Try-Out Ujian Tengah Semester

Jumat, 13 Oktober 2017

CSGE602040
Struktur Data & Algoritma
Fakultas Ilmu Komputer
Universitas Indonesia

NPM / Nama: _____ / _____

Peraturan UTS mengikuti aturan standar ujian Fasilkom UI. Tidak boleh menggunakan Internet ataupun alat bantu elektronik lainnya. Peserta ujian hanya boleh menggunakan catatan yang ditulis sendiri. Harap junjung kejujuran akademis.

Ujian ini terdiri dari 3 bagian:

1. Bagian A: Pilihan Ganda

   Terdapat 20 soal mencakup topik-topik perkuliahan SDA dari awal hingga pertengahan semester.

   Tuliskan jawaban dari soal-soal Pilihan Ganda di kotak jawaban yang tersedia di halaman pertama dokumen soal!

2. Bagian B: Isian Singkat

   Terdapat 6 soal mencakup topik-topik perkuliahan SDA dari awal hingga pertengahan semester.

   Tuliskan jawaban dari soal-soal Isian Singkat di kotak jawaban yang tersedia setelah setiap pertanyaan/soal!

3. Bagian C: Pemrograman Singkat

   Terdapat 4 soal mencakup topik pemrograman Java, rekursif, penggunaan ADT, dan implementasi *linked list*.

   Tuliskan jawaban dari soal-soal Pemrograman Singkat di kertas folio jawaban yang disediakan!

Kotak Jawaban Bagian A

No.	Jawaban		No.	Jawaban		No.	Jawaban		No.	Jawaban	
1			6			11			16		
2			7			12			17		
3			8			13			18		
4			9			14			19		
5			10			15			20		

SELAMAT MENGERJAKAN!

NPM / Nama: _____ / _____

Bagian A: Pilihan Ganda

1. Hasil analisa sebuah algoritma menyatakan bahwa ***total execution time*** algoritma tersebut merupakan fungsi dari dua input, yaitu: **m** dan **n**. Pilihlah fungsi Big-Oh dari fungsi *running time*: $\mathbf{f(m,n) = 5m \log n + 2n^2 + 10m + 150}$

   A. O($m \log n + n^2 + m$)
   C. O($m \log n + n^2$)
   E. O($m \log n + m$)
   B. O($m + n^2$)
   D. O($\log n + m$)

2. Sebuah algoritma memerlukan waktu 200 detik untuk jumlah input 20. Jika waktu yang dibutuhkan untuk memproses input sebanyak 160 adalah 4800 detik, maka kompleksitas dari algoritma tersebut adalah:

   A. O($n \log n$)
   C. O(n^2)
   E. O($(\log n)^2$)
   B. O(n)
   D. O(n^3)

3. Perhatikan fragmen program di bawah ini:

```
public static int methodKu(int n) {
    int sum = 0;
    for(int ii = -n; ii < n; ii++) {
        for(int jj = 0; jj < 100; jj++) {
            sum++;
        }
    }
    Return sum;
}
```

   Berapakah kompleksitas dalam notasi Big-Oh yang paling tepat untuk `methodKu`:

   A. O(n)
   C. O($n \log n$)
   E. O(200 n)
   B. O(n^2)
   D. O($2n + 100$)

4. Perhatikan fragmen program di bawah ini:

```
for(int ii = 0; ii < n; ii++) {
    for(int jj = 0; jj < n * n; jj++) {
        sum++;
    }
}
```

   Berapakah kompleksitas dalam notasi Big-Oh **yang paling tepat** untuk fragmen di atas:

   A. O(n^3)
   C. O($n \log n$)
   E. O($n * n^n$)
   B. O(n^2)
   D. O(n)

5. Apabila fungsi-fungsi berikut diurutkan dari yang laju pertumbuhannya paling kecil ke yang paling besar, manakah pernyataan berikut yang benar:

   A. $\log(N)$, N, $N \log(N)$, $N \log(N^2)$, N^2
   D. $\log(N)$, $N \log(N)$, N, $N \log(N^2)$, N^2
   B. N, $\log(N)$, N^2, $N \log(N)$, $N \log(N^2)$
   E. semua jawaban salah
   C. $\log(N)$, N, $N \log(N)$, N^2, $N \log(N^2)$

6. Sebuah algoritma perlu 10 detik untuk mengolah jumlah input 50. Jika algoritma tersebut memiliki kompleksitas *kubik*, berapa kira-kira waktu yang dibutuhkan untuk menyelesaikan masalah dengan jumlah input 100?
   A. 80 detik
   B. 70 detik
   C. 60 detik
   D. 50 detik
   E. 40 detik

7. Berapa jumlah maksimum iterasi yang dibutuhkan untuk mencari suatu bilangan dalam suatu *array* terurut (*sorted*) yang berisi 64 bilangan dengan menggunakan algoritma *binary search*?
   A. 6
   B. 10
   C. 3
   D. 8
   E. Semua jawaban salah

8. Berapa kompleksitas waktu yang dibutuhkan untuk melakukan proses *merging* dalam algoritma *merge sort*?
   A. Linear
   B. Logaritmik
   C. Kuadratik
   D. Kubik
   E. Tidak bisa ditentukan

9. Di antara algoritma pengurutan (*sorting*) berikut, manakah yang memiliki *running time* paling cepat secara umum?
   A. Quick sort
   B. Bubble sort
   C. Insertion sort
   D. Shell sort
   E. Selection sort

10. Di antara algoritma *sorting* berikut, manakah yang memiliki kompleksitas waktu mencapai O(*n*) pada kasus ekstrim?
    A. Bubble sort
    B. Quick sort
    C. Selection sort
    D. Shell sort
    E. Merge sort

11. Istilah *pivot* dikenal dalam algoritma *sorting*:
    A. Quick sort
    B. Bubble sort
    C. Insertion sort
    D. Merge sort
    E. Selection sort

12. *Worst case* pada pengurutan menaik dengan ***insertion sort*** terjadi pada data dengan pola:
    A. Terurut terbalik
    B. Acak
    C. Terurut
    D. Semua sama
    E. Setengah terurut

13. *Worst case* pada pengurutan menaik dengan ***quick sort*** selalu terjadi pada data dengan pola:
    A. Semua sama
    B. Terurut terbalik
    C. Acak
    D. Terurut
    E. Setengah terurut

14. Berikut ini adalah sejumlah masalah terkait algoritma rekursif, kecuali:
    A. Membutuhkan ruang memory untuk menyimpan berbagai informasi terkait eksekusi method sebelum melakukan pemanggilan.
    B. Setiap kemungkinan rekursif harus konvergent ke kasus-kasus dasar.
    C. Seringkali kompleksitas tersembunyi di balik simplisitas code rekursif.
    D. Seringkali terjadi pengulangan komputasi hal yang sama.
    E. Membutuhkan dukungan penggunaan struktur data stack secara eksplisit dalam implementasinya.

15. Berapa kali **misteri(0)** dipanggil ketika **misteri(10)**) dipanggil pada potongan kode di bawah?

```
void misteri(int N) {
    for(int i = N - 1; i >= 0; i--) {
        misteri(i);
    }
}
```

A. 10
B. 124
C. 55
D. 100
E. 40

Soa ini dibatalkan karena seharusnya jawabannya 512.

16. Suatu *linked-list* didefinisikan menggunakan suatu *class* bernama Node. Dalam Node terdapat *field* next untuk menunjuk Node berikutnya. Node pertama ditunjuk oleh variabel headNode. Jika suatu saat dalam *linked-list* sudah berisi minimal satu Node, suatu *node* baru yang ditunjuk newNode hendak disisipkan pada posisi *node* kedua dalam *linked-list*. Maka, manakah deretan perintah yang melakukan hal itu dengan benar:

A.
```
newNode.nextNode = headNode.nextNode;
headNode.nextNode = newNode.nextNode;
```

B.
```
headNode.nextNode = newNode.nextNode;
newNode.nextNode= headNode.nextNode;
```

C.
```
newNode.nextNode = headNode.nextNode;
headNode.nextNode = newNode;
```

D.
```
headNode.nextNode = newNode;
newNode.nextNode = headNode.nextNode;
```

E.
```
headNode = newNode.nextNode;
newNode = headNode.nextNode;
```

17. berikut ini merupakan kelebihan dari struktur data linked-list dibandingkan dengan array.
    A. Operasi pencarian umumnya lebih cepat dari array
    B. Operasi penghapusan node umumnya lebih lama dari array
    C. Operasi penyisipan umumnya lebih cepat dari array
    D. Operasi penghapusan di akhir umumnya lebih lama dari array
    E. Operasi penghapusah node pertama lebih cepat dari array

18. Diketahui suatu binary tree memiliki jumlah node 1500 dan **setiap** internal node di dalamnya memiliki degree 2 berikut ini adalah bilangan yang menyatakan tinggi binary tree. Jika H adalah tinggi tree, manakah dari berikut ini yang pasti bukan H dari binary tree tersebut?
    A. 65
    B. 129
    C. 605

D. 790 E. 21

Soal ini dibatalkan karena jumlah node 1500 tidak mungkin untuk jenis tree ini (jika semua internal node degree 2 maka jumlah node harus bilangan ganjil). Tapi jika jumlah node adalah 1501, maka tingginya antara 10 sampai dengan 750. Jadi tinggi yang tidak mungkin adalah 790.

19. Suatu binary tree berisi 6 node dan setian node dilabeli huruf-huruf unik. Pada tree dilakukan pencetakan label-label node secara **inorder** traversal dan diperoleh urutan pencetakan sebagai berikut: k, d, g, j, f, h. Jika dilakukan pencetakan semacam tapi secara **postorder** traversal, bagaimanakah urutan pencetakannya?

A. g, d, k, f, j, h
B. d, f, g, h, j, k
C. k, d, j, h, f, g
D. h, j, f, k, d, g
E. g, f, h, j, d, k

Soal ini ada dua jawaban yang benar.

20. Suatu Fibonacci Tree adalah binary tree yang memiliki sifat khusus, yakni pada setiap subtreenya tinggi dari subtree kiri dan subtree kanan (dari setiap subtree tersebut!) tepat berselisih satu. Berapa jumlah node pada suatu Fibonacci Tree dengan tinggi 6?

A. 10
B. 13
C. 15
D. 20
E. 2

Tidak ada jawaban, seharusnya 33, sepertinya ada kesalahan ketik, maksudnya tinggi 5, jawabannya 20.

Bagian B: Isian Singkat

1. Jalankan satu kali partisi (menurut Alg. Partisi versi 1 sesuai slide bahan kuliah) pada data 50, 38, 73, 91, 39, 58, 47, 74, 29, 18 dan tuliskan hasilnya saja (dipisahkan koma seperti semula)?

18, 38, 29, 47, 39, 50, 58, 74, 91, 73

2. Mengacu pada soal no 1 sebelumnya, jika dijalankan hingga selesai berapa kali algoritma partisi (Versi 1 itu) dijalankan pada data.

6 kali partisi Penjelasan: 1.Input: 50, 38, 73, 91, 39, 58, 47, 74, 29, 18 , Output: 18, 38, 29, 47, 39, 50, 58, 74, 91, 73 2.Input: 18, 38, 29, 47, 39, output: 18, 38, 29, 47, 39 3.Input: 38, 29, 47, 39, output: 29, 38, 47, 39 4.Input: 47, 39, outut 39, 47 5.Input: 58, 74, 91, 73, output: 58, 74, 91, 73 6.Input: 74, 91, 73, output: 73, 74, 91

3. Dalam operasi-operasi apakah (***insertion***, ***deletion***, ***update***, ***searching***, ***emptying***) yang menyebabkan sorted array lebih lambat dari unsorted array? Beri juga alasan singkat pendapat anda.

Insertion, deletion, update.

Penjelasan: insertion dan deletion memerlukan penggeseran. Update, jika yang diupdate keynya maka sama saja dengan delete dan insert kembali.

4. Sebuah algoritma memerlukan waktu 2 detik untuk jumlah input 100. Jika kompleksitas algoritma tersebut adalah O(n^3), berapa kira-kira waktu yang dibutuhkan untuk memproses input sebanyak 10.000?

2 juta detik
Penjelasan: data meningkat 100 kali maka waktu meningkat 100^3 kali atau sejuta kali dari semula

5. Berikut ini diberikan suatu fungsi (*method*):

```
public static int hitungan(int n) {
     int sum = 0;
     for (int i = -n; i < n; i++) {
          for (int j = 0; j < 1000; j++) {
               sum++;
          }
     }
     return sum;
}
```

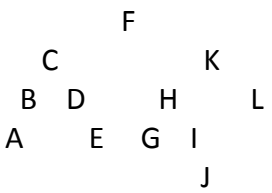
Berapakah kompleksitas **`hitungan(n)`** jika dinyatakan dalam notasi big-Oh yang paling sesuai?

O(n)

6. Suatu binary tree jika node-nodenya diprint secara inorder traversal menghasilkan urutan adalah: A, B, C, D, E, F, G, H, I, J, K, L. Sementara secara preordertraversal adalah F, C, B, A, E, D, K, H, G, I, J, K. Berapakah tinggi dari binary tree tersebut!

(Salah ketik: K ke dua seharusnya L, jadi kalau ada di ujian seperti ini harus dianulir kecuali jika sempat diralat!) Jadi kalau preordertraversalnya F, C, B, A, E, D, K, H, G, I, J, L maka jawabannya adalah: 4

Penjelasan: karena F adalah root, maka inorder subtree kiri A, B, C, D, E (preorder C, B, A, E, D) dan preorder subtree kanan G, H, I, J, K, L (preorder K, H, G, I, J, L). maka akan diketahui treenya:



Bagian C: Pemrograman Singkat

1. Buatlah sebuah *static method* **`isAlay(String nama)`** dengan *visibility* **`public`** yang menerima sebuah parameter bertipe **`String`**. Method ini mengembalikan nilai *boolean* **`true`** jika string yang masuk via parameter **`nama`** memenuhi kriteria string **alay**. Apabila string yang masuk tidak memenuhi kriteria alay, maka method mengembalikan *boolean* **`false`**.

   Sebuah string dikatakan **alay** apabila **memenuhi semua kriteria** berikut:

   - Terdapat kemunculan angka paling sedikit 2 kali.
   - Frekuensi kemunculan huruf besar lebih besar dari huruf kecil.

   Contoh:

String dalam parameter nama	Return value	Penjelasan
S4viRA M4hARan1 S4Ntoso	True	Angka muncul sebanyak 4 kali, frekuensi huruf besar > huruf kecil
Ivana Ir3n3 Thom4s	False	Angka muncul sebanyak 3 kali, frekuensi huruf besar < huruf kecil

   Panduan:

   Berikut ini adalah beberapa *method* dari kelas Character dan String milik *standard library* Java yang dapat Anda gunakan:

   - `Class Character: public static boolean isDigit(char ch)`
   - `Class Character: public static boolean isLowerCase(char ch)`
   - `Class Character: public static boolean isUpperCase(char ch)`
   - `Class String: public char charAt(int index)`

   Fungsi/kegunaan *method-method* di atas sudah cukup jelas dari *method signature* setiap *method*.

2. Perhatikan potongan kode berikut:

```
public static int sumRec(int[] arr, int index) {
    if(index >= arr.length) {
        return 0;
    } else {
        return arr[index] + sumRec(arr, index+1);
    }
}

public static int sumItr(int[] arr, int index) {
    // TODO Lengkapi saya!
```

```
}
```

Implementasikan *static method* **`sumItr(int[] arr, int index)`** dengan meniru mekanisme rekursif method **`sumRec`** dengan menggunakan bantuan *stack*. Pastikan implementasi **`sumItr`** Anda menggunakan *stack* dalam proses iterasi dan menghasilkan keluaran yang sama dengan **`sumRec`**.

Contoh 1 (dalam bentuk *pseudocode*):

```
arr_1 = [ 1, 2, 3, 4, 5, 6, 7 ]
result_rec = sumRec(arr_1, 0)    // Hasil: 28
result_itr = sumItr(arr_1, 0)    // Hasil: 28
print(result_rec == result_itr)  // Mencetak true
```

Contoh 2 (dalam bentuk *pseudocode*):

```
arr_2 = [ 1, 1, 3, 5, 8, 10 ]
result_rec = sumRec(arr_2, 3)    // Hasil: 23
result_itr = sumItr(arr_2, 3)    // Hasil: 23
print(result_rec == result_itr)  // Mencetak true
```

3. Perhatikan potongan kode berikut:

```
public static void merge(Queue<Integer> a, Queue<Integer> b, Queue<Integer> c) {
    // TODO Lengkapi saya!




}
```

Terdapat *static method* **`merge(Queue<Integer> a, Queue<Integer> b, Queue<Integer> c)`** yang bertujuan untuk menggabungkan setiap elemen integer dari *queue* **a** dan **b** ke dalam *queue* **c**. Proses penggabungan mengharuskan **setiap elemen yang masuk ke dalam *queue* c terurut dari terkecil hingga terbesar**.

Beberapa prekondisi terkait isi dari parameter-parameter *method* **`merge`** adalah sebagai berikut:

- Kapasitas *queue* **a** == kapasitas *queue* **b**
- Kapasitas *queue* **c** == kapasitas *queue* **a** + kapasitas *queue* **b**
- Elemen-elemen dalam *queue* **a** dan **b** dipastikan sudah terurut dari terkecil hingga terbesar. Dengan kata lain, elemen terkecil pasti terletak di posisi terdepan *queue* sedangkan elemen terbesar terletak di posisi terbelakang *queue*.

Method di atas masih kosong dan tugas Anda adalah melengkapi *method* tersebut!

Contoh 1 (dalam bentuk *pseudocode*):

```
queue_a = [ 1, 3, 5, 7, 9 ]
queue_b = [ 2, 4, 6, 8, 10 ]
queue_c = [ ]
merge(queue_a, queue_b, queue_c) // Isi queue_c setelah method merge dipanggil:
                                 // queue_c = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
```

Contoh 2 (dalam bentuk *pseudocode*):

```
queue_a = [ 1, 1, 2, 3, 5 ]
queue_b = [ 1, 2, 4, 8, 16 ]
queue_c = [ ]
merge(queue_a, queue_b, queue_c) // Isi queue_c setelah method merge dipanggil:
                                 // queue_c = [ 1, 1, 1, 2, 2, 3, 4, 5, 8, 16 ]
```

4. Perhatikan contoh kode implementasi *singly linked list* dengan *header node* berikut:

```
class SLinkedList<T> {

    private ListNode<T> header;

    public LinkedList() {
        header = new ListNode<T>(null, null);
    }

    public void addLast(T element) {
        // TODO Implement me!




    }
}
class ListNode<T> {

    T element;
    ListNode<T> next;

    public ListNode(T element, ListNode<T> next) {
        this.element = element;
        this.next = next;
    }

    public ListNode(T element) {
        this(element, null);
    }

    public ListNode() {
        this(null, null);
    }
}
```

Implementasikan *method* **`addLast(T element)`** yang akan membungkus elemen data sebagai objek **`ListNode`** baru dan menyisipkannya sebagai *node* terakhir dalam *linked list*!

SELESAI