(Answer Script)

Module 7: Theory Assignment 01

Basic Data Structures and Problem Solving Part-II

(nayeem.cse6.bu@gmail.com)

Question No. 01

Write down all the steps of Bubble Sort on the Following Array.

Index	0	1	2	3	4	5
Value	7	2	13	2	11	4

For example:

1st iteration:

1st step: **7 2** 13 2 11 4 -> 2 7 13 2 11 4
2nd step: 2 **7 13** 2 11 4 -> 2 7 13 2 11 4
3rd step: 2 7 **13 2** 11 4 -> 2 7 2 13 11 4
……..

Answer No. 01

Written below all the steps of Bubble Sort on the given array:

1st iteration:

1st step: **7 2** 13 2 11 4 -> 2 7 13 2 11 4
2nd step: 2 **7 13** 2 11 4 -> 2 7 13 2 11 4
3rd step: 2 7 **13 2** 11 4 -> 2 7 2 13 11 4
4th step: 2 7 2 **13 11** 4 -> 2 7 2 11 13 4
5th step: 2 7 2 11 **13 4** -> 2 7 2 11 4 13

2nd iteration:

1st step: **2 7** 2 11 4 13 -> 2 7 2 11 4 13
2nd step: 2 **7 2** 11 4 13 -> 2 2 7 11 4 13
3rd step: 2 2 **7 11** 4 13 -> 2 2 7 11 4 13
4th step: 2 2 7 **11 4** 13 -> 2 2 7 4 11 13
5th step: 2 2 7 4 **11 13** -> 2 2 7 4 11 13

3rd iteration:

1st step: **2 2** 7 4 11 13 -> 2 2 7 4 11 13
2nd step: 2 **2 7** 4 11 13 -> 2 2 7 4 11 13
3rd step: 2 2 **7 4** 11 13 -> 2 2 4 7 11 13
4th step: 2 2 4 **7 11** 13 -> 2 2 4 7 11 13
5th step: 2 2 4 7 **11 13** -> 2 2 4 7 11 13

[The array is now sorted. So, the below iterations are not needed for an efficient algorithm.]

4th iteration:

1st step: **2 2** 4 7 11 13 -> 2 2 4 7 11 13
2nd step: 2 **2 4** 7 11 13 -> 2 2 4 7 11 13
3rd step: 2 2 **4 7** 11 13 -> 2 2 4 7 11 13
4th step: 2 2 4 **7 11** 13 -> 2 2 4 7 11 13
5th step: 2 2 4 7 **11 13** -> 2 2 4 7 11 13

5th iteration:

1st step: **2 2** 4 7 11 13 -> 2 2 4 7 11 13
2nd step: 2 **2 4** 7 11 13 -> 2 2 4 7 11 13
3rd step: 2 2 **4 7** 11 13 -> 2 2 4 7 11 13
4th step: 2 2 4 **7 11** 13 -> 2 2 4 7 11 13
5th step: 2 2 4 7 **11 13** -> 2 2 4 7 11 13

So, the sorted array is: 2 2 4 7 11 13

Question No. 02

Write down two differences between array and vector in C++.

Answer No. 02

Written below the two differences between array and vector in C++:

Array	Vector
Array stores a fixed-size sequential collection of elements of the same type and it is index based.	Vector is dynamic in nature so size increases with insertion of elements.
Array is available in both C and C++	Vector is available in only C++

Question No. 03

Write down the time complexity with proper explanation of the following code segment.

```
for(int i=1;i<=n;i++)
{
        if(builtin_popcount(i) == 2)
        {
                for(int j=1;j<=n;j++)
                        cout<<i<<j<<endl;
        }
}
```

Note: builtin_popcount(i) returns the number of set bits in 'i'.
For example builtin_popcount(5) = 2. Because, 5 = $(101)_2$. So there are 2 set bits in 5.

Answer No. 03

The time complexity of the above code segment is O(n^2).

The outer loop runs n times and the inner loop also runs n times, so the total number of iterations of the inner loop is n * n = n^2.

The function "builtin_popcount(i)" takes a constant time to execute because it

simply counts the number of set bits in the given number. Therefore, the time complexity of the entire code segment is O(n^2).

Answer No. 04

Look at the following code which calculates the number of distinct elements. Are there any flaws in this code? If yes, write down the flaws with proper explanation.

```
#include<bits/stdc++.h>
using namespace std;
int main(){
        int n;
        cin>>n;
        vector<int>a(n);
        for(int i=0;i<n;i++)
                cin>>a[i];
        sort(a.begin(),a.end());
        int ans = 0;
        for(int i=0 ; i<=n ; i++)
                if(a[i]!=a[i-1])
                        ans++;
        cout<<ans;
        return 0;
}
```

Answer No. 04

There are some flaws in the given code:

Flaw-1:

The loop should run from 0 to n-1, not from 0 to n. This is because the size of the vector "a" is n and the valid indices for the vector are 0 to n-1. Running the loop from 0 to n will cause the program to access an element out of the bounds of the vector, leading to undefined behavior.

Flaw-2:

The initial value of the variable "ans" should be 1, not 0. This is because the first

element in the sorted vector is always a distinct element. If the initial value is 0, the code will not count the first element as a distinct element.

Flaw-3:

The condition in the if statement should be "a[i] != a[i+1]" instead of "a[i] != a[i-1]".

This is because the code is comparing the current element with the next element to check if they are distinct. If the condition is "a[i] != a[i-1]", the code will compare the current element with the previous element, which is not what we want.

Here is the modified code:

```
#include<bits/stdc++.h>
using namespace std;
int main(){
int n;
cin>>n;
vector<int>a(n);
for(int i=0;i<n;i++)
cin>>a[i];
sort(a.begin(),a.end());
int ans = 1;
for(int i=0 ; i<n-1 ; i++)
if(a[i]!=a[i+1])
ans++;
cout<<ans;
return 0;
}
```

Question No. 05

Write down the time and space complexity with proper explanation of the following code segment.

```
vector<int>d[n+1];
```

```
for(int i=1 ; i<=n ; i++)
        for(int j=i ; j<=n ; j = j+i )
                d[j].push_back(i);
```

Answer No. 05

Time Complexity:

The time complexity of the above code segment is O(n^2).

There are two nested loops and the time complexity of each loop is O(n). So, the time complexity of the code segment is O(n * n) = O(n^2).

```
vector<int>d[n+1];
for(int i=1 ; i<=n ; i++) // O(n)
     for(int j=i ; j<=n ; j = j+i ) // O(n)
          d[j].push_back(i);
```

Space Complexity:

The space complexity of the above code segment is O(n).

The space complexity is determined by the size of the data structures used in the code. In this case, the space complexity is determined by the size of the array d. The size of the array is n+1 because it has n+1 elements. So, the space complexity of the code segment is O(n).

```
vector<int>d[n+1]; // O(n + 1) = O(n)
for(int i=1 ; i<=n ; i++) // O(n)
     for(int j=i ; j<=n ; j = j+i ) // O(n)
          d[j].push_back(i);
```

Question No. 06

Fill up the following table with ‘YES’ or ‘NO’ in each cell in the context of public, private and protected access modifiers in C++ Class. First cell is already filled up for your convenience.

Name	Accessibility from own class	Accessibility from derived class	Accessibility from world
Public	YES		
Private			
Protected			

Answer No. 06

Filling up the given table with ‘YES’ or ‘NO’ in each cell in the context of public, private and protected access modifiers in C++ Class:

Name	Accessibility from own class	Accessibility from derived class	Accessibility from world
Public	YES	YES	YES
Private	YES	NO	NO
Protected	YES	YES	NO

Question No. 07

What is ‘new’ and ‘delete’ in C++.

Answer No. 07

In C++, the new and delete operators are used for dynamic memory allocation and deallocation, respectively.

'new' operator:

The 'new' operator is used to allocate memory at runtime for variables, arrays, and objects. It returns a pointer to the memory location it has allocated.

For example:

```
int* p = new int; // Allocates memory for an integer and returns a pointer to it.

int* arr = new int[55]; // Allocates memory for an array of 55 integers and returns a pointer to the first element.
```

'delete' operator:

The delete operator is used to deallocate the memory that was previously allocated using the new operator. It releases the memory back to the system so that it can be used by other programs.

For example:

```
delete p; // Deallocates the memory pointed to by p.

delete[] arr; // Deallocates the memory pointed to by arr.
```

It is important to use delete to deallocate the memory that was allocated using new, otherwise it can lead to memory leaks. Memory leaks occur when dynamically allocated memory is not deallocated, which can lead to insufficient available memory for the program to function correctly.

Question No. 08

Alice wrote a new algorithm which works in $O(n^3)$ where n can be atmost 10^6. Bob told Alice that it will take years to finish in the worst case. Do you agree with Bob? If yes,

then approximately how many years will it take to finish? Assume Alice's computer can run 10^9 instructions in 1 second.

Answer No. 08

Yes, I agree with Bob that it will take a long time for Alice's algorithm to finish in the worst case.

The time complexity of Alice's algorithm is O(n^3), which means that the running time will increase rapidly as the input size n increases.

In the worst case, when n is 10^6,

(10^6)^3
= 10^18 operations
= 10^18 / 10^9 [Alice's computer can run 10^9 operations in 1 sec]
= 10^19

So, the running time will be approximately ((10^6)^3)/(10^9) = approximately 1,000 years.

Question No. 09

Write down two differences between binary search and linear search.

Answer No. 09

Two differences between binary search and linear search:

Binary Search	Linear Search
Binary search is a faster searching algorithm than linear search, but it has a requirement that the data being	Linear search does not have this requirement and can be used on unsorted data.

searched must be sorted.	
The time complexity of binary search is O(log n) in the worst case, which means that the running time increases slowly as the input size n increases.	In contrast, the time complexity of linear search is O(n) in the worst case, which means that the running time increases linearly as the input size n increases.

Question No. 10

Suppose you wrote a code for a server which contains the following function.

```
void func()
{
    int* p = new int;
    return;
}
```

Are there any flaws in the function? If yes, then explain the flaw and modify the function so that there is no flaw in the function. You cannot delete any lines of code from the function. You are only allowed to write your own code to overcome the flaw.

Answer No. 10

There is a flaw in the function. The flaw is that the dynamically allocated memory pointed to by p is not deallocated before the function returns. This can lead to a memory leak.

To fix this flaw, the memory pointed to by p should be deallocated using the delete operator before the function returns.

Here is the modified function:

```
void func()
{
int* p = new int;
delete p; // modified
return;
}
```

This modified function will deallocate the memory pointed to by p before the function returns, preventing a memory leak.