

UNIVERSITY OF WESTMINSTER

School of Computer Science and Engineering
MOCK EXAM SEMESTER 2 2020/21

Module Code: 5SENG001W, 5SENG002C
Module Title: Algorithms: Theory, Design and Implementation
Module Leader: Klaus Draeger
Exam Start Time: N/A
Recommended Exam End Time: N/A
Submission Window: N/A
Submission Deadline: N/A (Exam time will be 2:40)

Instructions to Candidates:

Please read the instructions below before starting the paper

- Module specific information is provided below by the Module Leader
- The Module Leader will be available during the exam release time to respond to any queries via the Discussion Board in the Assessment area of the module's Blackboard site
- As you will have access to resources to complete your assessment any content you use from external source materials will need to be referenced correctly. Whenever you directly quote, paraphrase, summarise, or utilise someone else's ideas or work, you have a responsibility to give due credit to that person. Support can be found at:
<https://www.westminster.ac.uk/current-students/studies/study-skills-and-training/research-skills/referencing-your-work>
- This is an individual piece of work so do not collude with others on your answers as this is an academic offence
- Plagiarism detection software will be in use
- Where the University believes that academic misconduct has taken place the University will investigate the case and apply academic penalties as published in [Section 10 Academic Misconduct regulations](#).
- ***Once completed please submit your paper via the Assignment content. In case of problems with submission, you will have TWO opportunities to upload your answers and the last uploaded attempt will be marked. Note that instructions on how to compile and submit your handwritten and/or typed solutions will have been sent to you separately.***
- ***Work submitted after the deadline will not be marked and will automatically be given a mark of zero***

Module Specific Information

PLEASE WRITE YOUR STUDENT ID CLEARLY AT THE TOP OF EACH PAGE

You are advised (but not required) to spend the first ten minutes of the examination reading the questions and planning how you will answer those you have selected.

Answer ALL questions in Section A and at least THREE questions from Section B.

Section A is worth a total of 40 marks.

Section B is worth a total of 60 marks.

Each question in Section B is worth 20 marks.

In Section B, only the THREE questions with the HIGHEST MARKS will count towards the FINAL MARK for the EXAM.

Section A

Answer ALL questions from this section.

Question 1

- (a) For each of the following functions, decide which complexity class it belongs to:

1. $n * (n + \log n)$ [2 marks]

2. $2^n + n^4 + 7$ [2 marks]

Answer:

1. $n * (n + \log n)$ is in $O(n^2)$

2. $2^n + n^4 + 7$ is in $O(2^n)$

- (b) Given the following table of runtimes (in seconds) of two algorithms, determine their complexity. Give a short explanation of how you determined them. [6 marks]

| Input size | Algorithm 1 | Algorithm 2 |
|------------|-------------|-------------|
| 1000 | 4 | 21 |
| 2000 | 15 | 41 |
| 4000 | 62 | 85 |
| 8000 | 250 | 169 |

Answer: Algorithm 1 grows by about a factor $4 = 2^2$ each time the input size doubles, we would expect it to be in $O(n^2)$.

Algorithm 2 grows by about a factor 2 each time the input size doubles, we would expect it to be in $O(n)$.

[TOTAL 10]

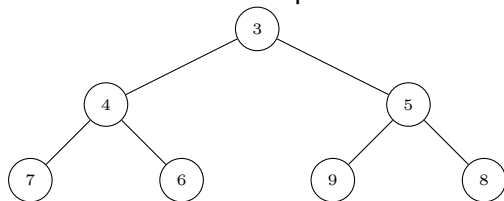
Question 2

Consider this array representation of a heap:

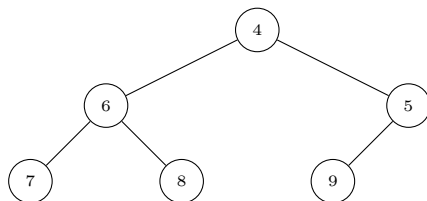
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|-------|---|---|---|---|---|---|---|
| value | 3 | 4 | 5 | 7 | 6 | 9 | 8 |

- (a) Draw the tree representation of this heap. [2 marks]
- (b) Perform the following sequence of operations, drawing the updated tree representation after each of them:
1. Pop the minimal element off the heap [2 marks]
 2. Insert the value 3 [2 marks]
 3. Insert the value 1 [2 marks]
 4. Insert the value 2 [2 marks]

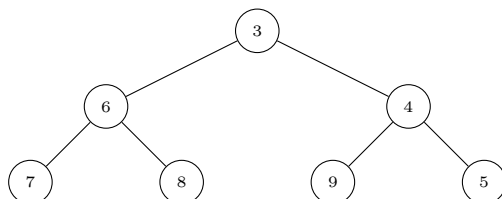
Answer: Initial heap:



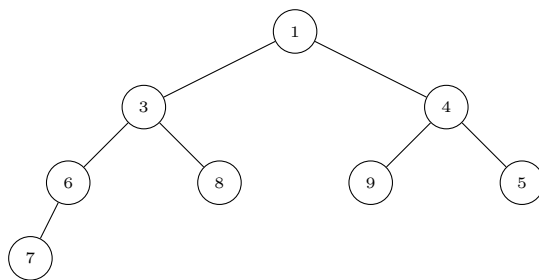
1. Pop the minimum:



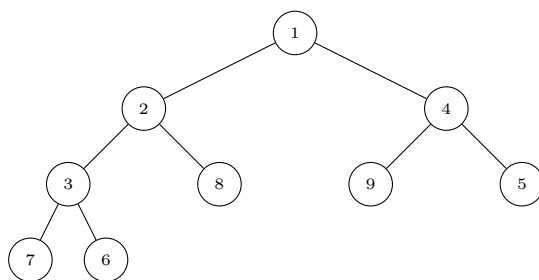
2. Insert 3:



3. Insert 1:



4. Insert 2:



[TOTAL 10]

Question 3

Write a method that removes duplicates in a **sorted** doubly linked list. For example, if the list originally contains the data

1,1,3,4,4,4,7,8

after calling the method it should contain

1,3,4,7,8

Complete either the Java or C++ version below. [10 marks]

- Java version:

```
public class DList{
    public class ListNode{
        public int data;
        public ListNode next, previous;
    }

    public ListNode first, last;

    public void removeDuplicates(){
        ListNode n = first;
        while(n != null){
            while(n.next != null && n.next.data == n.data){
                // remove the next node from the list
                n.next = n.next.next;
                if(n.next == null) // n is now the last node
                    last = n;
            }
            else
                n.next.previous = n;
            n = n.next;
        }
    }
}
```

- C++ version:

```
class DList{
public:
    class ListNode{
    public:
        int data;
        ListNode *next, *previous;
    };
};
```

```

ListNode *first, *last;

void removeDuplicates(){
    ListNode *n = first;
    while(n != null){
        while(n->next != null && n->next->data == n->data){
            // remove the next node from the list
            ListNode *temp = n->next;
            n->next = n->next->next;
            if(n->next == null) // n is now the last node
                last = n;
            else
                n->next->previous = n;
            delete temp; // avoid memory leak
        }
        n = n->next;
    }
}
};

```

[TOTAL 10]

Question 4

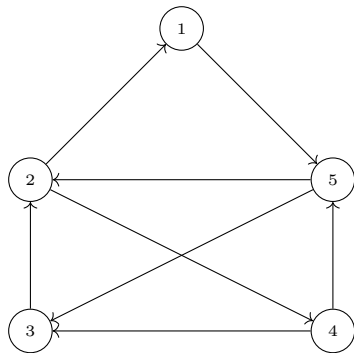
A *directed graph* is represented by the following *adjacency matrix* (where T represents True and - represents False):

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | - | - | - | - | T |
| 2 | T | - | - | T | - |
| 3 | - | T | - | - | - |
| 4 | - | - | T | - | T |
| 5 | - | T | T | - | - |

(a) Draw the above graph.

[5 marks]

Answer: The graph looks like this.



- (b) Does this graph have an *Euler path*? If yes, give one. If not, explain why not.

[5 marks]

Answer: The graph does have an Euler path, for example $4 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 5 \rightarrow 3$.

[TOTAL 10]

Section B

Answer THREE questions from this section.

Question 5

- (a) Consider the following unsorted array:

| | | | | | | | |
|-------|---|---|----|----|----|---|----|
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| value | 8 | 5 | 34 | 13 | 55 | 3 | 89 |

Suppose you try to find the value 8 in this array using binary search. For each iteration of the algorithm, give the values of the first, middle, and last indices, and a (short) explanation of what happens.

[10 marks]

Answer: The variable values per iteration are

1. first=0, middle=3, last=6
2. first=0, middle=1, last=2
3. first=2, middle=2, last=2
4. first=2, middle=1, last=1

At this point the algorithm terminates because `first > last`. Even though the value is in the array, the search fails because the array is not sorted, and binary search expects a sorted array.

- (b) Instead of finding a specific value, in this problem we are searching an array for a value within a given **range**.

Write a search function which gets a **sorted** array `A` and two numbers `low`, `high` as its input and returns an index `i` such that `low <= A[i] <= high` (or -1 if there is no such value in `A`).

For example, if the values in the array are `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]` and `low=11`, `high=15` then the function should return either 4 or 5 (because `A[4]=11`, `A[5]=13` are both in the given range).

Complete either the Java or C++ version below.

[10 marks]

Answer: The algorithm is a variation of binary search. Just like in binary search, we keep track of three positions first, middle, and last in the array. In each iteration we have three possible cases:

- The middle value is less than low. Restrict to the first half.
- The middle value is more than high. Restrict to the second half.
- Otherwise the middle value is in the range; return it as our result.

- Java version:

```
public class RangedSearch{
    public static int findInRange(int[] A, int low, int high){
        int first=0, last=A.length, middle=A.length/2;
        while(first<=last){
            if(A[middle]<low)
                last=middle-1;
            else if(A[middle]>high)
                first=middle+1;
            else
                return middle;
            middle=(first+last)/2;
        }
        return -1;
    }
}
```

- C++ version:

```
int findInRange(vector<int> &A, int low, int high){
    int first=0, last=A.size(), middle=A.size()/2;
    while(first<=last){
        if(A[middle]<low)
            last=middle-1;
        else if(A[middle]>high)
            first=middle+1;
        else
            return middle;
        middle=(first+last)/2;
    }
    return -1;
}
```

[TOTAL 20]

Question 6

(a) Consider the following array:

| | | | | | | | | | |
|-------|---|---|---|---|---|---|---|---|---|
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| value | 9 | 7 | 5 | 3 | 1 | 2 | 4 | 6 | 8 |

We want to sort this array using Selection Sort. Write down the contents of the array after each iteration of the main loop. [10 marks]

Answer: The contents of the array after each iteration are

| | | | | | | | | | |
|--------------|---|---|---|---|---|---|---|---|---|
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| iteration 1: | 8 | 7 | 5 | 3 | 1 | 2 | 4 | 6 | 9 |
| iteration 2: | 6 | 7 | 5 | 3 | 1 | 2 | 4 | 8 | 9 |
| iteration 3: | 6 | 4 | 5 | 3 | 1 | 2 | 7 | 8 | 9 |
| iteration 4: | 2 | 4 | 5 | 3 | 1 | 6 | 7 | 8 | 9 |
| iteration 5: | 2 | 4 | 1 | 3 | 5 | 6 | 7 | 8 | 9 |
| iteration 6: | 2 | 3 | 1 | 4 | 5 | 6 | 7 | 8 | 9 |
| iteration 7: | 2 | 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| iteration 8: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

(b) **Cycle Sort** is a sorting algorithm which works as follows: For each element in the array A, figure out where it should go (by counting how many smaller elements there are) and swap it with whatever is at that position. We assume that A contains **no duplicates** to simplify matters. In more detail:

- Maintain an additional boolean array B, initially all false; B[i] keeps track of whether A[i] is known to be in the right position.
- Starting at position i=0, do the following until i reaches the end:
 - If B[i] is true, increase i
 - Otherwise, count how many elements in A are less than A[i]. Call this number j, swap A[i] with A[j], and update B.

For example, starting with this array:

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| index | 0 | 1 | 2 | 3 | 4 | 5 |
| A | 5 | 3 | 1 | 2 | 6 | 8 |
| B | false | false | false | false | false | false |

First we look at A[0]=5. There are 3 values which are smaller, so this belongs at position 3. After swapping, we have

| index | 0 | 1 | 2 | 3 | 4 | 5 |
|-------|-------|-------|-------|------|-------|-------|
| A | 2 | 3 | 1 | 5 | 6 | 8 |
| B | false | false | false | true | false | false |

Now the 5 is in the right position, and A[0]=2. There is 1 value which is smaller, so this belongs at position 1, and we get

| index | 0 | 1 | 2 | 3 | 4 | 5 |
|-------|-------|------|-------|------|-------|-------|
| A | 3 | 2 | 1 | 5 | 6 | 8 |
| B | false | true | false | true | false | false |

and so on. Based on this description, complete either the Java or C++ version below. [10 marks]

• Java version:

```
public class CycleSort{

    public static void sort(int[] values){
        boolean[] inPlace = new boolean[values.length]; // initially all f
        for(int i=0; i<values.length; )
            if(inPlace[i])
                // this value is in the right place, move on
                i++;
            else{
                // determine where this value belongs
                int p=0;
                for(int j : values)
                    if(i < values[j])
                        p++;
                // swap entries at positions i and p
                int temp = values[i];
                values[i] = values[p];
                values[p] = temp;
                inPlace[p] = true;
            }
        }
    }
}
```

• C++ version:

```
void cycleSort(vector<int> &values){
    vector<boolean> inPlace(values.size(), false);
    for(int i=0; i<values.size(); )
        if(inPlace[i])
            // this value is in the right place, move on
```

```
        i++;  
    else{  
        // determine where this value belongs  
        int p=0;  
        for(int j : values)  
            if(i < values[j])  
                p++;  
        // swap entries at positions i and p  
        int temp = values[i];  
        values[i] = values[p];  
        values[p] = temp;  
        inPlace[p] = true;  
    }  
}
```

[TOTAL 20]

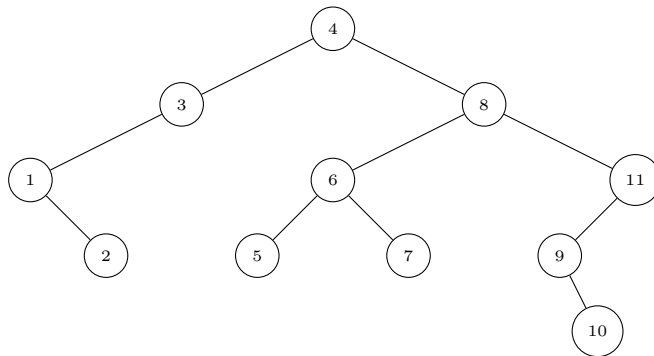
Question 7

- (a) Draw the *Binary Search Tree (BST)* which results from inserting the following values in the sequence given into an initially empty BST:

4,8,6,3,7,11,1,5,9,2,10

[4 marks]

Answer: The resulting tree is



- (b) Give the pseudo code for the **preorder** tree traversal algorithm. Apply the algorithm to the BST constructed for Part (a) and show the order in which nodes will be visited.

[4 marks]

Answer:

```

inorder(tree):
    IF ( tree not empty ) THEN
        print the root node
        inorder the left subtree
        inorder the right subtree
  
```

The result is: 4 3 1 2 8 6 5 7 11 9 10

- (c) Draw the BSTs which you get by removing the following three values, one at a time, from the tree created in Part (a):

(i) 7

[4 marks]

(ii) 1

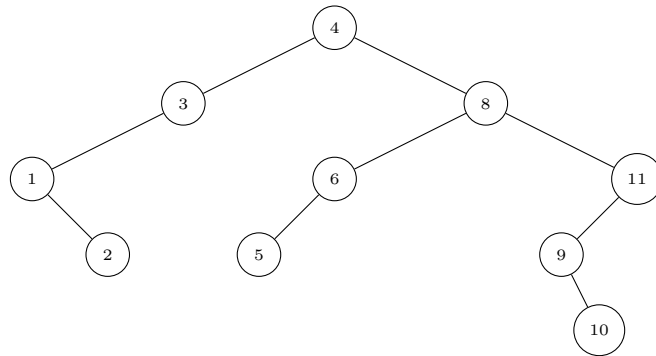
[4 marks]

(iii) 8

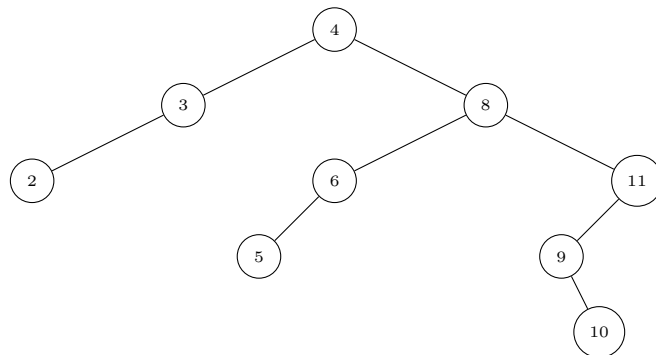
[4 marks]

Answer:

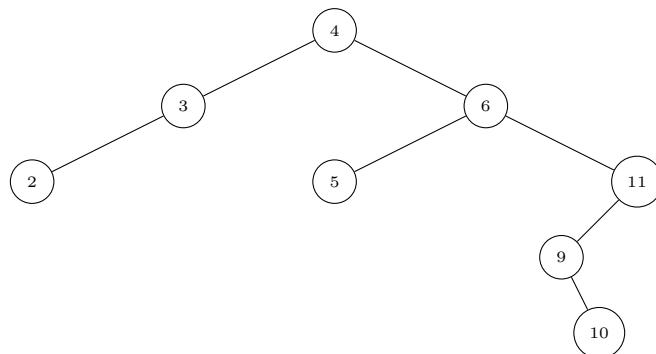
(i) After removing 7:



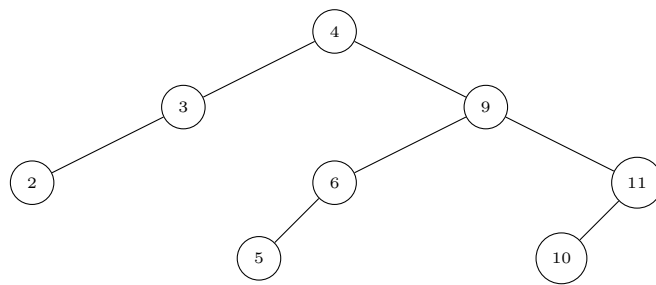
(ii) After removing 1:



(iii) After removing 8, we have **either**:



or:



[TOTAL 20]

Question 8

You are given the following adjacency matrix of a directed graph:

| id | 1 | 2 | 3 | 4 | 5 | 6 |
|----|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 0 |

- (a) Show the steps that are taken by breadth-first search when exploring this graph, starting at vertex 1. Do this by writing for each step which vertex is explored from which predecessor (e.g. "Exploring vertex 7 from vertex 9").

[10 marks]

Answer:

Exploring vertex 3 from vertex 1
Exploring vertex 5 from vertex 1
Exploring vertex 4 from vertex 3
Exploring vertex 2 from vertex 5
Exploring vertex 6 from vertex 4

- (b) Show the steps that are taken by iterative depth-first search when exploring this graph, starting at vertex 1. Do this the same way as in Part (a).

[10 marks]

Answer:

Exploring vertex 3 from vertex 1
Exploring vertex 4 from vertex 3
Exploring vertex 5 from vertex 4
Exploring vertex 2 from vertex 5
Exploring vertex 6 from vertex 4

[TOTAL 20]