本試題有 12 題選擇題（題號 1-12）。第 3 題及第 6 題為單選題，其餘為複選題。單選題答對得 5 分，答錯倒扣 1 分，未答者得零分。複選題每個選項單獨計分，選項答對者得該題五分之一分數，選項答錯者倒扣該題十分之一分數，另若該題整題未作答者，整題得 0 分。選擇題請於答案卡上作答。另有一題非選擇題（題號 13），請於試卷本上作答。

1. (5 points) Consider the following variation of the Selection Problem: Given a sequence $S = \langle a_1, a_2, \ldots, a_n \rangle$ of n distinct integers and two positive integers x and y, where each number in S falls between 1 and y, the goal is to report the smallest x numbers in S in ascending order. This scenario assumes the conventional single-processor, random access machine (RAM) model, with a machine word size of at least $\lceil \log y \rceil$ bits. To tackle this problem, five algorithms have been proposed. Please select the correct description(s) below.

   A. Algorithm A: The algorithm initializes an empty max-heap and sequentially inserts numbers from S. During insertion, if the size of the max-heap exceeds x, it removes the maximum element from the heap. Finally, the remaining numbers in the heap are sorted by heapsort and reported. If $x \leq n/\log n$, Algorithm A runs in $O(n)$ time in the worst case.

   B. Algorithm B: The algorithm initially constructs a min-heap and proceeds to extract the minimum element x times. If $x \leq n/\log n$, Algorithm B runs in $O(n)$ time in the worst case.

   C. Algorithm C: The algorithm begins by performing a base-n radix sort, followed by reporting the smallest x values. If $y \leq n^2$, Algorithm C runs in $O(n)$ time in the worst case.

   D. Algorithm D: The algorithm performs a counting sort by allocating an array C of length y, traversing through S and incrementing $C[a_i]$ for each a_i. It finally reports the smallest x values by scanning through the counter array. If $y \leq n^2$, Algorithm D runs in $O(n)$ time in the worst case.

   E. Algorithm E: The algorithm first applies quickselect, i.e., Randomized-Select, to determine the x-th smallest number in S. Subsequently, it traverses S for extracting all numbers smaller than or equal to the x-th smallest number. Finally, these x numbers are sorted by quicksort and reported. The time complexity of Algorithm E is $O(n + x \log x)$ in expectation.

2. (5 points) A Maximum Binary Tree (abbreviated as MaxBT) is defined as a binary tree that adheres to the max-heap property without the necessity of following the complete binary tree property. In a MaxBT, each node p consists of a value (p.data) and two links (p.left and p.right) pointing to the left and right children, respectively. A link to nil indicates the absence of further children.

   The provided pseudocode, although incomplete, implements a recursive function for merging two MaxBTs into a single MaxBT. This function takes the root nodes of the two MaxBTs (*root1* and *root2*) as input and returns the root node of the merged MaxBT.

```
1: function MergeMaxBinaryTrees(root1, root2)
2:     if root1 = nil then return ____(a)____ end if
3:     if root2 = nil then return ____(b)____ end if
4:     if root1.data > root2.data then
5:         ____(c)____ ← MergeMaxBinaryTrees(____(d)____, ____(e)____)
6:         return ____(f)____
7:     else
8:         ____(g)____ ← MergeMaxBinaryTrees(____(h)____, ____(i)____)
9:         return ____(j)____
10:    end if
11: end function
```

   Assuming all the blanks in the above function have been appropriately filled to ensure its proper functioning, please select the correct description(s) below.

   A. (b) and (f) are the same.

   B. (c), (d), and (e) are all distinct.

見背面

C. In the cases of (c) and (g), each has only one correct answer.

D. Exchanging the filled content of (h) and (i) leads to a correct implementation.

E. The function works correctly even when the two arguments passed into it refer to the same tree.

3. (5 points) A Dynamic Array (abbreviated as DArray) is a type of array that permits the insertion and deletion of elements at the end to dynamically adjust its size. DArray D maintains three crucial attributes, including the underlying array D.data, the logical data size D.size, and the actual array capacity D.capacity. The invariant $D.\text{size} \leq D.\text{capacity}$ holds at any given moment. The Insert(D, x) operation appends the element x to the end of D.data. In case $D.\text{size} = D.\text{capacity}$, it triggers a call to Resize(D) before appending x, where Resize(D) allocates a new and larger array, moves all the data to it, increases the value of D.capacity, sets up D.data, and finally frees the old array. Two implementations of Resize(D) are provided: ResizeA(D) increases D.capacity by a constant (e.g., 10) while ResizeB(D) doubles D.capacity each time.

   To initialize DArray D, we set D.size to 0, D.capacity to 1, and allocate a size 1 array to D.data. Assuming that both allocating and freeing arrays of length n take $O(n)$ time, and moving a single element takes $O(1)$ time, please select the correct description(s) below.

   A. DArray is not suitable for random access because its memory may be relocated.

   B. DArray can only be allocated in the heap memory due to its dynamic memory nature.

   C. If ResizeA(D) is considered, the amortized time complexity of Insert(D, x) is $O(1)$.

   D. If ResizeB(D) is considered, the amortized time complexity of Insert(D, x) is $O(1)$.

   E. In the Delete(D) operation, if $D.\text{size} > 0$, D.size is decreased by one. If, at this stage, $D.\text{size} < \lceil D.\text{capacity}/2 \rceil$, we create a new array d' with a size of $\lceil D.\text{capacity}/2 \rceil$, transfer the data to d', set up d' as the new data array for D.data, and subsequently free the old array. Then, the amortized time complexity of mixed Insert and Delete operations can be $O(1)$, where Resize(D) can be either ResizeA(D) or ResizeB(D).

4. (10 points) Consider a hash function h that maps numerous keys into an array A, indexed from 0 to $n-1$. The concern is that excessive collisions arising from the hashed keys can potentially cause inefficiency within the hash table. The following pseudocode, while not complete, implements a function that utilizes the divide-and-conquer strategy to identify the case where strictly more than half of the keys share the same hash value. The function takes the hash function h and the subarray A with the index range from *low* to *high* as input and returns the majority hash value shared by over $\lfloor (high - low + 1)/2 \rfloor$ keys. If the keys stored in the subarray $A[low..high]$ do not have such a majority hash value, the function returns -1.

```
1: function FindMajorityHashValue(h, A, low, high)
2:     if low = high then return h(A[low]) end if
3:     mid ← ⌊(low + high)/2⌋
4:     leftMajority ← FindMajorityHashValue(h, A, low, mid)
5:     rightMajority ← FindMajorityHashValue(h, A, mid + 1, high)
6:     leftCount ← CountElement(A, low, high, leftMajority)
7:     rightCount ← CountElement(A, low, high, rightMajority)
8:     [                                                    ]
9: end function
```

   In the pseudocode, CountElement(A, *low*, *high*, *val*) calculates and returns the count of keys in the subarray $A[low..high]$ whose hashed values are equal to the given *val*. Please select the correct description(s) below.

接次頁

A. If $leftMajority = rightMajority = -1$, $A[low..high]$ may still have a majority hash value.

B. If $A[low..high]$ has a majority hash value and $rightMajority = -1$, the function should return $leftMajority$.

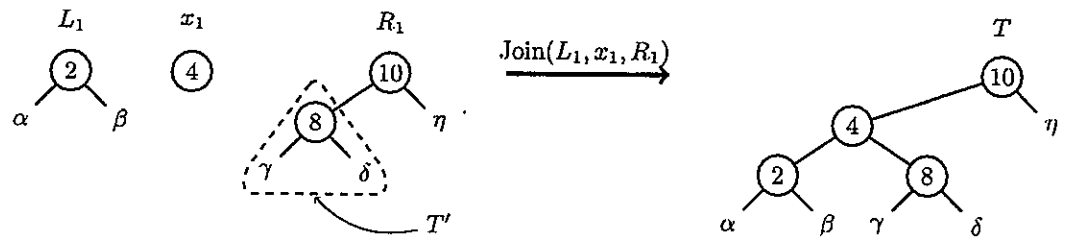
C. If $leftCount = rightCount$, the function should return -1.

D. Suppose that an implementation of CountElement runs within $\Theta(\ell)$ time, where $\ell = high - low + 1$. The time complexity of FindMajorityHashValue can be $O(n \log n)$.

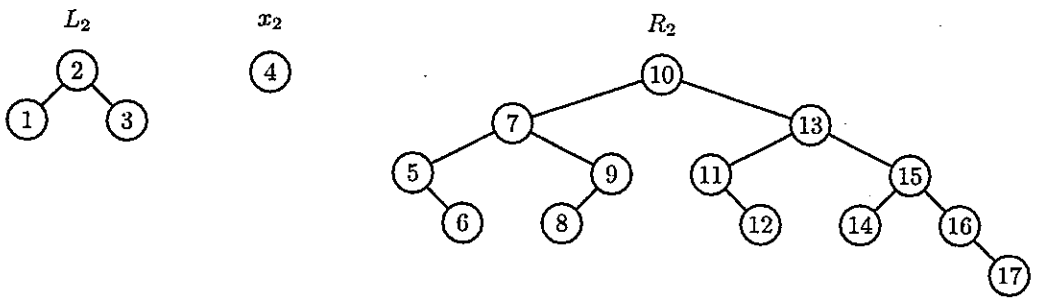
E. Suppose that a magical implementation of CountElement runs within $\Theta(\ell / \log \ell)$ time, where $\ell = high - low + 1$. The time complexity of FindMajorityHashValue can be $O(n)$.

5. (10 points) A Binary Search Tree (abbreviated as BST) is a tree structure comprised of nodes that maintain a specific order among their keys. Given a single node x and two BSTs L and R, in which l.key $< x$.key $< r$.key for all nodes $l \in L$ and $r \in R$, the objective is to design a Join(L, x, R) function that integrates the given input into a unified BST T with the nodes $L \cup \{x\} \cup R$. Be aware that AVL and Red-Black trees are specific types of BSTs. Given the constraint that T, L, and R must share the same type, they are required to be either all BSTs, all AVL trees, or all Red-Black trees.

   The Join(L, x, R) function is implemented with the steps: (1) selecting a subtree T' from either L or R, where a subtree is defined as potentially being empty, a portion, or the entire tree, (2) detaching T', (3) attaching T' under the node x, (4) attaching the other tree (which does not contain T') under x, and (5) re-attaching the tree rooted at x to the former parent of T'. The modified tree is then returned as T. If T' has no former parent, the tree rooted at x is simply returned as T. The following figure demonstrates an example result of Join(L_1, x_1, R_1), where the letters α, β, γ, δ, and η denote arbitrary subtrees. Inside a circle, the value represents the key of a node.



Please note that almost all steps, except for (1), have been finalized — there could be multiple ways to choose T' in (1). The tree returned from Join(L, x, R) may not be unique. Additionally, unlike AVL or Red-Black trees, Join(L, x, R) does not perform re-coloring and re-balancing. Consider the following arguments passed into Join(L_2, x_2, R_2). All nil nodes (considered as leaves in red-black trees) are omitted for brevity.



Please select the correct description(s) below.

A. If no additional balance is required for L_2 and R_2, i.e., both L_2 and R_2 are BSTs, there are 6 possible tree shapes that could be returned from Join(L_2, x_2, R_2).

見背面

B. If both L_2 and R_2 are AVL trees as observed, among all possible tree shapes returning from Join(L_2, x_2, R_2), there is a unique tree that is an AVL tree.

C. There are multiple ways to assign colors to the nodes in R_2 to make R_2 a red-black tree.

D. There exists a color assignment to all nodes in $L_2 \cup \{x_2\} \cup R_2$ such that: (1) both L_2 and R_2 are red-black trees, (2) a red-black tree can be returned from Join(L_2, x_2, R_2), and (3) the node with the key 4 is BLACK.

E. There exists a unique color assignment to all nodes in $L_2 \cup \{x_2\} \cup R_2$ such that: (1) both L_2 and R_2 are red-black trees, (2) a red-black tree can be returned from Join(L_2, x_2, R_2), and (3) the node with the key 1 is BLACK.

6. (5 points) We want to multiply a sequence of matrices $M_1, \ldots, M_n$ with the minimum number of multiplications. Since the matrix multiplication is associative, we can parenthesize the matrix multiplication in all possible orders. Let $m_{i,j}$ be the minimum number of multiplications to multiply $M_i, \ldots, M_j$. We can derive a recursion of $m_{i,j}$, where r_i and c_i are the numbers of rows and columns of the matrix M_i respectively.

$$m_{i,j} = \min_{i \le k < j} (m_{i,k} + m_{k+1,j} + r_i c_k c_j) \tag{1}$$

Now consider $m_{s,s+k}$, $1 < s < s+k < n$. For given s and k, let x be the number of $m_{i,j}$'s that have $m_{s,s+k}$ at the right hand side of Equation 1, and let y be the number of different m's that are at the right-hand side of Equation 1 when $i = s$ and $j = s+k$. What is the approximate value of $x + y$?

A. $s\binom{n-k}{2}$

B. $n + k$

C. $(n-s)(n-s-k)$

D. $\binom{k}{2} + \binom{n}{2}$

E. $\binom{n-k-s}{2} + \binom{k}{2}$

7. (5 points) When we compute all $m_{i,j}$'s with Equation 1, we must follow an order of i and j so that the m's on the right-hand side are all known when we compute $m_{i,j}$. Please select all the correct orders in the following choices.

A. decreasing order of $i + j$

B. increasing order of $i + j$

C. increasing order of $(i-j)^2$

D. decreasing order of $(i-j)^3$

E. increasing order of $|(i+j)(i-j)|$

8. (5 points) We define the height of a tree to be the maximum number of edges from the root to a leaf. We also define the level of a node v, denoted as $l(v)$ to be the height of the subtree rooted at v. Which of the following descriptions is correct?

A. If v is a leaf, then $l(v) = 1$.

B. If v is an internal node, $l(v) = \max_{s \in S} l(s)$, where S is the set of children of v.

C. We can compute $l(v)$ by starting a breadth-first-search (BFS) from v, traversing all nodes of the subtree rooted at v, and computing the maximum distance to a leaf as $l(v)$. Since there are no more than n v's and each BFS takes $O(n)$ time, the total time complexity is $O(n^2)$

D. We can compute the levels for all nodes by a depth-first-search (DFS) starting from the root of T. That is, we can compute the level of an internal node v after we compute the l values of all children of v.

E. Since there are n tree nodes to visit, the DFS from the root takes $O(n)$ time.

接次頁

9. (5 points) We have a binary tree T and for every node $v \in T$, we want to compute the new height if we select v as the new root.

   We consider the effect of making v the new root. First, the height of the subtree of v is still $l(v)$. Second, since v is now the new root, it could go up to its parent and then to the farthest leaf. We define this distance as $r(v)$.

   A. The new tree after selecting a new root is always a binary tree.

   B. The height of the tree after making v the new root is $\max(l(v), r(v))$.

   C. We can compute $r(v)$ for all tree node v's in a top-down manner. We assume that the r value of v's parent in T is known, so v only needs to consider the r value of its parent and the l value of its sibling in T, in order to compute its own r.

   D. The r value of the root of T is 1.

   E. We can compute the r values for all nodes in $O(n)$ time.

10. (5 points) Consider a sequence of numbers $(v_1, \ldots, v_n)$. Now we remove k numbers from the sequence so that we have $k+1$ non-empty segments of numbers. For example, consider $(2, 3, 8, 1, 4)$. If we remove 8 then we have two segments $(2, 3)$ and $(1, 4)$.

    Now we want to minimize the maximum sum of numbers of a segment. Let $m(1, n, k)$ be the answer, then what is the correct recursion for m when $1 \le i < j \le n$ and $0 < k \le (j-i)/2$?

    A. $m(i, j, k) = \min_{x=i}^{j} \max(\sum_{y=x}^{j} v_y, m(i, x, k-1))$

    B. $m(i, j, k) = \min_{x=i+1}^{j-1} \max(\sum_{y=x+1}^{j} v_y, m(i, x-1, k-1))$

    C. $m(i, j, k) = \min_{x=i+1}^{j-1} \max(m(x+1, j, k-1), \sum_{y=i}^{x-1} v_y)$

    D. $m(i, j, k) = \min_{v+w=k-1, v, w \ge 0} \min_{x=i+1}^{j-1} (\max(m(i, x-1, v), m(x+1, j, w))$

    E. $m(i, j, k) = \min_{v+w=k-1, v, w \ge 0} \min_{x=i}^{j} (\max(m(i, x, v), m(x, j, w))$

11. (5 points) Which of the following descriptions is true for the previous problem?

    A. We can always find an optimal solution that removes one of the largest v.

    B. We can always find an optimal solution that does not remove one of the smallest v.

    C. The initial value for $m(i, i, 0)$ is v_i.

    D. The initial value for $m(i, j, k)$ when $k > (j-i)/2$ is 0.

    E. The initial value for $m(i, i+2, 1)$, $1 \le i \le n-2$ is $\max(v_i, v_{i+2})$.

12. (5 points) Consider a directed acylic graph $G = (V, E)$ of n tasks where every node is a task and every edge is a dependency. If there is an edge from a task v to another task w, that means we need to finish v before starting w. We have p processors of the same capability and every processor can finish any task in one unit of time.

    We define a schedule function s to map a task v to a positive integer time step $s(v)$. Since we have only p processors, so the system can only process at most p tasks in one time step. Also, a schedule needs to respect the dependency of edges.

    A. For every positive integer i, $|\{v | s(v) \le i\}| \le p$.

    B. If there is a path $(v_{i_1}, \ldots, v_{i_k})$ in G, then $s(v_{i_j}) < s(v_{i_k})$ for $1 \le j \le k$.

    C. The number of time steps required is at least $\min(\lceil n/p \rceil, L)$, where L is the number of tasks along the longest path in G.

    D. Without loss of generality, we can always assume $s(v) = 1$ for all v's without incoming edges.

    E. If the number of processors is infinite, then we can finish all tasks in L time steps, where L is the number of tasks along the longest path in G.

見背面

13. (30 points) A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$ (or both). The size of a vertex cover is defined as the number of vertices in it. The vertex-cover problem is to find a vertex cover of minimum size in a given undirected graph. It has been shown to be NP-hard, thus we cannot find a polynomial-time algorithm for solving it exactly unless NP = P. Fortunately, some polynomial-time approximation algorithms have been proposed for solving the vertex-cover problem.

    The following two approximation algorithms take as input an undirected graph and compute a vertex cover for it. Initially, all edges in the graph are uncovered for both Algorithm VC-Arbitrary and Algorithm VC-Max-Degree. Once a vertex v is put into the cover, all edges incident on v are covered and removed.

    Algorithm VC-Arbitrary picks an arbitrary uncovered edge (u, v), puts both u and v into the cover, throws out all edges incident on either u or v, and repeats the process until there are no uncovered edges left.

    Algorithm VC-Max-Degree picks a vertex v that covers the most number of uncovered edges, puts v into the cover, throws out all edges incident on v, and repeats the process until there are no uncovered edges left.

    (a) (10 points) Give an example for which Algorithm VC-Arbitrary yields a solution that is at least double the size of a minimum-size vertex cover. Give another example for which Algorithm VC-Max-Degree will definitely yield a solution with the size more than that of a minimum-size vertex cover no matter how it breaks the tie. Each example should contain no more than ten vertices. You have to justify your solution by showing the vertex cover found by the algorithm and its corresponding minimum-size vertex cover.

    (b) (10 points) Prove or disprove that Algorithm VC-Arbitrary has a smaller ratio bound than Algorithm VC-Max-Degree. The ratio bounds in your proof or disproof should be as tight as possible.

    (c) (10 points) In the weighted vertex-cover problem, you are given an undirected graph $G = (V, E)$ and a weight $w(v)$ for each vertex $v \in V$, and the objective is to find a vertex cover (covering all edges in E) of minimum total weight. In other words, the goal is to find a vertex cover C of G such that $\sum_{v \in C} w(v)$ is minimized among all possible vertex covers of G. You are required to formulate the weighted vertex cover problem as an integer linear program.

試題隨卷繳回