Computer Science, an Overview

Final Exam (A)

January 4, 2018

Department: Class: Name: Student No.:

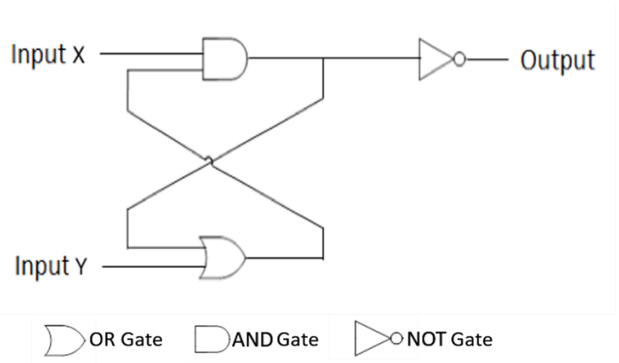
Read First Please

- Please fill in your **Department**, **Class**, **Name** and **Student No.**!
- The exam is divided into 4 parts, for a total of 100 points.
- You should work alone and **MUST NOT** copy others'.
- **The format of 8-bit floating-point notation**, **V8 machine language** and **the algorithm for a control system using heuristics** can be found on the reference page. You may tear the last page (empty) out and write anything on it.

1st PART: Single Choice (3' for each question, 30' in total)

Write your choice for each question on the underline

1. Here is the logic circuit of a flip-flop. Which of following will set Output to 0 no matter with the original Output? _____



A. Set Input X to 0 and Input Y to 0
B. Set Input X to 1 and Input Y to 1
C. Set Input X to 1 and Input Y to 0
D. Set Input X to 0 and Input Y to 1

2. What are the results of $64 + 64$ and $-64 - 64$ when they are calculated using the eight-bit two's complement representation introduced in class? _____

A. $128,\ 128$ B. $128,\ -128$ C. $-128,\ 128$ D. $-128,\ -128$

3. We have two variables in 8-bit floating-point notation: a = 01011010 and b = 10101010. What is the result of (a + b) in the same notation? _____

A. 11011000 B. 01011000
C. 01001000 D. 11001000

4. Which of the following component in Von Neumann Architecture contains Program counter and Instruction register? _____

A. Arithmetic/logic unit B. Control unit
C. Memory D. Output devices

5. Which of the following component is in charge of starting an operating system? _____

A. Memory manager B. Boot loader
C. File system D. Scheduler and dispatcher

6. Suppose each time slice in a **time-sharing** system is 50 ms and each context switch requires 3 μs(1ms = 1000μs). If there are a lot of processes each of which executes an I/O request 1 μs after its time slice begins (assume OS terminates a process performing I/O operation and allows another process to run while the first is waiting), what fraction of the machine's time is spent actually running these processes? _____

A. 99.998% B. 50%
C. 25% D. 5%

7. The following algorithm merges two sequential files. Assume that one input file contains records with key values equal to B and E while the other contains records with key values equal to A, C, D and F. How many records will be copied **before** executing the last line of the algorithm? _____

```
// A procedure for merging two sequential files
procedure MergeFiles (InputFile1, InputFile2, OutputFile)
if (both input files at EOF) then (Stop, with OutputFile empty)
if (InputFile1 not at EOF) then (Declare its first record to be its current record)
if (InputFile2 not at EOF) then (Declare its first record to be its current record)
while (neither input file at EOF) do
    (Copy the current record with the "smaller" key field value to OutputFile;
    if (that current record is the last record in its corresponding input file)
        then (Declare that input file to be at EOF)
        else (Declare the next record in that input file to be the file's current record)
    )
Starting with the current record in the input file that is not at EOF, copy the remaining records to OutputFile.
```

A. 3 B. 4
C. 5 D. 6

8. The figure and table below give a Turing Machine and its expected initial configuration. Inputs to the Turing Machine are written as X_i that may be 0 or 1. Which of the four following statements is correct? _____

Initial configuration:

*	*	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	*	*

↑
Initial State = START

Current State	Current Cell content	Value to write	Direction to move	New state to enter
START	*	*	LEFT	COMPUTE1
COMPUTE1	1	0	LEFT	COMPUTE1
COMPUTE1	0	1	NO MOVE	HALT
COMPUTE1	*	*	NO MOVE	HALT

A. The Turing Machine increments the input value by 1 when the input is not all 1s.
B. The Turing Machine sets the leftmost bit to 1 while keeping the other bits unchanged
C. The Turing Machine decrements the input value by 1 when the input is not all 0s.
D. The Turing Machine applies the NOT operation to all bits of the input.

9. Will a V8 machine executing the following program, starting from 0x00, halt? _____

Address	00	01	02	03	04	05	06	07	08	09	0A	0B
Contents	20	00	21	02	52	12	B2	0A	B0	04	C0	00

A. It will always halt
B. It will never halt
C. It will terminate in some cases but not always. Whether it will terminate depends on the initial contents in the general registers
D. It is impossible to predict if the machine will halt even the initial contents in all general registers are known

10. Each of the following statement describes a set of problems. Which of them is empty as far as we currently know? _____

A. Non-polynomial problems that are also NP problems
B. Polynomial problems that are also NP problems
C. Non-polynomial problems that are not NP problems
D. Polynomial problems that are not NP problems

(Hint: Consider the following problems as examples: 1. the time complexity of insertion sort is $\Theta(n^2)$, 2. traveling salesman problem belongs to NP problems, 3. printing all subsets of a given set needs **at least** time of $\Theta(2^n)$.)

2nd PART: Eight Puzzle (19' for this part)

Here is an unsolved eight-puzzle:

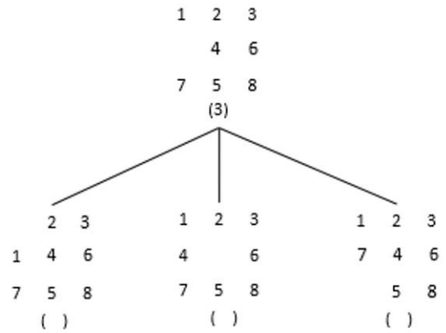
1	2	3
	4	6
7	5	8

1. (3') As we want to implement the eight-puzzle solving algorithm, we need first store an eight-puzzle state. If each state is stored as a two-dimension array (the empty space is represented by 0), what is the index (start from 0) of the number in *i-th* row and *j-th* column (both started from 1) if the state is stored in row major order? If each element is an 8-bit integer and all values are stored in the memory, how many bytes are needed to save one state?

2. (2') The following algorithm builds a search tree in **a depth-first manner**. The formal parameter *InitialNode* is the eight-puzzle we want to solve. What kind of data structure should be used in this algorithm?
   A. Stack B. Queue

```
1  procedure DepthFirstSearch (InitialNode)
2      ds ← an empty _______
3      insert InitialNode into ds
4      do (
5          s ← The next element to remove from ds
6          if (s is the goal node) then
7              (report a success and exit the current loop)
8          else
9              (insert the next not-yet-visited node that can be reached by a single production into ds)
10         end if
11         remove s from ds
12     ) while (true)
```

3. (9') Complete the **heuristic-based** search tree on the next page. The heuristic is **the number of out of place numbers**, i.e. the numbers that are placed differently as they are in our goal. For example, in the above eight-puzzle state, the numbers '4', '5' and '8' are not where they should be, while the others are in the correct places. Thus, the heuristic of the state is 3. Note that heuristic values of each state should also be included in your search tree. The algorithm can be found on the reference page.



4. (5’) Here is an algorithm for printing the solution to the eight puzzle when a search tree is given. The search tree can be accessed via global variables.

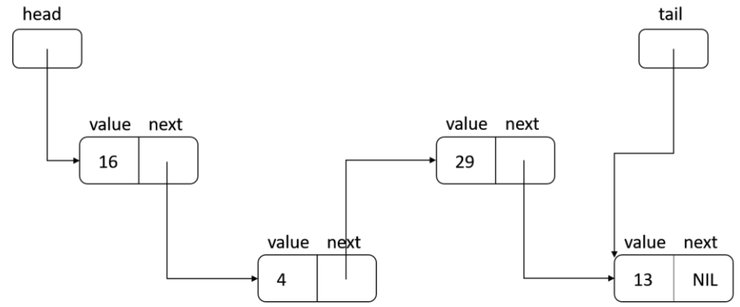
```
1   procedure PrintSolutionAux (n)
2       Print the string “(enter Print)”
3       if (n is not the root of the search tree) then (
4           p ← The parent node of n
5           op ← production leading p to n
6           apply procedure PrintSolutionAux to p
7           Print the production op
8       )
9       Print the string “(exit Print)”
10
11  procedure PrintSolution()
12      g ← the goal node in the search tree
13      PrintSolutionAux(g)
```

Productions are printed as ‘Move [the number] [direction]’, such as ‘Move 2 up’ or ‘Move 8 right’. What will be printed after executing the procedure PrintSolution on the search tree you have constructed in problem 3?

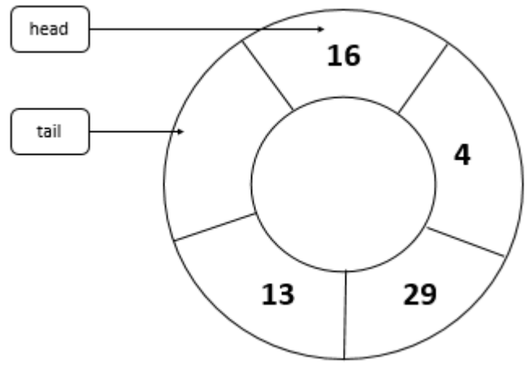
3rd PART: Queue (18’ for this part)

Queues are a kind of useful data structure with the property that all **additions** are performed at the end and all **removals** at the front. There are several ways to implement a queue. Here are two examples, using a **linked list** and a **one-dimension array** respectively.

A queue implemented using a linked list (when the queue is empty, both the head and tail pointers contain the value NIL)



A queue implemented using a one-dimension array (A circular queue introduced in class)



The structures of **LinkedList** and **Node** in a linked list are given as follows.

```
struct LinkedList {
    Node *head;
    Node *tail;
}
```

```
struct Node {
    int value;
    Node *next;
}
```

We will use the following notations in pseudo code.

- Node* means the pointer to a Node.
- *node*.next is used as the field **next** in the **Node** structure that the variable *node* pointed to.
- The integers and pointers are 4-byte long each.

1. (4’) Each following statement compares the two different implementations of queues. Determine whether each of them is correct or not. Write a tick (✓) in the parenthesis if it is correct. Otherwise write a cross (X).

Note: The *capacity* of a circular queue is the number of elements in the queue when it is full.

( ) The number of elements in a circular queue introduced in class is limited to the size of the array as long as we do not ask for a larger array after the circular queue is created. A queue implemented using a linked list, however, can grow as long as there is unused memory cells available.

( ) When there is only one element in the queue, queues implemented using linked lists use more memory cells than queues implemented using arrays with a capacity of 100.

( ) When there are 100 elements in the queue, queues implemented using linked lists use more memory cells than queues implemented using arrays with a capacity of 100.

( ) To determine if a queue is empty or not, we need more information, in addition to where the head and tail pointers point to, no matter the queue is implemented using linked lists or arrays.

2. (4') What is the time complexity of algorithms that add / remove an element into / from a queue? Complete the following table by filling the blanks in the rightmost column using the options given below. The n in the options is the number of elements in the queue.

A. $\Theta(1)$ B. $\Theta(lgn)$ C. $\Theta(n)$ D. $\Theta(n^2)$

Type of queue	Operation	Class in which the algorithm falls
circular queues implemented using arrays	Addition	
	Removing	
queues implemented using linked lists	Addition	
	Removing	

3. (8') In a **non-empty** queue implemented by a linked list, the following algorithms **add a new node at the tail** and **remove the node at the head and return the pointer to the removed node** respectively. The formal parameters of EnqueueNonempty is a pointer to a LinkedList (*queue*) and a pointer to a Node (*node*). Note that pointers in *node* is not initialized and thus **can be any value**. The formal parameter of DequeueNonempty is a pointer to a LinkedList (queue). Please fill in the blanks with appropriate code to show how these procedures can be implemented by updating pointers in the queue and its nodes.

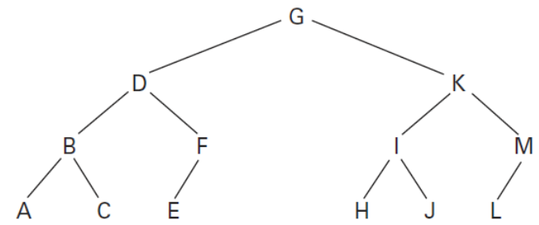
procedure EnqueueNonempty (*queue*, *node*)
    head ← *queue*.head (*head* is a local variable in this procedure)
    tail ← *queue*.tail (*tail* is a local variable in this procedure)

procedure DequeueNonempty (*queue*)
    head ← *queue*.head (*head* is a local variable in this procedure)
    tail ← *queue*.tail (*tail* is a local variable in this procedure)

4. (2') Can the procedure *EnqueueNonempty* you have implemented above add a new node to an empty queue implemented using linked list? If not, what are the missing operations?

4th PART: Binary Search Tree (33' for this part)

A **binary search tree** is a binary tree with a value stored in each node such that, for any node in the search tree, the value stored in the node is greater than all values stored in the left sub-tree, and smaller than all values stored in the right sub-tree. The following figure shows how letters A through M can be stored in a binary search tree, assuming 'A' < 'B' < 'C' < … < 'M'.



As an example, consider the node storing the letter 'D'. The left sub-tree of the node stores the letters 'A', 'B' and 'C' which are all smaller than 'D'. The right sub-tree of the node stores 'E' and 'F' which are both greater than 'D'.

We can implement a binary search tree using a **one-dimension array**, as has been introduced in class. In this part, we will store positive integers in the nodes and use -1 to indicate that no child exists. Given a **non-empty binary search tree** implemented in the above way, we can efficiently search for an entry using the following algorithm.

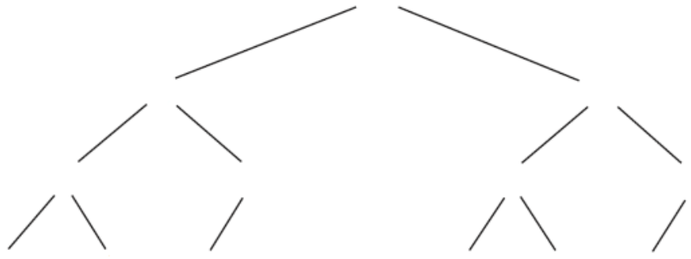
```
1   procedure Search (Tree, TargetValue)
2       i ← 0 (which is the index of the root node of Tree, as Tree is implemented as an array)
3       v ← 0
4       while (true) (
5           v ← The value stored at index i in Tree
6           if (v equals to TargetValue or v equals to -1) then ( break the loop )
7           if (TargetValue < v) then
8               (i ← the index of the left child of i, which is _____)
9           else
10              (i ← the index of the right child of i, which is _____)
11      )
12      if (v does not equal to -1) then (report the search a success)
13      else (report the search a failure)
```

We now show you how the above algorithm can be implemented in V8. The following table shows the contents in the memory (all in hexadecimal notation). The content in a table cell is the bit pattern stored in the memory cell whose address is the sum of row and column headers of that table cell. For example, the lower right corner of the table says the bit pattern FF is stored in the memory cell whose address is 0x70 + 0xF = 0x7F. Memory cells whose addresses range from 0x20 to 0x59 store the program, which **starts at 0x20**. Memory cells whose addresses range from 0x60 to 0x7F store a binary search tree implemented as a one-dimensional array.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20	21	01	22	02	23	80	24	FF	25	60	26	42	27	00	58	57
30	38	33	18	20	40	40	B8	54	40	60	B8	56	9A	84	5A	A1
40	5A	A6	8A	A3	20	00	BA	4E	A7	07	57	71	B0	2E	A7	07
50	57	72	B0	2E	27	FF	37	5F	C0	00	00	00	00	00	00	00
60	32	25	3D	0C	2D	37	46	05	1A	2B	FF	33	3A	42	FF	FF
70	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

1) (4') The algorithm in pseudo code does not show how the left or right child of a node should be calculated. Complete the algorithm by filling the underlines on line 8 and 10 with the following options.
   A. $2i - 1$
   B. $2i$
   C. $2i + 1$
   D. $2i + 2$

2) (8') The following figure shows the shape of the binary search tree stored in memory cells whose addresses start from 0x60. Complete the tree by filling the integers stored in each node. You can keep the integers **in hexadecimal notation**.



3) The V8 program shown above can be divided into three parts, listed as follows.
   - Code in memory cells whose addresses range from 0x20 to 0x2B initializes some constants that are never changed in the program. **Register 6 stores *TargetValue* in the algorithm in pseudo code.**
   - Code in memory cells whose addresses range from 0x2C to 0x53 implements the while loop part (from line 4 to line 11) of the algorithm in pseudo code. The instructions in memory cells whose addresses range from 0x2E to 0x33 load "the value stored at index *i* in Tree".
   - Code in memory cells whose addresses range from 0x54 to 0x59 stores the result to memory cell whose address is 0x5F and halts.

a) (4') In the V8 program, register _____has the address of root node of the *Tree*, register _____ acts as the index *i*, register _____ has the value stored in the node and register _____ has the result of "*TargetValue* - v".

b) (4') The instructions in memory cells whose addresses range from 0x34 to 0x3B determine the conditions on line 6 in the algorithm. If "*v* equals to *TargetValue*", the contents in the program counter will be changed to _____. If "*v* equals to -1", the contents in the program counter will be changed to _____.

c) (4') The instructions in memory cells whose addresses range from 0x3C to 0x47 compare the value stored in the node (i.e. *v*) with *TargetValue*. If *v* is greater than *TargetValue*, after executing the instruction in the memory cell whose address is 0x42, the contents in register A will be _____. If *v* is smaller than *TargetValue*, the contents in register A will be _____.

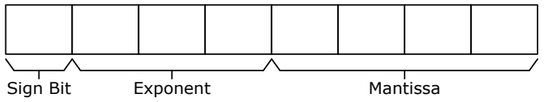
4) (7') If a V8 machine starts with 0x20 in the program counter and the above contents in memory, how many times will the instruction in memory cells whose addresses are 0x32 and 0x33 be executed when the machine halts? What are the contents in the memory cell whose address is 0x33 each time that instruction is executed? Answer the above questions by filling the following table. You can add more lines if you need.

The # time instruction at 0x32 is executed	Contents in memory cell at 0x33
1st	
2nd	

5) (2') What is the content in the memory cell whose address is 0x5F after the machine halts?

Reference page:

1. 8-bit Floating Point Notation



An Excess Four Conversion Table	
Bit Pattern	Value Represented
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

2. V8 Machine Language

Op-code	Operand	Description
1	**RXY**	**LOAD** the register R with the bit pattern found in the memory cell whose address is XY.
2	**RXY**	**LOAD** the register R with the bit pattern XY.
3	**RXY**	**STORE** the bit pattern found in register R in the memory cell whose address is XY.
4	**0RS**	**MOVE** the bit pattern found in register R to register S.
5	**RST**	**ADD** the bit pattern in registers S and T as though they were two's complement representations and leave the result in register R.
6	**RST**	**ADD** the bit pattern in registers S and T as though they represented values in floating-point notation and leave the floating-point result in register R. The rightmost bits in the result will be lost if the result is too long to be filled into the mantissa field. E.g. $0.11_2 + 0.00111_2$ will be 0.1111_2.
7	**RST**	**OR** the bit patterns in registers S and T and place the result in register R.
8	**RST**	**AND** the bit patterns in registers S and T and place the result in register R.
9	**RST**	**XOR** the bit patterns in registers S and T and place the result in register R.
A	**R0X**	**ROTATE** the bit pattern in register R one bit to the right X times.
B	**RXY**	**JUMP** to the instruction located in memory address XY if the bit pattern in register R is equal to the bit pattern in register 0.
C	**000**	**HALT** execution.

3. An algorithm for a control system using heuristics

1 Establish the start node of the state graph as the root of the search tree and record its heuristic value
2 **while** (the goal node has not been reached) **do** (
3 Select the leftmost leaf node with the smallest heuristic value of all leaf nodes
4 To this selected node attach as children those nodes that can be reached by a single production
5 Record the heuristic of each of these new nodes next to the node in the search tree
6)

Scratch page:

You are free to tear this page out and write anything on it. Please remember to hand this page in at the end of the exam.