Answer Script

Question No. 01

1. Write a program to reverse an array.

10

Sample input	Sample output
5 6 2 3 3 5	5 3 3 2 6

Answer No. 01

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];

    for(int i=n-1; i>=0; i--)
        cout<<a[i]<<" ";

    return 0;
}
```

Question No. 02

2. Write a program to remove duplicate numbers from an array and print the remaining elements in sorted order. You have to do this in O(nlogn). 15

Sample input	**Sample output**
5 6 3 2 3 5	2 3 5 6

Answer No. 02

CODE:

```
#include<bits/stdc++.h>
using namespace std;

vector<int> merge_sort(vector<int> a)
{
    if(a.size()<=1)
        return a;

    int mid = a.size()/2;

    vector<int> b;
    vector<int> c;
    for(int i=0; i<mid; i++)
        b.push_back(a[i]);

    for(int i=mid; i<a.size(); i++)
        c.push_back(a[i]);

    vector<int> sorted_b = merge_sort(b);
    vector<int> sorted_c = merge_sort(c);

    vector<int> sorted_a;
    int idx1 = 0;
    int idx2 = 0;

    for(int i=0; i<a.size(); i++)
    {
        if(idx1 == sorted_b.size())
        {
            sorted_a.push_back(sorted_c[idx2]);
            idx2++;
        }
        else if(idx2 == sorted_c.size())
        {
            sorted_a.push_back(sorted_b[idx1]);
            idx1++;
        }
        else if(sorted_b[idx1] < sorted_c[idx2])
        {
            sorted_a.push_back(sorted_b[idx1]);
            idx1++;
        }
        else
        {
            sorted_a.push_back(sorted_c[idx2]);
            idx2++;
        }
    }

    return sorted_a;

}

int main()
{
    int n;
    cin>>n;
    vector<int> a(n);
    for(int i=0; i<n; i++)
        cin>>a[i];

    vector<int> ans = merge_sort(a);
    int j = 0;
    for(int i=1; i<n; i++)
    {
        if(ans[i] != ans[j])
        {
            ans[++j] = ans[i];
        }
    }

    for(int i=0; i<=j; i++)
    {
        cout<<ans[i]<<" ";
    }
    return 0;
}
```

Question No. 03

3. Write a program to sort the numbers in non-increasing order using quick sort. You have to take random index as a pivot element.

15

Sample input	Sample output
5 6 3 2 3 5	6 5 3 3 2

Answer No. 03

CODE:

```
#include<bits/stdc++.h>
using namespace std;

vector<int> quick_sort(vector<int>&a)
{
   if(a.size()<=1)
      return a;

   int pivot = rand()%(a.size());
   vector<int> b, c;

   for(int i=0; i<a.size(); i++)
   {
      if(i == pivot)
         continue;
      if(a[i]<=a[pivot])
         b.push_back(a[i]);
      else
         c.push_back(a[i]);
   }

   vector<int>sorted_b = quick_sort(b);
   vector<int>sorted_c = quick_sort(c);

   vector<int> sorted_a;
   for(int i=0; i<sorted_b.size(); i++)
      sorted_a.push_back(sorted_b[i]);

   sorted_a.push_back(a[pivot]);

   for(int i=0; i<sorted_c.size(); i++)
```

```
        sorted_a.push_back(sorted_c[i]);

    return sorted_a;
}

int main()
{
    int n;
    cin>>n;

    vector<int> a(n);
    for(int i=0; i<n; i++)
        cin>>a[i];

    vector<int> sorted_a = quick_sort(a);
    for(int i=sorted_a.size()-1; i>=0; i--)
    {
        cout<<sorted_a[i]<<" ";
    }
    return 0;
}
```

Question No. 04

4. Write a recursive function to check if a given word is a palindrome.
15

Sample input	Sample output
abcba	Yes
abcaa	No

A palindrome is a word which reads the same forward and backward.

Answer No. 04

CODE:

```
#include<bits/stdc++.h>
using namespace std;

bool isPalindrome(string s, int start, int end)
{

   if (start >= end)
   {
      return true;
   }

   if (s[start] != s[end])
   {
      return false;
   }

   return isPalindrome(s, start + 1, end - 1);
}

int main()
{
   string s;
   cin >> s;

   if (isPalindrome(s, 0, s.length() - 1))
      cout<< "Yes";
   else
      cout<<"No";

   return 0;
}
```

Question No. 05

5. Write a recursive function to find the maximum element in an array.

15

Sample input	Sample output
5 1 3 5 2 4	5

Answer No. 05

CODE:

```
#include <bits/stdc++.h>
using namespace std;

int findMax(vector<int> &a, int n)
{
   if (n == 1)
   {
      return a[0];
   }
   int max = findMax(a, n - 1);
   if(a[n-1]>max)
      return a[n-1];
   else
      return max;
}

int main()
{
   int n;

   cin>>n;
   vector<int> a(n);

   for(int i = 0; i < n; i++)
   {
      cin>>a[i];
   }

   cout<<findMax(a, n)<<"\n";
   return 0;
}
```

Question No. 06

6. Take the Singly linked-list class from Github.
   15
   Link: https://github.com/phitronio/Data-Structure-Batch2/blob/main/Week%204/Module%2013/1.cpp

Add the following functions to the class.

- **int getLast()** -> This function will return the last node of the linked list. If the linked list is empty then return -1.
  Sample Input: [3, 2, 6, 4, 5]
  Sample Output: 5
- **double getAverage()** -> This function will return the average of all elements in the linked list.
  Sample Input: [3, 2, 6, 4, 7]
  Sample Output: 4.4

Answer No. 06

CODE:

```
#include<bits/stdc++.h>

using namespace std;

class node
{
public:
   int data;
   node * nxt;
};

class LinkedList
{
public:
   node * head;
   int sz;
   LinkedList()
   {
      head = NULL;
      sz=0;
   }
```

```
//Creates a new node with data = value and nxt= NULL
node* CreateNewNode(int value)
{
    node *newnode = new node;
    newnode->data = value;
    newnode->nxt = NULL;
    return newnode;
}

// Insert new value at Head
void InsertAtHead(int value)
{
    sz++;
    node *a = CreateNewNode(value);
    if(head == NULL)
    {
        head = a;
        return;
    }
    //If head is not NULL
    a->nxt = head;
    head = a;
}

//Prints the linked list
void Traverse()
{
    node* a = head;
    while(a!= NULL)
    {
        cout<<a->data<<" ";
        a = a->nxt;
    }
    cout<<"\n";
}

//Search for a single value
int SearchDistinctValue(int value)
{
    node* a = head;
    int index = 0;
    while(a!= NULL)
    {
        if(a->data==value)
        {
                return index;
            }
            a = a->nxt;
            index++;
        }
        return -1;
    }

    //Search all possible occurrence
    void SearchAllValue(int value)
    {
        node* a = head;
        int index = 0;
        while(a!= NULL)
        {
            if(a->data==value)
            {
                cout<<value<<" is found at index "<<index<<"\n";
            }
            a = a->nxt;
            index++;
        }
    }

    //Returns number of elements in the linked list
    int getSize()
    {
        //O(1)
        return sz;


        //O(size of linked list) = O(n)
//      int sz = 0;
//      node *a = head;
//      while(a!=NULL)
//      {
//          sz++;
//          a = a->nxt;
//      }
//      return sz;
    }

    //Insert a value at the given index
    void InsertAtAnyIndex(int index, int value)
    {
```

```
    if(index <0 || index > sz)
    {
        return;
    }
    if(index==0)
    {
        InsertAtHead(value);
        return;
    }
    sz++;
    node *a = head;
    int cur_index = 0;
    while(cur_index!=index-1)
    {
        a = a->nxt;
        cur_index++;
    }
    node *newnode = CreateNewNode(value);
    newnode->nxt = a->nxt;
    a->nxt = newnode;
}

//Delete the first element of a linked list
void DeleteAtHead()
{
    if(head == NULL)
    {
        return;
    }
    sz--;
    node *a = head;
    head = a->nxt;
    delete a;
}

//Delete the value at the given index
void DeleteAnyIndex(int index)
{
    if(index <0 || index > sz-1)
    {
        return;
    }
    if(index==0)
    {
        DeleteAtHead();
        return;
    }
    sz--;
    node *a = head;
    int cur_index = 0;
    while(cur_index != index-1)
    {
        a = a->nxt;
        cur_index++;
    }
    node *b = a->nxt;
    a->nxt = b->nxt;
    delete b;
}

void InsertAfterValue(int value, int data)
{
    node *a = head;
    while(a != NULL)
    {
        if(a->data == value)
        {
            break;
        }
        a = a->nxt;
    }
    if(a== NULL)
    {
        cout<<value<<" doesn't exist in linked-list.\n";
        return;
    }
    sz++;
    node *newnode = CreateNewNode(data);
    newnode->nxt = a->nxt;
    a->nxt = newnode;
}

//Print the Reverse Order from node a to last
void ReversePrint2(node *a)
{
    if(a==NULL)
    {
        return;
    }
    ReversePrint2(a->nxt);
```

```
        cout<<a->data<<" ";
    }
    void ReversePrint()
    {
        ReversePrint2(head);
        cout<<"\n";
    }

    int getLast()
    {
        if(sz==0)
            return -1;

        int lastValue;
        node* a = head;
        while(a!= NULL)
        {
            lastValue = a->data;
            a = a->nxt;
        }
        return lastValue;
    }

    double getAverage()
    {

        double total = 0;
        double avg;

        node* a = head;
        while(a!= NULL)
        {
            total += a->data;
            a = a->nxt;
        }
        avg = total/sz;
//      return total;
        return avg;
    }

};

int main()
{
    LinkedList l1;
```

```
   l1.InsertAtHead(5);
   l1.InsertAtHead(4);
   l1.InsertAtHead(6);
   l1.InsertAtHead(2);
   l1.InsertAtHead(3);
//    l.Traverse();
   cout<<l1.getLast()<<"\n";


   LinkedList l2;
   l2.InsertAtHead(7);
   l2.InsertAtHead(4);
   l2.InsertAtHead(6);
   l2.InsertAtHead(2);
   l2.InsertAtHead(3);

   cout<<l2.getAverage()<<"\n";

//    l.ReversePrint();
//    l.Traverse();
   return 0;
}
```

Question No. 07

7. Take the Doubly linked-list class from Github.
   15
   Link:
   https://github.com/phitronio/Data-Structure-Batch2/blob/main/Week%204/Module%2014/1.cpp

Add the following functions to the class.

- **void swap(i , j)** -> This function will swap the i-th index and j-th index.
  Sample Input: [3, 2, 6, 4, 7], i = 1, j = 4
  Sample Output: Doubly Linked list containing the elements [3,7,6,4,2]
- **void deleteZero()** -> This function will delete all the nodes that have data=0.
  Sample Input: [0, 2, 0, 0, 5]
  Sample Output: Doubly linked list containing the elements [2, 5]

Answer No. 07

CODE:

```
#include<bits/stdc++.h>

using namespace std;



class node
{
public:
   int data;
   node * nxt;
   node * prv;
};

class DoublyLinkedList
{
public:
   node *head;
   int sz;
   DoublyLinkedList()
   {
      head = NULL;
```

```
    sz = 0;
}

//Creates a new node with the given data and returns it O(1)
node * CreateNewNode(int data)
{
    node *newnode = new node;
    newnode->data = data;
    newnode->nxt = NULL;
    newnode->prv = NULL;
    return newnode;
}

//Inserts a node with given data at head O(1)
void InsertAtHead(int data)
{
    sz++;
    node *newnode = CreateNewNode(data);
    if(head == NULL)
    {
        head = newnode;
        return;
    }
    node *a = head;
    newnode->nxt = a;
    a->prv = newnode;
    head = newnode;
}

//Inserts the given data at the given index O(n)
void Insert(int index, int data)
{
    if(index > sz)
    {
        return;
    }
    if(index==0)
    {
        InsertAtHead(data);
        return;
    }
    node *a = head;
    int cur_index = 0;
    while(cur_index!= index-1)
    {
        a = a->nxt;
        cur_index++;
    }
    // a = cur_index - 1
    node *newnode = CreateNewNode(data);
    newnode->nxt = a->nxt;
    newnode->prv = a;
    node *b = a->nxt;
    b->prv = newnode;
    a->nxt = newnode;
    sz++;
}

//Deletes the given index O(n)
void Delete(int index)
{
    if(index >= sz)
    {
        cout<<index<<" doesn't exist.\n";
        return;
    }
    node *a = head;
    int cur_index = 0;
    while(cur_index != index)
    {
        a = a->nxt;
        cur_index++;
    }
    node *b = a->prv;
    node *c = a->nxt;
    if(b!=NULL)
    {
        b->nxt = c;
    }
    if(c!= NULL)
    {
        c->prv = b;
    }
    delete a;
    if(index==0)
    {
        head = c;
    }
    sz--;
}
```

```
    //Prints the linked list O(n)
    void Traverse()
    {
       node *a = head;
       while(a!=NULL)
       {
          cout<<a->data<<" ";
          a = a->nxt;
       }
       cout<<"\n";
    }

    // Returns the size of linked list O(1)
    int getSize()
    {
       return sz;
    }

    //Reverse the doubly linked list O(n)
//   void Reverse()
//   {
//      if(head==NULL)
//      {
//         return;
//      }
//      node *a = head;
//      int cur_index = 0;
//      while(cur_index != sz-1)
//      {
//         a = a->nxt;
//         cur_index++;
//      }
//      // last index is in a
//
//      node *b = head;
//      while(b!= NULL)
//      {
//         swap(b->nxt, b->prv);
//         b = b->prv;
//      }
//      head = a;
//   }
```

```
void swap(int i, int j)
{
    if(i == j || head == NULL || head->nxt == NULL)
        return;

    node* iTh = head;
    node* jTh = head;

    int iIndex = 0;
    while(iIndex != i)
    {
        iTh = iTh->nxt;
        iIndex++;
    }

    int jIndex = 0;
    while(jIndex != j)
    {
        jTh = jTh->nxt;
        jIndex++;
    }

    int temp;
    temp = iTh->data;
    iTh->data = jTh->data;
    jTh->data = temp;

}

void deleteZero()
{
    node* a = head;
    node* nxt;

    int i = 0;
    while(a != NULL)
    {
        if(a->data == 0)
        {
            nxt = a->nxt;
            Delete(i);
            a = nxt;
            i--;
        }
        else
```

```
      {
        a = a->nxt;
      }
      i++;
    }
  }
};


int main()
{
  DoublyLinkedList dl;
//   dl.InsertAtHead(10);
//   dl.InsertAtHead(5);
//   dl.InsertAtHead(1);
//   dl.Traverse();
//   dl.Insert(2,100);
//   dl.Traverse();

//   dl.Reverse();
//   dl.Traverse();

/// SWAP FUNCTION CALL
  dl.InsertAtHead(7);
  dl.InsertAtHead(4);
  dl.InsertAtHead(6);
  dl.InsertAtHead(2);
  dl.InsertAtHead(3);

  dl.swap(1, 4);
  dl.Traverse();

/// DELETE ZERO FUNCTION CALL
  DoublyLinkedList dl1;
  dl1.InsertAtHead(5);
  dl1.InsertAtHead(0);
  dl1.InsertAtHead(0);
  dl1.InsertAtHead(2);
  dl1.InsertAtHead(0);

  dl1.deleteZero();
  dl1.Traverse();

  return 0;
}
```