Republic of the Philippines

SULTAN KUDARAT STATE UNIVERSITY

Isulan Camus, Isulan Sultan Kudarat

College of Computer Studies

Bachelor of Science in Information System

CC 114 – Data Structure and Algorithm

Final Exam

Name: ___________________________________ Course/Yr/Section______________________ Score: __________

Instructions:

- Read each question carefully and encircle the letter of the correct answer.
- You have 2 hours to complete the exam.

Chapter 4: Algorithms

Lesson 01: Evaluating Expressions (PEMDAS, Infix, Prefix, Postfix)

1. What does PEMDAS stand for in expression evaluation?
   A. Parentheses, Exponents, Multiplication and Division, Addition and Subtraction
   B. Parentheses, Exponents, Multiplication, Division, Addition, Subtraction
   C. Parentheses, Exponents, Multiplication and Division, Addition, Subtraction
   D. Parentheses, Exponents, Multiplication, Division, Addition and Subtraction
2. Which notation places operators between operands?
   A. Infix
   B. Prefix
   C. Postfix
   D. Both A and B
3. What is the correct order of operations for the expression 3 + 5 * 2?
   A. 3 + (5 * 2)
   B. (3 + 5) * 2
   C. 3 + 5 + 2
   D. 5 * (3 + 2)
4. Which notation places operators after operands?
   A. Infix
   B. Prefix
   C. Postfix
   D. Both B and C
5. What is the prefix notation for the infix expression A + B * C?
   A. +ABC
   B. +A*BC
   C. *AB+C
   D. ABC+
6. What is the postfix notation for the infix expression A + B * C?
   A. ABC+
   B. +AB*C
   C. +A*BC
   D. ABC*+
7. Which notation eliminates the need for parentheses?
   A. Infix
   B. Prefix
   C. Postfix
   D. Both B and C
8. What is the result of evaluating the postfix expression 3 4 + 5 ?
   A. 35
   B. 23
   C. 17
   D. 15
9. Which notation is also known as Polish notation?
   A. Infix
   B. Prefix
   C. Postfix
   D. Both A and B
10. What is the infix notation for the prefix expression * + 3 4 5?
    A. (3 + 4) * 5
    B. 3 + 4 * 5
    C. 3 + (4 * 5)
    D. 3 + 4 + 5
11. Which code snippet correctly converts the infix expression A + B * C to postfix notation?
    A. ABC+*
    B. +A*BC
    C. +AB*C
    D. ABC*+
12. What is the result of evaluating the prefix expression * + 3 4 5?
    A. 35
    B. 23
    C. 17
    D. 15
13. *Which code snippet correctly evaluates the postfix expression 3 4 + 5 ?
    A. 35
    B. 23
    C. 17
    D. 15
14. What is the prefix notation for the infix expression (A + B) * C?
    A. *+ABC
    B. +A*BC
    C. *AB+C
    D. ABC+*

15. What is the postfix notation for the infix expression (A + B) * C?
    A. A B+ C*
    B. +A *B C
    C. * A B+ C
    D. A B C* +
16. Why is postfix notation advantageous for computer evaluation of expressions?
    A. It eliminates the need for parentheses.
    B. It follows the natural order of human reading.
    C. It is easier to parse and evaluate using a stack.
    D. It is more intuitive for beginners.
17. What is the purpose of using a stack in evaluating postfix expressions?
    A. To store operands and operators in the correct order.
    B. To handle parentheses in infix expressions.
    C. To convert infix to prefix notation.
    D. To manage memory allocation.
18. How does the PEMDAS rule affect the evaluation of infix expressions?
    A. It ensures the correct order of operations.
    B. It eliminates the need for parentheses.
    C. It converts infix to postfix notation.
    D. It simplifies the evaluation of prefix expressions
19. .What is the advantage of prefix notation over infix notation?
    A. It eliminates the need for parentheses.
    B. It is easier to parse and evaluate using a stack.
    C. It follows the natural order of human reading.
    D. It is more intuitive for beginners.
20. Why is infix notation the most commonly used notation in programming?
    A. It is easier to parse and evaluate using a stack.
    B. It eliminates the need for parentheses.
    C. It follows the natural order of human reading.
    D. It is more intuitive for beginners
21. .Write a C++ function to convert an infix expression to postfix notation.
    A. string infixToPostfix(string infix) { /* implementation */ }
    B. int infixToPostfix(string infix) { /* implementation */ }
    C. void infixToPostfix(string infix) { /* implementation */ }
    D. char infixToPostfix(string infix) { /* implementation */ }
22. Write a C++ function to evaluate a postfix expression.
    A. int evaluatePostfix(string postfix) { /* implementation */ }
    B. string evaluatePostfix(string postfix) { /* implementation */ }
    C. void evaluatePostfix(string postfix) { /* implementation */ }
    D. char evaluatePostfix(string postfix) { /* implementation */ }
23. What is the result of evaluating the prefix expression + *3 4 33?
    A. 45
    B. 35
    C. 27
    D. 23
24. What is the postfix notation for the infix expression A + B * (C + D)?
    A. A B C D+*+
    B. +A* B +C D
    C. +A B* +C D
    D. A B+ *C D+
25. What is the prefix notation for the infix expression A + B * (C + D)?
    A. +A*B+CD
    B. +AB*+CD
    C. +A*B+CD
    D. AB+*CD+
    E.
26. Which notation is most suitable for human reading and writing?
    A. Infix
    B. Prefix
    C. Postfix
    D. Both A and B
27. Why is it important to follow the PEMDAS rule in infix expressions?
    A. To ensure the correct order of operations.
    B. To eliminate the need for parentheses.
    C. To convert infix to postfix notation.
    D. To simplify the evaluation of prefix expressions.
28. What is the primary advantage of using a stack in expression evaluation?
    A. It eliminates the need for parentheses.
    B. It follows the natural order of human reading.
    C. It is easier to parse and evaluate using a stack.
    D. It is more intuitive for beginners.
29. Which sorting algorithm has a time complexity of O(n^2)?
    A. Selection Sort
    B. Bubble Sort
    C. Insertion Sort
    D. All of the above
30. Which sorting algorithm is based on the divide-and-conquer strategy?
    A. Selection Sort
    B. Bubble Sort
    C. Merge Sort
    D. Insertion Sort
31. Which sorting algorithm uses a pivot to partition the array?
    A. Selection Sort
    B. Bubble Sort
    C. Quick Sort
    D. Insertion Sort

32. Which sorting algorithm is stable?
A. Selection Sort
B. Bubble Sort
C. Merge Sort
D. Insertion Sort

33. Which sorting algorithm has a time complexity of O(n log n)?
A. Selection Sort
B. Bubble Sort
C. Merge Sort
D. Insertion Sort
E.

34. Why is Selection Sort inefficient for large datasets?
A. It has a time complexity of O(n^2).
B. It is a stable sorting algorithm.
C. It uses a pivot to partition the array.
D. It is based on the divide-and-conquer strategy.

35. Why is Merge Sort preferred for large datasets?
A. It has a time complexity of O(n^2).
B. It is a stable sorting algorithm.
C. It uses a pivot to partition the array.
D. It is based on the divide-and-conquer strategy.

36. What is the role of the pivot in Quick Sort?
A. It helps in partitioning the array.
B. It ensures the stability of the sort.
C. It divides the array into two halves.
D. It compares adjacent elements.

37. Why is Bubble Sort considered inefficient?
A. It has a time complexity of O(n^2).
B. It is a stable sorting algorithm.
C. It uses a pivot to partition the array.
D. It is based on the divide-and-conquer strategy.

38. What is the advantage of Insertion Sort for small datasets?
A. It has a time complexity of O(n^2).
B. It is a stable sorting algorithm.
C. It uses a pivot to partition the array.
D. It is based on the divide-and-conquer strategy.

39. Which code snippet correctly implements Selection Sort?
A. for (int i = 0; i < n; i++) { for (int j = i + 1; j < n; j++) { if (arr[i] > arr[j]) swap(arr[i], arr[j]); } }
B. for (int i = 0; i < n - 1; i++) { int minIndex = i; for (int j = i + 1; j < n; j++) { if (arr[j] < arr[minIndex]) minIndex = j; } swap(arr[i], arr[minIndex]); }
C. for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]); } }
D. for (int i = 1; i < n; i++) { int key = arr[i]; int j = i - 1; while (j >= 0 && arr[j] > key) { arr[j + 1] = arr[j]; j--; } arr[j + 1] = key; }

40. Which code snippet correctly implements Bubble Sort?
A. for (int i = 0; i < n; i++) { for (int j = i + 1; j < n; j++) { if (arr[i] > arr[j]) swap(arr[i], arr[j]); } }
B. for (int i = 0; i < n - 1; i++) { int minIndex = i; for (int j = i + 1; j < n; j++) { if (arr[j] < arr[minIndex]) minIndex = j; } swap(arr[i], arr[minIndex]); }
C. for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]); } }
D. for (int i = 1; i < n; i++) { int key = arr[i]; int j = i - 1; while (j >= 0 && arr[j] > key) { arr[j + 1] = arr[j]; j--; } arr[j + 1] = key; }

41. Which code snippet correctly implements Insertion Sort?
A. for (int i = 0; i < n; i++) { for (int j = i + 1; j < n; j++) { if (arr[i] > arr[j]) swap(arr[i], arr[j]); } }
B. for (int i = 0; i < n - 1; i++) { int minIndex = i; for (int j = i + 1; j < n; j++) { if (arr[j] < arr[minIndex]) minIndex = j; } swap(arr[i], arr[minIndex]); }
C. for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]); } }
D. for (int i = 1; i < n; i++) { int key = arr[i]; int j = i - 1; while (j >= 0 && arr[j] > key) { arr[j + 1] = arr[j]; j--; } arr[j + 1] = key; }

42. Which code snippet correctly implements Merge Sort?
A. void mergeSort(int arr[], int left, int right) { if (left < right) { int mid = left + (right - left) / 2; mergeSort(arr, left, mid); mergeSort(arr, mid + 1, right); merge(arr, left, mid, right); } }
B. for (int i = 0; i < n; i++) { for (int j = i + 1; j < n; j++) { if (arr[i] > arr[j]) swap(arr[i], arr[j]); } }
C. for (int i = 0; i < n - 1; i++) { int minIndex = i; for (int j = i + 1; j < n; j++) { if (arr[j] < arr[minIndex]) minIndex = j; } swap(arr[i], arr[minIndex]); }
D. for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]); } }

43. Which code snippet correctly implements Quick Sort?
A. int partition(int arr[], int low, int high) { int pivot = arr[high]; int i = (low - 1); for (int j = low; j < high; j++) { if (arr[j] < pivot) { i++; swap(arr[i], arr[j]); } } swap(arr[i + 1], arr[high]); return (i + 1); } void quickSort(int arr[], int low, int high) { if (low < high) { int pi = partition(arr, low, high); quickSort(arr,

low, pi - 1); quickSort(arr, pi + 1, high); } }

B. for (int i = 0; i < n; i++) { for (int j = i + 1; j < n; j++) { if (arr[i] > arr[j]) swap(arr[i], arr[j]); } }

C. for (int i = 0; i < n - 1; i++) { int minIndex = i; for (int j = i + 1; j < n; j++) { if (arr[j] < arr[minIndex]) minIndex = j; } swap(arr[i], arr[minIndex]); }

D. for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]); } }

44. Why is Merge Sort considered a stable sorting algorithm?

A. It maintains the relative order of equal elements.

B. It has a time complexity of O(n^2).

C. It uses a pivot to partition the array.

D. It is based on the divide-and-conquer strategy

45. .What is the advantage of Quick Sort over Merge Sort?

A. It has a time complexity of O(n^2).

B. It is a stable sorting algorithm.

C. It uses a pivot to partition the array.

D. It is based on the divide-and-conquer strategy.

46. Why is Bubble Sort considered inefficient for large datasets?

A. It has a time complexity of O(n^2).

B. It is a stable sorting algorithm.

C. It uses a pivot to partition the array.

D. It is based on the divide-and-conquer strategy.

47. What is the advantage of Insertion Sort for small datasets?

A. It has a time complexity of O(n^2).

B. It is a stable sorting algorithm.

C. It uses a pivot to partition the array.

D. It is based on the divide-and-conquer strategy.

48. Why is Selection Sort inefficient for large datasets?

A. It has a time complexity of O(n^2).

B. It is a stable sorting algorithm.

C. It uses a pivot to partition the array.

D. It is based on the divide-and-conquer strategy.

49. Which of the following C++ function implements Selection Sort.

A. void selectionSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { int minIndex = i; for (int j = i + 1; j < n; j++) { if (arr[j] < arr[minIndex]) minIndex = j; } swap(arr[i], arr[minIndex]); } }

B. void selectionSort(int arr[], int n) { for (int i = 0; i < n; i++) { for (int j = i + 1; j < n; j++) { if (arr[i] > arr[j]) swap(arr[i], arr[j]); } } }

C. void selectionSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]); } } }

D. void selectionSort(int arr[], int n) { for (int i = 1; i < n; i++) { int key = arr[i]; int j = i - 1; while (j >= 0 && arr[j] > key) { arr[j + 1] = arr[j]; j--; } arr[j + 1] = key; } }

50. Write a C++ function to implement Bubble Sort.

A. void bubbleSort(int arr[], int n) { for (int i = 0; i < n; i++) { for (int j = i + 1; j < n; j++) { if (arr[i] > arr[j]) swap(arr[i], arr[j]); } } }

B. void bubbleSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { int minIndex = i; for (int j = i + 1; j < n; j++) { if (arr[j] < arr[minIndex]) minIndex = j; } swap(arr[i], arr[minIndex]); } }

C. void bubbleSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]); } } }

D. void bubbleSort(int arr[], int n) { for (int i = 1; i < n; i++) { int key = arr[i]; int j = i - 1; while (j >= 0 && arr[j] > key) { arr[j + 1] = arr[j]; j--; } arr[j + 1] = key; } }

51. Which of the following C++ implements Insertion Sort.

A. void insertionSort(int arr[], int n) { for (int i = 0; i < n; i++) { for (int j = i + 1; j < n; j++) { if (arr[i] > arr[j]) swap(arr[i], arr[j]); } } }

B. void insertionSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { int minIndex = i; for (int j = i + 1; j < n; j++) { if (arr[j] < arr[minIndex]) minIndex = j; } swap(arr[i], arr[minIndex]); } }

C. void insertionSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]); } } }

D. void insertionSort(int arr[], int n) { for (int i = 1; i < n; i++) { int key = arr[i]; int j = i - 1; while (j >= 0 && arr[j] > key) { arr[j + 1] = arr[j]; j--; } arr[j + 1] = key; } }

52. Which of the following C++ implements Merge Sort.

A. void mergeSort(int arr[], int left, int right) { if (left < right) { int mid = left + (right - left) / 2; mergeSort(arr, left, mid); mergeSort(arr, mid + 1, right); merge(arr, left, mid, right); } }

B. void mergeSort(int arr[], int n) { for (int i = 0; i < n; i++) { for (int j = i + 1; j < n; j++) { if (arr[i] > arr[j]) swap(arr[i], arr[j]); } } }

C. void mergeSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { int minIndex = i; for (int j = i + 1; j < n; j++) { if (arr[j] <

arr[minIndex]) minIndex = j; } swap(arr[i], arr[minIndex]); } }

D. void mergeSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]); } } }

53. Which of the following C++ implements Quick Sort.

A. int partition(int arr[], int low, int high) { int pivot = arr[high]; int i = (low - 1); for (int j = low; j < high; j++) { if (arr[j] < pivot) { i++; swap(arr[i], arr[j]); } } swap(arr[i + 1], arr[high]); return (i + 1); } void quickSort(int arr[], int low, int high) { if (low < high) { int pi = partition(arr, low, high); quickSort(arr, low, pi - 1); quickSort(arr, pi + 1, high); } }

B. void quickSort(int arr[], int n) { for (int i = 0; i < n; i++) { for (int j = i + 1; j < n; j++) { if (arr[i] > arr[j]) swap(arr[i], arr[j]); } } }

C. void quickSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { int minIndex = i; for (int j = i + 1; j < n; j++) { if (arr[j] < arr[minIndex]) minIndex = j; } swap(arr[i], arr[minIndex]); } }

D. void quickSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]); } } }

54. Which sorting algorithm is most suitable for small datasets?

A. Selection Sort
B. Bubble Sort
C. Insertion Sort
D. Merge Sort

55. Which sorting algorithm is most suitable for large datasets?

A. Selection Sort
B. Bubble Sort
C. Merge Sort
D. Insertion Sort

56. Which sorting algorithm is based on the divide-and-conquer strategy?

A. Selection Sort
B. Bubble Sort
C. Merge Sort
D. Insertion Sort

57. Which sorting algorithm uses a pivot to partition the array?

A. Selection Sort
B. Bubble Sort
C. Quick Sort
D. Insertion Sort

58. Which sorting algorithm is stable?

A. Selection Sort
B. Bubble Sort
C. Merge Sort
D. Insertion Sort

59. What is hashing?

A. A technique to encrypt data.
B. A technique to map data to specific indices in a data structure.
C. A technique to sort data.
D. A technique to compress data.

60. What is a hash function?

A. A function that converts input data into a fixed-size integer.
B. A function that encrypts data.
C. A function that sorts data.
D. A function that compresses data.

61. What is a hash table?

A. A data structure that stores data in key-value pairs.
B. A data structure that encrypts data.
C. A data structure that sorts data.
D. A data structure that compresses data.

62. What is a collision in hashing?

A. When two keys produce the same hash code.
B. When two keys are encrypted differently.
C. When two keys are sorted differently.
D. When two keys are compressed differently.

63. What is the purpose of a hash function?

A. To convert input data into a fixed-size integer.
B. To encrypt data.
C. To sort data.
D. To compress data.

64. Why is hashing important in data retrieval?

A. It allows for fast data retrieval.
B. It encrypts data.
C. It sorts data.
D. It compresses data.

65. What is the role of a hash table in data storage?

A. It stores data in key-value pairs.
B. It encrypts data.
C. It sorts data.
D. It compresses data.

66. What is the purpose of a collision handling method?

A. To resolve conflicts when two keys produce the same hash code.
B. To encrypt data.
C. To sort data.
D. To compress data.

67. What is the advantage of using a hash function?
   A. It allows for fast data retrieval.
   B. It encrypts data.
   C. It sorts data.
   D. It compresses data.

68. What is the purpose of a hash table?
   A. To store data in key-value pairs.
   B. To encrypt data.
   C. To sort data.
   D. To compress data.

69. Which code snippet correctly implements a hash function?
   A. int hashFunction(string key, int tableSize) { int hash = 0; for (char c : key) { hash = (hash + int(c)) % tableSize; } return hash; }
   B. int hashFunction(string key, int tableSize) { int hash = 0; for (char c : key) { hash = (hash + int(c)) * tableSize; } return hash; }
   C. int hashFunction(string key, int tableSize) { int hash = 0; for (char c : key) { hash = (hash + int(c)) / tableSize; } return hash; }
   D. int hashFunction(string key, int tableSize) { int hash = 0; for (char c : key) { hash = (hash + int(c)) + tableSize; } return hash; }

70. Which code snippet correctly implements a hash table?
   A. list<pair<string, string>> table[TABLE_SIZE];
   B. map<string, string> table;
   C. vector<pair<string, string>> table;
   D. array<pair<string, string>> table;

Prepared by:

ALEXIS D. APRESTO
Faculty
BSIS. Program Chairperson

Approved by:

ELBREN O. ANTIONIO, DIT
Dean, College of Computer Studies