# CITY UNIVERSITY OF HONG KONG

---

Course Code & Title : EE2331 Data Structures and Algorithms

Session : Semester B, 2010/11

Time Allowed : Two hours

---

This paper has 5 pages (including this cover page).

Answer ALL questions in this paper.

---

Material, aids & instruments permitted to be used during examination:

1. Approved calculators

**Question 1:** The initial contents of an integer array `H[]` representing a max-heap are shown below:

```
H[] = {83, 52, 68, 44, 37, 49, 21, 15, 39, 28}
```

(a) Show the contents of `H[]` after the largest number in the max-heap has been removed.

**(3 marks)**

(b) Referring to the above given initial contents of `H[]`, show the contents of `H[]` after the value 71 has been added to the max-heap.

**(3 marks)**

**Question 2:** Reconstruct the binary tree from the given inorder and postorder traversal sequences. Each node label is a single letter in 'A' to 'Z'. **(6 marks)**

| | |
|---|---|
| Inorder sequence: | E X B K D L A Y T C P Q U |
| Postorder sequence: | X E K L D B Y T Q P U C A |

**Question 3:** A binary search tree can be used to sort an array of numbers. The sorting method is called the *tree-sort* algorithm. The sorting procedure is as follow:

1) Insert the numbers (one at a time) into an initially empty binary search tree.
2) Produce the sorted sequence in ascending order by performing an inorder traversal of the resultant binary search tree.

(a) Use the above strategy to sort the given list of numbers `{44, 21, 37, 52, 49, 83, 68, 15, 39, 28}`. Show the structure of the binary search tree after each insertion.

**(6 marks)**

(b) What is the best case time complexity of the tree-sort algorithm? Under what condition will you obtain the best case time complexity?

**(4 marks)**

(c) What is the worst case time complexity of the tree-sort algorithm? Under what condition will you obtain the worst case time complexity?

**(4 marks)**

**Question 4:** Consider a hash table of size 13, and the keys are integers. The hash function is $h(key) = key \% 13$. Collisions are resolved by double hashing. The second hash function is $h2(key) = 1 + key \% 5$. The initial contents of the hash table are shown below.

| | |
|---|---|
| 0 | 29 |
| 1 | |
| 2 | |
| 3 | 16 |
| 4 | |
| 5 | 5 |
| 6 | |
| 7 | |
| 8 | 8 |
| 9 | 74 |
| 10 | 23 |
| 11 | 37 |
| 12 | |

(a) Show the contents of the hash table after the ***key* = 57** has been inserted.

**(3 marks)**

(b) What is the average number of probes for successful search with respect to the hash table obtained in part (a) after the insertion of the *key* value 57?

**(7 marks)**

**Question 5:**

(a) Consider the recursive function given below, show the outputs generated by calling **what(7)**.

```
void what(int y)
{
      printf("%d, ", y);
      if (y > 0)
      {
            what(y/2);
            what(y - 3);
      }
}
```

**(8 marks)**

(b) Redesign the above recursive function **without** using recursion. Your program will make use of a stack of integer. You may assume the stack is already defined, and your program can call the standard stack operators as given below:

```
typedef struct _stack Stack; //Stack of integer
void stackInit(Stack *s);    //initialize s to empty state
void push(Stack *s, int e);  //push an element e to the top of s
int pop(Stack *s);           //remove and return the top element from s
int stackEmpty(Stack *s);    //return 1 if s is empty; return 0 otherwise
int stackFull(Stack *s);     //return 1 if s is full; return 0 otherwise
```

**(15 marks)**

**Question 6:**

(a) A general tree can be specified by the nested-parenthesis format.

Tree = (D (K (T)) (A (M) (N (F) (B)) (E)) (S (C) (X)))

Draw the structure of the above tree using the conventional linked representation.

**(3 marks)**

(b) A general tree can be physically represented using a binary tree. Draw the structure of the binary tree that corresponds to the general tree of part(a).

**(3 marks)**

(c) The structure of tree node is defined as follows:

```
struct _node {
    char data;
    struct _node *left, *right;
}
typedef struct _node Node;
typedef struct _node * NodePtr;
```

Write a recursive function to count the number of nodes on a given level of a general tree represented by a binary tree. The root of the tree is defined to be on level 0, and the function signature is given below.

```
int countNodeByLevel(NodePtr tree, int level);
```

**(15 marks)**

**Question 7:**

A sequence of integers is stored in a singly **circular** linked list **without** dummy header. The external pointer points to the first number in the sequence. The structure of the list node is defined as follows:

```
struct _node {
    int data;
    struct _node *next;
}
typedef struct _node Node;
typedef struct _node * NodePtr;
```

Design a function to search if a list *L* contains a **non-empty** *pattern*, where the pattern is also a list of integers. Both *L* and *pattern* are represented as circular linked list without header node.

**(20 marks)**

```
NodePtr search(NodePtr L, NodePtr pattern);
/* Precondition: pattern is not empty

   If the pattern is found, the function returns the
   pointer to the first element of the searched pattern
   in L.
   If the pattern is not found, the function returns the
   value NULL. */
```

Example 1:
*L* = {5, 9, 3, 5, 5, 2, 6, 1, 0, 2}
*pattern* = {5, 2, 6, 1}
*pattern* is found in *L*.

Example 2:
*L* = {5, 9, 3, 5, 5, 2, 6, 1, 0, 2}
*pattern* = {5, 2, 6, 2}
*pattern* is **not** found in *L*.

Example 3:
*L* = {5, 9, 3, 5, 5, 2, 6, 1, 0, 2}
*pattern* = {2, 5, 9}
*pattern* is found in *L* (since *L* is a circular list)

**- THE END -**