UNIVERSITAS INDONESIA
FAKULTAS ILMU KOMPUTER
Veritas, Probitas, Justitia

CSGE602040 – Struktur Data & Algoritma
UTS Semester Gasal 2019/2020
26 Oktober 2018 09.00 – 11.30

Pengajar: Suryana Setiawan, Rahmad Mahendra,
Iis Solichah, Dipta Tanaya

PERNYATAAN KESANGGUPAN MENAATI TATA TERTIB UJIAN

"Saya telah membaca dan memahami ketentuan tata tertib berikut ini, serta menyatakan bahwa jawaban ujian ini adalah hasil pekerjaan saya sendiri. Saya menyetujui jika melakukan pelanggaran atas ketentuan tersebut, saya bersedia diproses sesuai ketentuan yang berlaku (SK DGB UI No.1 Tahun 2014) dengan sanksi maksimal **nilai akhir E**."

Nama & Tanda-tangan:	Kelas:	Nomor Pokok Mahasiswa:

TATA TERTIB UJIAN

- Semua alat komunikasi elektronik dalam kondisi non-aktif (dimatikan), dimasukkan ke dalam tas dan diletakkan pada tempat yang telah disediakan.
- Peralatan ujian yang boleh dibawa adalah alat tulis dan yang diperbolehkan sesuai sifat ujian.
- Peserta ujian menempati tempat duduk yang telah ditentukan.
- Peserta ujian menuliskan nama dan NPM pada setiap lembar jawaban ujian.
- Peserta mulai membuka soal dan mengerjakan ketika pengawas mengatakan ujian dimulai dan berhenti bekerja (meletakkan alat tulis) ketika pengawas mengatakan waktu habis.
- Peserta tidak berkomunikasi dalam bentuk apapun dengan peserta lain selama berada di ruang ujian, termasuk pinjam meminjam alat tulis, serta tidak memberi atau menerima bantuan dari siapapun selama ujian berlangsung.
- Peserta yang meninggalkan ruang ujian dianggap selesai mengerjakan. Jika karena kondisi medis khusus tidak bisa memenuhi ketentuan ini, peserta wajib melaporkan kepada pengawas sebelum ujian dimulai.
- Setelah selesai mengerjakan atau setelah waktu habis, peserta segera meninggalkan berkas soal dan lembar jawaban ujian di meja masing-masing, mengambil tas dan segera keluar tanpa mengganggu peserta lain serta tanpa berkomunikasi dengan peserta lain.
- Jawaban ujian ini tidak akan dinilai jika pernyataan di atas ini tidak ditandatangani.

Informasi Tambahan

- Sifat ujian: open notes (1 lembar A4)

Nama		Kelas	
NPM		No. Meja	

Bagian A (40 poin)

Bagian A terdiri atas soal isian pendek dan pilihan ganda. Untuk isian pendek, jawablah di tempat yang telah disediakan. Untuk pilihan ganda, lingkari jawaban yang benar.

1. Tentukan kompleksitas *method* guessMe(int N) dalam notasi Big-O.

```
public static void guessMe (int N) {
    int i = 10;
    while (i <= N){
        for (int j = 4; j <= N; j *= 4)
            System.out.println("i, j adalah " + i + ", " + j);
        i++;
    }
}
```

O(N log N)

2. Tentukan kompleksitas *method* guessWhat(int M, int N) dalam notasi Big-O.

```
public static void guessWhat (int M, int N) {
    for (int i = 5; i <= M; i += 4)
        System.out.println("i adalah " + i);
    for (int j = 5; j <= N; j *= 4)
        System.out.println("j adalah " + j);
}
```

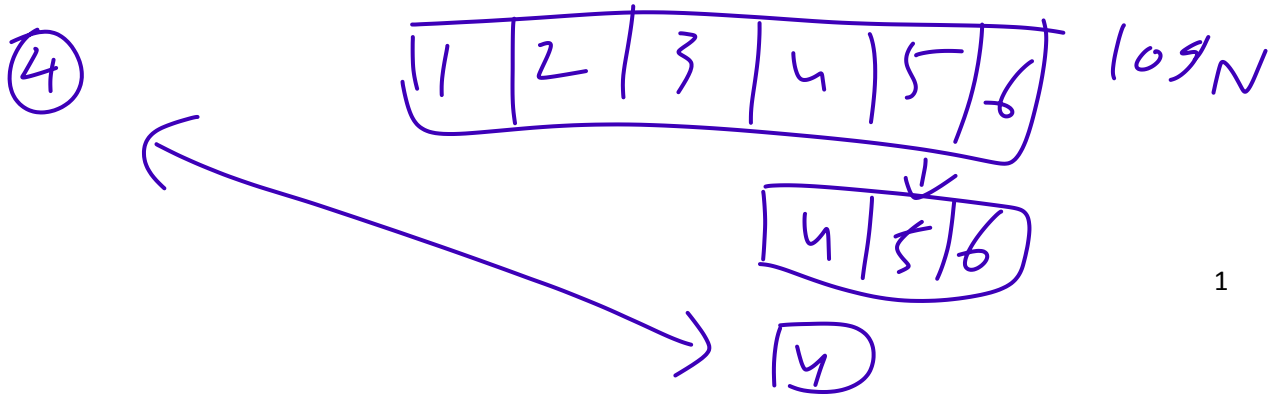
3. Eksekusi sebuah algoritme memerlukan waktu 300 detik untuk jumlah masukan (N) sebanyak 30. Jika waktu yang dibutuhkan untuk memproses masukan sebanyak 240 adalah 7200 detik, maka kompleksitas dari algoritme tersebut dalam notasi Big-O adalah:
(Asumsikan bahwa *running time* hanya dipengaruhi oleh kompleksitas algoritma dan ukuran *input*)

O(n log n)

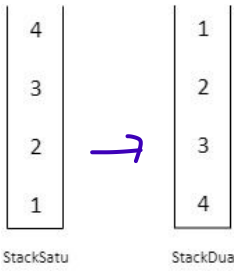
4. Sebuah algoritme memerlukan waktu 10 detik untuk memproses sejumlah 75 data. Jika kompleksitas algoritme tersebut adalah $O(N^2)$, berapakah waktu yang dibutuhkan untuk mengolah sejumlah 150 data?
(Asumsikan bahwa *running time* hanya dipengaruhi oleh kompleksitas algoritma dan ukuran *input*)

40

5. Pencarian elemen pada Abstract Data Type List yang memiliki kompleksitas logaritmik dikenal sebagai algoritme binary search



6. Lengkapi potongan *code* berikut agar kita dapat memindahkan isi StackSatu ke StackDua dengan posisi kebalikannya seperti pada gambar berikut:




```
while(!StackSatu.empty()){
    Stackdua.push( Stacksatu.pop() )
}
```

7. Diberikan implementasi suatu ADT yang diimplementasikan menggunakan ArrayList sebagai berikut. ADT apakah yang dimaksud?

```
public class ADTYangDimaksud{
    ArrayList daftar;

    public ADTYangDimaksud(){
        daftar = new ArrayList();
    }

    public void tambahElemen(Object temp){
        daftar.add(temp);
    }

    public Object ambilElemen(){
        if(!daftar.isEmpty())
            return daftar.remove(daftar.size()-1);
        return null;
    }
}
```

remove paling blkg

Jawaban : Stack

8. *Output* program di bawah ini adalah

```
import java.util.Set;
import java.util.HashSet;

public class BelajarHimpunan {
   public static void main(String[] args) {
      Set<String> kotaku = new HashSet<String>();
      kotaku.add("Padang");
      kotaku.add("Makassar");
      kotaku.add("Banjarmasin");
      kotaku.add("Bandung");
      kotaku.add("Banjarmasin");
      kotaku.remove("Banjarmasin");
      kotaku.remove("Makassar");
      System.out.print(kotaku.size());
   }
}
```

1. padang
2. padang, makassar
3. padang, makassar, banjarmasin
4. padang, makassar, banjarmasin, bandung
5. padang, makassar, banjarmasin, bandung
6. padang, makassar, bandung
7. padang, bandung

ouput= 2

Output :

9. Berapakah kompleksitas dari algoritme Selection Sort (dalam notasi Big-O) jika data *input* sudah terurut? ..O(n²)..

Nama		Kelas	
NPM		No. Meja	

10. Operasi *swap* adalah penukaran posisi dua elemen pada sebuah *array*
    Diberikan data awal: 26 45 31 18 36 65 54
    Berapa kali operasi *swap* yang dilakukan oleh algoritma Selection Sort (cari max di dalam *unsorted part*, lalu sisipkan max ke dalam *sorted part*) sampai array dalam posisi terurut (*ascending*/menaik)?

    0: 26 45 31 18 36 65 54
    1: 26 45 31 18 36 54 65
    2: 26 45 31 18 36 54 65
    3: 26 36 31 18 45 54 65
    4: 26 18 31 36 45 54 65
    5: 26 18 31 36 45 54 65
    6: 18 26 31 36 45 54 65

11. Diberikan *array* berikut ini:
    52 23 10 40 5 12 6
    Tuliskan isi *array* setelah dilakukan *3-sort* (tahapan dari algoritma Shell Sort dengan *gap* = 3):

12. Kompleksitas proses *merging* pada algoritme Merge Sort adalah ...

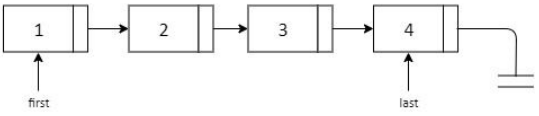
13. Berikut ini adalah method implementasi operasi merge terhadap dua `int[]` yang sudah terurut, yaitu A dan B, menjadi sebuah `int[] C`. Diketahui `C.length = A.length + B.length`.

```
private static void merge(int[] A, int[] B, int[] C){
    int cA = 0; int cB = 0; int cC = 0;
    while(cA < A.length && cB < B.length)
    ..............................
    while(cA < A.length){
        C[cC] = A[cA]; cC = cC+1; cA = cA+1;
    }
    while(cB<B.length){
        C[cC] = B[cB]; cC = cC+1; cB = cB+1;
    }
}
```

    Manakah kode yang benar untuk melengkapi baris yang dikosongkan di atas?

    a. `C[cC++] = (A[cA]<B[cB]) ? A[cA++] : B[cB];`
    b. `C[++cC] = (A[cA]<B[cB]) ? A[++cA] : B[++cB];`
    c. `C[cC++] = (A[cA]<B[cB]) ? A[cA++] : B[cB++];`
    d. `C[cC++] = (A[cA]<B[cB]) ? A[cA] : B[cB++];`
    e. `C[++cC] = (A[cA]<B[cB]) ? A[cA++] : B[cB++];`

14. Diberikan sebuah Singly Linked-list. Terdapat dua pointer tambahan yaitu **first** dan **last**. **First** menunjuk elemen pertama pada Linked-list dan **last** menunjuk elemen terakhir dari Linked-list seperti gambar berikut.

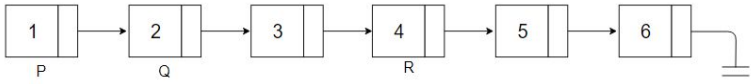


    Dari operasi berikut, operasi mana yang tidak dapat dilakukan dalam waktu O(1) / konstan?

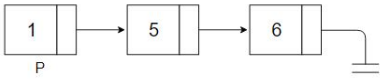
    a. *Insertion* sebuah elemen pada awal Linked-list
    b. *Deletion* elemen pertama Linked-list
    c. *Insertion* sebuah elemen di akhir Linked-list
    d. *Deletion* elemen terakhir Linked-list

15. Dalam implementasi sebuah Linked-list, terdapat tiga *pointer* yang menunjuk ListNode yang berbeda: P, Q, dan R. P != Q, P != R, dan Q != R. Posisi *node* yang ditunjuk oleh P dijamin berada di sebelum *node* yang ditunjuk Q dan posisi *node* yang ditunjuk Q dijamin berada di sebelum *node* yang ditunjuk R. Posisi *node* yang ditunjuk Q dijamin berada tepat setelah posisi *node* yang ditunjuk P.
    Berikan perintah (kode singkat) agar penghapusan serangkaian *node* sesuai ilustrasi di bawah ini (*node* yang dihapus dari Q sampai R) dapat dilakukan

    Linked-list awal:

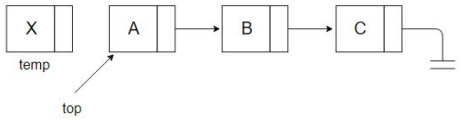
    

    Linked-list setelah proses penghapusan:

16. Sebuah struktur data Queue diimplementasikan dengan menggunakan *circular array*.
    Dua variabel digunakan untuk menandai posisi **first** (posisi data yang akan di-*dequeue*) dan **last** (posisi data yang akan di-*enqueue* ditempatkan).
    Jika ukuran *array* = 7, serta variabel **first** dan **last** ketika diinisialisasi pertama kali masing-masing diset 0.
    Berapa nilai variabel **last** setelah dilakukan serangkaian operasi pada queue sebagai berikut:
    Enqueue 5x, dilanjutkan dengan dequeue 2x, kemudian enqueue 3x, dan dequeue 4x

17. Berikut ini digambarkan sebuah struktur data Stack yang dibangun dengan Linked-list. Letak *top* sesuai dengan ilustrasi, yaitu di sebelah kiri. Top harus selalu menunjuk ke elemen yang siap di-*pop* dari Stack. Terdapat sebuah ListNode baru yaitu temp, yang berisi data X.

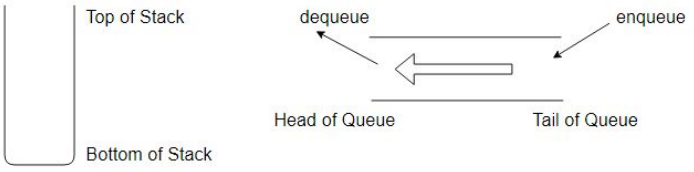
    

    Lengkapilah *code* di bawah ini untuk mengimplementasikan *method* push untuk menambahkan sebuah ListNode ke dalam Stack, dan menempatkan top pada posisi yang tepat.

```
public void pushToStack(ListNode temp){

}
```

Nama		Kelas	
NPM		No. Meja	

18. Diketahui Stack dan Queue yang masing-masing sudah menyimpan N buah data.
Penamaan posisi pada Stack dan Queue mengacu pada gambar di bawah ini.



Jika pada Stack hanya diperbolehkan operasi push dan pop, dan pada Queue juga hanya diperbolehkan operasi enqueue dan dequeue (sesuai teori), manakah pernyataan berikut yang benar?

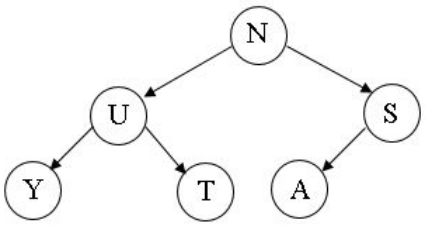
a. Operasi penyisipan elemen baru di posisi bottom of Stack lebih mahal dibanding penyisipan pada head of Queue
b. Operasi penghapusan elemen di posisi top of Stack lebih mahal dibanding penghapusan pada posisi tail of Queue
c. Operasi penghapusan elemen di posisi bottom of Stack lebih mahal dibanding penghapusan pada head of Queue
d. Operasi penyisipan elemen di posisi top of Stack lebih mahal dibanding penyisipan pada tail of Queue
e. Operasi penyisipan elemen baru pada posisi head of Queue lebih mahal dibanding penyisipan pada bottom of Stack

19. Method **search** menandai posisi dalam Maze yang dikunjungi sampai penjelajahan menemukan jalan keluar. Kompleksitas method **search** dalam notasi Big-O adalah ...

```
public class Maze {
   char[][] maze;
   /*
   maze diinisialisasi seukuran m x n
   maze[x][y] menyatakan posisi koordinat (x,y)
   nilainya 'V' jika posisi tersebut sudah pernah dikunjungi
   'W' berarti dinding labirin, 'E' berarti jalan keluar
   */

   public void search(int x, int y) {
      if(maze[x][y] == 'E') {
         System.out.println("Successfully exit from maze");
         return;
      }
      if(x >= 0 && y >= 0 && x < maze.length && y < maze[0].length){
         if(maze[x][y] != 'V' && maze[x][y] != 'W') {
            maze[x][y] = 'V';
            search(x+1, y);
            search(x-1, y);
            search(x, y+1);
            search(x, y-1);
         }
      }
   }
   ...
}
```

Jawaban :

20. Cetaklah elemen *tree* di bawah ini dengan mengeksekusi method **printSpecificOrder**



Jawaban:

```
class BinaryNode<E> {
   BinaryNode right, left;

   void printSpecificOrder() {
      if(right != null)
         right.printSpecificOrder();

      System.out.println(element);

      if(left != null)
         left.printSpecificOrder();
   }
}
class BinaryTree<E> {
   BinaryNode root;

   public void printSpecificOrder() {
      if(root != null)
         root.printSpecificOrder();
   }
}
```

Nama		Kelas	
NPM		No. Meja	

Bagian B (45 poin)

1. [9 poin] Binary Tree adalah *tree* di mana setiap *node*-nya memiliki *degree* maksimal 2. Ada sebuah *binary tree* yang setiap *node*-nya menyimpan suatu *integer*. Setiap *node* dari *binary tree* tersebut dicetak datanya menggunakan metode *in-order* dan *post-order.* Berikut ini adalah hasil cetak masing-masing:

   Hasil cetak *in-order traversal* sebagai berikut:
   3, 40, 12, 9, 13, 6, 7, 50, 4, 10, 5, 11, 8

   Hasil cetak *post-order traversal* sebagai berikut:
   3, 12, 13, 9, 6, 40, 50, 10, 4, 8, 11, 5, 7

   a. Gambarkan *binary tree* yang dimaksud:
      Jawab:

   b. Bagaimana hasil cetak *binary tree* tersebut jika menggunakan *pre-order traversal*?

2. [4 poin] Apa manfaat *header node* dalam implementasi Linked List? (Jelaskan maksimal 50 kata)

3. [16 poin] Perhatikan *code* berikut ini.

```
public class Something{
   public static void main(String[] args) throws Exception{
      int[] arr = {9, 8, 7, 6, 5, 4, 3};
      doSomething(arr);
   }

   static void cetak(int[] arr){
      for(int x : arr){
            System.out.print(x + " ");
      }
      System.out.println("");
   }

   static void doSomething(int a[]) throws Exception {
       for (int i = a.length-1; i>=0; i--) {
           boolean swapped = false;
           for (int j = 0; j<i; j++) {
               if (a[j] > a[j+1]) {
                   int temp = a[j];
                   a[j] = a[j+1];
                   a[j+1] = temp;
                   swapped = true;
               }
           }
           if (!swapped) return;
           swapped = false;
           for (int j = i-1; j > 0; j--) {
               if (a[j] < a[j-1]) {
                   int temp = a[j];
                   a[j] = a[j-1];
                   a[j-1] = temp;
                   swapped = true;
               }
           }
           cetak(a);
           if (!swapped) return;
       }
   }
}
```

a. Apa *output* dari program tersebut?

b. Apa yang dilakukan oleh *method* `doSomething(int[] a)`? (Jelaskan dalam maksimal 20 kata)

c. Saat kondisi *array* seperti apa yang dapat menyebabkan *method* `doSomething(int[] a)` mencapai kompleksitas O(N)? (N adalah *size* dari *array*)

Nama		Kelas	
NPM		No. Meja	

4. [9 poin]

```
static int iniRekursif (int angka) {
     if(angka > 1)
          return iniRekursif(angka/2) * iniRekursif(angka/2);
     return angka;
}
```

a. Implementasi method **iniRekursif** pada kode di atas kurang efisien. Jelaskan apa yang tidak efisien dalam implementasi di atas

Perbaiki kode di atas menjadi method rekursif yang lebih efisien

```
// implementasi yang lebih efisien
static int iniRekursif (int angka) {









}
```

b. Buatlah sebuah method non-rekursif yang fungsionalitasnya sama dengan method **iniRekursif**

```
static int bukanRekursif (int angka) {









}
```

Halaman ini tidak berisi soal

Nama		Kelas	
NPM		No. Meja	

5. [7 poin] Lengkapi potongan *code* berikut agar dapat menghitung kemunculan kata dalam sebuah String dan menyimpannya ke dalam TreeMap<String, Integer> (*key* dari TreeMap merupakan kata yang akan disimpan jumlah kemunculannya, *value* dari TreeMap merupakan jumlah kemunculan kata tersebut dalam String).

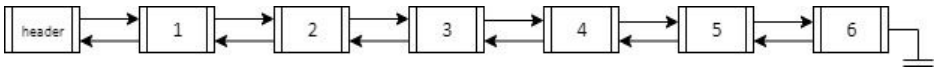
   Hint: Anda dapat menggunakan method berikut
   (source: https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html)

Modifier and Type	Method	Description
`boolean`	`containsKey(Object key)`	Returns true if this map contains a mapping for the specified key
`V`	`put(K key, V value)`	Associates the specified value with the specified key in this map
`V`	`get(Object key)`	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key

```
String text = "Good morning. Have a good day!";
Map<String, Integer> map = new TreeMap<String, Integer>();
String[] words = text.split("[ \n\t\r.,;:!?(){}]");

for (int i = 0; i < words.length; i++) {
    String key = words[i].toLowerCase();

    if (key.length() > 0) {









    }
}
```

Bagian C (20 poin)

1. Diketahui sebuah *sorted* Doubly LinkedList yang berisi bilangan bulat positif unik. Buatlah sebuah *method* yang menerima masukan **int X** dan **Node header**, yang mencetak pasangan data yang jumlahnya sama dengan **X**. Misal pasangan data yang dicetak adalah **a** dan **b**, maka posisi **a** berada di sebelah kiri posisi **b**. Contoh:



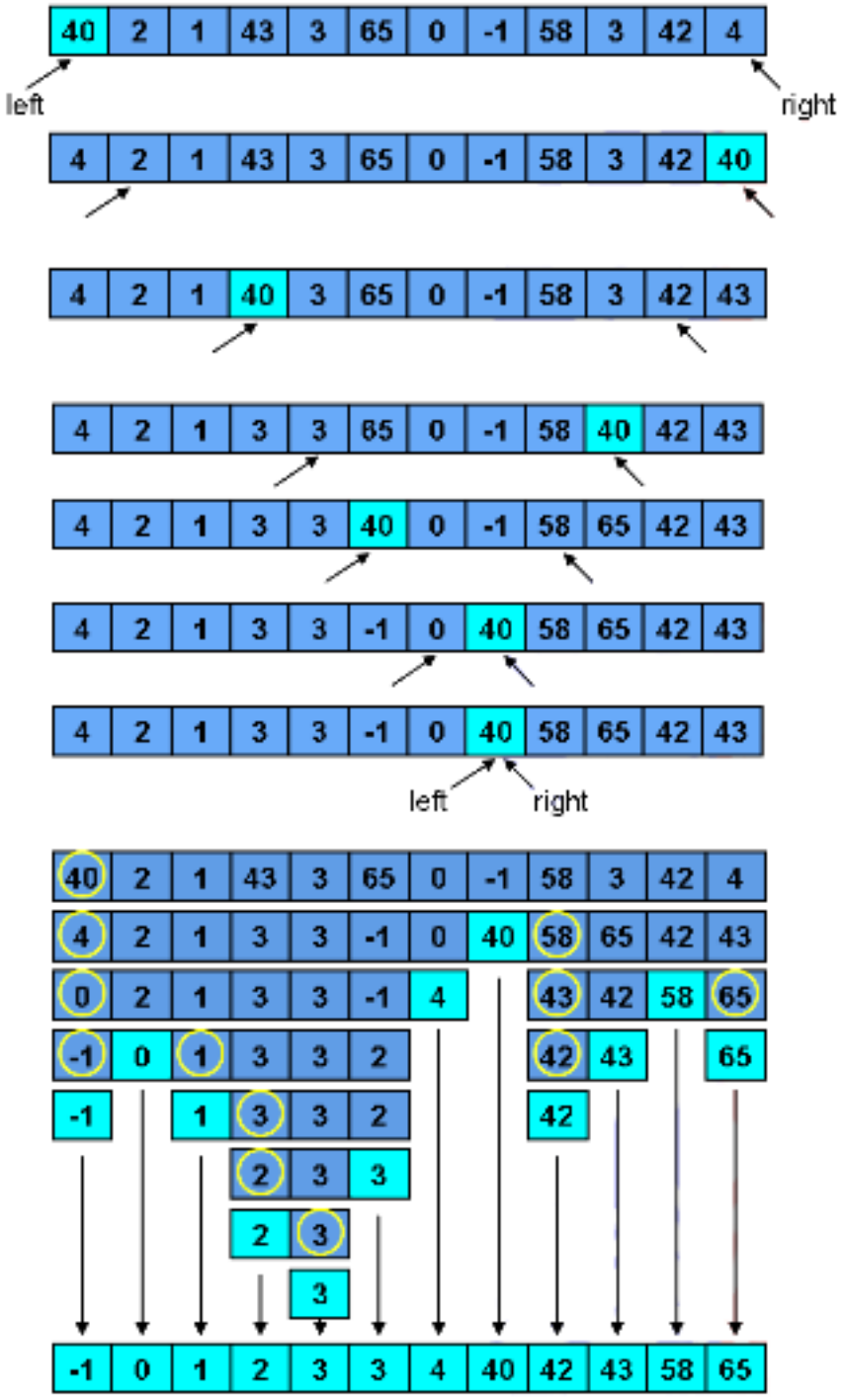
Pada Doubly LinkedList tersebut, jika nilai **X** yang diberikan adalah 5, maka *method* akan mencetak:
(1 4)
(2 3)

```
class Node{
    int data;
    Node next, prev;
}
class DoublyLinkedList{
    Node header;

    //[TO-DO] Anda dapat menambahkan argumen method bila diperlukan
    void cetakPasanganBilangan(Node header, int x               ){









    }
}
```

Nama		Kelas	
NPM		No. Meja	

2. Implementasikan kode Quick Sort dengan Partition Algorithm #1 yang dibahas di *slide* kuliah.
   - Pivot dipilih dari elemen pertama dalam (sub)-arr*ay* yang akan diurutkan
   - Pivot dibandingkan dengan elemen-elemen lain di array (ditandai dengan **left** / **right**).

   Sebagai ilustrasi, perhatikan contoh dari slide sebagai berikut.



```
static void quickSort(int[] arr, int left, int right) {




}
```

Selamat mengerjakan :)