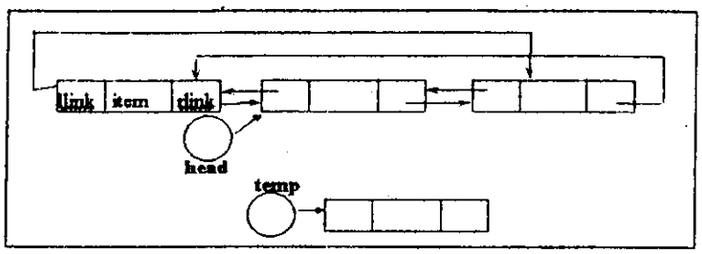
試題是否可以使用計算機： □可使用 ， ☑不可使用 （請命題老師勾選）

1. A queue is designed as a cycled two-way linked list, whose structure definitions and relations are shown as following:

```
typedef struct node *node_pointer;
typedef struct node {
    node_pointer llink;
    element item;
    node_pointer rlink;
}
```



Please write a pseudo code for inserting the "temp" node into the logic place between top and tail. (8%)

2. Please use the structure in question 1 to write a recursive function for describing binary tree inorder traversal (8%)

3. Write two programs in any language (10%), both recursive and iterative, of the following equation. EvenTotal = 2 + 4 + 6 + … + N, and explain the difference between program writing in recursive and iterative. (6%)

4. Assume Data[100][100] is a two-dimension integer array whose start address is A0A0 $_{(16)}$. Please explain the possible addresses of Data[3][5] (6%) (integer : 4 bytes)

5. Please discuss the average case time complexity of sequential search in big O. (8%)

6. Please discuss the importance of Data Structure and Algorithm, and explain their relations. (4%)

(背面仍有題目,請繼續作答)

7 (24%) True or False, and EXPLAIN
Circle T or F for each of the following statements to indicate whether the statement is true or false, respectively. If the statement is correct, briefly state why. If the statement is wrong, explain why or give a counter example. Answers without reasons will get at most 1 point.

(a) (**T** , **F**) To sort n distinct numbers with any sorting algorithm takes $\Omega(n)$ time.
(b) (**T** , **F**) To sort n distinct numbers with any comparison sorting algorithm takes $\Omega(n \lg n)$ time. So, $\Theta(n \lg n)$ is the best complexity for any sorting algorithm.
(c) (**T** , **F**) It is possible for a min-heap of $n \geq 3$ nodes to have height (i.e. the number of edges on a longest simple path from the root down to a leaf) equal to $n - 1$.
(d) (**T** , **F**) A complete graph K_n of even number of nodes (i.e. n is even) is a bipartite graph.
(e) (**T** , **F**) In a complete graph K_n, solving the all pairs shortest paths using Dijkstra's algorithm is less efficient than using Floyd-Warshall's algorithm.
(f) (**T** , **F**) Given an undirected graph G of n nodes and m arcs, one may use the adjacency matrix to store this graph and apply the Floyd-Warshall's algorithm to compute all pairs shortest paths in $\Theta(n^3)$ time. This complexity can NOT be further improved even if we only store the upper half triangle of its adjacency matrix.

8 (6%) Given a red-black tree T of n interal nodes (i.e. not the leaves which are $nil[T]$ elements with black color) Define the height $h(x)$ of a node x to be the number of edges in a longest path to a leaf. Define the black height $bh(x)$ of a node x to be the number of black nodes (excluding x but including the leaf (i.e. $nil[T]$)) on the path from x to a leaf. Also, define the red height $rh(x)$ of a node x to be the number of red nodes (excluding x) on a longest path from x to a leaf. It is obvious that $h(x) = bh(x) + rh(x)$ (check the example in Figure 1) Let the height of a tree be the height of its root node.Explain why the following statements are true, as well as how to use these statements to explain the upper bound of the height for a red-black tree:

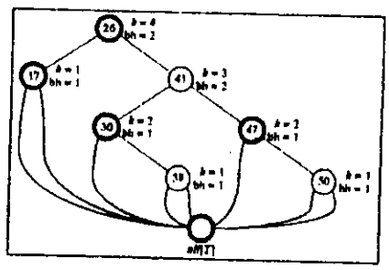


Figure 1: A red-black tree example

(a) [2%] Any node x with height $h(x)$ has black-height $bh(x) \geq h(x)/2$
(b) [2%] The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal nodes (hint: starting from x is a leaf, then use mathematical induction)
(c) [2%] Using (a) and (b) to explain why a red-black tree with n internal nodes has height $\leq 2 \lg(n + 1)$

9 (8%) Suppose you have n objects with index from 1 to n. Suppose you are given a function $RAND(s,t)$, and $RAND(s,t)$ can generate a random integer number uniformly distributed in the range of $[s,t]$ in $O(1)$ time. (i.e. if $k = RAND(1,n)$, k is a random integer in $[1,n]$)

(a) Give an $O(n \lg n)$ method to generate a random permutation for these n objects. (you may explain your method in words or in pseudo code)

(b) Give an $O(n)$ method to do the same thing as (a).

10 (12%) Let $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$, $B(x) = b_0 + b_1x + b_2x^2 + \ldots + b_nx^n$, and $C(x) = A(x) \cdot B(x) = c_0 + c_1x + c_2x^2 + \ldots + c_{2n}x^{2n}$, where n is even, $c_i = \sum_{k=0}^{i} a_k b_{i-k}$ for $i = 0, \ldots, 2n$, and $a_j \equiv 0$, $b_j \equiv 0$ for $j > n$.
To compute $C(x)$, we define $P(x), Q(x), R(x)$, and $S(x)$ such that $A(x) = P(x) + x^{\frac{n}{2}}Q(x)$, $B(x) = R(x) + x^{\frac{n}{2}}S(x)$. For example, if $A(x) = 2 + 5x - 4x^2 + 3x^3 - 7x^4$, $B(x) = -1 - 4x + 3x^2 - 2x^3 + x^4$, we will have $P(x) = 2 + 5x$, $Q(x) = -4 + 3x - 7x^2$, $R(x) = -1 - 4x$, and $S(x) = 3 - 2x + x^2$.
Note that computing each of $P(x) \cdot R(x)$, $P(x) \cdot S(x)$, $R(x) \cdot Q(x)$, and $Q(x) \cdot S(x)$ is similar to $A(x) \cdot B(x)$ except the former terms are products of polynomials with smaller orders (up to $\frac{n}{2}$), whereas the latter term is a product of polynomials with order n. This problem seeks the possibility to improve the complexity of multiplication of polynomials via decomposition techniques.

Let $T(n)$ be the running time of some algorithm computing for $C(x)$. Answer the following questions **with explanation**:

(a) [4%] If we compute $C(x)$ by directly computing $A(x) \cdot B(x)$, what is the tight upper bound for $T(n)$?

(b) [8%] If we compute $C(x)$ by computing $P(x) \cdot R(x) + x^{\frac{n}{2}}(P(x) \cdot S(x) + R(x) \cdot Q(x)) + x^nQ(x) \cdot S(x)$, we can analyze the complexity by $T(n) = aT(n/b) + f(n)$. What are a, b, and $f(n)$? Is this method more efficient than the method in (a)? (you may use Θ-notation for $f(n)$)