Section 2: Theory of Computation [15]

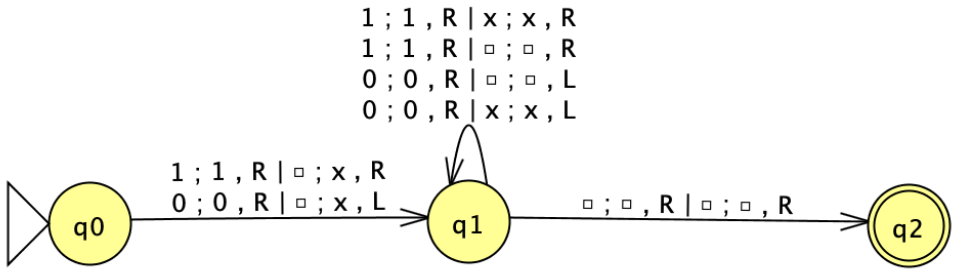
Question 8. **[6]**

Design a Turing Machine that accepts an unequal number of 0s and 1s. Thus, it should accept strings like 00111 and 10110, but reject strings like 0011 and 1010. You may choose to do that with any type of Turing Machine that has been covered in the module.
Provide the *full definition* of the TM. The transition function may be drawn (using consistent common notation), listed in a list of functions (adhering to the definition of the function), or shown in a table with the states (first column the states, first row the alphabet).

ANSWER

This is fairly similar to exercise 8.2.2 and 8.4.6, but then the other way around (*un*even rather than even). The multi-tape is the easiest, and instead of making it go to the final state upon the # (or whatever the marker is on the scratch tape), it should go to the final state if it the cell with # is *not* under the tape head, i.e., if the cell on the scratch tape has a blank.

Here's a drawing in JFLAP, where the box is the blank (b) and the x is the marker on the scratch tape:



Note for marking: either the diagram or the table, or as a list of functions (delta(q0, [0,B]) = (q1, [0, x], [R,L]) etc etc). For the definition: M = ({q0,q1,q2},{0,1},{0,1,B},q0,q2)

Marks: 1 for listing all the components, then for the remaining 5: give 5 for everything correct, 2.5 if it does about half of it, 0 if wrong (accepts a different language, incomplete). Deduct 0.5 mark for inconsistent notation, for missing the arrow/forgetting to indicate the start symbol, forgetting arrows in the transitions.

END ANSWER.

Question 9. **[4]**

L1 reduces to L2. We know that L2 is a recursive language. What can be said about L1? Prove it.

ANSWER

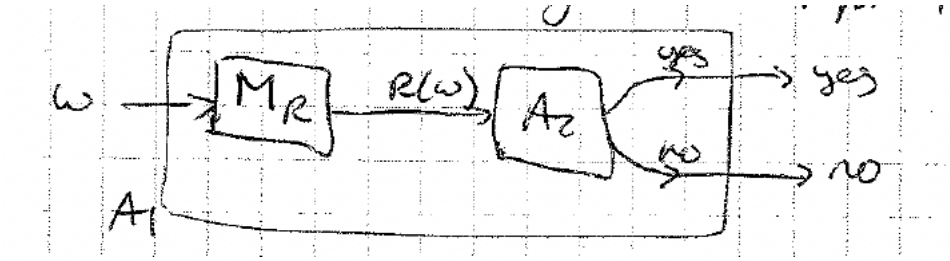
L1 is also recursive. [1 mark]

Proof]:

L2 is recursive, so has an algorithm for it, which returns y/n for sure [1 mark].

Since it does reduce (given in statement), there must be an algorithm A_R for that reduction converting w in to 'R(w)' (A2's input) [1 mark].

Then we can construct an algorithm A1 such that M1's input is converted to one for M2 for A2, and we can use A2's output for A1'a output. So then w in L1 equiv R(w) in L2 [1 mark]. Schematically, optionally.alternatively:



qed

NOTE FOR MARKING: just the figure plus a note that w in L1 equiv R(w) in L2 would be fine as well to gain the 3 marks for the proof. missing the "w in L1 equiv R(w) in L2" would lose 0.5 marks.

END ANSWER

Section 3: Theory of Algorithms [10]

Question 10. [6]

Consider the following algorithm:

```
ALGORITHM what(int [] A, int i, int j)
   if i == j then
      return A[i]
   if i == j-1 then
      if A[i] < A[j] then
         return A[i]
      else
         return A[j]

   k = (i+j) / 2
   m1 = what(A, i, k)
   m2 = what(A, k+1, j)
   if m1 < m2 then
      return m1
   else
      return m2
```

i) What does the algorithm above do? What strategy (algorithmic pattern) does this algorithm use? [2]

ii) Determine the performance of this algorithm by setting up a recurrence relation for the basic operations being performed and solving it using the Master Theorem (see Appendix A). Show all your work. [4]

i) It uses recursion to find the minimum value in an array [1]. It employs a divide and conquer strategy [1].

ii) Basic operation is comparison [1]. In the worst case 3 comparisons are done on each step and 2 recursive calls, each of which operates on half the array, thus:

$C(n) = 2\ C(n/2) + 3$ [1]

In terms of the Master Theorem a = 2, b = 2, d = 0 [1].

This fits the $a > b^d$ case so efficiency is

$\Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \Theta(n)$ [1].

Question 11. **[4]**

Given the following adjacency matrix:

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

i) Show the steps in the execution of Warshall's algorithm for transitive closure applied to this matrix. [3]

ii) From the outcome, list a pair of nodes (a, b) where node b is not reachable from node a. Note that a and b should be numbers in the range 1, …, 3. [1]

Solution:

i)

$R1 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ (no change)

[1]

$R2 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$

[1]

$R3 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$

[1]

ii) either (2, 1) or (3, 1). [1]

Appendix A – Master Theorem

If we have a recurrence of this form: $T(n) = aT(n/b) + f(n)$, $f(n) \in \Theta(n^d)$ and $T(1) = c$ then

$T(n) \in \Theta(n^d)$ if $a < b^d$

$T(n) \in \Theta(n^d \log n)$ if $a = b^d$

$T(n) \in \Theta(n^{\log_b a})$ if $a > b^d$

Question 2 [10 Marks]

a) Explain your answers in each case below:

i. Is $O(2^n) \subseteq O(n!)$? [1]

ii. Is $O(\log_2 n) \subseteq O(1)$? [1]

iii. Is $\Theta(n) \subseteq \Theta(n!)$? [1]

Answer:

i) Yes. (½ mark). n! grows faster than 2^n (need to make an argument for growth rate) but Big Oh provides an upper bound (1/2 mark)

ii) No (1/2 mark). $\log_2 n$ grows faster than a constant (provide some kind of argument for this) (1/2 mark)

iii) No (1/2 mark). n! grows faster than n (make an argument for this) so they are in different efficiency classes. And the class is a subset of itself. (1/2 mark)

b) Consider the pseudocode below for Strassen's algorithm for matrix multiplication:

Fill A and B up with zeroes so as to get two n x n matrices where n is a power of 2. Then apply the algorithm below. [4]

```
Strassen(A, B):
if n == 1 return A00 × B00
else
  Split matrices into 8 n/2 X n/2 parts:
                    A00 , B00 , . . . , A11 , B11
  M3 = Strassen(A00, B01 – B11)
  M5 = Strassen(A00 + A01 , B11 )
  M2 = Strassen(A10 + A11 , B00 )
  M4 = Strassen(A11 , B10 – B00 )
  M1 = Strassen(A00 + A11 , B00 + B11 )
  M7 = Strassen(A01 – A11 , B10 + B11 )
  M6 = Strassen(A10 – A00 , B00 + B01 )
  C11 = M1 + M4 – M5 + M7
  C12 = M3 + M5
  C21 = M2 + M4
  C22 = M1 + M3 – M2 + M6
  return C
```

Set up a recurrence relation for the number of basic operations being performed by this algorithm and solve it using the Master Theorem.

Answer: $T(n) = 7T(n/2)+\text{Theta}(n^2)$. (1 mark) a=7, b=2, d=2. (1 mark) So $7>2^2$. (1 mark) So T(n) is in $\text{Theta}(n^{(\log_2 7)})=\text{Theta}(n^{2.807})$. (1 mark). Don't need to get the 2.807 precise.

c) Give pseudocode for Floyd's algorithm to find the shortest path between all vertices, given a weighted connected graph. [2]

d) Explain how a weighted connected graph needs to be represented for Floyd's algorithm to work. [1]

Answer:

a.
```
floyd(W[1..n,1..n]
    for k=1 to n
      for i=1 to n
        for j=1 to n
          W[i,j] = min(W[i,j],W[i,k]+W[k,j])
```

b. Adjacency matrix where infinity represents no edge between two vertices, other values represent distance between vertices (so 0 between vertex x and itself).

Question 3 [10 Marks]

a) Can every NFA N be converted to an NFA N' with a single accept state that accepts the same language as N? If so, explain how to do so. If not, given an example. [2]

Yes it can. (1/2 mark) Informally, add a new state p (1/2 mark), add an epsilon transition from every accept state of N to p (1/2 mark) , and make p the only accept state. (1/2 mark)

b) Construct a context-free grammar that generates the following language: [4]

$L = \{w \mid w \in \{a,b,c\}^* \text{ and } w = w^r\}$. So every word in L is a palindrome made up of the letters a, b, and c.

$S \rightarrow X$ (1 mark)

$X \rightarrow aXa \mid bXb \mid cXc$ (1 mark)

$X \rightarrow a \mid b \mid c$ (1 mark)

$X \rightarrow \varepsilon$ (1 mark)

c) Prove that the context-free languages are closed under the * operation. [4]

Let G1 be a CFG that describes the language L1. Give each of the variables in G1 the subscript 1. Let S_1 be the start symbol for G1.

We construct a new CFG G from G1 that accept L1*. (1 mark)

Let S be a new variable (not occurring in G1). S is the start symbol for G. (1 mark) Add the following rules: $S \rightarrow \varepsilon \mid SS_1$. (1 mark for each rule)

Question 4 [10 Marks]

a) Cohesion and coupling are commonly referred to when talking about architectural design patterns. Explain what each of those two concepts mean, and what the goal of architectural design is in terms of those concepts. [3]

a. Answer: Cohesion is the degree to which communication takes place within the module [1]

b. Coupling is the degree to which communication takes place between modules [1]

c. Architectural design is about, Minimize coupling while maximizing cohesion [1]

b) What property defines a strict layered architecture? [1]

d. Answer: In a strict layered architecture, a layer only calls upon the services of the layer directly below it.

c) What is the lollipop notation used for in UML? [1]

e. Answer: To indicate type

d) Explain the concept of “separation of concerns” mean? [1]

f. Answer: separating a system into distinct sections, such that each section addresses a separate concern.

e) Which architectural design pattern is used for distributed systems over the internet such as email and the world wide web. [1]

Answer: Client Server

f) A compiler is a typical example of what kind of application architecture? [1]

Language processing systems

g) Briefly explain the Gang of Four Observer pattern and how it relates to the common MVC pattern. [2]

Answer: the object containing the data is separate from the objects which display the data [1]
Generalization of the MVC Pattern [1]

Section 2. Theory of Algorithms [11 Marks]

Question 4 [11 Marks]

a. Brute force and divide-/decrease-and-conquer are techniques most commonly used for problem solving. Name and briefly describe ***any two other*** techniques. [2]

Space-time: use additional data structures/memory/variables/space in order to make a faster solution/algorithm possible.

Dynamic programming: solve overlapping subproblems by storing intermediate results in an array where they are later accessed instead of being recomputed

Transform and conquer: solve a problem more easily by changing it to a more convenient instance of the same problem, or using a different representation, or transforming it to another problem whose solution can be transformed into an answer to the original (any 1 of the three will do)

Greedy: solve a problem by making the optimal choice at each step

b. Consider the Master Theorem below and answer the questions that follow: [3]

If $T(n) = aT(n/b) + f(n)$ then $T(n)$ is $O(n^d)$ if $a < b^d$

$T(n)$ is $O(n^d \log n)$ if $a = b^d$

$T(n)$ is $O(n^{\log_b a})$ if $a > b^d$

i. Which letter indicates how much recursion an algorithm does: a, b or d?

ii. Which indicates how many operations each recursive call does: a, b or d?

iii. What does the remaining letter indicate?

a

d

what fraction of the input is the argument to each recursive call

c. Answer ***either one*** of the following questions: [4]

i. Show the ***shift table*** that would be used by the Boyer-Moore and Horspool algorithms when searching for the string DESEEDED in a text; and state by ***how many positions each algorithm would shift*** the pattern (DESEEDED) after their very first attempt i.e. when they fail to find that string at the start of the text "HE-FOUND-AND-DESEEDED-IT" (state clearly which answer is BM and which Horspool).

Array: D = 2; E = 1; S = 5. Horspool moves 2; BM moves 7.

ii. The initial array A(1) of Floyd's algorithm for finding shortest paths is shown below (∞ represents infinity). Show the ***next array A(2)*** that would result, and show what ***the other array*** used by this algorithm would contain at that stage i.e. when A(2) has been constructed, what does the other array contain?

	1	2	3	4
1	**0**	∞	**3**	∞
2	**2**	**0**	∞	∞
3	∞	**7**	**0**	**1**
4	**6**	∞	∞	**0**

Solution (where "any" can be 0 or null any value other than 1, 2, 3 or 4):

	1	2	3	4
1	**any**	**any**	**any**	**any**
2	**any**	**any**	**1**	**any**
3	**any**	**any**	**any**	**any**
4	**any**	**any**	**1**	**any**

	1	2	3	4
1	**0**	∞	**3**	∞
2	**2**	**0**	**5**	∞
3	∞	**7**	**0**	**1**
4	**6**	∞	**9**	**0**

d. You have N rolls of material, all 1 metre wide but all of different lengths. You need to cut K strips of material all 1 metre wide and want each of the K strips to be the same length. Given K and N, and the length of each of the N rolls, how do you find the maximum length your K strips can be? You can NOT join material from different rolls together, you can only cut. Assume K and N are at most 100.

Outline briefly in words (about 1 line each) two (2) algorithms that have different efficiency classes, and both solve this problem, stating which is more efficient. [2]

Try each possible length in turn going from 1 to 100 (or from 100 to 1)

Do a binary search, which is more efficient than the above.

Section 3. Theory of Computation [8 Marks]

Question 5 [8 Marks]

a. Let N be an NFA, and let N' be an NFA obtained from it by letting the accept states of N' be the reject states of N, and the reject states of N' the accept states of N (so N' is obtained from N by swapping around the accept and reject states). Is the language accepted by N' the complement of the language accepted by N? If it is, explain why it is. If it is not, give an example to show that it isn't. [3]

No it's not. Many examples. Consider the following NFA N over {0,1}. It accepts the language {0}. N' accepts the language {ε}, which is not the complement of L(N). The problem cases are those where the NFA gets stuck

b. Give a regular expression that accepts the following language: [2]

L = { w | w has an odd number of 1s, and ends with a 0}.

*One of many options: 0*10*(0*10*10*)*0*

c. Complete the following proof to prove that the following language is not regular: [3]

L = { $a^n b^n a^n$ | n≥0}.

Proof: Assume L is regular and consider the word w = …… where p is the pumping length. By the pumping lemma, w must be a word of the form xyz where the length of xy is no greater than p and y has a length greater than 0. This means that the word xy^2z is not a word in L because…….: a contradiction.

Assume L is regular and consider the word w = $a^p b^p a^p$ where p is the pumping length. By the pumping lemma, w must be a word of the form xyz where the length of xy is greater than p, and y has a length greater than 0. This means that the word xy^2z is not a word in L because it has the form $a^q b^p a^p$ where q>p (since xy consists of 0s only): a contradiction.

Section 2. Theory of Algorithms [24 Marks]

Question 2 [24 Marks]

a. Which **TWO** of the following statements are **TRUE**? [2]

i. $\Theta (n + \log n) = \Theta (n)$

ii. if $x \in O (n \log n)$ then $x \in \Theta (n \log n)$

iii. if $x \in O (n \log n)$ then $x \in O (n^2)$

iv. if $x \in O (n \log n)$ then $x \in O (n)$

i. $\Theta (n + \log n) = \Theta (n)$ *[1]*

iii. if $x \in O (n \log n)$ then $x \in O (n2)$ *[1]*

b. Show the different approaches of the algorithmic techniques below, using the example of exponentiation (a^N). Each answer should take one line. [4]

i. Brute force

ii. Divide and conquer

iii. Decrease and conquer

iv. Transform and conquer

i. *a * a * a .... * a N times* *[1]*

ii. $(a^{N/2}) * (a^{N/2})$ *[1]*

iii. $(a^{N-1}) * a$ OR $(a^{N/2})^2$ *[1]*

iv. Horner's rule OR Binary Exponentiation *[1]*

c. Two other techniques are Greedy Algorithms and Space-Time-Trade-off. Briefly describe these 2 approaches to problem solving (1 line each). [2]

Space-time: use additional data structures/memory/variables/space in order to make a faster solution/algorithm possible *[1]*

Greedy: make the optimum choice at each step. *[1]*

d. We have the academic records of all UCT students who have taken any CSC course during the past decade. We wish to sort them by alphabetical order of their Faculty (so all Commerce students first, then all Engineering ones, … finally all Science students). Students from all 6 of UCT's Faculties have taken a CSC course.

i. Outline (do not just name it) a method of solving this problem which is more efficient than quicksort and also uses key comparisons (i.e. do not describe radix sort). [2]

ii. What is the complexity of your more efficient algorithm above? [1]

i. Go through input counting how many from each faculty. [1 mark].

Then set the 6 end-position values for each Faculty accordingly. Finally go through the input again from end to start, using the end-position of their Faculty to put them in the sorted array, and decrement that value to indicate that position has been filled. [1 mark]

ii. linear i.e. O(N). [1]

e. You are searching for the first occurrence of the string REFERER in a text.

i. Show the shift table that Horspool's algorithm would use. [2]

ii. Show the good-suffix shift table that the Boyer-Moore algorithm would use for this search string REFERER. [2]

i.1 mark for characters (E,F,R) 1 mark for values. Elements in any order

E	F	R
1	4	2

ii. 1 mark for indexes (1 to 6). 1 mark for values. Elements in any order.

1	2	3	4	5	6
6	2	6	6	6	6

f. You are given a row of integers. You can choose as many as you like except that you cannot take any 2 integers that are adjacent (next to each other). For example, if the row has values 4, 2, 8, 6 then you can take the 4 only of you don't take the 2, you can take the 2 only if you take neither the 8 nor the 4, … you can take the 6 only if you don't take the 8.

i. State in words or pseudocode how you would you use Dynamic Programming to find the maximum value you can take [2]

ii. Show what the array would contain after solving the example below: [1]
4, 2, 8, 6
Maximum is 12 (take the 4 and the 8).

i. Go from start to end of list, each time choosing max(with new value, without new value) and storing it in the array [1 mark]

For the j'th item, A[j] is set to max(A[j-2]+newval , A[j-1]) [1 mark]

ii. Array [1 mark]

1	2	3	4
4	4	12	12

g. The results of a race are recorded in a file of 100 000 records, where each record contains the position that person came and the time they took, e.g. < 42, 1234> means the person came 42nd and took exactly 1234 seconds to run the race. The file is in order from winner to loser, i.e. in increasing order of position starting at 1.

Consider the problem of knowing the time you took in the race and wanting to look up your position i.e. the input is the number of seconds (e.g. 1234) and the output is the corresponding position (e.g. 42). Assume there were no ties.

Outline briefly in words (about one line each) ***three*** (3) different algorithms you could use to solve this problem and give the complexity of each. [6]

Go through the entire list from start to end which is linear or O(N). *[2 marks]*

Do a binary search which is logarithmic O(log N). *[2 marks]*

Use linear interpolation which is O(log log N). *[2 marks]*

Section 3. Theory of Computation [9 Marks]

Question 3 [9 Marks]

a. Prove, using the formal definitions of Deterministic Finite Automata and Non-Deterministic Finite Automata, that for every Deterministic Finite Automaton there is a Non-Deterministic Finite Automaton that accepts exactly the same language. [4]

Let D = (Q, Sigma, delta, q0, F). We need to find an NFA N such that N accepts the same *language as D.* *[1]*

We define N as: (Q,Sigma,delta', q0, F) where q' is defined as follows: *[1]*

For every q in Q and every s in Sigma, delta'(q,s) = {delta(q,s)} *[1]*

For every q in Q, delta'(q,epsilon) = emptyset *[1]*

b. Suppose we want to prove that for any regular language A, A* is also regular. We know that if A is regular, then there is a Non-Deterministic Finite Automaton N such that N accepts A. Now, from N we construct a new Non-Deterministic Automaton N' as follows: for every accept state s of N, add an epsilon-transition from s to the start state of N (if there isn't one already). Does N' accept A*? If it does, explain why in detail. If it does not, explain why not, indicate what language it accepts, and explain how to modify N' so that it accepts A*. [5]

N' does not always accept A.* *[1]*

If A contains the empty string, then N' accepts A. If not, then N' accepts every element of A* except the empty string.* *[1]*

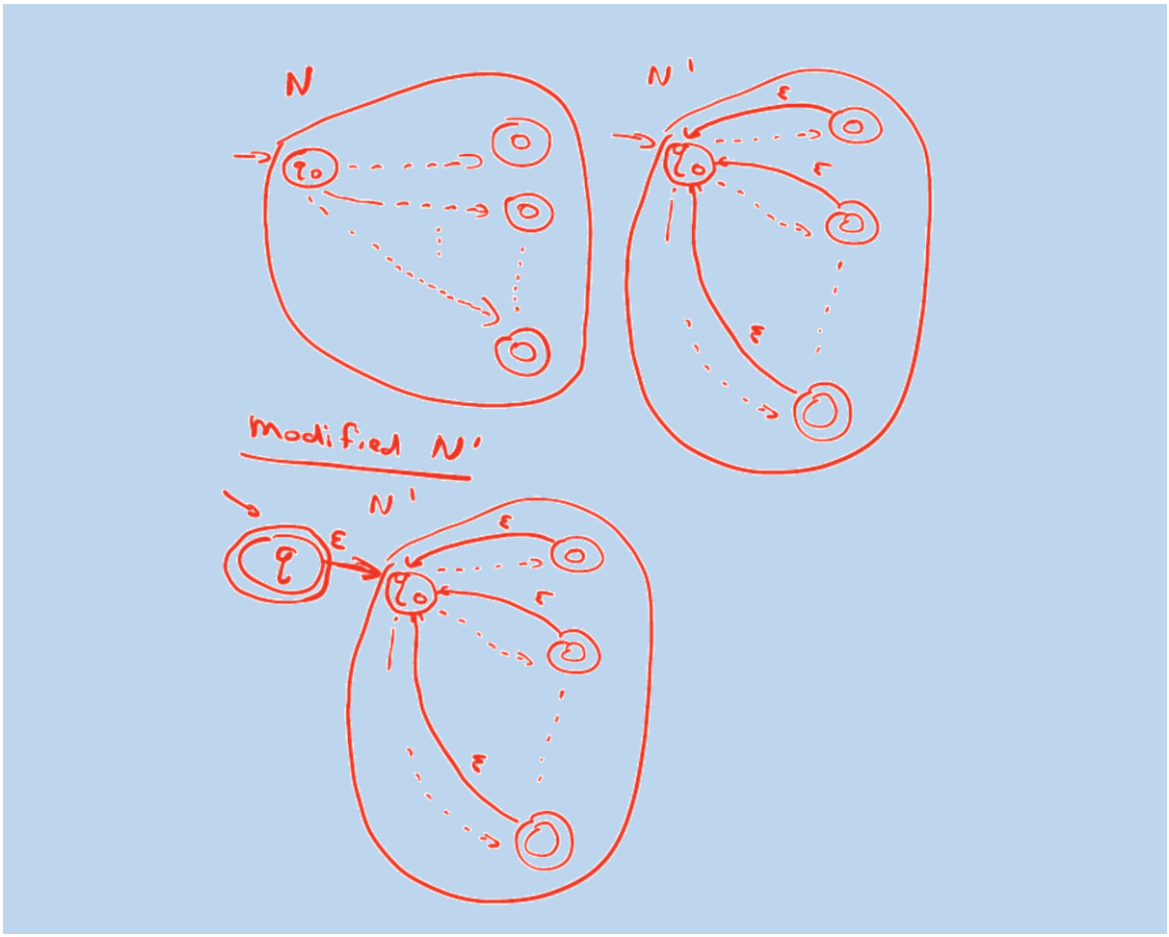
To ensure that N' always accepts A:*

- add a new start state, say q, to N'; *[1]*

- make q an accept state; *[1]*

- add an epsilon-transition from q to the old start state of N'. *[1]*

It's also fine if they draw the NFA N'.


N
N'
ε
ε
ε
Modified N'
N'
q
ε
ε
ε
ε

Question 1 [15 Marks]

a. Briefly explain the techniques below. Be sure the difference between the two problem-solving strategies in each pair is clear. [6]

1. *divide-and-conquer* and *decrease-and-conquer*
2. *space-and-time tradeoff* and *problem reduction*
3. *dynamic programming ("DP")* and *DP with memory functions*

div-conq: solve problem by combining solutions of smaller subproblems

decrease-conq: solve problem by solving smaller instance of that same problem

space-time: solve problem faster by using more space (or vice versa, if space is the problem)

prob reduction: solve problem A by transforming it to an instance of another problem B

DP: solve problem by combining solutions of overlapping smaller subproblems

DP with mem func: as above, but proceed top down instead of bottom up

b. State whether each of the following is true or false: [5]

1. $\Theta(n + \log n) = \Theta(n)$
2. $O(n \log n) = O(n)$
3. if $p \in O(n \log n)$ then $p \in O(n^2)$
4. if $p \in \Theta(n \log n)$ then $p \in \Theta(n^2)$
5. if $p \in \Omega(n \log n)$ then $p \in \Omega(n)$

1. **T** 2. **F** 3. **T** 4. **F** 5. **T**

c. Exponentiation function a^n calculates a to the power n. Show how the problem of finding a^n is tackled using each of the following techniques (1 line each): [2]

1. brute force
2. decrease by a constant
3. decrease by a constant factor
4. divide and conquer

brute force: a * a * … * a (n times)

decrease by constant: $a^{n-1} * a$

decrease by constant factor: $(a^{n/2})^2$

divide and conquer: $a^{n/2} * a^{n/2}$

d. Use the Master Theorem to find the complexity of *waat* below, **showing your working**.

waat (x) { if x < 3 return 'a' else return waat(2x/3) ++ waat(2x/3) } // ++ is concatenate

Master Theorem: If $T(n) = aT(n/b) + f(n)$ then

$T(n) \in \Theta(n^d)$ if $a < b^d$

$T(n) \in \Theta(n^d \log n)$ if $a = b^d$

$T(n) \in \Theta(n^{\log_b a})$ if $a > b^d$ [2]

a is 2, b is 3/2, d is 0

so $a > b^d$ so $\Theta(n^{\log_{3/2} 2})$

Question 2 [17 Marks]

a. Consider searching for the first occurrence of the octal pattern 061606 in octal text 02040606061606

1. Show the shift table that would be used if Horspool's algorithm is applied. [2]
2. Show the good-suffix table that Boyer-Moore would use for this search. [4]
3. The search starts by aligning the pattern with the first character of the text. Where will it move the pattern to next (i.e. for the 2nd alignment) using: [2]
   i. Horspool's algorithm
   ii. the Boyer-Moore algorithm
4. Can the Boyer-Moore algorithm ever have a good-suffix shift table with elements S[j] and S[k] such that j < k but S[j] > S[k]? If no, explain why not. If yes, give an example of such a search pattern. [1]
5. Which algorithm is more efficient: Horspool or Boyer-Moore? [1]

1. Horspool shift table:

0	1	2	3	4	5	6	7
1	**3**	6	6	6	6	**2**	6

2. Good-suffix table:

1	2	3	4	5
2	**4**	**4**	**4**	**4**

3i Horspool shifts up <whatever number they had in HorspoolShiftTable[6] above> or **2**

3ii Boyer-Moore shifts up <whatever number they had in GoodSuffixTable[2] above> or **4**

4. **Yes,** many examples e.g. **anon** or **teepee** or **ceded** other pattern where suffix occurs earlier in string - but not if that pattern's first char/s are also how it ends eg **baba** or **ar**tisanal-b**ar**

5. **Boyer-Moore** (is more efficient).

1. Show the first 2 arrays (only!) that would be generated by Warshall's transitive closure algorithm (i.e. after initialising, and after the 1st iteration of the algorithm). [4]

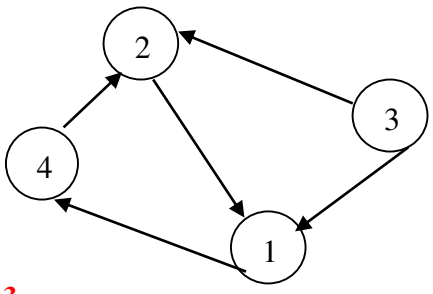
0	0	0	**1**
1	0	0	0
1	**1**	0	0
0	**1**	0	0

0	0	0	1
1	0	0	**1**
1	1	0	**1**
0	1	0	0

or blue below if had rows as "to" and columns as "from"

0	**1**	**1**	0
0	0	**1**	**1**
0	0	0	0
1	0	0	0

0	1	1	0
0	0	1	1
0	0	0	0
1	**1**	**1**	0



2. What is the time complexity of Warshall's algorithm? n^3 [1]

3. What is the space complexity of Warshall's algorithm? n^3 **(but I'll accept n^2 too)** [1]

4. Can its space complexity be improved? If yes state how, if no state why not. [1]

Yes, can use the same one array on each iteration and just overwrite values instead of making an extra new array each time.

Question 3 [8 Marks]

a. You are in a race to find the treasure on a map. Each time you guess a point by giving its X and Y co-ordinates, you are told "YES!" or given the direction (from your guess) in which the treasure is. This direction can be North, South, East, West, North-East, North-West, South-East or South-West. The X values of the map range from -N in the West to +N in the East; the Y values of the map range from -N in the South to +N in the North.

1. Explain briefly how a brute force approach would go about solving this problem. [1]
2. Explain briefly another algorithm that can solve this problem more efficiently than your brute force one. [3]
3. What is the complexity of your brute force algorithm? [1]
4. What is the complexity of your other (more efficient) algorithm? [1]

1. Guess each possible point using 2 nested loops that both go from –N to N until get "YES!"

2. Use **binary search (or decrease and conquer).**

Initialise:Top=N, Botm=(-N), Lef=(-N), Right=N.

Repeatedly **guess** X=Lef + (Right-Lef)/2, Y=Botm + (Top-Botm)/2 and reduce the search area based on reply.

E.g. if reply is "East" set Lef = X+1 and Top=Botm=Y; - - - ; if it's "North" set Botm=Y+1 and Lef=Right=X; - - -;

if it is "South-East" set Lef=X-1 and Top = Y-1.

If reply is "YES!" exit and display X,Y as location of treasure.

1 mark for underlined blue stated or described. **½ point for each underlined red aspect.**

3. N^2

4. log N

b. The *Dutch Flag Problem* is to arrange an array of characters B, O, W (blue, orange, white) so that all the O's come first, then all the W's next, and finally all the B's last. Briefly describe a linear time algorithm for this problem. [2]

Create 3-element array X initialised with zeroes: X[0] is number of O's found, X[1] is number of W's found, X[2] is number of B's found. Do linear scan of array of characters incrementing value of corresponding X element each time. Finally scan X[j] for j=0 to 2 and output X[0] O's, then X[1] W's and lastly X[2] B's.

If just says use distribution sort, or just says use sorting by counting, then only 1 mark not 2.

Question 1 [14]

a) What does the following algorithm do? [1]

```
ALGORITHM mystery(int x,int y)
   while y ≠ 0 do
     t ← x mod y
     x ← y
     y ← t
   return x
```

Answer: It computes the gcd of x and y.

b) Explain your answers in each case below:

i) Is $O(n!) \subseteq O(2^n)$? [2]

ii) Is $O(n \log_2 n) \subseteq O(n!)$? [2]

iii) Is $\Theta(n^3) \subseteq \Theta(n^2)$? [2]

Answer

i) No (1 mark). n! grows faster than 2^n (need to make an argument for growth rate). (1 mark)

ii) Yes (1 mark) . $n\log_2 n$ grows slower than n! (provide some kind of argument for this) but big Oh provides an upper bound. (1 mark)

iii) No (1 mark). n^3 grows faster than n^2 (make an argument for this) so an n^3 function is not in $\Theta(n^2)$. So there are elements of $\Theta(n^3)$ that are not in $\Theta(n^2)$. (1 mark)

c) Consider the following algorithm:

```
ALGORITHM whoknows(int x,int y, int z, int a)
  if a > 0 do
    whoknows(x,z,y,a-1)
    print x, z, a
    whoknows(y,x,z,a-1)
```

i) What does the algorithm above do? [1]

ii) Set up a recurrence relation for the number of basic operations being perfomed and solve it. Show all your work. [3]

Answer:

i) It solves the Towers of Hanoi problem. (1 mark)

ii) T(0) = 0. (1 mark) T(n) = T(n-1) + 1 + T(n-1) = 2T(n-1) + 1 = 2(2T(n-2) + 1) + 1 =… 2^kT(n-k) + (2^k-1). (1 mark) Set k=n. Then T(n) = 2^nT(0) + 2^n-1 = 2^n-1 (1 mark)

d) Consider the two algorithms below. Are they in the same efficiency class or not? Justify your answer. [3]

```
ALGORITHM A2(int n > 0)
if n=1 return 1 else return 1 + A2(n/2)

ALGORITHM A5(int n > 0)
if n=1 return 1 else return 1 + A5(n/5)
```

Answer:

Yes they are. (1 mark) A2 performs roughly $\log_2 n$ basic operations and A2 performs roughly $\log_5 n$ basic operations. (1 mark) Logarithmic functions grow at the same rate, regardless of the base (need to provide some argument for this). (1 mark)

Question 2 [10]

a) Strassen's algorithm for matrix multiplication make use of the fact that two 2x2 matrices can be multiplied as below. Use this result to write an algorithm in pseudocode for matrix multiplication of an n x m matrix by an m x k matrix (yielding an n x k matrix). [8]

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix}$$

$$\begin{bmatrix} m_1+m_4-m_5+m_7 & m_3+m_5 \\ m_2+m_4 & m_1+m_3-m_2+m_6 \end{bmatrix}$$

$$\begin{aligned} m_1 &= (a_{00}+a_{11})*(b_{00}+b_{11}) \\ m_2 &= (a_{10}+a_{11})*b_{00} \\ m_3 &= a_{00}*(b_{01}-b_{11}) \\ m_4 &= a_{11}*(b_{10}-b_{00}) \\ m_5 &= (a_{00}+a_{01})*b_{11} \\ m_6 &= (a_{10}-a_{00})*(b_{00}+b_{01}) \\ m_7 &= (a_{01}-a_{11})*(b_{10}+b_{11}) \end{aligned}$$

Answer:

Fill A and B up with zeroes so as to get two n x n matrices where n is a power of 2. (1 mark)
Then apply the algorithm below.

```
STRASSEN(A, B):

If n = 1 Output A00 × B00 (½ mark)

Else

  Split matrices into 8 n/2 X n/2 parts: A00 , B00 , . . . , A11 , B11

      (1/2 mark)

  M3 = Strassen(A00, B01 – B11) ½ mark

  M5 = Strassen(A00 + A01 , B11 ) ½ mark

  M2 = Strassen(A10 + A11 , B00 ) ½ mark

  M4 = Strassen(A11 , B10 – B00 ) ½ mark

  M1 = Strassen(A00 + A11 , B00 + B11 ) ½ mark

  M7 = Strassen(A01 – A11 , B10 + B11 ) ½ mark

  M6 = Strassen(A10 – A00 , B00 + B01 ) ½ mark

  C11 = M1 + M4 – M5 + M7 ½ mark

  C12 = M3 + M5 ½ mark

  C21 = M2 + M4 ½ mark

  C22 = M1 + M3 – M2 + M6 ½ mark

  Output C ½ mark
```

b) Consider the algorithm below. What is the efficiency class of this algorithm in the worst case? Explain what the input will look like in the worst case. [2]

```
ALGORITHM X(A[0..n-1])
  for i=1 to n-1
    v = A[i]
    j = i-1
    while j ≥ 0 and A[j]>v do
      A[j+1] = A[j]
      j = j – 1
```

```
A[j+1] = v
```

Answer: Efficiency class in the worst case is $\Theta(n^2)$. (1 mark) This will occur if the elements in A are in decreasing order. (This is insertion sort.) (1 mark)

Question 3. **[4]**

Consider the following pseudo-code to find the shortest distance between a set of points on a Euclidean plane, each with an x and y coordinate.

```
1.  Sort the points by x order (tiebreak on the y value) into P.
2.  Sort the points by y order (tiebreak on the x value) into Q.
3.  EfficientClosestPair(P, Q)
4.  EfficientClosestPair(P, Q):
5.    if n <= 3: return minimal distance found by brute force
6.    copy the first ceiling[n/2] points of P to Pl
7.    copy the same points of Q to Ql
8.    copy the remaining floor[n/2] points of P to Pr
9.    copy the same points of Q to Qr
10.   dl = EfficientClosestPair(Pl, Ql)
11.   dr = EfficientClosestPair(Pr, Qr)
12.   .....
13.   m = P[ceiling(n/2)]-1].x
14.   copy all points of Q for which |x-m|<d into S[0..num-1]
15.   dminsq = d * d
16.   for i = 0 to num – 2:
17.     k = i + 1
18.     while k <= num – 1 and (S[k].y – S[i].y)^2 < dminsq:
19.       dminsq = min((S[k].x-S[i].x)^2+(S[k].y-S[i].y)^2, dminsq)
20.       k = k + 1
21.   return sqrt(dminsq)
```

a. Complete line 12. [1]

Answer: d = min(dl, dr) (1 mark)

b. Lines 16 to 20 appear to be $\Theta(n^2)$ in the worst case. Explain why this isn't true, and what the worst case efficiency actually is for these lines of code. [3]

Answer: The inner while loop executes at most five times because there can be at most six points in the region of the plane under consideration. (1 mark) Consequently lines 17 to 19 are in a constant efficiency class. (1 mark) The efficiency class for these lines of code is determined by the for loop on line 15 which is $\Theta(n)$. (1 mark)

Question 4. **[8]**

Consider the following code:

```
1.  X(A[l .. r]):
```

```
2.    p = A[l]
3.    i = l
4.    j = r + 1
5.    repeat:
6.      repeat ++i until A[ i ]>= p
7.      repeat --j until A[ j ]<= p
8.      swap(A[ i ], A[ j ])
9.    until i >= j
10.   swap(A[ i ], A[ j ])
11.  swap(A[l],A[j]
12. return j
```

a. Explain what the algorithm does and give a high-level description of how it accomplishes this. [3]

b. The algorithm contains a bug in lines 6 and 7. Explain what the bug is and how to fix it. [2]

c. Explain why line 10 is necessary. [2]

Answer:

a. Rearranges the list so that it is partitioned into those elements smaller (or equal to) the pivot element p (the value initially in A[l] and those greater or equal to p, with those ≤ p to the left of p, and those ≥ p to the right of p (but not necessarily sorted). (1 mark) It accomplishes this by scanning the list simultaneously from the left, looking for the first value ≥ p (using the index i) and from the right, looking for the first value ≤ p (using the index j), and swapping these. (1 mark) This continues until the indices cross (or are equal). (1 mark) Some explanation such as this.

b. Problem in line 6 if all values in A are less than p. (1/2 mark) Similarly, problem in line 7 if all values are greater than p. (1/2 mark). Solution:

repeat ++i until A[i]>= p or i==r (1/2 mark)

repeat --j until A[j]<= p or j==l (1/2 mark)

c.

If i==j the swap makes no difference (it doesn't swap anything). (1/2 mark) If i>j we will have that A[j] ≤ A[i] before the last swap in the outer repeat loop, and therefore A[j] ≥ A[i] after exiting the outer repeat loop. (1/2 mark) This means we may end up with a situation for which A[j] > A[i], and therefore we could have A[j] > p or A[i] < p. (1/2 mark) Swapping A[l] and A[j] would then produce incorrect results. (1/2 mark) By swapping A[i] and A[j] back just outside of the outer repeat loop we ensure that the new A[i] ≥ p and the new A[j] ≤ p, ensuring that the swapping of A[l] and A[j] produces the correct result. (1 mark)

Question 5. **[5]**

Prove, using the formal definitions of Deterministic Finite Automata and Non-deterministic Finite Automata, that for every Deterministic Finite Automaton there is a Non-deterministic Finite Automaton that accepts exactly the same language.

Answer: Let D = (Q, Sigma, delta, q0, F) be a DFA. (1 mark) We need to find an NFA N such that N accepts the same language as D. (1 mark) We define N as: (Q,Sigma,delta', q0, F) (1 mark) where q' is defined as follows:
For every q in Q and every s in Sigma, delta'(q,s) = {delta(q,s)} (1 mark)
For every q in Q, delta'(q,epsilon) = emptyset (1 mark)

Section 2. Theory of Algorithms [11 Marks]

Question 4 [11 Marks]

a. Brute force and divide-/decrease-and-conquer are techniques most commonly used for problem solving. Name and briefly describe ***any two other*** techniques. [2]

Space-time: use additional data structures/memory/variables/space in order to make a faster solution/algorithm possible.

Dynamic programming: solve overlapping subproblems by storing intermediate results in an array where they are later accessed instead of being recomputed

Transform and conquer: solve a problem more easily by changing it to a more convenient instance of the same problem, or using a different representation, or transforming it to another problem whose solution can be transformed into an answer to the original (any 1 of the three will do)

Greedy: solve a problem by making the optimal choice at each step

b. Consider the Master Theorem below and answer the questions that follow: [3]

If $T(n) = aT(n/b) + f(n)$ then $T(n)$ is $O(n^d)$ if $a < b^d$

$T(n)$ is $O(n^d \log n)$ if $a = b^d$

$T(n)$ is $O(n^{\log_b a})$ if $a > b^d$

i. Which letter indicates how much recursion an algorithm does: a, b or d?

ii. Which indicates how many operations each recursive call does: a, b or d?

iii. What does the remaining letter indicate?

a

d

what fraction of the input is the argument to each recursive call

c. Answer ***either one*** of the following questions: [4]

i. Show the ***shift table*** that would be used by the Boyer-Moore and Horspool algorithms when searching for the string DESEEDED in a text; and state by ***how many positions each algorithm would shift*** the pattern (DESEEDED) after their very first attempt i.e. when they fail to find that string at the start of the text "HE-FOUND-AND-DESEEDED-IT" (state clearly which answer is BM and which Horspool).

Array: D = 2; E = 1; S = 5. Horspool moves 2; BM moves 7.

ii. The initial array A(1) of Floyd's algorithm for finding shortest paths is shown below (∞ represents infinity). Show the ***next array A(2)*** that would result, and show what ***the other array*** used by this algorithm would contain at that stage i.e. when A(2) has been constructed, what does the other array contain?

	1	2	3	4
1	**0**	∞	**3**	∞
2	**2**	**0**	∞	∞
3	∞	**7**	**0**	**1**
4	**6**	∞	∞	**0**

Solution (where "any" can be 0 or null any value other than 1, 2, 3 or 4):

	1	2	3	4
1	**any**	**any**	**any**	**any**
2	**any**	**any**	**1**	**any**
3	**any**	**any**	**any**	**any**
4	**any**	**any**	**1**	**any**

	1	2	3	4
1	**0**	∞	**3**	∞
2	**2**	**0**	**5**	∞
3	∞	**7**	**0**	**1**
4	**6**	∞	**9**	**0**

d. You have N rolls of material, all 1 metre wide but all of different lengths. You need to cut K strips of material all 1 metre wide and want each of the K strips to be the same length. Given K and N, and the length of each of the N rolls, how do you find the maximum length your K strips can be? You can NOT join material from different rolls together, you can only cut. Assume K and N are at most 100.

Outline briefly in words (about 1 line each) two (2) algorithms that have different efficiency classes, and both solve this problem, stating which is more efficient. [2]

Try each possible length in turn going from 1 to 100 (or from 100 to 1)

Do a binary search, which is more efficient than the above.

Section 3. Theory of Computation [8 Marks]

Question 5 [8 Marks]

a. Let N be an NFA, and let N' be an NFA obtained from it by letting the accept states of N' be the reject states of N, and the reject states of N' the accept states of N (so N' is obtained from N by swapping around the accept and reject states). Is the language accepted by N' the complement of the language accepted by N? If it is, explain why it is. If it is not, give an example to show that it isn't. [3]

No it's not. Many examples. Consider the following NFA N over {0,1}. It accepts the language {0}. N' accepts the language {ε}, which is not the complement of L(N). The problem cases are those where the NFA gets stuck

b. Give a regular expression that accepts the following language: [2]

L = { w | w has an odd number of 1s, and ends with a 0}.

*One of many options: 0*10*(0*10*10*)*0*

c. Complete the following proof to prove that the following language is not regular: [3]

L = { $a^n b^n a^n$ | n≥0}.

Proof: Assume L is regular and consider the word w = …… where p is the pumping length. By the pumping lemma, w must be a word of the form xyz where the length of xy is no greater than p and y has a length greater than 0. This means that the word xy^2z is not a word in L because…….: a contradiction.

Assume L is regular and consider the word w = $a^p b^p a^p$ where p is the pumping length. By the pumping lemma, w must be a word of the form xyz where the length of xy is greater than p, and y has a length greater than 0. This means that the word xy^2z is not a word in L because it has the form $a^q b^p a^p$ where q>p (since xy consists of 0s only): a contradiction.

University of Cape Town
Department of Computer Science

Computer Science CSC3003S
Class Test 2, 2021

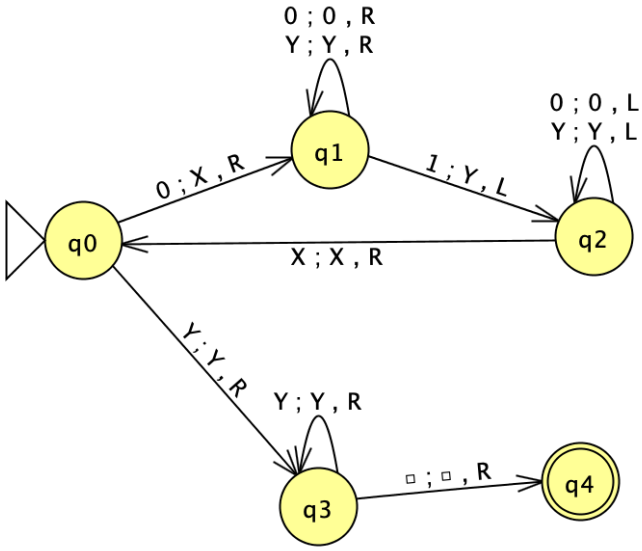
Answer all questions.
Approximate marks per question are shown in brackets.

Time: 40 minutes **Marks: 40**

Section 1: Theory of Computation

Question 1. **[3]**

Consider the following Turing Machine for some language or decision problem that accepts by final state, as depicted in the figure below.



a) Show the computation of the input string 011, using the notation of *instantaneous descriptions*. [1.5]

b) Describe the situation that the Turing Machine is in when it is at that last instantaneous description. [1.5]

a) $q_0011 \vdash Xq_111 \vdash q_2XY1 \vdash Xq_0Y1 \vdash XYq_31$ (and then no more moves are possible)
b) the TM halts in a non-final state and so rejects the input (alternative terminology that is also fine: it dies/stops, and that it will answer 'no' to the question whether the input string is in the language that the TM is designed for)

Question 2. **[7]**

Design a basic – i.e., one tape, one track, deterministic – Turing Machine that, given some sequence of 0s and 1s as input string on its tape, deletes the first symbol (i.e., a 0 or a 1) and shifts all the following 0's and 1's one position to the left, and then halts. For instance, and with B the blank symbol, if there's some piece of tape with just the input on it at the beginning, …BB1010BB…, then it should halt with …BB010BB… on its tape.

Provide the *full definition* of the TM. The transition function may be drawn (using consistent common notation), listed in a list of functions (adhering to the definition of the function), or shown in a table with the states (first column the states, first row the alphabet).

ANSWER - begin

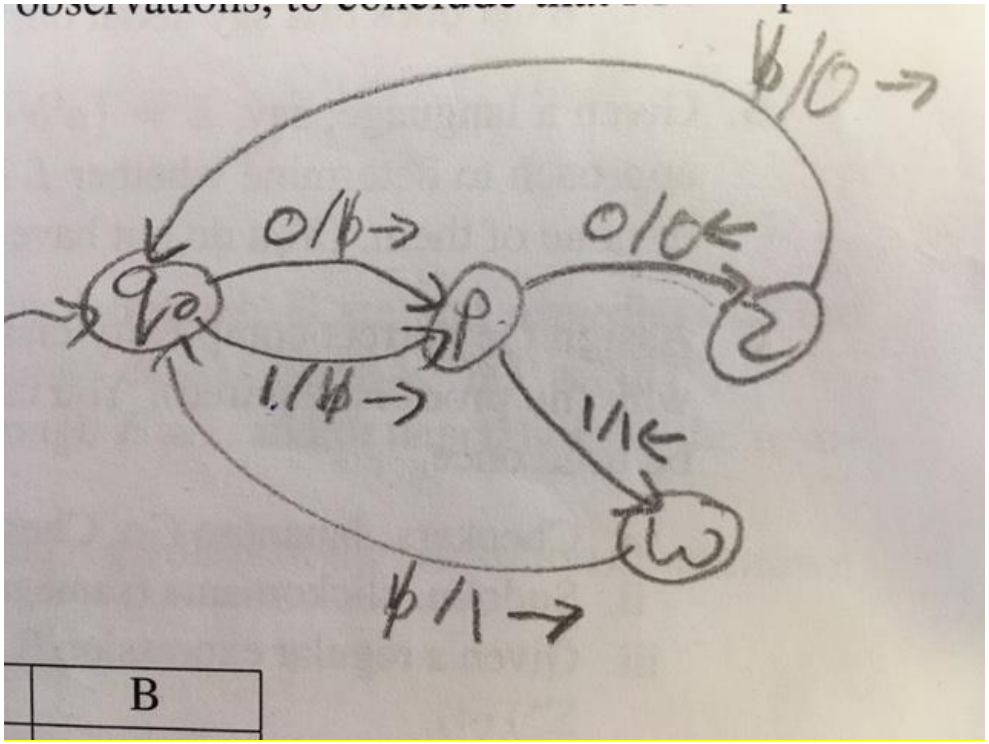
$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where:

- Q is the set of states $\{q_0, p, z, w\}$
- Σ input alphabet $\{0, 1\}$
- Γ tape alphabet $\{0, 1, B\}$
- B the blank symbol
- q_0 start state
- F set of final states
- δ transition function: see Table 2.

Table 2:

State	0	1	B
q_0	(p, B, R)	(p, B, R)	
p	$(z, 0, L)$	$(w, 1, L)$	
z			$(q_0, 0, R)$
w			$(q_0, 1, R)$

Instead of a table with the transition functions, here's my drawing:



Note for marking: either the diagram or the table, or as a list of functions (delta(q0,0) = (p,B,R) etc etc)

Marks: 2 for listing all the TM components (listed before the table above). For the remaining 5 marks: give 5 for everything correct, 2.5 if it does about half of it, 0 if wrong (accepts a different language, incomplete). Deduct 0.5 mark for inconsistent notation, 0.5 for missing the arrow/forgetting to indicate the start symbol, 0.5 for forgetting arrows in the transitions.

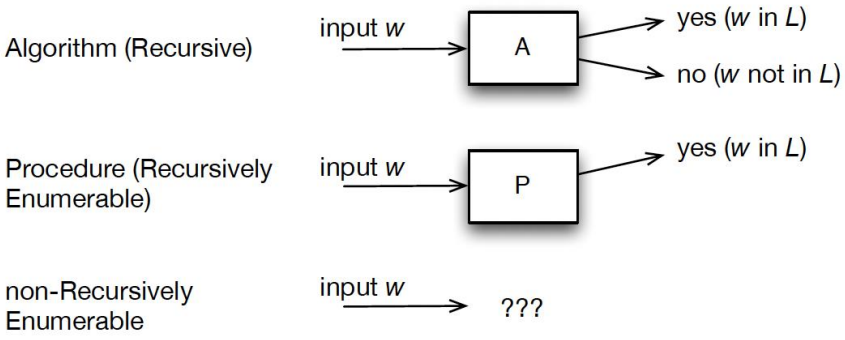
NOTE: accepting by final state is also an option though not fully according to specs. If they added a final state, then that can only be with the transition delta(q0,B) = (qf,B, R) (or L instead of R – doesn't matter). Give 4 marks in that case.

NOTE: it also can be done with a 2 tape machine, but the question says explicitly one tape, one track DTM.

Question 3. **[5]**

Sketch or describe the differences between recursive, recursively enumerable, and non-recursively enumerable languages in the context of Turing machines, and how they relate to something like the Hello World Problem specifically and to algorithms and procedures more generally.

The easiest would be to draw something like the following figure, which would get full marks when the non-RE one has the HWP there:



Alternatively, to describe that the recursive ones correspond to the algorithms, always a y or n on any given input [2]. The RE ones [that are not recursive] correspond to the procedures that can give a y but that's it [2], and non-RE gets into knots (like the HWP or Foo problem) and no TM can do anything with it [1].

Marking note: the "w in L" does not need to be represented. Input "I" and just yes/no is also fine.

Section 2: Theory of Algorithms

Question 4. **[12]**

a) Answer True or False to each assertion below and explain your answers in each case:

i) $\Theta(n \log_2 n) \subseteq \Theta(n)$ [2]

ii) if $y \in O(1)$ then $y \in O(n)$ [2]

iii) if $x \in \Omega(n!)$ then $x \in \Omega(n^2)$ [2]

i) False [1]. Big-theta is an exact bound so it is not possible for an algorithm to be in two different theta efficiency classes [1].

ii) True [1]. Big-O is an upper bound so that if an algorithm is in a better efficiency class (e.g., constant) it is also bound above by a worse efficiency class (e.g., linear) [1].

iii) True [1]. Big-Omega is a lower bound so that if an algorithm is in a worse efficiency class (e.g., factorial) it is also bound below by a better efficiency class (e.g., quadratic) [1].

c) Consider the following algorithm:

```
ALGORITHM strange(int [] A, int i, int j, int x)
  if i >= j do
      k = i + (j - i)/2
      if A[k] == x do
            return TRUE
```

```
    if A[k] > x do
        return strange(A, i, k-1, x)
    else
        return strange(A, k+1, j, x)
else
  return FALSE
```

i) What does the algorithm above do, assuming A is a sorted array? What strategy (algorithmic pattern) does this algorithm use? [2]

ii) Assuming worst-case performance, set up a recurrence relation for the basic operations being performed and solve it using the Master Theorem (see Appendix A). Show all your work. [4]

i) It uses a binary search to determine if x is in the array A and returns TRUE if it is or FALSE otherwise [1]. This is an example of a decrease and conquer algorithm [1].

ii) Basic operation is comparison [1]. In the worst case x is not found in the list. Two comparisons are done and the size of the array is halved on each recursive step, thus:

$C(n) = C(n/2) + 2$ [1]

In terms of the Master Theorem a = 1, b = 2, d = 0 [1].

This fits the $a = b^d$ case so worst-case efficiency is $\theta(n \log n)$ [1].

Question 5. **[7]**

Consider the following problem description:

You are provided with the following set of numbers $s = \{1, 2, 5, 10, 50\}$ and an integer x. Your task is to choose numbers from the set s so that their sum matches x. The numbers chosen may repeat but you must use the fewest numbers possible as part of the summation.

a) Provide detailed pseudo-code for an algorithm that efficiently solves this problem. [5]
b) Run your algorithm for $x = 116$ and show the output. [1]
c) By what name is this problem commonly known? [1]

a)

```
ALGORITHM makechange(int x, int [] s):
    tot = 0
    while tot != x do        [1]
        i = 0
```

while i < len(s) **and** s[i] < (x-tot) **do**: [1/2 for bounds check, 1 for total check]

i= i+1 [1/2]

print s[i-1] [1]

tot = tot + s[i-1] [1]

b) output: 50+50+10+5+1 [1]

c) Change Making [1]

Question 6. **[6]**

Given the text *t* = RATATATTAT

And the pattern *p* = ATT

a) Construct the shift table for Horspool's algorithm. [2]

b) Show the steps taken when the pattern *p* is run against the text *t*. [2]

c) What algorithmic pattern/strategy is in use here? Explain your answer. [2]

a) A = 2, T = 1 [1 for each]

b)

```
RATATATTAT
ATT                  [1/2]
 ATT                 [1/2]
   ATT               [1/2]
     ATT (match)     [1/2]
```

c) Space-time Tradeoff [1]. The pattern is pre-processed to build a table (using extra space) in order to speed up the matching process (reducing time), hence trading off additional space for reduced time [1].

Section 3: Theory of Algorithms [9]

Question 5. **[2]**

You are given a recurrence relation for an algorithm:

$$T(n) = 3T\left(\frac{n}{4}\right) + n^3 + 1, for\ n > 1, and\ T(1) = 5$$

Solve this using the Master Theorem.

ANSWER:

a = 3, b = 4, c = 5, d = 3 1 mark

→ a = 3 < b^d = 4^3 0.5 mark

→ T(n) ∈ $\Theta(n^3)$ 0.5 mark

Question 6. **[3]**

Given the text *t* = BIB_A_BITT

And the pattern *p* = BIT

Construct the shift table for Horspool's algorithm for the alphabet {A-Z, _}, then run the matching algorithm, showing the steps/shifts taken when the pattern *p* is run against the text *t*

ANSWER:

Table:

B	I	Other letters, _
2	1	3

½ mark for B

½ mark for I

½ mark for other alphabet characters

Apply

```
BIB_A_BITT
BIT
  BIT (2 shift)                    ½ mark {reason: T(B) = 2}
     BIT  (3 shifts)                 ½ mark {reason: T(A) = 3}
      BIT (1 shifts = match!)     ½ mark {reason: T(I) = 1}
NOTE: reason need not be written for the mark, shifts must be
correct though
```

Question 7. **[3]**

You are given the following knapsack problem solution, for up to 5 items (with indicated weights and values) and a knapsack of maximum capacity 6.

		capacity j						
	i	0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
$w_1 = 3, v_1 = 25$	1	0	0	0	25	25	25	25
$w_2 = 2, v_2 = 20$	2	0	0	20	25	25	45	45
$w_3 = 1, v_3 = 15$	3	0	15	20	35	40	45	60
$w_4 = 4, v_4 = 40$	4	0	15	20	35	40	55	60
$w_5 = 5, v_5 = 50$	5	0	15	20	35	40	55	65

Use **back-tracing** to determine the optimal set of items included in a knapsack of up to **3** items and a capacity of **6**. Show each step.

ANSWER:

We need to start with F(3,6) = 60, ½ mark
Then, check if F(3,6) == F(2,6) → NO. This means **item 3** is in the set, ½ mark
We then move to cell i = 2, j = 6-w_3 = 5 ½ mark
Then we check if F(2,5) == F(1,5) → NO. **So item 2** is in the set ½ mark
We then move to cell i = 1, j = 5-w_2 = 3 ½ mark
Then we check if F(1,3) == F(0,3) →NO, so **item 1** is in set ½ mark
We then move to cell i = 0, j = 3-w_1 = 0 -> done

Question 8. **[1]**

Give an example of where the "greedy change making" algorithm fails to give the least amount of change for a desired amount, assuming currency denomination = {1, 15, 25}.

ANSWER:

many – example Value = 30, gives 30-25, then 5 X 1 coins for 6 coins, instead of 2 x 15 coin. (any example where the change is not minimal, based on common sense!)

Section 4: Theory of Computation [9]

Question 9. **[1]**

Give a regular expression for the language of binary numbers i.e. strings comprising any number of 0s and 1s in any order.

ANSWER:

(0|1)*

Question 10. **[4]**

Draw a transition diagram for a Turing Machine that converts an input binary string into its one's complement form – i.e. every 1 is replaced by a 0 and every 0 is replaced by a 1. Assume that the tape head starts at the cell where the input binary string starts. The computation must end with the tape head at the first blank cell encountered.

ANSWER:

1 mark for correctly showing initial and halt state

1 mark for correctly showing nodes, edges and edge labels

2 marks (all or nothing here) if correct for any input (including blank tape i.e. no input).

Machine should have 3 states, I'll call them F (first, the initial state), D (done a digit) and H (halt).

Arc from F to H labelled blank/blank

2 arcs from F to D: one labelled 0/1 and the other labelled 1/0

2 arcs from D to F: one labelled 0/R and the other labelled 1/R

Question 11. **[2]**

Give ***any one*** entry that would appear in the state table for your Turing Machine above.

ANSWER:

1 mark for columns; 1 mark for values in the example entry exactly matching any arc in their TM (whether or not their TM does one's complement correctly or not)

Columns are:

current state; current symbol; next state; action

action is 1 of these, can indicate which in any way:

L/move Left or R/move Right or O (write 0) or 1 (write 1) or blank (write blank)

e.g. the arc from F to H in my example would have entry as below:

F, blank, H, blank

etc.

Question 12. **[2]**

Briefly state the difference between an acceptable language and a decidable language.

ANSWER:

Same as in original test.

Acceptable: there is an algorithm that can halt and say Yes if and only if it is given a string in the language

Decidable: there is an algorithm that will halt and say Yes if it is given a string in the language and will halt and say No if it is given a string that is not in the language

Appendix A – Master Theorem

If we have a recurrence of this form: $T(n) = aT(n/b) + f(n)$, $f(n) \in \Theta(n^d)$ and $T(1) = c$ then

$T(n) \in \Theta(n^d)$ if $a < b^d$

$T(n) \in \Theta(n^d \log n)$ if $a = b^d$

$T(n) \in \Theta(n^{\log_b a})$ if $a > b^d$

Section 1: Theory of Algorithms [22]

Question 1. **[8]**

a) Explain the difference between *best case*, *worst case* and *average case* time efficiency for an algorithm. [2]

b) Answer True or False to each assertion below and briefly explain your answers in each case:

i) $\Theta(\log_2 n) \subseteq \Theta(n)$ [2]

ii) if $y \in O(n)$ then $y \in O(n^2)$ [2]

iii) $n + \log n \in O(n)$ [2]

ANS:

a) The best/worst/average time efficiencies are the times (in terms of basic operations) that result when specific arrangements of *inputs* are passed into the algorithm; specifically the inputs that produce the fastest running time (best case), the slowest running time (worst case) and the average running time (average case). [at least 1 mark for noting role of INPUTS in deciding this; 2 marks if the distinction is made between best/average/worse case]

b)

i) False (1); log n has lower asymptotic growth than n, they belong to different asymptotic growth classes (big-theta is an exact bound, not upper/lower bound) (1)

ii) True (1); n is bounded above by n*n; in terms of O(.) definition, all functions in O(n) are also in O(n*n) (1)

iii) True (1); for big-oh, we can disregard terms with lower growth rates (can also use (O(max{n, log n}), which is proved in textbook) (1)

Question 2. **[6]**

Consider *ternary search* which uses the following algorithm for searching in a *sorted array* A[0..n − 1]:

If n = 1, compare the search key K with the single element of the array; otherwise, search recursively by comparing K with $A[\lfloor n/3 \rfloor]$, and if K is larger, compare it with $A[\lfloor 2n/3 \rfloor]$ to determine in which third of the array to continue the search.

i) What strategy (algorithmic pattern) does this algorithm use? [1]

ii) What is the basic operation? [1]

iii) Assuming worst-case performance, and $n = 3^k$, set up a recurrence relation (including base case) for the number of basic operations being performed and solve it using the Master Theorem (Appendix A). Show all your work. [4]

ANS:

i) decrease by constant factor (3 here) and conquer (1)

[Note: if divide and conquer is stated, award ½ mark]

ii) The basic operations is key comparison (1)

iii)

We do TWO key comparisons per stage to determine where to continue (1/ 2)

Then we recursively investigate the relevant 1/3 of the array (1/2)

For 1 element, we need to check that single element: C(1) = 1 (1/2)

Thus number of comparisons is

C(n) = 2 + C(n/3) (1)

For Master Theorem: (recall: $T(n) = aT(n/b) + f(n)$, $f(n) \in \Theta(n^d)$ and $T(1) = c$)

a = 1, b = 3, c = 1, d = 0 (1/2)

Because $a = b^d$,(1 = 3^0) ,

we use form $T(n) \in \Theta(n^d \log n)$ (1/ 2)

→ $C(n) \in \Theta(\log n)$ (1/ 2)

[Aside: In fact $C(n) = 2 \log_3 n + 1$, but the extra comparison means it loses out to binary search]

Question 3. **[4]**

Given the text *t* = RABID_RABBIT

And the pattern *p* = ABB

Construct the shift table for Horspool's algorithm for the alphabet {A-Z, _}, then run the matching algorithm, showing the steps/shifts taken when the pattern *p* is run against the text *t*.

ANS:

Table:

A	B	Other letters, _
2	1	3

½ mark for A

½ mark for B

1 mark for other alphabet characters

Apply

```
RABID_RABBIT
```

```
ABB  (fail, then 1 shift from T(B)                    1/2 mark
 ABB (fail, then 3 shifts from T(I))                  1/2 mark
    ABB  (fail, then 3 shifts from T(R) )             1/2 mark
       ABB   (match!)                                 1/2 mark
[Note: the explanations do not have to be included, but shifts
must be correct]
```

Question 4. **[4]**

Apply Warshall's algorithm to find the transitive closure of the directed graph defined by the following adjacency matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Show your working by constructing the intermediate matrices and indicating the rows/columns used for each stage.

ANS:

1 mark per stage for R1, R2, R3, R4 - 4 marks (bold shows added entries per stage); if correct rows/cols shown to compute each stage (e.g. row 1, col 1 of R(0) needed for R(1)…) but final answer is incorrect, award 1 mark at end

$$R^{(0)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad R^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R^{(2)} = \begin{bmatrix} 0 & 1 & \mathbf{1} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad R^{(3)} = \begin{bmatrix} 0 & 1 & 1 & \mathbf{1} \\ 0 & 0 & 1 & \mathbf{1} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R^{(4)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = T$$

Section 2: Theory of Computation [13]

Question 5. **[4]**

Briefly state the difference between:

i) An acceptable language and a decidable language [2]

ii) An upper bound and a lower bound for a problem [2]

ANS:

1 mark per "definition"

i) Acceptable: machine will answer YES & halt iff string in language.

Decidable: acceptable & machine will answer NO & halt if string not in language.

ii) Upper bound: complexity of most efficient algorithm solving it.

Lower bound: proven complexity of any algorithm solving it.

Question 6. **[5]**

Prove that the language {Turing Machines that halt when given themselves as input} is not decidable.

ANS:

Assume M^H can do this

Construct M^0 as below:

Feed input to M^H i.e. $M^H(T,T)$ where T is a TM (1)

if M^H stops with answer YES, loop forever. (1/2)
if M^H stops with answer NO, halt. (1/2)

Consider feeding M^0 as input to itself (1)

if it halts, M^H gives YES so it doesn't halt. (1/2)
if it doesn't halt, M^H gives NO so it halts. (1/2)

Contradiction. (1)

Question 7. **[4]**

Draw a transition diagram for a Turing Machine with alphabet {◊,1} that does natural number addition, and stops with the tape head at the cell immediately right of that total. A number N is represented by a string of N consecutive 1s. The tape head will start on a blank cell, followed by the numbers to add with a single blank cell between them. Example input for 2 + 4:

◊11◊1111◊

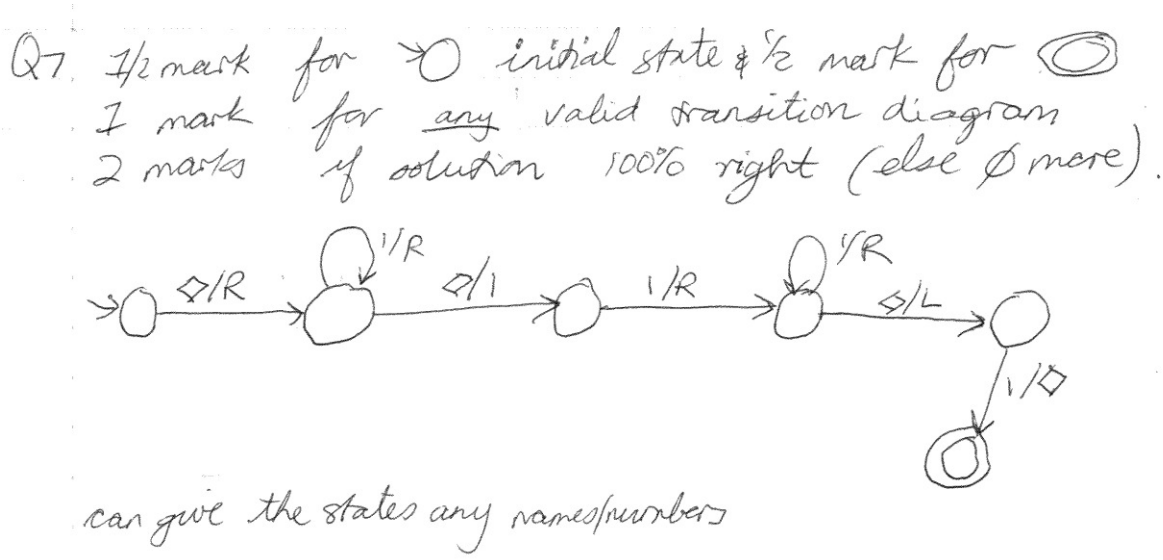
which must generate output:

◊111111◊◊

ANS:

½ mark for initial state & and ½ mark for end state
1 mark for any valid transition diagram
2 mars if solution 100% right (else no more)

Can give the states any names/numbers



Appendix A – Master Theorem

If we have a recurrence of this form: $T(n) = aT(n/b) + f(n)$, $f(n) \in \Theta(n^d)$ and $T(1) = c$ then

$T(n) \in \Theta(n^d)$ if $a < b^d$

$T(n) \in \Theta(n^d \log n)$ if $a = b^d$

$T(n) \in \Theta(n^{\log_b a})$ if $a > b^d$

Section 3. Theory of Algorithms (TOA) [9 marks]

Question 4 **[6]**

a) Which statement(s), among the following, is FALSE about breadth-first search when it is used on graph-based problems [3]

(i) A queue must be used to keep track of adjacent nodes and the order in which they need to be visited.
(ii) The algorithm's efficiency for adjacency lists is BigTheta(|V|) where represents the vertices/nodes
(iii) There are two orderings of the nodes that can be obtained via the algorithm
(iv) Topological sorting cannot be achieved using BFS since only Depth-First Search is appropriate.
(v) It can be used to detect cycles in directed graphs but not undirected graphs.

(ii, iii, iv, v) are all false. 3 marks if the answer is FULLY correct, 1 mark if the answer is partially correct.

b) **Fake-Coin problem:** consider the following algorithm for finding a single coin that weighs less than all others from an array of n coins where all others have the same weight and answer the question that follows. [2]

```
ALGORITHM findFakeCoin(coins):
   for i in range(1,n-2) do
        left = coins[i-1]
        mid = coins[i]
        right = coins[i+1]
        if left + mid < mid + right do
              return min(left, mid)
        else left + mid > mid + right do
              return min(mid, right)
```

Is the algorithm an example of a divide-and-conquer strategy? Explain your answer.

No. The input is never split into m equal sub-problems. Instead, the for-loop iterates over n-2 of the items and compares the current coin to its left and right hand side.

c) Consider the text "This is a problem" and the pattern "prob". How many unsuccessful alignments of the text and pattern will be conducted by Horspool's algorithm before it finds the pattern? [1]

"prob" compared to "This" and is unsuccessful
"prob" compared to " is " and is unsuccessful
"prob" compared to "a pr" and is unsuccessful
"prob" compared to "prob" but it is successful
ANS: Three

Question 5 **[3 marks]**

Consider the following scenario and answer the question that follows:

You are given the following messages and each message's associated measure of information content.

Message 1 = ('a', 2)
Message 2 = ('bb', 4)
Message 3 = ('bca', 16)

The cost of sending a single character is 1 cent and you want to transmit messages with the most amount of information (measured in bits).

Write down the recurrence relation, if you are asked to use Dynamic Programming, to calculate the maximum amount of information you can transmit if you can only transmit at most n cents worth of information.

Let i ($0 \leq i \leq 3$) represent the transfer of i messages and j ($0 \leq j \leq n$) represent the amount of information being sent. Furthermore, let the cost of sending i messages be represented by $cost_i$ and the information amount associated with sending i messages be $info_i$ then the optimal solution can calculated as follows:

$$C(i,j) = \begin{cases} max\{C(i-i,j), info_i + C(i-1, j-cost_i)\} & \text{if } j \geq cost_i \\ C(i-i,j) & \text{if } j < cost_i \end{cases}$$

Where $C(0,j) = 0$ for $j \geq 0$ and $C(i,0) = 0$ for $i \geq 0$

(1 mark for each condition in the equation, 1 mark for specifying what the variables are)

Section 4. Theory of Computation (TOC) [8 Marks]

Question 9 [1]

Consider the following Turing machine with states q, p, and r, where q is the start state and r the final state, Γ = {0, 1, □} (where the box "□" indicates the blank symbol), Σ = {0,1}, and the following transition functions:

State	Tape Symbol	Move
q	0	(q,0,R)
q	1	(p,0,R)

q	□	(q, □,R)
p	0	(q,0,L)
p	1	(r,1,R)
p	□	(p,0,R)

Describe a property of the input strings that makes this TM accept by final state. Then select the string from the options listed which make this TM accept.

a. 01100
b. 010□1□1
c. 0001
d. 1010

Answer: A

Question 10 **[3]**

Suppose we have the following four languages and all we know about them is the following:

- A is Recursive
- B Recursively Enumerable but not Recursive.
- C is Recursively Enumerable
- D is non-Recursively Enumerable.

Select all the correct statements from the options below:

a. There definitely exists a function R such that C reduces to A.
b. There cannot be a reduction from D to B, because else it would prove that B is non-Recursively Enumerable, which it is not.
c. If $\overline{B}$ (complement of B) is also Recursively Enumerable, then both $\overline{B}$ and B are recursive.
d. Either concatenating A and B or taking the union of A and B may give us C.

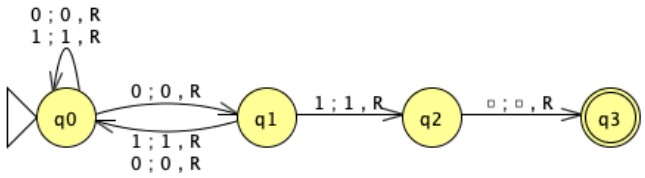
Answer: B and C. 1.5 mark for each.

Question 11 [4 Marks]

Design a Turing Machine that accepts all strings that have 01 as its last two characters.

Hint: first describe in short statements the three component tasks that the TM needs to accomplish. Then draw the state diagram that accomplishes that.

Answer: an NTM is the easiest:



Marking guide: **deduct 0.5** for not indicating start state (an incoming arrow, -->), and **deduct 0.5** for not indicating final state (double lined circle). **A state diagram as above gains the 4 marks.** If they write all the transition functions as functions, then **also** give the full marks.

If they draw a TM that accept only 01 (i.e., no loop on q0 and no transition back from q1 to q0), then give **1 mark**.

If they can describe how their TM would solve it even if their state diagram is wrong or is not present, Give **2 marks** when says something along the line of:

1) It steps over the 0s and 1s for any number of times
2) At some point, it guesses that it will be the last 0 of the input;
3) Then it guesses that the next symbol is the final 1;
4) When it sees a blank afterward, it goes to the final state.

With **0.5** for each sub-task. Marking note: talking of *traces* instead of *guesses* is fine as well, like “at least one trace”.

-END-

Section 2. ToA: Theory of Algorithms [16 Marks]

Question 4 [3Marks]

You have been given five variants of an algorithm for achieving the same task. You have analysed the number of times the basic operation is executed based on the input and determined that it can described as follows, for each variant:

Variant 1: n^2
Variant 2: 6n
Variant 3: n!
Variant 4: n log n
Variant 5: 2^n

Order the algorithm variants from best to worse efficiency --- for large input sizes.

In order:

6n
nlogn
n^2
2^n
n!

3 marks if completely correct, 2 marks if any 1 error, 1 mark if any 1 correct

Question 5 [3 Marks]

Given the multiplication algorithm (à la Russe) below:

```
ALGORITHM multiply(n, m)
   t  = 0
   while n != 1 do
         if n % 2 != 0 do
               t += m
         m = m * 2
         n = n // 2
   t += m
   return t
```

a. To which algorithmic pattern/strategy, among the following, does the algorithm belong? [1]

(a) divide and conquer
(b) decrease by a constant
(c) **decrease by a constant factor**
(d) brute force
(e) transform and conquer

b. Does the algorithm exhibit the same performance, from a time perspective, when the values of n and m are swapped (e.g., n=3 and m=10 vs. n = 10 and m = 3)? Explain. [2]

No **[1]**. The number of executed steps is determined by how many times n must be halved until it reaches 1 (i.e., problem size is characterised by n). As such, a larger value of n implies a larger number of execution steps. For instance, consider the case where n=3 and m=10. When n=3, a single loop is sufficient whereas three are necessary for n=10 **[1]**

Question 6 [10 Marks]

Given the algorithm below:

```
ALGORITHM func(A, l, r, i):
  if l < r:
        v = (l + r) // 2
        if A[v] % i == 0 do
              PRINT v
        else:
              func(A, v + 1, r, i)
              func(A, l, v - 1, i)
```

a. What does the algorithm do? [2]

Uses divide and conquer to search and print some element(s) from A that are divisible by i. The values must be at the centre of each segment of A.

b. Write pseudo-code for the brute-force version of the algorithm that prints all the values that satisfy line 4's condition? [2]

```
(let n be the size of A)
for idx in [0 ... n] do
   if i%A[idx] == 0 do
         PRINT i
```

c. Use the master theorem to show that both the algorithm as presented and its expanded brute-force version belong to the same efficiency class in the worst case (Show your work).
[6]

Solve using the master theorem

If $T(n) = aT(n/b) + f(n)$ and $f(n) \in \Theta(n^d)$ then

- $T(n) \in \Theta(n^d)$ if $a < b^d$
- $T(n) \in \Theta(n^d \log n)$ if $a = b^d$
- $T(n) \in \Theta(n^{\log_b a})$ if $a > b^d$

In the case of the brute-force solution, worst case performance is seen when the element is at the last position (n-1) **[1].** We can compute the sum from 0 to n-1 of doing a single comparison 1 + 1 + …. + 1 = n hence T(n) in BigO(n) **[1]**

In the case of the divide and conquer approach, we have the recurrence relations T(0) = 0 and T(n) = aT(n/b) + f(n) where a=2**[1],** b=2**[1]** and f(n) in BigTheta(n^0) so d = 0 **[1].** T(n) in BigTheta(n^log_2^2) = BigTheta(n). Hence T(n) in BigO(n) --- by definition of BigTheta **[1].**

Section 3. ToC: Theory of Computation [9 Marks]

Question 9 [1 Marks]

Consider the following Turing machine with states q and p, where q is the start state, Γ = {0, 1, □} (where the box "□" indicates the blank symbol), Σ = {0,1}, and the following transition functions:

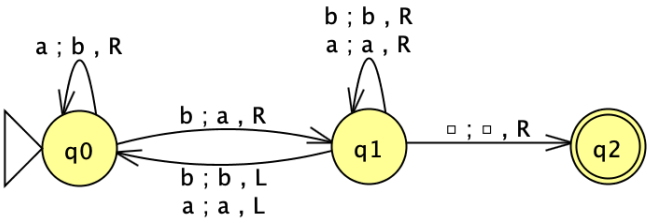
State	Tape Symbol	Move
q	0	(q,0,R)
q	1	(p,0,R)
q	□	(q, □,R)
p	0	(q,0,L)
p	1	none (halt)
p	□	(q,0,L)

Describe a property of the input strings that makes this TM halt. Then select the string from the options listed which make this TM halt.

a. 10010
b. 0100
c. 0001
d. 10110

Question 10 [2 Marks]

Consider the following state diagram of a Turing machine, with q_0 the start state and q_2 the final state, and where the □ (small square box) denotes blank.



The TM with these states and transitions is given input w = aaababa. Examine the TM on this input and then select the correct answer from the following options:

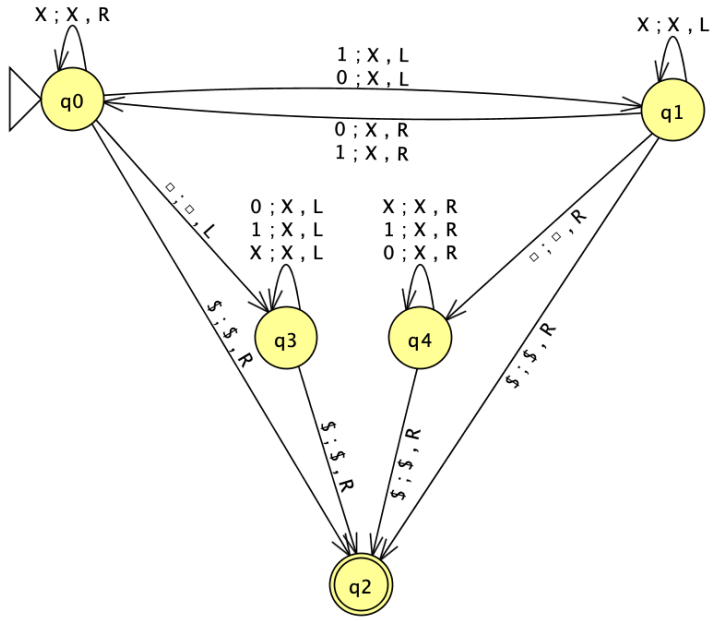
a. Reachable IDs include bbbaaaq$_1$a and bbbaaaq$_1$b.
b. Reachable IDs include bbbaaq$_1$ba and bbbaaba□q$_2$□.

Question 11 [6 Marks]

Design a deterministic Turing Machine such that, when its head is on some arbitrary position somewhere on the input that is of pattern w$w with w from the alphabet {0,1}, it will find the centre symbol $ and accept. So, it should accept, e.g., 0001$0001 and 1$1 on the tape, but reject 11.

Hint: first describe in short statements the three component tasks that the TM needs to accomplish. Then draw the state diagram that accomplishes that.

Answer: alike the one from the exercises with the blanks and using #, but then with marking off 0 and 1 with an X. Marking note: blank can be indicated with a □, B or a b with a diagonal line through it.



Marking guide: **deduct 0.5** for not indicating start state (an incoming arrow, -->), and **deduct 0.5** for not indicating final state (double lined circle). **A state diagram as above gains the 6 marks.**

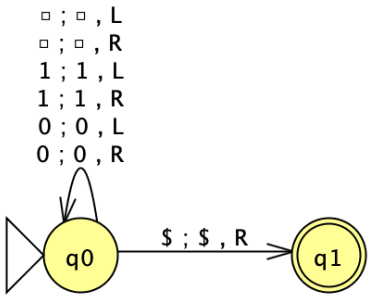
Give **2 marks** if they can describe how their TM would solve it even if their state diagram is wrong or is not present. It should say something along the line of:

1) it alternates looking right and left (*marking note*: or vv) for $, marking off each cell it has seen;
2) when it encounters the blank, it will go only in the other direction;
3) when it encounters $, it goes to the final state.

With **0.5** for each sub-task and **another 0.5** for having all three.

If they write all the transition functions as functions, then also give the full marks.

If they draw an NTM that is correct ('guess whether $ is to the left or to the right' + 'goto final upon $'), give **1 mark**. The 'simplest' NTM is:



-END-

Section 2. Theory of Algorithms [24 Marks]

Question 2 [24 Marks]

a. Which **TWO** of the following statements are **TRUE**? [2]

i. $\Theta (n + \log n) = \Theta (n)$

ii. if $x \in O (n \log n)$ then $x \in \Theta (n \log n)$

iii. if $x \in O (n \log n)$ then $x \in O (n^2)$

iv. if $x \in O (n \log n)$ then $x \in O (n)$

i. $\Theta (n + \log n) = \Theta (n)$ *[1]*

iii. if $x \in O (n \log n)$ then $x \in O (n2)$ *[1]*

b. Show the different approaches of the algorithmic techniques below, using the example of exponentiation (a^N). Each answer should take one line. [4]

i. Brute force

ii. Divide and conquer

iii. Decrease and conquer

iv. Transform and conquer

i. *a * a * a .... * a N times* *[1]*

ii. $(a^{N/2}) * (a^{N/2})$ *[1]*

iii. $(a^{N-1}) * a$ OR $(a^{N/2})^2$ *[1]*

iv. Horner's rule OR Binary Exponentiation *[1]*

c. Two other techniques are Greedy Algorithms and Space-Time-Trade-off. Briefly describe these 2 approaches to problem solving (1 line each). [2]

Space-time: use additional data structures/memory/variables/space in order to make a faster solution/algorithm possible *[1]*

Greedy: make the optimum choice at each step. *[1]*

d. We have the academic records of all UCT students who have taken any CSC course during the past decade. We wish to sort them by alphabetical order of their Faculty (so all Commerce students first, then all Engineering ones, … finally all Science students). Students from all 6 of UCT's Faculties have taken a CSC course.

i. Outline (do not just name it) a method of solving this problem which is more efficient than quicksort and also uses key comparisons (i.e. do not describe radix sort). [2]

ii. What is the complexity of your more efficient algorithm above? [1]

i. Go through input counting how many from each faculty. [1 mark].

Then set the 6 end-position values for each Faculty accordingly. Finally go through the input again from end to start, using the end-position of their Faculty to put them in the sorted array, and decrement that value to indicate that position has been filled. [1 mark]

ii. linear i.e. O(N). [1]

e. You are searching for the first occurrence of the string REFERER in a text.

i. Show the shift table that Horspool's algorithm would use. [2]

ii. Show the good-suffix shift table that the Boyer-Moore algorithm would use for this search string REFERER. [2]

i.1 mark for characters (E,F,R) 1 mark for values. Elements in any order

E	F	R
1	4	2

ii. 1 mark for indexes (1 to 6). 1 mark for values. Elements in any order.

1	2	3	4	5	6
6	2	6	6	6	6

f. You are given a row of integers. You can choose as many as you like except that you cannot take any 2 integers that are adjacent (next to each other). For example, if the row has values 4, 2, 8, 6 then you can take the 4 only of you don't take the 2, you can take the 2 only if you take neither the 8 nor the 4, … you can take the 6 only if you don't take the 8.

i. State in words or pseudocode how you would you use Dynamic Programming to find the maximum value you can take [2]

ii. Show what the array would contain after solving the example below: [1]
4, 2, 8, 6
Maximum is 12 (take the 4 and the 8).

i. Go from start to end of list, each time choosing max(with new value, without new value) and storing it in the array [1 mark]

For the j'th item, A[j] is set to max(A[j-2]+newval , A[j-1]) [1 mark]

ii. Array [1 mark]

1	2	3	4
4	4	12	12

g. The results of a race are recorded in a file of 100 000 records, where each record contains the position that person came and the time they took, e.g. < 42, 1234> means the person came 42nd and took exactly 1234 seconds to run the race. The file is in order from winner to loser, i.e. in increasing order of position starting at 1.

Consider the problem of knowing the time you took in the race and wanting to look up your position i.e. the input is the number of seconds (e.g. 1234) and the output is the corresponding position (e.g. 42). Assume there were no ties.

Outline briefly in words (about one line each) ***three*** (3) different algorithms you could use to solve this problem and give the complexity of each. [6]

Go through the entire list from start to end which is linear or O(N). *[2 marks]*

Do a binary search which is logarithmic O(log N). *[2 marks]*

Use linear interpolation which is O(log log N). *[2 marks]*

Section 3. Theory of Computation [9 Marks]

Question 3 [9 Marks]

a. Prove, using the formal definitions of Deterministic Finite Automata and Non-Deterministic Finite Automata, that for every Deterministic Finite Automaton there is a Non-Deterministic Finite Automaton that accepts exactly the same language. [4]

Let D = (Q, Sigma, delta, q0, F). We need to find an NFA N such that N accepts the same *language as D.* *[1]*

We define N as: (Q,Sigma,delta', q0, F) where q' is defined as follows: *[1]*

For every q in Q and every s in Sigma, delta'(q,s) = {delta(q,s)} *[1]*

For every q in Q, delta'(q,epsilon) = emptyset *[1]*

b. Suppose we want to prove that for any regular language A, A* is also regular. We know that if A is regular, then there is a Non-Deterministic Finite Automaton N such that N accepts A. Now, from N we construct a new Non-Deterministic Automaton N' as follows: for every accept state s of N, add an epsilon-transition from s to the start state of N (if there isn't one already). Does N' accept A*? If it does, explain why in detail. If it does not, explain why not, indicate what language it accepts, and explain how to modify N' so that it accepts A*. [5]

N' does not always accept A.* *[1]*

If A contains the empty string, then N' accepts A. If not, then N' accepts every element of A* except the empty string.* *[1]*

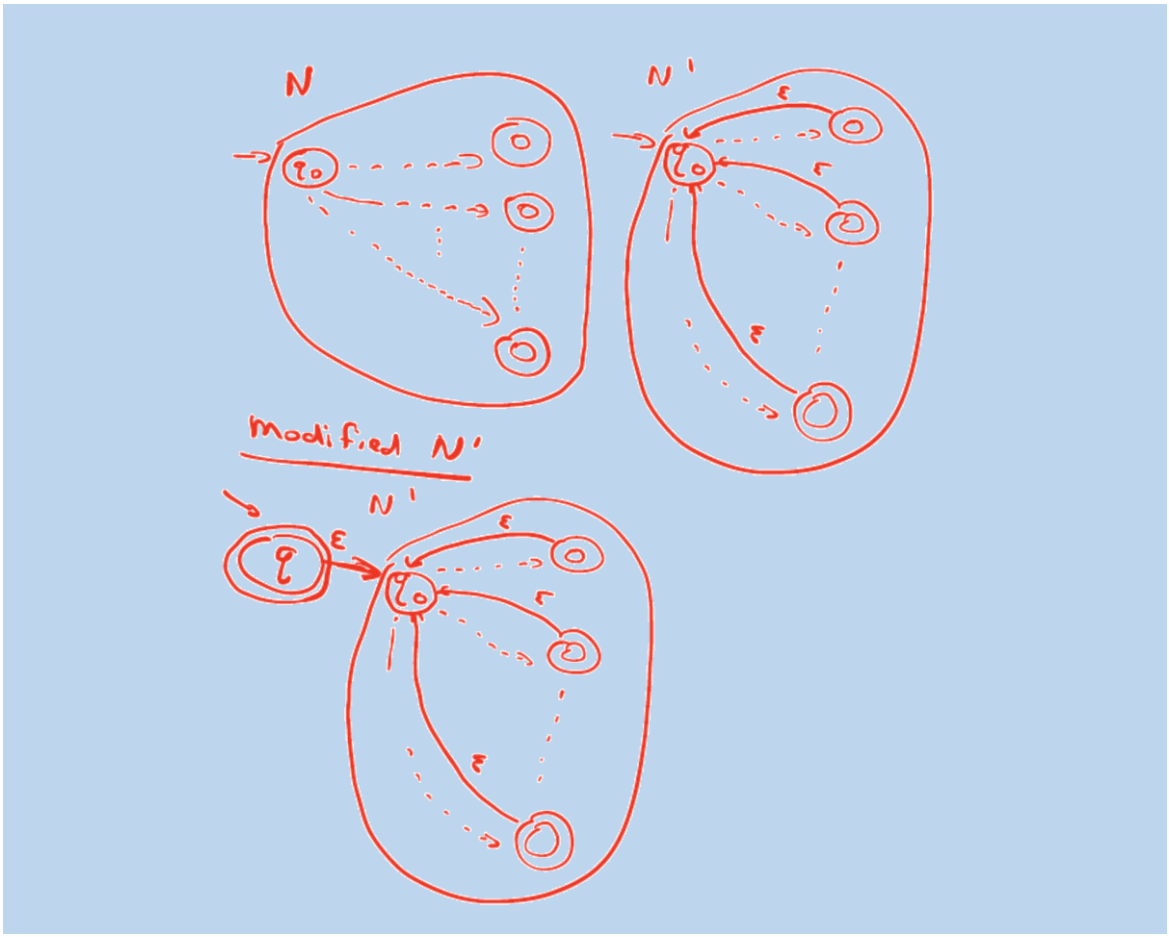
To ensure that N' always accepts A:*

- add a new start state, say q, to N'; *[1]*

- make q an accept state; *[1]*

- add an epsilon-transition from q to the old start state of N'. *[1]*

It's also fine if they draw the NFA N'.


N
N'
ε
ε
ε
Modified N'
N'
q
ε
ε
ε
ε

University of Cape Town
Department of Computer Science

Computer Science CSC3003S
Class Test 2, 2025

Answer all questions.
Marks per question are shown in brackets.

Time: 45 minutes (+5 minutes reading time) **Marks: 35**

Section 1: Theory of Algorithms [20]

Question 1. **[4]**

Select all options that apply in the following questions.

a) if $T(n) = n^2 + (\log n)^2$, then T(n) is in

i. $O(n^3)$
ii. $\Theta(n^2)$
iii. both i. and ii.
iv. none of the above

ANS: iii (1 mark) or i (0.5 mark and ii 0.5 marks)

b) Is the following statement True or False?

"We can use the Master Theorem to solve a recurrence of the form $T(n) = aT(n-b) + f(n)$, where f(n) is in $\Theta(n^d)$"

ANS: False (1 mark)

c) Given $T(n) = T(n/3) + 3$, the solution to the recurrence is in

i. $\Theta(n^2)$
ii. Θ (log n)
iii. Θ (n)
iv. Θ (1)

ANS: ii (1 mark) simple application of Master Theorem at end of Qn paper)

d) Solving the Travelling Salesman (TSP) problem, between *n* cities, by brute force enumeration (exhaustive search) is in

i. Θ (2^n)
ii. Θ (n^n)
iii. Θ (n!)
iv. none of the above

ANS: iii (1 mark) (reason: ~ n! paths through all cities – see notes)

Question 2. **[3]**

You are given an unsorted array of n elements and asked to find the k^{th} smallest element.

a) Explain briefly how you could do this *without* first sorting the array. [2]
b) Why will this approach not always be better than presorting? [1]

ANS:

a) recursively apply a partitioning scheme like Hoare or Lomuto with a suitable pivot until you reach a state where we have k-1 elements to left of pivot and remainder on right. The pivot is then the kth smallest in array

(**1 mark** for recursive partitioning about pivot),

(1 mark for final partition with pivot in the kth position – note it is also fine - for 0-indexed arrays - to say the pivot must be in (k-1)th (array index) position)

The details of the recursive algorithm – quickselect – are not needed.

Also, just mentioning quickselect is only worth **0.5 mark**.

b) for some arrays contents and pivot strategies this can degenerate to $\Theta/O(n^2)$ comparisons – so sort O(n log n) would be better for these array types. (**1 mark** for noting it can degenerate to $\Theta/O(n^2)$)

Question 3. **[2]**

Why does *interpolation search* not always reduce the search domain by a constant factor like binary search?

ANS: Interpolation search estimates the key index position by linearly interpolating the *key values* of current search array end points - this can lead to a split where the new left and right sub-arrays are of different sizes **(1 mark – emphasize the interpolation causes the uneven split)**

This is different from binary search which always estimates the key position as the middle of the previous search interval. **(1 mark)**

Question 4. **[4]**

You are given an array of elements $[s_1, ..., s_n]$ and asked to write an algorithm to sum all the elements. Your classmate claims you can improve on brute force summation (n-1 adds) by using a simple *divide and conquer* algorithm to do the summation in "log n operations"! Show that this is not true by defining and solving the recurrence relation for the number of *additions* A(n) required to compute the full sum using the obvious divide-and-conquer approach for n elements. After you have defined the full recurrence relation, you can assume that $n = 2^k$ to solve it – you may use the Master Theorem. NOTE: this is a *serial* solution, so no parallel sums.

ANS: Split array into two halves of size floor(n/2), ceiling(n/2) **(1 mark)**
(note – just saying n/2 is fine for **0.5 mark**)

Recurrence: A(n) = A(floor(n/2)) + A(ceil(n/2)) + 1
(**1 mark** for full recursive formulae – **or 0.5 mark** if floor(.)/ceil(.) not used))
A(1) = 0 **(0.5 marks)**
if $n = 2^k$:
We can use back-substitution (which is long – so not advised) or just use Master Theorem
A(n) = 2 A(n/2) + 1 **(0. 5 mark)**
a = 2, b = 2, d = 0
- ⇨ 2 > 2^0 =1
- ⇨ Θ(n^log_b a) = Θ (n^1) = Θ (n)) **(1 mark)**
- ⇨ (which is not Θ /O(log n)….)

Question 5. **[4]**

Use Horner's algorithm for exponentiation to compute the number of multiplies needed to compute a^{13}. **Show all your work**.

ANS:

13 = 8+4+1 = 1101 in binary **(1 mark)**

p = a

bit	1	1	0	1	DONE!
	P=a	P ← (P*P)*a	P ←P*P	P ←P * P*a	
Num mults		2	1	2	

1 mark for showing #multiplications in **each** highlighted columns above. **= 3 marks**

(Just stating 5 multiplications get 1 mark only.)

Question 6. **[3]**

You are given the following solution table for the knapsack problem of capacity W = 6 and n=5 items.

		capacity j						
	i	0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
$w_1 = 3, v_1 = 25$	1	0	0	0	25	25	25	25
$w_2 = 2, v_2 = 20$	2	0	0	20	25	25	45	45
$w_3 = 1, v_3 = 15$	3	0	15	20	35	40	45	60
$w_4 = 4, v_4 = 40$	4	0	15	20	35	40	55	60
$w_5 = 5, v_5 = 50$	5	0	15	20	35	40	55	65

Use this information determine the most valuable items that can fit into a knapsack of capacity W'= **5**. Show your reasoning – based on table values - used to determine the solution set – guess will not be accepted.
ANS:

NOTE: The table contains *all* sub-problems for W < 6 and up to 4 items – so we just backtrack from the appropriate position.

Start at i = 5, j = 5 .
T(5,5) = T(4,5) -> so item 5 not in knapsack **0.5 mark**
Continue from T(4,5) **0.5 mark**
T(4,5) > T(3,5) -> so **item 4** is in knapsack. **0.5 mark**
Now, move w_4 (=4) steps left from T(3,5) to get to next start trace position
Continue from T(3,1) **0.5 mark**
T(3,1) > T(2,1) **-> item 3** is in knapsack! **0.5 mark**
Now, move w_3 (=1) left from T(2,1) = 0 -> terminate. **0.5 mark**

So knapsack contains {item 3, item4}

(if nothing else is shown, but its observed that we startsat T(5,5) that can be awarded 0.5 marks)

[NB: MUST SHOW WORKING – by inspection the answer is clear, so that is not enough, the backtracing process must be followed or clear from the presented answer.]

Section 2: Theory of Computation [15]

Question 7. **[7]**

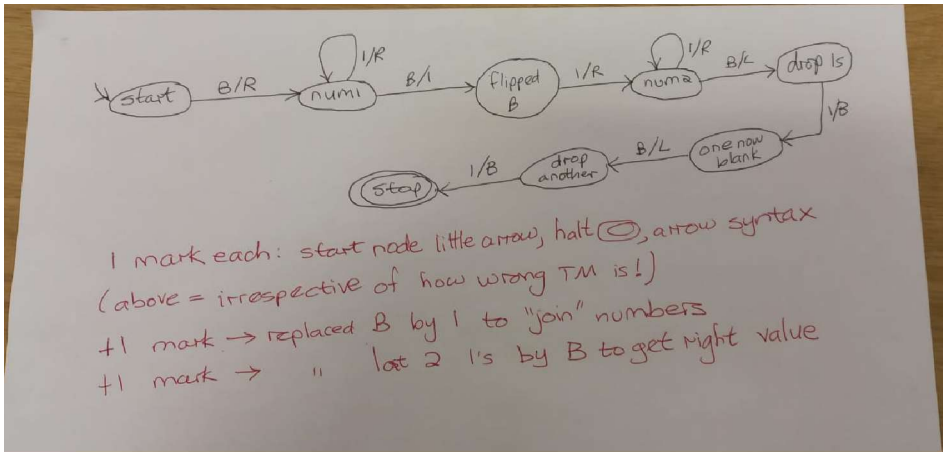
a) Using B to represent a blank cell, **draw a transition diagram for a Turing Machine** with alphabet {B,1} that does natural number addition for unary numbers, and stops with the tape head at the cell immediately right of the total. A number N is represented by a string of N+1 consecutive 1s. The tape head will start on a blank cell, followed by the two numbers, with a single blank cell between them. Example input for 3 + 1:

```
B1111B11BB- - -
↑
```

which must generate output:

```
B11111BBBB- - -
     ↑
```

[5]



b) **How many rows** would your Turing Machine above have in its State Table? [1]

As many rows as there are arrows in THEIR diagram (whether diagram is right or wrong, if answer here is number of arrows it has, then they know it's a row per arrow)

c) Give an example of **any one row** in your Turing Machine's State Table. [1]

Again, even if their TM is wrong, mark this based on THEIR machine. Answer must contain 4 things for any one arrow in their diagram; mark right no matter what order the 4 things are given in. Don't miss the arrow they are using!! No half marks: 0 or 1

Current state: state where arrow starts,
Current/tape symbol: symbol before the /,
Next state: state where arrow ends,
Action: symbol after the /

Question 8. **[3]**

Give a regular expression (**regex**) that accepts the language AskLess:

AskLess = { w | w comprises 2 binary numbers separated by either "<" or "<=" }

Example strings in this language: 10101<=11011 or 11011<=10101 etc. – i.e. a language for stating that kind of Boolean expression.

(1|0)+<=?(1|0)+ or (1|0)+[<|<=](1|0)+ or (1|0)+<[=| ε](1|0)+

i.e. always (1|0)+ on each end, and the comparator in between can be represented as <=?
[< | <=]
< [= | ε]
they could also (optionally) put explicit concatenate dots between things
One mark or zero, no half marks

Question 9. **[1]**

We know there are languages no Turing Machine can recognize because (A) the set of all languages is *uncountably* infinite; and (B) the set of all Turing Machines is *countably* infinite. **In one sentence**, **explain** briefly how we know (B) is true.

Because everyTM can be represented by a binary number [½]
And binary numbers are countable [½]

Question 10. **[4]**

Which of the following statements are **false**? **Give the letter(s) of the false** statement(s), with a brief reason for your answer. **Negative marking employed**: right minus wrong! Minimum mark zero.

a) If the input of a problem is of size N, then the lower bound of that problem must be at least Ω(N).
b) Using the adversary argument in a "Guess the 2-digit number" game, one can keep changing the correct answer after each guess. For example, start with 48, then make it 62, then 55, etc.
c) Any language that is decidable is also acceptable.
d) If problem A has a lower bound of N log N, and we can show that problem B is reducible to problem A in linear time, then problem B also has a lower bound of N log N.

ANSWER:

(a) is false, e.g. binary search
(b) is false, as 2nd value must be <48, because fewer 2-digit numbers below 48 than above it
(d) is false, we'd need to show A is reducible to B, i.e. the other way round is correct

MARKING:

(a) is false [1] (no marks for reason)
(b) is false because 62 is wrong [1] **or (b) is false because changed values are wrong [1]**
or (b) is false because changed values too big [1]
or (b) is false because changed values not < 48 [1]
i.e. one mark for knowing why it is false, else no marks
(d) is false [1] (no marks for reason)
[1] for NOT listing (c) (know it's true)
or MINUS ½ FROM ABOVE TOTAL if they say (c) is false
Minimum zero

Appendix A – Master Theorem

If we have a recurrence of this form: $T(n) = aT(n/b) + f(n)$, $f(n) \in \Theta(n^d)$ and $T(1) = c$ then

$T(n) \in \Theta(n^d)$ if $a < b^d$

$T(n) \in \Theta(n^d \log n)$ if $a = b^d$

$T(n) \in \Theta(n^{\log_b a})$ if $a > b^d$

Section 2: Theory of Algorithms [9]

Question 4. **[2]**

You are told that the time complexity of a particular algorithm is

$T(n) = 5\ T(n/5) + n^2$

for *n* inputs.

Derive the complexity class of this algorithm.

ANS: Apply Master Theorem - a = 5, b = 5, d = 2 (0.5 marks)

- ⇨ 5 < 5^2, case 1 (0.5 mark)
- ⇨ T(n) in $\Theta(n^2)$ (1 mark)

Question 5. **[2]**

Use Horner's algorithm for exponentiation to find the *number of multiplies* needed to compute a^7. Show your working.

ANS:

7 = 111 in binary **(0.5 mark)**

P = a, then

P = (P*P)*a = (a*a)*a = a^3 **(0.5 mark)**

P = (P*P)*a = (a^3*a^3)*a = a^7 **(0.5 mark)**

So we have 2+2 = 4 multiplies. **(0.5 mark)**

[NOTE: can also generate the Horner table/tabulation explicitly, but the main thing is clear evidence for the number of multiplies per step, 2 mult + 2 mult, over 2 steps]

Question 6. **[2]**

Generate the Horspool Table for the pattern **RATTLE** assuming A-Z as the underlying alphabet.

ANS

R	A	T	L	E and others
5	4	2	1	6

2 marks of correct table, -0.5 mark per error (down to 0) otherwise

Question 7. **[3]**

You are given the following solution table for the knapsack problem of capacity W = 3 and n=4 items, with the indicated weights (w_i) and values (v_i) for the i^{th} item:

	Capacity (j)				
		0	**1**	**2**	**3**
Items (n)	**0**	0	0	0	0
w_1=2, v_1=12	**1**	0	0	12	12
w_2=1, v_2=10	**2**	0	10	12	22
w_3=3, v_3=20	**3**	0	10	12	22
w_4=2, v_4=15	**4**	0	10	15	**25**

Use this information determine the *most valuable items* that can fit into the knapsack of capacity 3. Show your reasoning – based on table values - used to determine the solution set – guess will not be accepted.

ANS:

Backtrace from n= 4, j= 3 (last entry)

T(n=4,j=3) **!=** T(n=3, j=3) => item 4 is in knapsack **(0.5 mark)**
Move to T(n=3,j=3), then move left by w4 = 2 units **(0.5 marks)**
Restart trace at T(n=3, j=1)
T(n=3, j=1) **==** T(n=2, j=1) => item 3 NOT included, **(0.5 mark)**
Restart trace at T(n=2,j=1)
T(n=2, j=1) != T(n=1, j=1) => item 2 is in knapsack ***(0.5 mark)***
Move to T(n=1, j=1), then move left w2=1 unit to T(n=1,j=0) **(0.5 mark)**
STOP
So final knapsack contains {item2, item4} **(0.5 mark)**

[NOTE **if solution is wrong**: if some understanding of backtracing is shown i.e. starting at last cell, and testing previous row entry, award **0.5 mark**; if the 'move left' behavior is also shown then award another **0.5 mark**, even if value used is not correct)

Section 3: Theory of Computation [9]

Question 8. **[9]**

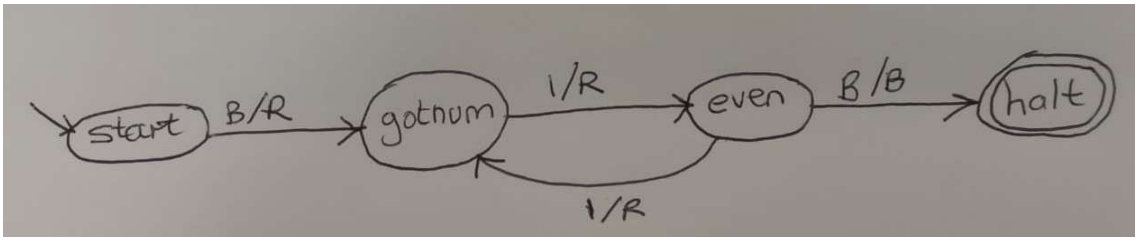
a) Using B to represent a blank cell, draw a transition diagram for a Turing Machine with alphabet {B,1} that ***accepts even unary numbers***. A unary number N is represented by a

string of N+1 consecutive 1s. The tape head will start on a blank cell, followed immediately by the number. The machine must stop on the first blank cell after the unary number if it is even, and must crash if it is not even. Example input for unary 2 (tape head shown below the tape):

B111BB- - -
↑

which, because 2 is even, must halt with the tape head at the first blank after its input:

B111BB- - -
↑ [4]



b) Give an example of any one row in your Turing Machine's State Table. [1]

Any one row representing any one arc/edge in your TM. For mine:

CURRENT STATE	INPUT	ACTION	NEXT STATE
start	**B**	**R**	**gotnum**
gotnum	**1**	**R**	**even**
even	**1**	**R**	**gotnum**
even	**B**	**B**	**halt**

Odd numbers will not be accepted, as there is no arc leaving "gotnum" for when a B is encountered in that state = crash = not accepted.

c) Explain what is meant by the *lower bound* and the *upper bound* of a problem. [2]

Lower bound = proven worst case complexity of the problem; best possible efficiency any algorithm solving that problem can have

Upper bound = worst case complexity of best known algorithm that solves the problem; minimum worst case complexity of all known algorithms that solve the problem.

d) We showed that the Element Uniqueness problem (call it U) is reducible to the Minimum Spanning Tree problem (call it T) in polynomial time. Which of the following statements follow from this? Give the statement number, and a brief reason for your answer.

1. The lower bound of U is no worse than the lower bound of T (U is no harder than T)
2. The lower bound of T is no worse than the lower bound of U (T is no harder than U)

[2]

1 follows from this, because this means we have a way of solving U by doing a little work (polynomial time reduction) and then running an algorithm that solves T.

Could set up contradiction (but wasn't expected): Suppose U has a lower bound of UB. By definition, this means U cannot be solved by any algorithm with a complexity less than UB. SupposeT has a lower bound of TB, such that UB is worse than TB. Since the polynomial time reduction of U to T, followed by the solving of T, is a solution to U, this combined process is of complexity TB. But UB is worse than TB, so a solution of complexity TB is not possible if the lower bound of U has been proven to be UB. Contradiction.

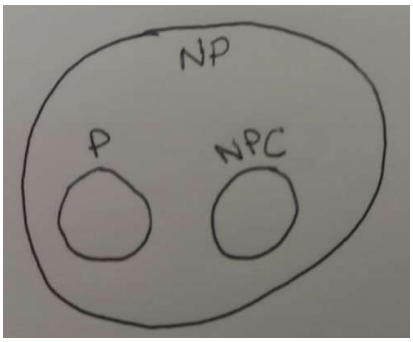
Simplify to yourself by reading "reducible in polynomial time" as "is no harder than"

The question below was uploaded by Cael, but was in an earlier version of the Concession Test, not the final version. So here is its question and expected answer.

d) Suppose in the future someone proves, for some NP problem Q, that Q is **not** solvable in polynomial time. Draw a Venn diagram to show what that would mean in terms of the set/subset relationships between the classes P, NP and NPC (NP-Complete). [2]

Only the Venn diagram below was required, but here is an explanation of why this would be the situation if Q were proven superpolynomial:

P and NPC are contained in NP because they have certificates that can be checked in polynomial time. P is only a *subset* of NP because Q is in NP and NOT in P (i.e. NOT solvable in polynomial time). NPC is *separate* from P for the following reason: Q can be polynomially reduced to any NPC problem (by definition of NPC). This means that Q is no harder than NPC problems (by definition of polynomial reducibility). So if Q has been proven as not solvable in polynomial time, then NPC problems are also not solvable in polynomial time.



Appendix A – Master Theorem

If we have a recurrence of this form: $T(n) = aT(n/b) + f(n)$, $f(n) \in \Theta(n^d)$ and $T(1) = c$ then

$T(n) \in \Theta(n^d)$ if $a < b^d$

$T(n) \in \Theta(n^d \log n)$ if $a = b^d$

$T(n) \in \Theta(n^{\log_b a})$ if $a > b^d$