mereka "sama". Tidak memeriksa lagi apakah mereka sudah diurutkan sehingga perbandingan 'n-1' dibuat untuk algoritma ini.

- Jika semua keys 'equal' maka waktu berjalan untuk insertion sort adalah **$O(n)$** .

b) Shell sort

Pada shell sort, setiap nilai suburutan diatur ulang untuk mendapatkan daftar yang diurutkan hingga mencapai nilai '1'. Kinerja shell sort akan ditingkatkan hanya setelah menjalankan insertion sort. Pada array input yang diurutkan tidak diperlukan pertukaran untuk setiap pass, tetapi dengan melewati loop kedua akan membutuhkan waktu " $O(n)$ ". Total pass yang diperlukan untuk input sorted adalah " $O(\log n)$ ".

- Maka, running time untuk shell sort adalah **$O(n \log n)$** .

c) Merge sort

Pada merge sort, elemen array dibagi secara rekursif hingga mencapai elemen tunggal. Di merge sort, hanya pointer 'i' yang bergeser dalam array sedangkan pointer 'j' tidak berubah. Perbandingan untuk merge sort direduksi menjadi ' $n/2$ ' daripada mereduksi 'n'. Untuk menggabungkan setiap array diperlukan waktu " $O(n)$ ".

- Maka, running time yang diperlukan oleh merge sort adalah **$O(n \log n)$** .

d) Quick sort

Pada quick sort membutuhkan nilai pivot, yang mana akan ada pointer 'i' yang bergerak ke kanan untuk mencari nilai terbesar lebih dari pivot dan 'j' yang bergerak ke kiri untuk mencari nilai terkecilnya. Di quick sort, array dibagi menjadi dua bagian yang sama sehingga memiliki ukuran ' $n/2$ '. Pertukaran angka akan terjadi pada setiap langkah rekursif jika nilainya lebih besar atau kecil dari pivot. Namun, bernilai sama tidak akan mengalami perubahan.

- Jika semua key elemen sudah 'equal' maka waktu yang dibutuhkan selama quick sort adalah " **$O(n \log n)$** ".

- 8.5** When the input has been sorted, what is the running time of
- Insertion sort
 - Shellsort
 - Mergesort
 - Quicksort

Jawab :

a) Insertion sort

Misalkan terdapat array "A", Insertion sort akan mengurutkan array $A[i]$ dengan memilih elemen sebelumnya yakni $A[i-1]$. Kemudian akan memeriksa apakah mereka equal/sama. Ini tidak mengecek lagi apakah telah di sorted.

Perbandingan " $n-1$ " dibuat untuk algoritma. Sorted list akan menghasilkan case yang terbaik untuk insertion sort.

- Running time saat sorted list untuk insertion sort adalah : **$O(n)$** .

b) Shellsort

Pada "Shellsort", setiap nilai yang diurutkan diatur ulang untuk mendapatkan list yang diurutkan hingga mencapai nilai 1. Shellsort akan ditingkatkan apabila telah menjalankan insertion sort.

Untuk array input yang diurutkan, pertukaran tidak diperlukan untuk setiap pass. Namun, ketika melewati loop kedua, akan membutuhkan " $O(n)$ ". Total passes yang dibuat ulang untuk sorted input adalah : $O(\log n)$.

- Maka, running time saat sorted list untuk Shellsort adalah: **$O(n \log n)$** .

c) Mergesort

Pada Mergesort, element array dibagi secara rekursif hingga mencapai elemen tunggal. Dalam Mergesort, array diurutkan dengan menggabungkan list. Data dipecah menjadi 2 bagian, pertama bagian ini merupakan setengah genap(jika datanya adalah genap) atau setengah minus satu (jika datanya adalah ganjil) dari seluruh data yang ada.

Dalam Mergesort, data digabungkan kembali dengan cara membandingkan pada bagian blok yang sama, apakah data pertama itu lebih besar dari pada data ke-tengah+1. Jika lebih besar, maka data yang ke-tengah+1 dipindah sebagai data yang pertama, kemudian data pertama sampai tengah digeser menjadi data kedua sampai ketengah+1. Untuk menggabungkan setiap array, akan menghabiskan waktu $O(n)$.

- Maka, running time saat sorted list untuk Mergesort adalah: **$O(n \log n)$** .

d) Quicksort

Pada Quicksort, untuk memulai iterasi pengurutan, pertama-tama sebuah elemen dipilih dari data, kemudian elemen-elemen data akan diurutkan diatur sedemikian rupa.

Algoritma ini mengambil salah satu elemen secara acak (biasanya dari tengah) yang disebut dengan pivot. Kemudian angka yang lebih besar dari elemen pivot akan bergerak ke kanan, sedangkan angka lainnya akan bergerak ke kiri. Hal ini dilakukan secara rekursif terhadap elemen di sebelah kiri dan kanannya sampai semua elemen sudah terurut.

Wilayah yang dipartisi akan memiliki ukuran " $n/2$ ". Pertukaran angka terjadi untuk semua langkah rekursif karena dibandingkan dengan simbol "<" dan ">". Perbandingan tidak akan dilakukan dengan simbol "="

- Running time dari sorted list untuk Quicksort adalah : **$O(n \log n)$** .

- 8.6 When the input has been sorted in reverse order, what is the running time of
- Insertion sort
 - Shellsort
 - Mergesort
 - Quicksort

Jawab :

a) Insertion sort

Pada Insertion sort, ketika input telah diurutkan dalam urutan terbalik akan menghasilkan perbandingan case yang terburuk karena elemen yang diurutkan dalam arah sebaliknya akan menukar angka secara proporsional dengan angka yang saat ini yang ada dalam sorted list. Dengan demikian kompleksitas waktu “dalam kasus terburuk” yaitu $O(n^2)$

- Maka running time dari insertion sort untuk urutan terbalik adalah : **$O(n^2)$** .

b) Shellsort

Pada Shellsort, setiap nilai sub urutan diatur ulang untuk mendapatkan list yang diurutkan hingga mencapai nilai 1. Kinerja Shellsort akan ditingkatkan hanya setelah menjalankan insertion sort. “worst case” untuk shellsort ini didasarkan pada urutan kenaikan.

- Maka, running time dari Shellsort untuk urutan terbalik adalah : **$O(n \log n)$** .

c) Mergesort

Pada Mergesort, elemen array dibagi secara rekursif hingga mencapai elemen tunggal. Dalam Mergesort, array diurutkan dengan menggabungkan list. Untuk menggabungkan setiap array akan menghabiskan waktu “ $O(n)$ ”. Kompleksitas waktu untuk Mergesort adalah sama untuk semua yang terbaik, rata-rata, dan “worst case”.

- Maka, running time dari Mergesort untuk urutan terbalik adalah : **$O(n \log n)$** .

d) Quicksort

Pada Quicksort, nilai pivot diidentifikasi, maka bilangan yang lebih besar dari elemen pivot akan bergerak ke kanan, sedangkan bilangan lainnya akan bergerak ke kiri. Dalam Quicksort, array dipartisi menjadi dua bagian yang sama.

Memilih elemen pivot sebagai yang terakhir akan memperlambat proses. Karena itu akan menghasilkan kompleksitas “worst case / kasus terburuk”

- Maka, running time dari Quicksort untuk urutan terbalik adalah : **$O(n \log n)$** .

3. Kerjakan soal 8.29, silahkan didiskusikan dengan teman satu kelompok solusi yang paling sesuai untuk menyelesaikan persoalan pada soal ini. Algoritma apa yang kalian gunakan? Kenapa?

8.29 Write a simple sorting utility, *sort*. The *sort* command takes a filename as a parameter, and the file contains one item per line. By default the lines are considered strings and are sorted by normal lexicographic order (in a case-sensitive manner). Add two options: The *-c* option means that the sort should be case insensitive; the *-n* option means that the lines are to be considered integers for the purpose of the sort.

Kode Program :

```
package sorting_11s20018_11s20043;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Scanner;

/**
 *
 * @author 11S20018_11S20043
 */
```

```

public class Sort_11S20018_11S20043 {
    public static void main(String[] args) throws Exception{
        Scanner s = new Scanner(System.in);
        String ch;
        System.out.println("c- case insensitive\nn- case sensitive");
        ch = s.next();

        //checks for the option "n"
        if (ch.charAt(0) == 'n'){
            //declare and initialize reader
            BufferedReader reader = null;
            try{
                InputStream is =
Sort_11S20018_11S20043.class.getResourceAsStream("./superwings.txt");
                reader = new BufferedReader(new InputStreamReader(is));
                String[] sortarray = new String[8];
                String temp;
                //loop executes until "i"
                for (int i = 0; i <= 7; i++){
                    //read the file line by line store in arraylist
                    sortarray[i] = reader.readLine();
                }

                //loop executes until "i" is less than
"sortarray.length"
                for(int i = 0; i < sortarray.length; i++){
                    //loop executes until "j" is less than
"sortarray.length"
                    for(int j = i + 1; j < sortarray.length; j++){
                        if (sortarray[i].compareTo(sortarray[j])>0){
                            //assign "sortarray[i]" to "temp"
                            temp = sortarray[i];
                            //assign "sortarray[i]" to "sortarray[j]"
                            sortarray[i] = sortarray[j];
                            //assign "temp" to "sortarray[j]"
                            sortarray[j] = temp;
                        }
                    }
                }

                //print statement
                System.out.print("Names in Sorted Order\n");
                //checks for the condition
                for (int i = 0; i < sortarray.length; i++){
                    //print statement

```

```

        System.out.println(sortarray[i]);
    }

    //print statement
    System.out.println("-----");
}

//exeception is thrown
catch (IOException e){
    //prints Stack Trace
    e.printStackTrace();
}

finally{
    try{
        //checks whether "reader" not equal to null
        if (reader != null){
            //closing of reader file
            reader.close();
        }
    }

    // IO exception
    catch (IOException e){
        //prints Stack Trace
        e.printStackTrace();
    }
}

//checks for the option "c"
else if (ch.charAt(0) == 'c'){
    //declare and initialize reader
    BufferedReader reader = null;
    //executes try option
    try{
        //object is created
        ArrayList line = new ArrayList();
        //read the data from file
        InputStream is =
Sort_11S20018_11S20043.class.getResourceAsStream("./superwings.txt");
        reader = new BufferedReader(new InputStreamReader(is));
        //reads the file line by line and store in arraylist"
        String currentLine = reader.readLine();
        //checks for the condition

```

```

        while (currentLine != null){
            //adds the currentLine
            line.add(currentLine);
            //reads the file line by line and store in currentLine
            currentLine = reader.readLine();
        }

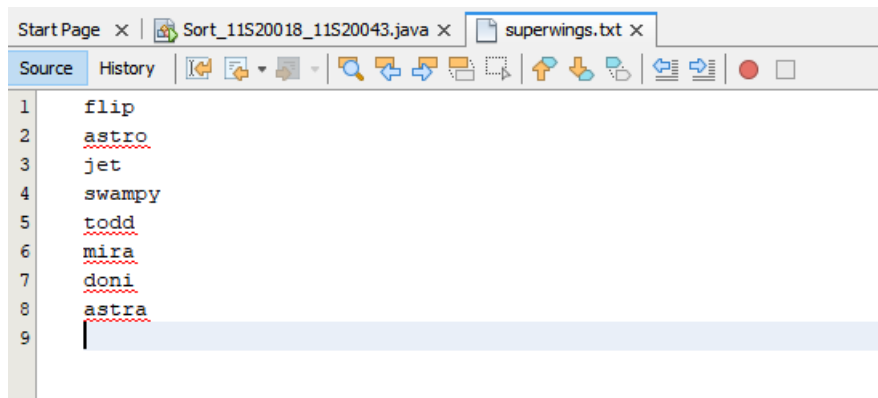
        //sort the string with insensitive order
        Collections.sort(line, String.CASE_INSENSITIVE_ORDER);
        System.out.println("Names in Sorted Order");
        //loop executes until "line"
        for(Object num : line){
            System.out.println(num);
        }
    }

    //exception is thrown
    catch (IOException e){
        //prints Stack Trace
        e.printStackTrace();
    }

    finally{
        //executes try option
        try{
            //checks for the condition
            if (reader != null){
                //closing of reader file
                reader.close();
            }
        }
        //exception is thrown
        catch (IOException e){
            //prints StackTrace
            e.printStackTrace();
        }
    }
}
}
}

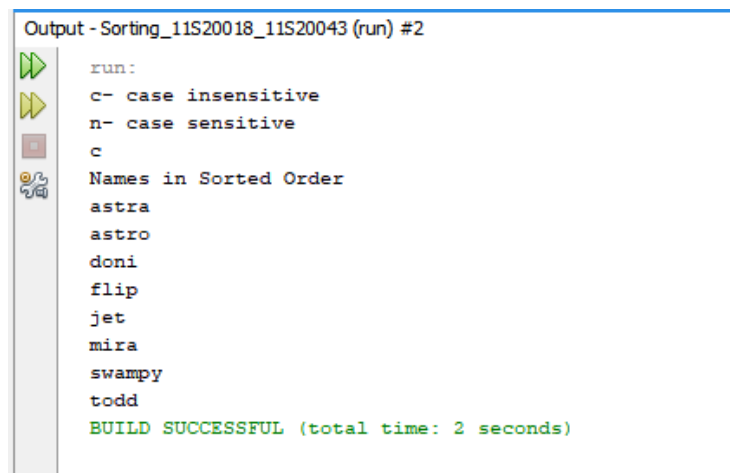
```

Input files data :

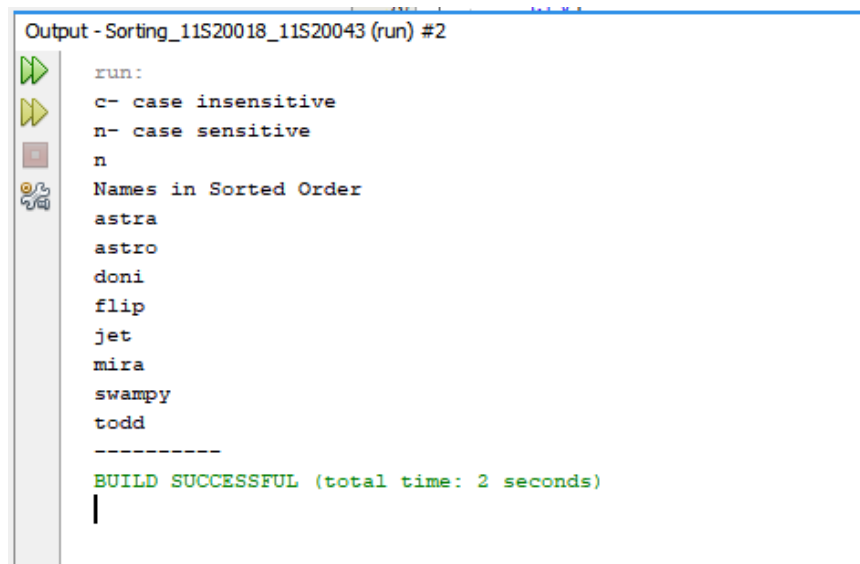


```
Start Page x | Sort_11S20018_11S20043.java x | superwings.txt x
Source History
1 flip
2 astro
3 jet
4 swampy
5 todd
6 mira
7 doni
8 astra
9 |
```

Output :



```
Output - Sorting_11S20018_11S20043 (run) #2
run:
c- case insensitive
n- case sensitive
c
Names in Sorted Order
astra
astro
doni
flip
jet
mira
swampy
todd
BUILD SUCCESSFUL (total time: 2 seconds)
```



```
Output - Sorting_11S20018_11S20043 (run) #2
run:
c- case insensitive
n- case sensitive
n
Names in Sorted Order
astra
astro
doni
flip
jet
mira
swampy
todd
-----
BUILD SUCCESSFUL (total time: 2 seconds)
|
```

Algoritma:

- Sertakan file header yang diperlukan.
- Class dibuat untuk "Sort".
 - Deklarasi operasi utama.
- Cetak pernyataan untuk memeriksa opsi "c" dan "n".
- Periksa apakah karakter yang ditentukan sama dengan "n".
- Deklarasikan dan inisialisasi pembaca dan dapatkan data dari file.
- Simpan nilai array dalam daftar array.
- Loop dijalankan sampai "i" kurang dari atau sama dengan tujuh.
 - Membaca file dan menyimpannya dalam daftar array.
- Loop dijalankan sampai "i" kurang dari "sortarray.length".
 - Loop dijalankan sampai "j" kurang dari "sortarray.length".
- Perbandingan dibuat untuk "sortarray[i]" dan "sortarray[j]".
- Swapping dilakukan untuk "sortarray[i]" ke "temp".
- Assign "sortarray[j]" ke "sortarray[i]".
- Tetapkan "temp" ke "sortarray[j]".
- Cek sampai kondisi "i ".
- Mencetak "sortarray[i]".
- Pengecualian dilempar dan kemudian mencetak StackTrace.
- Jalankan opsi coba dan kemudian periksa apakah "pembaca" tidak sama dengan nol.
- Tutup "reader file".
 - Menangkap untuk melempar IOExceptions.
- Mencetak "StackTrace".
- Sebaliknya Periksa apakah karakter yang ditentukan sama dengan "c".
- Deklarasikan dan inisialisasi pembaca dan dapatkan data dari file.
- Membaca file baris demi baris dan menyimpannya dalam daftar array.
- Periksa kondisi "currentLine" tidak sama dengan null.
 - Menambahkan "currentLine".
 - Membaca file baris demi baris dan menyimpannya di currentLine.

- Urutkan string dengan urutan yang tidak sensitif.
- Mencetak pernyataan "garis".
- Pengecualian dilempar dan kemudian mencetak StackTrace.
- Mengeksekusi opsi try dan kemudian memeriksa apakah "reader" tidak sama dengan null.
- Tutup “reader file”.
- Menangkap untuk melempar IOExceptions.
- Mencetak "StackTrace”