Data Structure, Final

A. Single Choice Questions (5% each, 60%)

(1) Consider the following recursive function in Python and C++.

```python
def value(n):
    if n <= 1:
        return n
    return value(n - 1) + 2 * value(n - 2)
```

```cpp
int value(int n) {
    if (n <= 1) return n;
    return value(n - 1) + 2 * value(n - 2);
}
```

What is `value(5)`?

(A) 5

(B) 8

(C) 11

(D) 16

(C)

`value(0)=0`, `value(1)=1`, `value(2)=1`, `value(3)=3`, `value(4)=5`, and `value(5)=11`.

(2) Ordered sequential search is used on

`[4, 9, 13, 20, 28, 35]`
to search for `26`. How many comparisons are made before stopping?

(A) 4

(B) 5

(C) 6

(D) 2

(B)

The search compares 4, 9, 13, 20, and 28. Since 28 is larger than 26, it can stop.

(3) A hash table of size `7` and uses the remainder method with separate chaining which appends new keys to the end of each chain. Which chain stores keys `8, 15, 22` after insertion?

(A) slot 0

(B) slot 1

(C) slot 2

(D) slot 6

(B)

All three keys have remainder 1 modulo 7, so they are stored in the chain at slot 1.

(4) Which statement is FALSE about binary search trees (BST)?

(A) An inorder traversal of a BST visits keys in sorted order.

(B) Searching a BST compares the target key with the current node and then moves left or right.

(C) A BST also be a complete binary tree.

(D) All keys in the left subtree of a node are less than that node's key, under the usual no-duplicate convention.

(C)

A BST does not need to be complete; it can become unbalanced depending on insertion order.

(5) One Shell sort pass with gap `3` is applied to

`[31, 14, 52, 9, 43, 20, 37, 11, 28]`
What is the list after this gap pass?

(A) `[9, 11, 20, 31, 14, 28, 37, 43, 52]`

(B) `[9, 14, 20, 31, 43, 28, 37, 11, 52]`

(C) `[14, 31, 9, 20, 43, 11, 28, 37, 52]`

(D) `[9, 11, 20, 28, 31, 37, 43, 52, 14]`

(A)

Sort subsequences at indices 0,3,6; 1,4,7; and 2,5,8 independently.

(6) Which statement is TRUE about recursive maze DFS and explicit stack-based maze DFS?

(A) Recursive DFS cannot backtrack.

(B) Explicit stack DFS cannot mark visited cells.

(C) Recursive DFS stores unfinished work in stack frames; explicit stack DFS stores it in a stack object.

(D) Both always return the same path no matter what neighbor order is used.

(C)

The two approaches store equivalent search state in different places; neighbor order can change the first path found.

(7) The following is the adjacency-list representation of a graph. Each row lists the neighbors of one vertex. Neighbors are processed in alphabetical order.

```
R: S, T
S: U, V
T: W
U:
V: W
W:
```

What is the BFS visit order starting from `R`?

(A) `R, S, U, V, W, T`

(B) `R, S, T, U, V, W`

(C) `R, T, W, S, V, U`

(D) `R, S, T, W, V, U`

(B)

BFS visits R, then S and T, then the next layer U, V, and W.

(8) Which statement is FALSE about recursion?

(A) Every recursive function needs at least one base case that can stop the recursion.

(B) A recursive call usually works on a smaller or simpler version of the original problem.

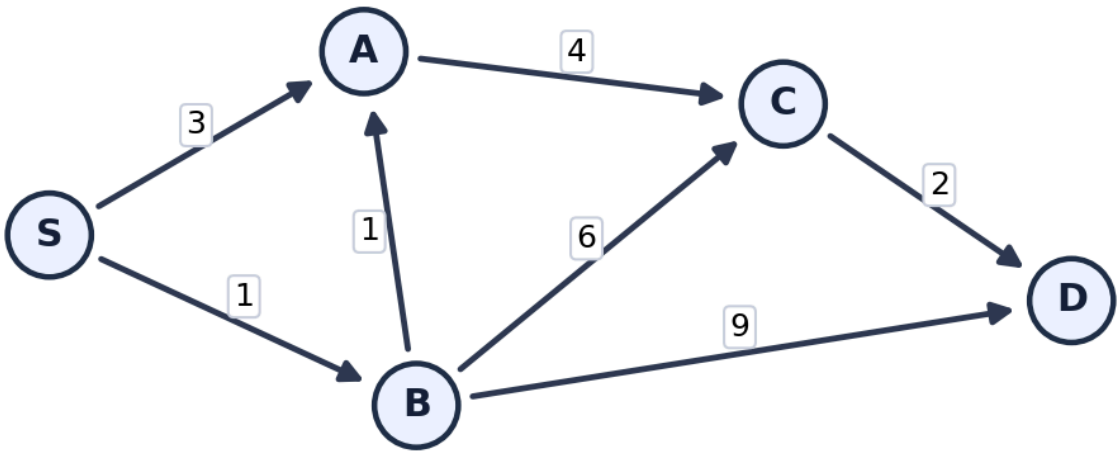
(C) Recursion uses stack frames to remember unfinished function calls.

(D) Recursion always uses less memory than an iterative solution.

(D)

Recursive calls store active calls in stack frames, so recursion does not always use less memory than iteration.

(9) Run Dijkstra's algorithm from `S` on the following directed graph.



In what order are vertices removed from the priority queue and marked visited/finalized?

(A) `S, A, B, C, D`

(B) `S, B, A, C, D`

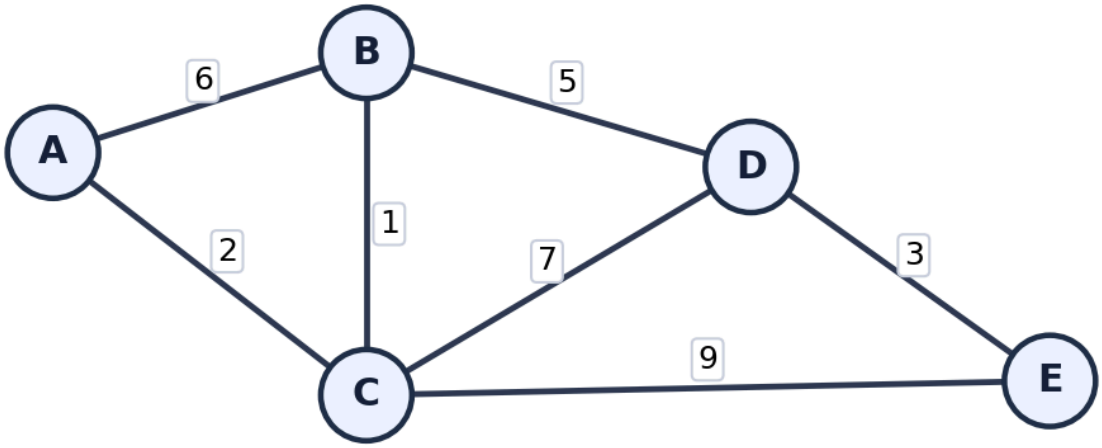
(C) `S, B, C, A, D`

(D) `S, A, C, B, D`

(B)

Distances finalize as S=0, B=1, A=2, C=6, and D=8.

(10) Prim's algorithm starts from `A` on the following undirected weighted graph.



Which edge is added last?

(A) `B-C`

(B) `A-C`

(C) `D-E`

(D) `B-D`

(C)

The edge order is A-C, B-C, B-D, then D-E.

(11) The lecture `heapify()` method builds a min-heap by calling `_perc_down()` from the last non-leaf node back to the root. Starting from the list

`[14, 9, 21, 3, 18, 7, 11]`
what is the final heap's list representation after `heapify()`?

(A) `[3, 9, 7, 14, 18, 21, 11]`

(B) `[7, 3, 11, 9, 18, 21, 14]`

(C) `[3, 7, 9, 14, 18, 21, 11]`

(D) `[14, 3, 7, 9, 18, 21, 11]`

(A)

`heapify()` calls `_perc_down()` at indices 2, 1, and 0. The resulting heap's list representation is `[3, 9, 7, 14, 18, 21, 11]`.

(12) Insert `30, 18, 42, 12, 24, 36, 48, 20` into an empty BST. Which nodes are visited when searching for `20`?

(A) `30, 42, 36, 20`

(B) `30, 18, 12, 20`

(C) `30, 18, 24, 20`

(D) `30, 24, 18, 20`

(C)

20 is less than 30, greater than 18, and less than 24.

B. Short-answer questions, please provide the derivation for each question along with your answer (8% each, 40%).

(13) Briefly explain each of the following terms.

(a) Stack frame

(b) Quadratic probing

(c) Priority queue

(d) Heap order property

(a) A **stack frame** is the information stored for one active function call, including parameters, local variables, and where to return.

(b) **Quadratic probing** is an open addressing collision resolution method. After a collision, it probes positions offset by successive perfect squares, such as `h + 1`, `h + 4`, `h + 9`, and so on modulo the table size. This helps reduce the clustering problem of linear probing.

(c) A **priority queue** removes the item with highest priority, such as the smallest tentative distance in Dijkstra's algorithm.

(d) The **heap order property** requires each parent in a min-heap to be less than or equal to its children.

(14) A hash table has size `11` and uses the remainder method with linear probing. Insert the keys in this order:

`22, 1, 13, 24, 35, 46, 57`
Show the final table. Use `empty` for unused slots.

The home slots are:

```
22 % 11 = 0
1  % 11 = 1
13 % 11 = 2
24 % 11 = 2
```

```
35 % 11 = 2
46 % 11 = 2
57 % 11 = 2
```
Final table:

```
0: 22      1: 1       2: 13      3: 24
4: 35      5: 46      6: 57      7: empty
8: empty   9: empty  10: empty
```

(15) The following parse tree represents an arithmetic expression.

```
        *
      /   \
     -     /
    / \   / \
   9   + 8   4
      / \
     2   1
```

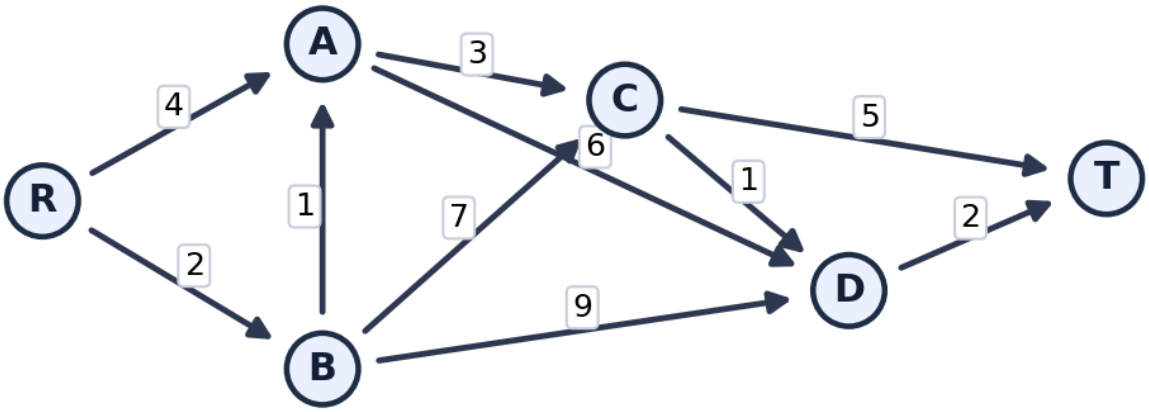
Recover the fully parenthesized expression and evaluate it.

The fully parenthesized expression is:

```
( ( 9 - ( 2 + 1 ) ) * ( 8 / 4 ) )
```
Evaluate bottom-up:

```
2 + 1 = 3
9 - 3 = 6
8 / 4 = 2
6 * 2 = 12
```
The value is `12`.

(16) The following directed weighted graph is separate from the earlier Dijkstra question.



Starting from `R`, use Dijkstra's algorithm to find the final distance and previous vertex for every vertex. Then give one shortest path from `R` to `T`.

Finalized order:

```
R, B, A, C, D, T
```

Final distance and previous table:

```
vertex  distance  previous
R       0         -
B       2         R
A       3         B
C       6         A
D       7         C
T       9         D
```

One shortest path to `T` is:

```
R, B, A, C, D, T
```

(17) A binary tree has the following traversal sequences:

```
Inorder:   D B E A F C G
Postorder: D E B F G C A
```

Construct the tree.

Postorder gives root `A`. Inorder splits the left subtree as `D B E` and the right subtree as `F C G`.

The left postorder part is `D E B`, so `B` has left child `D` and right child `E`. The right postorder part is `F G C`, so `C` has left child `F` and right child `G`.

```
      A
    /   \
   B     C
  / \   / \
 D   E F   G
```

C. Bonus

Searching and sorting input order (+8%, 4% each)

(a) Suppose an unsorted list has 10,000 numbers and you need to search for one target only once. Should you first sort the list and then use binary search, or directly use sequential search? Explain using Big-O and the condition needed for binary search. (4%)

(b) In a sorting benchmark, why should we test sorted, inverse-sorted, and random input lists instead of only random input lists? Give one example of an algorithm or variant whose behavior can change a lot on sorted or inverse-sorted input. (4%)

(a) Direct sequential search is usually better for one search.

Sequential search on an unsorted list is `O(n)`. Binary search is `O(log n)`, but it requires the list to be sorted first. Sorting first costs about `O(n log n)` for common efficient sorts, so doing `O(n log n) + O(log n)` for one search is usually worse than doing one `O(n)` sequential search.

(b) Random input alone does not show the full performance picture because sorted and inverse-sorted inputs can expose best-case or worst-case behavior.

For example, quicksort can behave poorly on already sorted or inverse-sorted data if the pivot choice repeatedly creates very unbalanced partitions. The median-of-three pivot rule is meant to choose a better pivot when the list is somewhat sorted. Insertion sort can also behave very differently on nearly sorted input because fewer shifting operations may be needed.