Problem 1. Pengetahuan Tambahan

Algoritma sorting yang sudah Anda pelajari (*insertion sort*, *shell sort*, *merge sort*, *quick sort*, dsb) merupakan algoritma sorting yang berbasis pada **perbandingan elemen (*comparison*)**, artinya algoritma tersebut pasti membutuhkan operasi perbandingan 2 buah elemen. Apakah ada algoritma sorting yang **tidak** menggunakan operasi **perbandingan** ?

Hint: lihat penjelasan algoritma **Bucket Sort** pada link berikut:

https://www.cs.cmu.edu/~adamchik/15-121/lectures/Sorting%20Algorithms/sorting.html

Problem 2. Implementasi Algoritma Sorting

Pelajari implementasi algoritma sorting yang diberikan bersamaan dengan latihan ini ! (**ShellSort.java**, **QuickSort.java**, dsb.)

Problem 3. Analisis Algoritma Sorting

Lengkapi tabel di bawah ini ! untuk setiap algoritma sorting yang disebutkan pada tabel, tuliskan juga kapan ***Best Case*** dan ***Worst Case*** terjadi ?

Tujuannya adalah agar Anda tidak hanya "menghafal" kompleksitas algoritma pada setiap kasus, tetapi Anda juga harus memahami proses analisis yang terjadi.

Algoritma	Best Case	Average Case	Worst Case
Bubble Sort		$O(N^2)$	
Selection Sort		$O(N^2)$	
Insertion Sort		$O(N^2)$	
Merge Sort		O(N log N)	
Quick Sort		O(N log N)	

Untuk Shell Sort, analisis *kasus rata-rata* dan *worst case* cukup kompleks. Kompleksitas Shell Sort sangat bergantung pada **pemilihan gap** (*increment sequence*).

Problem 4. Generic Sort

Perhatikan kode **InsertionSort.java** dan **GenericInsertionSort.java** ! File yang pertama merupakan implementasi *insertion sort* untuk array yang hanya berisi bilangan bulat. File yang kedua merupakan implementasi *insertion sort* untuk generic array, yaitu array yang berisi elemen/objek yang dapat dibandingkan (***Comparable***).

Sebenarnya, ada dua cara untuk mengimplementasikan algoritma sorting pada **generic array**. Cara yang pertama adalah seperti yang sudah disampaikan sebelumnya:

```
public static void genericSort(Comparable[] arr) {
    ...
}
```

Cara yang kedua adalah dengan menggunakan **Type Parameter**:

```
public static <T extends Comparable<? super T>>
                  void genericSort(T[] arr) {
    ...
}
```

Tugas Anda adalah implementasikan **generic sort** menggunakan algoritma *Quick Sort*, *Merge Sort*, dan *Shell Sort* ! Silakan pilih satu dari dua cara di atas !

Problem 5. QuickSelect

Sebuah permasalahan yang mirip dengan *sorting* adalah *selection* (pemilihan), atau **mencari elemen terkecil ke-k** dalam sebuah array. Misal, mencari sebuah median, atau elemen terkecil **ke-N/2**. Dengan mengubah sedikit *quick sort*, kita bisa menyelesaikan masalah *selection* ini dalam **O(N) secara rata-rata**.

Langkah-langkah untuk **Quickselect(array, k)** adalah sebagai berikut:

1. Jika banyaknya elemen di dalam array adalah 1, dan **k** juga bernilai 1, kita kembalikan elemen tersebut sebagai hasil.
2. Pilih elemen pivot **v** dari array.
3. Partisi array ke subarray **L** dan subarray **R** berdasarkan pivot **v**, persis sama dengan proses partisi pada *quick sort*.
4. Ada tidak kasus:
    a. Jika **k** kurang dari atau sama dengan banyaknya elemen di **L**, maka elemen yang kita cara ada di **L**. Panggil **Quickselect(L, k)** secara rekursif.
    b. Jika **k = 1 + |L|**, maka pivot **v** adalah elemen terkecil ke-**k**.
    c. Selain itu, elemen yang kita cari pasti ada di **R**. Kita dapatkan elemen tersebut dengan **Quickselect(R, k - |L| - 1)**.

Implementasikan algoritma **Quickselect** ini !