**PAST/SAMPLE QUESTIONS**

**Question 1**

(a) What is recursion?  
(b) What are the three requirements for successful recursion in C++?

(c) The **fibonacci** function is defined mathematically as:  
Fib(n) = Fib(n − 1) + Fib(n − 2), Fib(1) = Fib(0) = 1, n >= 2. Write a C++ function to implement this with the signature int fibonacci(int n).

(d) We often refer to recursion being less efficient than an iterative solution. Why is the recursive Fibonacci less efficient than its iterative counterpart? Provide a diagram to support your discussion.

**Question 2**

(a) What is the complexity of the following function, **f**? You must fully justify your answer. You may assume that the function g has a complexity of O(n).

int f(int N)

{

int result, i=0;

if (N <= 0)

result = g(N);

else

for (i=0; i < N; i++)

result = result + g(result)

return result;

}

(b) Give a brief explanation why the complexity of Binary Search is O(log(N)).

(c) Consider the following statement: “A function with a complexity of O(n) will always be faster then one with complexity O(n2)”. Is this statement true or false? Clearly explain your answer.

**Question 3**

(a)  Explain the terms *upper bounds* and *lower bounds* in terms of an algorithm.

(b)  Why are we interested in the bounds of an algorithm?

(c)  Give the complexity for the following functions:

i. f(n)=n3 +n+3

ii. f(n)=3

(d)  What is the complexity of the following function? Explain your answer.

int getHandshakes (int number){

int handshakes = 0;

for (int i = 0; i < number; i++){

for (int j = i+1; j < number; j++){

handshakes++;

} }

return handshakes;

}

(e)  Binary Search is a more efficient way to search *sorted* data.

i. Write C++ code for a function with the signature

int binarysearch(int [] myArr, int target, int size)  
that implements Binary Search, searching for element target and returning its position, if found, -1 otherwise.  
You may assume that size contains the size of the array myArr.

ii. What is the complexity of Binary Search.

**Question 4**

(a) Define a linked list containing n nodes as follows:

struct Node {

int data;

Node \*link;

}

What is the complexity for deleting a node at the end of the linked list? Please also provide the pseudo-code.

(b)  Stacks and Queues are often implemented based on linked lists. i. What is a stack?

ii. What is a queue?  
iii. What are the common operations of the stack?

(c)  What is the complexity for adding a node in the middle of the linked list?

(d)  A binary tree is a tree with at most 2 child nodes.

i. Please provide the C++ code for a node in a binary tree.

ii. An important application of binary trees is the use in searching. Please pro- vide pseudo code for the search function in a binary search tree.

**Question 5**

Define a tree node as follows:

struct Node {

int data;

Node \*left;

Node \*right;

}

(a) What is a binary search tree?

(b) Starting with an empty tree, show the process of adding the list{3,6,1,2,5,4} (in order) to the tree.

(c) Write a function bool search(struct Node \*root, int obj) that takes as input a binary search tree root and a value of obj. The function re- turns whether obj exists in the tree or not.

**Question 6**

Define a linked list containing n nodes as follows:

struct Node {

int data;

Node \*link;

}

(a)  Stacks and Queues are often implemented based on linked lists. i. What is a stack?

ii. What is a queue?

iii. What does FIFO represent in the context of algorithms and data structures?

(b)  How do you use singly linked list to efficiently implement a stack?

(c)  How do you use singly linked list to efficiently implement a queue?

(d)  Given a singly linked list, if you have access to a pointer that points to the middle of the linked list (besides the head pointer and the tail pointer), then what is the complexity for deleting a node at the end of the linked list?

**Question 7**

Text

Description automatically generated

**Question 8**

Text, letter

Description automatically generated

Q3 (b)

Upper and lower bounds have to do with the minimum and maximum "complexity" of an algorithm (I use that word advisedly since it has a very specific meaning in complexity analysis).

Take, for example, our old friend, the bubble sort. In an ideal case where all the data are already sorted, the time taken is f(n), a function dependent on n, the number of items in the list. That's because you only have to make one pass of the data set (with zero swaps) to ensure your list is sorted.

In a particularly bad case where the data are sorted in the opposite to the order you want, the time taken becomes f(n2). This is because each pass moves one element to the right position and you need n passes to do all elements.

In that case, the upper and lower bounds are different, even though the big-O complexity remains the same.

As an aside, the bubble sort is much maligned (usually for good reasons) but it can make sense in certain circumstances. I actually use it in an application where the bulk of the data are already sorted and only one or two items tend to be added at a time to the end of the list. For adding one item, and with a reverse-directional bubble sort, you can guarantee the new list will be sorted in one pass. That illustrates the lower bound concept.

In fact, you could make an optimization of the bubble sort that sets the lower bound to f(1), simply by providing an extra datum which indicates whether the list is sorted. You would set this after sorting and clear it when adding an item to the end.