

%Exam : Data Structures and Algorithms 3 (DSA 3)%

Duration: 2h - Documents: not allowed

Academic year : 2025 / 2026

Date : 17/01/2026 (10 : 30 – 12 : 00)

Corrected

Level : L2

Semester : 5

First Name	Last Name	Group
.....		

Exercise No.	1	2	3	4	5	6	Total
Marks Obtained	1.5	5	5	4	4.5	2	20

Exercise 1 : (1.5 Points / 10 Minutes)Write the values of the **int** variables when the following program ends.

#include <stdio.h>

void main() {

int a[] = {2, 3, 6, 11, 18};

int *p, *q;

int i, j, k, l, m, n;

q = &a[4];

p = q - 4;

i = *q;

j = *p++;

k = *--q;

l = p[1];

m = *(q-2);

n = q - p;

}

Answer (0.25 × 6 = 1.5 pts)

i =18.....

j =2.....

k =11.....

l =6.....

m =3.....

n =2.....

Exercise 2 : (5 Points / 20 Minutes)

Q1) Prove that $T(n) = n^2 + \log(n^2)$ is $O(n^2)$ using the formal definition of the Big-Oh notation. [Hint: Find a constant c and threshold n_0 such that $T(n) \leq cn^2$ for $n \geq n_0$.]

However, we can show that $T(n) \in O(n^2)$. (1 pt)We have : $T(n) = n^2 + \log(n^2)$ $\leq n^2 + n^2$ for all $n \geq 1$ $= 2n^2$ Choosing $c = 2$ and $n_0 = 1$, for all $n \geq n_0$, we have : $T(n) = n^2 + \log(n^2) \leq c.n^2$ so that $T(n) \in O(n^2)$.

Q2) Calculate the complexity of the following sources fragments; briefly justify your answer:

Pieces of code	Complexity
<pre>int sum = 0; for(int i = n*n*n; i >= 1; i /= 3) sum++;</pre> (0.25 + 0.25 = 0.5pts)	i starts at n^3 - Each step: i = i / 3 - Number of iterations $\approx \log_3(n^3)$ $\log(n^3) = 3 \log n \Rightarrow O(\log n)$
<pre>for(int i = 1; i < size; i++){ current = arr[i]; for(j = i; j>0 && arr[j-1]>current; j--){ arr[j] = arr[j-1]; } arr[j] = current; }</pre> (0.25 + 0.25 = 0.5pts)	Total operations: $1 + 2 + \dots + (n - 1) = \sum_{i=1}^{n-1} i$ $\frac{n(n-1)}{2} \Rightarrow O(n^2)$
<pre>int sum = 0; for (int i = 0; i < n/2; i++){ for (int j = 0; j < n; j++){ sum++; } for (int j = 1; j < n * n; j++){ for(int k = 1; k < 2026; k++){ sum++; } } }</pre> (0.25 + 0.25 = 0.5pts)	- Outer loop runs $n/2 \rightarrow O(n)$ - First inner loop runs $n \rightarrow O(n)$ - Second inner loop runs $n^2 \rightarrow O(1 \times n^2) = O(n^2)$ because - j loop runs n^2 - k loop runs 2025 times (constant) $\text{Max}(O(n), O(n^2)) \times O(n) = O(n^3)$
<pre>int func(int n){ for (int i = 0; i < n; i++) // some operations if (n <= 0) return 1; else return 1 + func(n-1) + func(n-1); }</pre> (0.5 + 0.5 = 1pt)	Recurrence relation $T(n) = \begin{cases} O(1) & \text{if } n = 0 \\ 2T(n-1) + O(n) & \text{otherwise} \end{cases}$ This is a binary recursion. - The $2T(n-1)$ term dominates - The recursion tree doubles at each level - Depth of recursion = n Total number of calls: $2^n \Rightarrow O(2^n)$
<pre>int count = 0; for (int i = n; i > 0; i /= 2) for (int j = 0; j < i; j++) count++;</pre> (0.25 + 0.25 = 0.5pts)	Total operations: $n + \frac{n}{2} + \frac{n}{4} + \dots + \frac{n}{2^k} = \sum_{k=0}^{\log n} \frac{n}{2^k}$ $\sum_{k=0}^{\log n} \frac{n}{2^k} \leq 2n \Rightarrow O(n)$

Q3) You must find **1000** most expensive items from an unsorted price list containing **10^7** different items. Two schemes of solution are as follows.

- **Scheme A:** repeat **1000 times** the sequential search (with the linear complexity **$O(n)$**).
- **Scheme B:** convert the list into an array (the same complexity **$O(n)$** as for search), then sort the array (complexity **$O(n \log n)$**) and fetch **1000** top items.

Which scheme would you prefer assuming that searching for **100 items** in the unsorted list takes **0.1 millisecond (ms)** and sorting of **100 items** also takes **0.1 ms**? **Time for fetching data should not be taken into account.** (0.25 + 0.25 + 0.5 = 1pt)

The scaling factors for the linear search and the sort are $\frac{0.1}{100} = 0.001$ and

$\frac{0.1}{100 \log 100} = \frac{1}{200} = 0.0005$, respectively (you may use any convenient base of the logarithm for computing the running time).

Then the overall time for the **scheme A** is

$$TA = 0.001 \times 10^7 \times 10^3 = 10^7 ms = 10^4 sec . \quad (0.25 pts)$$

The overall time for **scheme B** consists of the time to convert the list into an array:

$$TB1 = 0.001 \times 10^7 = 10^4 ms = 10 sec$$

and the time for sorting the array:

$$TB2 = 0.0005 \times 10^7 \log(10^7) = 35 \times 10^3 ms = 35 sec$$

$$\text{So, } TB = TB1 + TB2 = 10 + 35 = 45 sec \quad (0.5 pts)$$

Therefore, we have $TB < TA$ the scheme of choice is B. (0.25 pts)

Exercise 3: (5 Points / 20 Minutes)

The stooge sort is a recursive sorting algorithm that can be written quite simply:

```
void stooge_sort(int Tab[], int start, int end) {
    if (Tab[start] > Tab[end]) {
        swap(Tab[start], Tab[end]);
    }
    if (end - start + 1 > 2) {
        int t = (end - start + 1) / 3;
        stooge_sort(Tab, start, end - t);
        stooge_sort(Tab, start + t, end);
        stooge_sort(Tab, start, end - t);
    }
}
```

- Q1)** Run the algorithm for `stooge_sort([7, 8, 1, 6], 0, 3)`. During the execution, you must rewrite the contents of the array **Tab** each time values are swapped. (2 pts)

For each new recursive call, you must indicate the values of the **start** and **end** indices.

Call <code>stooge_sort(Tab, start, end)</code>	Tab are swapped
<code>stooge_sort([7, 8, 1, 6], 0, 3)</code>	[6, 8, 1, 7]
<code>stooge_sort([6, 8, 1, 7], 0, 2)</code>	[1, 8, 6, 7]
<code>stooge_sort([1, 8, 6, 7], 0, 1)</code>	
<code>stooge_sort([1, 8, 6, 7], 1, 2)</code>	[1, 6, 8, 7]
<code>stooge_sort([1, 6, 8, 7], 0, 1)</code>	
<code>stooge_sort([1, 6, 8, 7], 1, 3)</code>	
<code>stooge_sort([1, 6, 8, 7], 1, 2)</code>	
<code>stooge_sort([1, 6, 8, 7], 2, 3)</code>	[1, 6, 7, 8]
<code>stooge_sort([1, 6, 7, 8], 1, 2)</code>	
<code>stooge_sort([1, 6, 7, 8], 0, 2)</code>	
<code>stooge_sort([1, 6, 7, 8], 0, 1)</code>	
<code>stooge_sort([1, 6, 7, 8], 1, 2)</code>	
<code>stooge_sort([1, 6, 7, 8], 0, 1)</code>	

- Q2)** Give the recurrence equation for the number of recursive calls performed by `stooge_sort(Tab, 0, n - 1)`, where **n** is the number of elements in **Tab**.

$$T(n) = \begin{cases} O(1) & \text{if } end - start + 1 \leq 2 \\ 3T\left(\frac{2n}{3}\right) + O(1) & \text{otherwise} \end{cases} \quad (1 \text{ pt})$$

- Q3)** Deduce the asymptotic complexity of sorting from the recurrence equation.

For this recurrence, we have $a = 3$, $b = 3/2$, and $f(n) = O(1) \Rightarrow c = 0$ (1 pt)

Since $a > b^c$, $[3 > (3/2)^0]$

As per **case 1** of the master theorem, the solution is : $T(n) = O(n^{\log_{3/2}(3)}) = O(n^{2.7095})$

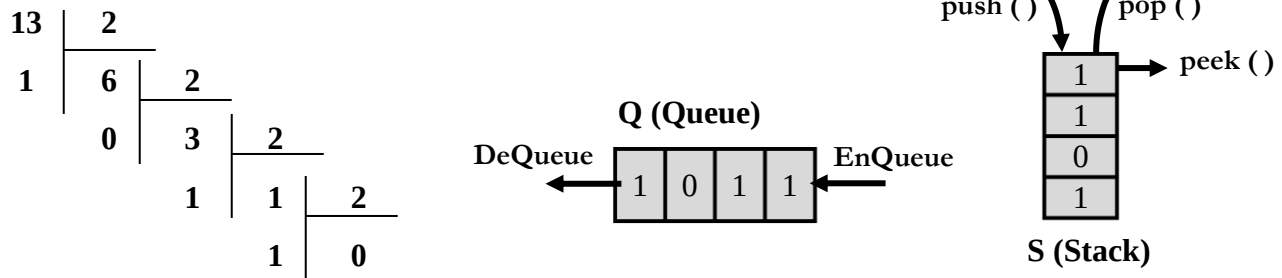
- Q4)** What should we think about the complexity of this sorting method compared to Bubble sort and Quick sort?

The asymptotic complexity of the stooge sort is **worse than** that of Bubble sort ($O(n^2)$) also worse than that of Quick sort in the worst case (also ($O(n^2)$)). (1 pt)

Therefore, Stooge Sort is a **particularly inefficient sorting algorithm**.

Exercise 4 : (4 Points / 25 Minutes)

In the example below, we have a **queue Q** and a **stack S**, each containing the binary representation of the integer number 13 (1101).



Q1) Provide the declaration for this queue and this stack. (0.25 + 0.25 + 0.25 = 0.75 pts)

Node	Queue	Stack
<pre> struct Node { bool val; struct Node* next; }; </pre>	<pre> typedef struct { struct Node *first; struct Node *last; int size; } QUEUE; </pre>	<pre> typedef struct { struct Node *top; int size; } STACK; </pre>

In all questions, you can use all functions seen in course without needing to redefine them.

Q2) Write the function **binary(x, Q)** which receives a decimal number **x** and creates a queue **Q** containing its binary representation. (1 pt)

```

void binary(int x, QUEUE *Q) {
    if (x == 0) { // Handle special case x = 0
        EnQueue(Q, 0);
        return;
    }
    while (x > 0) {
        EnQueue(Q, x % 2);
        x = x / 2;
    }
}

```

Q3) Write a **Stack_Reverse(S)** that destructively returns a stack with the same elements as its input stack but in reverse order. (0.75 pts)

```

STACK Stack_Reverse (STACK S) {
    STACK Reverse = init_Stack();           // NULL
    while (!isEmpty(S)) {
        push(Reverse, pop(S));
    }
    return Reverse;
}

```

Q4) Write the function **equal(Q, S)** that makes it possible to verify whether the queue **Q** and the stack **S** contain binary representations of the same integer number. (1.5 pts)

```

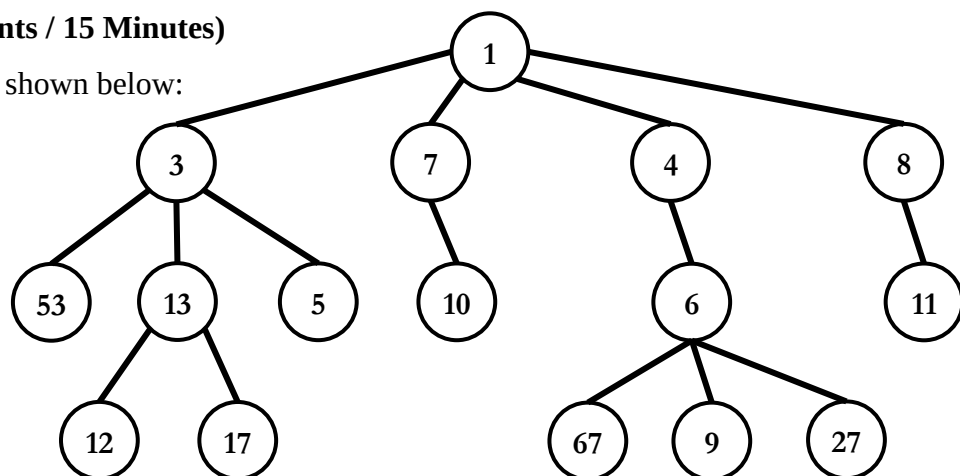
bool equal(Queue Q, STACK S) {
    if (Q.size != S.size)           // Sizes must be equal
        return false;

    STACK temp = Stack_Reverse(S);
    while (Q.size > 0 && S.size > 0) {
        if (DeQueue(Q) != pop(S))   // Compare each bit
            return false;
    }
    return true;
}

```

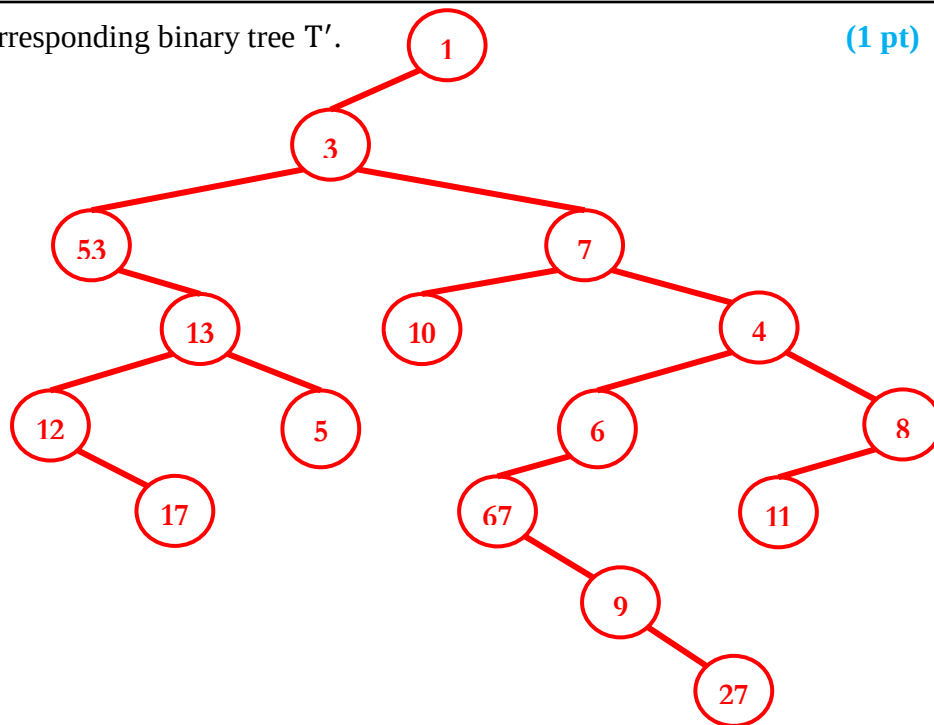
Exercise 5 : (4.5 Points / 15 Minutes)

For the general tree **T** shown below:



Q1) Find the corresponding binary tree T' .

(1 pt)

Q2) In the binary tree T' , write the pre-order, in-order and post-order traversals of each one.

(0.5 + 0.5 + 0.5 = 1.5 pts)

Pre-order	1 → 3 → 53 → 13 → 12 → 17 → 5 → 7 → 10 → 4 → 6 → 67 → 9 → 27 → 8 → 11
In-order	53 → 12 → 17 → 13 → 5 → 3 → 10 → 7 → 67 → 9 → 27 → 6 → 4 → 11 → 8 → 1
Post-order	17 → 12 → 5 → 13 → 53 → 10 → 27 → 9 → 67 → 6 → 11 → 8 → 4 → 7 → 3 → 1

Q3) Is the T' a binary search tree? Why? (0.5 pts)

No, T' is **not a binary search tree**, because the node with value 53 is in the **left subtree of node 3**. Which violates the **BST property**.

A node in a binary tree is defined as follows:

```

struct BTreeNode {
    int data;
    BTreeNode *left;
    BTreeNode *right;
};

typedef BTreeNode *BTREE;
  
```

Q4) Write a **Height(BTree)** function to find a height of any node in the binary tree.

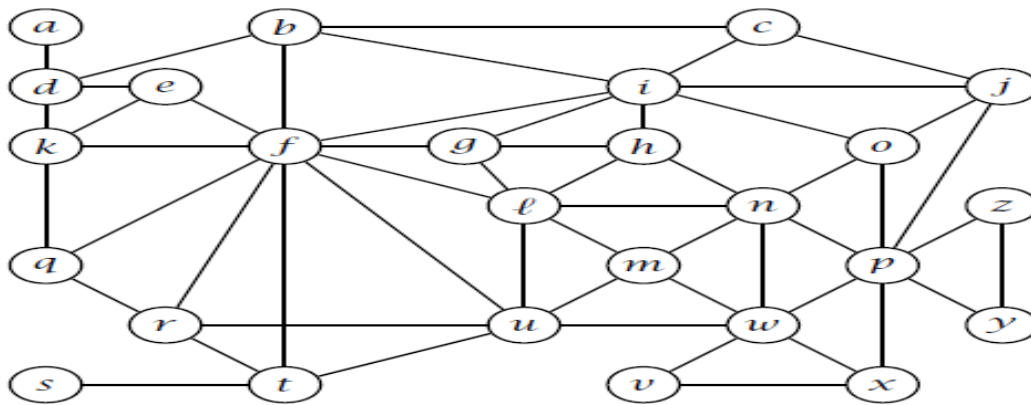
```

int Height(BTREE BTree) {
    if (BTree == NULL)
        return -1;
    else
        if ((BTree ->left== NULL) && (BTree ->right== NULL))
            return 0;
        return max(Height(BTree->left), Height(BTree->right)) + 1;
}

```

Exercise 6 : (2 Points / 10 Minutes)

Consider the following graph. Suppose we want to traverse it, starting at node **g**.



Q1) If we traverse this using **breadth-first search (BFS)**, what is a possible order of the nodes we visit? When processing the neighbors of a vertex, process them in alphabetical order.

(1.5 pts)

g→**f**→**h**→**i**→**l**→**b**→**e**→**k**→**q**→**r**→**t**→**u**→**n**→**c**→**j**→**o**→**m**→**d**→**s**→**w**→**p**→**a**→**v**→**x**→**y**→**z**

Q2) Write the structure of the graph to represent it using the adjacency list. (0.5 pts)

```

struct GNode{
    int vertex;
    GNode* next;
};
// Structure to represent the graph
struct Graph {
    unsigned int numVertices;
    GNode** adjLists;
    bool isDirected;
};

```

Good Luck