SDA - Latihan Rekursif

Kerjakan pada buku catatan masing-masing

1. **sum**

a. Implementasikan method **rekursif sum (N)** yang mengembalikan nilai

sum(N) = 1 + 2 + 3 + ... + N

```
public static int sum (int N) {



}
```

b. Hitunglah **Running Time / Kompleksitas Waktu** dari algoritma rekursif **sum** di atas menggunakan **relasi rekurensi** !

c. Implementasikan versi **Tail-Recursive** dari fungsi **sum** sebelumnya !

d. Mana versi yang lebih efisien ? versi pada soal a (non-tail-recursive) atau soal c (tail-recursive) ? jelaskan mengapa ?

2. **Palindrome Array**

Anda pasti sudah mengenal **Palindrom**. Berikut adalah contoh dari Array palindrom.

23	5	3	10	3	5	23

a. Implementasikan method **rekursif palindrom** yang mengembalikan **true** jika **array of integers** masukan bersifat palindrom, dan **false** jika sebaliknya.

```
public static boolean palindrom (int[] arr, int start, int end) {



}
```

b. apakah method rekursif yang Anda implementasikan sebelumnya sudah dalam bentuk **Tail-Recursive** ?

c. berapakah running time/kompleksitas waktu dari algoritma yang Anda buat sebelumnya ?

3. **Fibonacci**

```
public static int fib (int n) {
    if (n<=1) return n;
    return fib(n-1) + fib(n-2);
}
```

a. Algoritma rekursif fibonacci di atas **tidak efisien**. Mengapa tidak efisien ?

b. Versi yang lebih efisien dapat diimplementasikan menggunakan prinsip **Dynamic Programming**. Apa itu prinsip **Dynamic Programming** ? Silakan Anda latihan membuat versi algoritma yang menggunakan Dynamic Programming ! (silakan lihat slide untuk solusinya)

hasil komputasi dari masalah/input yang lebih kecil disimpan untuk dapat digunakan oleh masalah/input yang lebih besar (menghindari duplikasi proses untuk input yang sama).

c. Versi yang lebih efisien berikutnya adalah dengan menggunakan **Tail-Recursive**. Coba Anda implementasikan versi Tail-Recursive dari algoritma fibonacci yang tidak efisien tersebut !

4. **Fungsi Power Efisien**

Berikut adalah fungsi **power (x, n)** untuk **n ≥ 0**:

```
public static int power (int x, int n) {
    if (n == 0) {
       return 1;
    } else {
       return x * power (x, n-1);
    }
}
```

a. Hitunglah running time/kompleksitas waktu dari fungsi di atas !

b. Ubahlah fungsi rekursif di atas menjadi versi ***Tail-Recursive*** !

c. Ubahlah fungsi rekursif di atas menjadi versi yang lebih efisien dalam hal kompleksitas waktu, yaitu **O(log n)** !

5. **Binary Search**

a. Lengkapi potongan kode di bawah sehingga menghasilkan algoritma **binary search <u>versi rekursif</u>** untuk mencari elemen target pada **array yang sudah terurut membesar**.

```
public static int binarySearch(int[] data, int target, int low, int high) {
      if (low <= high) { // mengapa ini diperlukan ?
      } else {
      }
}
```

b. Hitunglah running time/kompleksitas waktu dari algoritma di atas menggunakan **relasi rekurensi** !

c. Bisakah kita katakan binary search versi rekursif tersebut menerapkan prinsip **Divide and Conquer** ? mengapa ?

6. Perhatikan method misteri berikut:

```
public static int misteri(int[] a) {
    return misteriHelp(a, 0, a.length-1);
}

private static int misteriHelp(int[] a, int left, int right) {
    if (left == right) {
       return a[left];
    } else if (right-left == 1) {
       return a[left] < a[right] ? a[left] : a[right]
    } else {
       int center = (left + right) / 2;
       int part1  = misteriHelp(a, left, center);
       int part2  = misteriHelp(a, center+1, right);
       return part1 < part2 ? part1 : part2;
   }
}
```

a. Apakah yang dilakukan oleh method tersebut ? **Tips:** lakukan tracing untuk input kecil.

b. Apakah algoritma tersebut menerapkan prinsip ***Divide and Conquer*** ?