NWU®

Requirements for this paper/Benodigdhede vir hierdie vraestel:			
Answer scripts/ Antwoordskrifte:	□	Multi-choice cards (A5)/ Multikeusekaarte (A5):	□
Attendance slips (Fill-in paper)/ Presensiestrokies (Invulvraestel):	X	Multi-choice cards (A4)/ Multikeusekaarte (A4):	□
Scrap paper/ Rofwerkpapier:	□	Graph paper/ Grafiekpapier:	□

Calculators/Sakrekenaars: **No/Nee**

Other resources/Ander hulpmiddels:

Type of assessment/ Tipe assessering:	**Semester Test**	Qualification/ Kwalifikasie:	**BSc**
Module code/ Modulekode:	**ITRW222**	Duration/ Tydsduur:	**2 hour** **2 uur**
Module description/ Modulebeskrywing:	**Data Structures and Algorithms**	Max/ Maks:	**60**
Examiner(s)/ Eksaminator(e):	**R Goede**	Date/ Datum:	**25/09/2018**
Internal/Interne moderator(s):		Time/ Tyd:	**14:00**

Submission of answer scripts/Inhandiging van antwoordskrifte: **Ordinary/Gewoon**

Details of student / Besonderhede van student:

Title/ Titel: __________ Initials/ Voorletters: __________ Surname/ Van: ____________________

University number/ Universiteitsnommer: □□□□□□□□

Contact number/ Kontaknommer: □□□□□□□□□□

Signature/ Handtekening: ____________________

Examination instructions / Eksamenvoorskrifte

1. Students are allowed into the venue in the first half hour of a session, but no extra time is granted.
2. No student is allowed to leave the venue before half an hour of the examination session has elapsed.
3. Students bring bags to the venue at **own risk**, and must put them in front of the room.
4. Students may not have cell phones/electronic devices with them and/or handle them.
5. No refreshments are allowed in the examination venue.
6. Students may not leave the room for a smoke break.
7. Write on both sides of each page.
8. Write in black or blue ink only.
9. No pages may be removed from the answer scripts.
10. Students may not have unauthorised material with them during a session, e.g. notes and/or objects that contain notes.
11. No items may be borrowed during the session.
12. Students may not attempt to assist another student, or attempt to obtain assistance.
13. Students **must** hand in their answer scrips to invigilators before they leave the venue.
14. The attendance slip on the back cover that also serves as an undertaking, **must** be completed and handed in.

1. Studente mag in die eerste halfuur van 'n sessie tot die lokaal toegelaat word, maar geen ekstra tyd word toegestaan nie.
2. Geen student word toegelaat om die lokaal te verlaat binne die eerste halfuur van 'n eksamensessie nie.
3. Studente bring sakke na lokaal op **eie risiko**, en moet dit voor in die lokaal neersit.
4. Studente mag nie selfone/elektroniese toestelle by hulle hê en/of hanteer nie.
5. Geen verversings word in 'n eksamenlokaal toegelaat nie.
6. Studente mag nie die lokaal verlaat om te gaan rook nie.
7. Skryf op beide kante van die bladsye.
8. Skryf slegs in swart of blou ink.
9. Geen bladsye mag uit die antwoordskrif verwyder word nie.
10. Studente mag nie ontoelaatbare materiaal by hulle hê tydens 'n sessie nie, bv. notas en/of objekte wat notas bevat nie.
11. Geen items mag tydens die sessie geleen word nie.
12. Studente mag nie 'n ander student probeer help of probeer om hulp te kry nie.
13. Studente **moet** hul antwoordskrifte aan toesighouers oorhandig voordat hulle die lokaal verlaat.
14. Die presensiestrokie op die agterblad, wat ook as onderneming geld, **moet** voltooi en ingegee word.

Question 1 / Vraag 1 (20)

Use the detailed model (τ-notation) to determine the running time of the following program lines. (7)	Gebruik die gedetailleerde model (τ-notasie) om die looptyd van die volgende programlyne te bepaal. (7)
1. for (int i=1; i<n; i++) { 2. b=arr[i]*2; }	
1a $\tau_{\text{fetch}} + \tau_{\text{store}}$	1b $(2\tau_{\text{fetch}} + \tau_{<}) * n$
1c $(2\tau_{\text{fetch}} +\tau_{+} + \tau_{\text{store}}) * (n-1)$	2 $(4\tau_{\text{fetch}} +\tau_{x} + \tau_{[\cdot]} + \tau_{\text{store}}) * (n-1)$

Determine the running time of **the identified lines** in context of this program segment. You need not simplify the expressions. **Use the simplified model and asymptotic analysis.** Carefully check the line numbers. (8)	Bepaal die looptyd van die **aangeduide lyne** in konteks van hierdie programdeel. Jy hoef nie die uitdrukkings te vereenvoudig nie. **Gebruik die vereenvoudigde model en asimptotiese ontleding.** Kyk versigtig na die lynnommers. (8)

```
1        public class Question1_2
2        {
3          public static int numbers (int n)
4          {
5          int ans = 1;
6          for (int i=0;  i<n;  i++ )
7            {
8              for ( int j=1;  j<=i; ++j)
9                ans =ans-i;
10           }
11         return prod;
12       }
13       }
```

6b	3 (n+1)	O(n)
8b	$3\sum_{i=0}^{n-1}(\mathrm{i}+1)$	O(n^2)
8C	$4\sum_{i=0}^{n-1}\mathrm{i}$	O(n^2)

Proof the following equation (3)	Bewys die volgende gelykheid: (3)
$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$	
$\sum_{i=1}^{n} i = 1+2+3+..........+(n-2)+(n-1)+n$ and √ $\sum_{i=1}^{n} i = n+(n-1)+(n-2)+.......+3+2+1$ When you add these two rows together you get *n* pairs that each adds to *(n+1) thus:* √ $2\sum_{i=1}^{n} i = n(n+1)$ and √ $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$	

Give the definition for the asymptotic upper bound – big Oh. (2)	Gee die definisie van die asimptotiese bo-grens – groot O. (2)
f(n) is O(g(n)) if there exist positive numbers *c* and *N* such that f(n)<=*c*g(n) for all n>=*N*	

Question 2 / Vraag 2 (5)

Design a method for the class Queue called: *dequeue()*. (5)	Ontwerp 'n metode vir die klas Queue genaamd: *dequeue()*. (5)
Assume the following code exists: / Aanvaar die volgende kode bestaan:	

```
public class Queue
{
   private Listing[] data;
   private int size;
   private int numOfNodes;
   private int front;
   private int rear;
   public Queue()
   {
      size = 100;
      numOfNodes = 0;
      front = 0;
      rear = 0;
      data = new Listing[100];
   }
```

QUESTION 2

public Listing dequeue()	0	1	
if(numOfNodes == 0)			
return null; // ** overflow error **			
else			
frontlocation = front;		1	
front = (front +1) % size;		1	
numOfNodes = numOfNodes -1;	0	1	
return data[frontlocation]	0	1	
		5	

Question 3 / Vraag 3 (35)

Study the following class:	Bestudeer die volgende klas:

```
public class SinglyLinkedList<T extends Comparable<? super T>>
{
   private Element head;  // list header
   private Element tail;
   public SinglyLinkedList()
   {
       head = null;
       tail = null;
   }
    public boolean delete(T item)
    {
    // working code: you my use this method without supplying code
    }
  public class Element
  {
      private T data;
      private Element next;
      public Element(T param)
     {
         data = param;
     }
   }// end of inner class Element
}//end SinglyLinkedList outer class
```

Provide the code for a method *append(..)* that will add an object to the back of the list. (7)	Gee die kode vir 'n metode *append(…)* wat 'n objek aan die agterkant van die lys sal byvoeg. (7)

public boolean append(T newElement)	0	1
Element temp = new Element(newElement);	0	1
if(temp == null) // out of memory		
return false;		
else		
if (head==null)	0	1
head = temp;	0	1
tail = temp;	0	1
else		
tail.next = temp;	0	1
tail = temp	0	1
		7

Design and code a method for the class SinglylLinkedList called *rangeFIlter(..)* that will remove all values of the calling list between the 2 parameter values. The method should return the number of items deleted. You may assume that a delete(item) method already exist. Example list before rangeFilter(3,6): 5; 3;4;3;6;3;4; After: 5; 4; 4 removed 3elements	Ontwerp en kodeer ‘n metode vir die klas SinglyLinkedList genaamd rangeFilter(..) wat al die waardes in die roepende lys moet verwyder wat tussen die 2 parameter waardes is. Die metode moet die aantal elemente wat verwyder is terugstuur. Jy mag aanvaar dat daar reeds ’n delete(item) metode bestaan. Voorbeeldlys voor rangeFilter(3,6): 5;3;4;3;6;3;4; Daarna: 5; 4; 4 verwyder 3 elemente.
Draw a diagram to assist your design. (3)	Teken ’n diagram om jou ontwerp te ondersteun. (3)

- Diagram should be linked list;
- It should clearly show addresses and nodes separately
- It should indicate the head and the tail
- It should indicate the *ptr* for traversing
- It should show 2 separate parameter variables which are NOT a list!!!

Write down the general case and describe the steps to be taken. (3)	Skryf die algemene geval neer en beskryf die stappe wat gevolg moet moet word. (3)

- Traverse list with *ptr* that starts at head
- Check each element’s data field – if it is larger than the first parameter and smaller than the second – use delete to remove it and increment the counter that counts the number of values deleted
- Return the counter

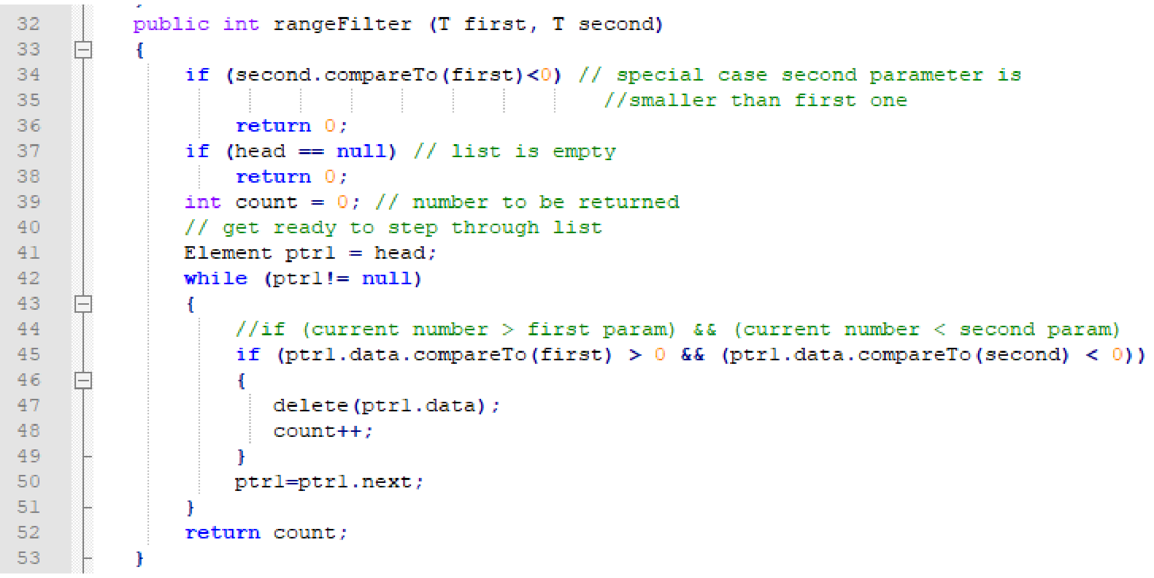
Write down the special cases and describe the steps to be taken. (3)	Skryf die spesiale gevalle neer en beskryf die stappe wat gevolg moet moet word. (3)

- Delete handles the following special cases:
- list empty; delete head; delete tail; value to be deleted not present
- Special cases for Filter:

  First parameter value might be larger than the second one;
  List is empty – return 0 not null!

Give the java code for the method. (10)	Gee die java kode vir die metode. (10)

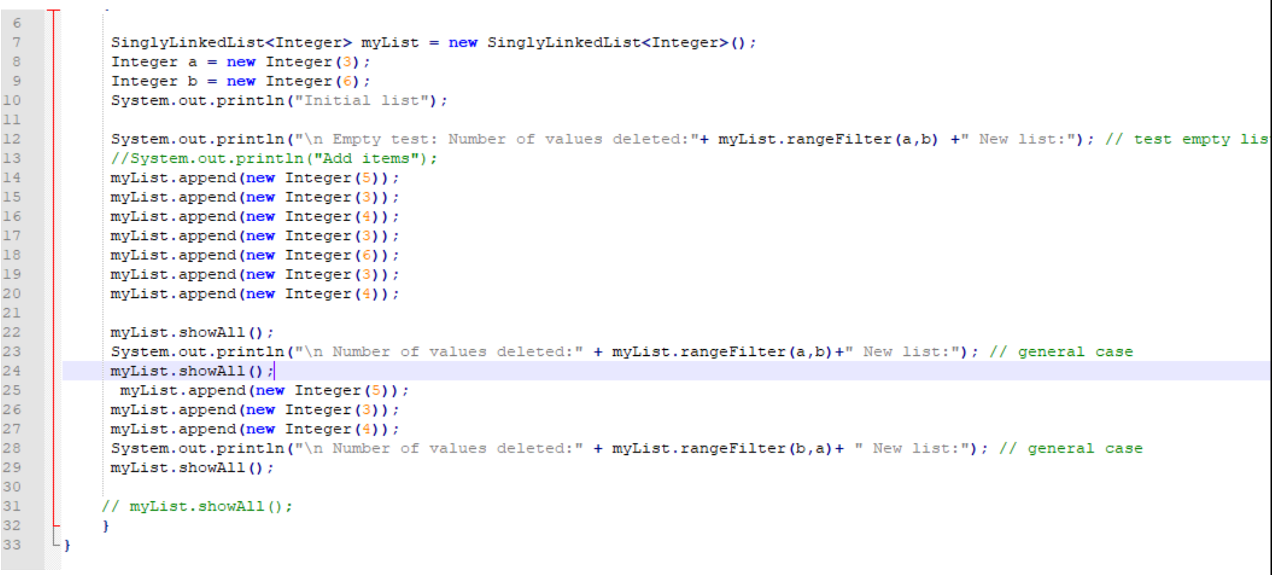
public int rangeFilter (T first, T second)	0	1
if (second.compareTo(first)<0)	0	1
if (head == null)	0	1
int count = 0; // number to be returned		
Element ptr1 = head;	0	1
while (ptr1!= null)	0	1
if (ptr1.data.compareTo(first) > 0 && (ptr1.data.compareTo(second) < 0))	0	1
delete(ptr1.data);	0	1
count++;	0	1
ptr1=ptr1.next;	0	1
return count;	0	1
		10



```
public int rangeFilter (T first, T second)
{
    if (second.compareTo(first)<0) // special case second parameter is
                                    //smaller than first one
        return 0;
    if (head == null) // list is empty
        return 0;
    int count = 0; // number to be returned
    // get ready to step through list
    Element ptr1 = head;
    while (ptr1!= null)
    {
        //if (current number > first param) && (current number < second param)
        if (ptr1.data.compareTo(first) > 0 && (ptr1.data.compareTo(second) < 0))
        {
            delete(ptr1.data);
            count++;
        }
        ptr1=ptr1.next;
    }
    return count;
}
```

Complete the test program to test the method thoroughly. (9)	Voltooi die toetsprogram om die metode deeglik te toets. (9)

public class Driver
{
        public static void main(String [] args)
        {



```
        SinglyLinkedList<Integer> myList = new SinglyLinkedList<Integer>();
        Integer a = new Integer(3);
        Integer b = new Integer(6);
        System.out.println("Initial list");

        System.out.println("\n Empty test: Number of values deleted:"+ myList.rangeFilter(a,b) +" New list:"); // test empty lis
        //System.out.println("Add items");
        myList.append(new Integer(5));
        myList.append(new Integer(3));
        myList.append(new Integer(4));
        myList.append(new Integer(3));
        myList.append(new Integer(6));
        myList.append(new Integer(3));
        myList.append(new Integer(4));

        myList.showAll();
        System.out.println("\n Number of values deleted:" + myList.rangeFilter(a,b)+" New list:"); // general case
        myList.showAll();
         myList.append(new Integer(5));
        myList.append(new Integer(3));
        myList.append(new Integer(4));
        System.out.println("\n Number of values deleted:" + myList.rangeFilter(b,a)+ " New list:"); // general case
        myList.showAll();

       // myList.showAll();
       }
}
```

Create list	0	1	2	
correct Integer parameters	0	1		
Test with empty list	0	1		
Add values	0	1		
second number smaller than first number	0	1		
general case	0	1		
Output number deleted	0	1		
Good screen output	0	1		
		9		

TOTAL/TOTAAL: 60