Answer Script

Basic Data Structures and Problem Solving Part-II

Week-5, Module 19: Lab Mid Term Exam

(nayeem.cse6.bu@gmail.com)

Question No. 01

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

C++ Program to Reverse an Array:

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

using namespace std;

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

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

   reverse(v.begin(), v.end()); // built in function to reverse the
array

   for(int i = 0; i < n; i++) {
    cout << v[i] << " ";
   }
}
```

Question No. 02

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

C++ Program to remove the duplicate numbers from an array and printing the remaining elements in sorted order:

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

using namespace std;

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

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

  set<int> s; // built in function to store the unique elements
  for(int i = 0; i < n; i++) {
     s.insert(v[i]);
  }
```

```
  for(auto i : s) {
   cout << i << " ";
  }
}
```

Question No. 03

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

C++ Program to sort the numbers in non-increasing order using quick sort:

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

using namespace std;

// function of quick sort
vector<int> quick_sort(vector<int>v) {
    if(v.size() <= 1) {
        return v;
    }

    int pivot = rand() % v.size(); // random index as a pivot element

    vector<int> a, b;

    // dividing the v vector and putting into a and b vector
```

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

    // sorting the divided vectors by recursion
    vector<int> sorted_a = quick_sort(a);
    vector<int> sorted_b = quick_sort(b);

    vector<int> sorted_v; // to merge the divided vectors

    // merging the divided vectors: sorted array + pivot + sorted
array
    for(int i = 0; i < a.size(); i++) {
        sorted_v.push_back(sorted_a[i]);
    }

    sorted_v.push_back(v[pivot]);

    for(int i = 0; i < b.size(); i++) {
        sorted_v.push_back(sorted_b[i]);
    }

    return sorted_v;
}

// driver code
int main() {

    int n;
    cin >> n;
```

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

    vector<int> ans = quick_sort(v);

    for(int i = 0; i < ans.size(); i++) {
        cout << ans[i] << " ";
    }

return 0;
}
```

Question No. 04

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

Recursive function to check the given word palindrome or not:

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

using namespace std;

// recursive function to check the word palindrome or not
bool is_palindrome(string word) {
```

```
    if(word.size() <= 1) {
        return true;
    }
    else if(word[0] != word[word.size()-1]) {
        return false;
    }
    else {
        return is_palindrome(word.substr(1, word.size()-1));
    }
}

// driver code
int main() {

    string word;
    cin >> word;

    if(is_palindrome(word)) {
        cout << "YES\n";
    }
    else {
        cout << "NO\n";
    }

}
```

Question No. 05

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

Recursive Function to find the maximum element in the given array:

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

using namespace std;

// recursive function to find the maximum element from the given array
int mx_element(vector<int> v, int n) {
    if(n == 1) {
        return v[0];
    }
    int mx = mx_element(v, n-1);
    if(v[n - 1] > mx) {
        return v[n - 1];
    }
    else {
        return mx;
    }
}

// driver code
int main() {

  int n;
  cin >> n;

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

  cout << mx_element(v, n);

}
```


Question No. 06
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

Added the given functions to the singly linked list class from github:

```
#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";
    }

    // return the last node of the linked list
    int getLast() {
        node* a = head;
        if(sz == 0) {
            return -1;
        }
        while(a->nxt != NULL) {
            a = a->nxt;
        }
        return a->data;
    }


    // return the average of all elements in the linked list
    double getAverage() {
        node* a = head;
        double sum = 0;
        while(a != NULL) {
            sum += a->data;
            a = a->nxt;
        }
        return sum / sz;
    }

};

int main()
{
    LinkedList l;
    l.InsertAtHead(3);
```

```
    l.InsertAtHead(2);
    l.InsertAtHead(6);
    l.InsertAtHead(4);
    l.InsertAtHead(7);
    l.Traverse();

    l.ReversePrint();
    l.Traverse();

    cout << "The Last Element: " << l.getLast() << "\n";
    cout << "The Average: " << l.getAverage() << "\n";
    return 0;
}
```

Question No. 07
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
Added the given functions to the doubly linked list class from github: `#include<bits/stdc++.h>`

```
using namespace std;



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

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

    //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;
}

// swap the i-th index and j-th index
void SWAP(int i, int j)
{
    node* a = head;
    node* b = head;

    if(i == j)
    {
        return;
    }

    int idx = 0;
    while(a != NULL && idx != i)
    {
        a = a->nxt;
        idx++;
    }
    idx = 0;
    while(b != NULL && idx != j)
    {
        b = b->nxt;
        idx++;
    }

    if(a == NULL && b == NULL)
    {
        return;
    }

    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}
```

```
    // delete all the nodes that have data=0
    void deleteZero()
    {
        node* current = head;
        while (current != NULL)
        {
            if (current->data == 0)
            {
                if (current == head)
                {
                    head = current->nxt;
                    head->prv = NULL;
                }
                else if (current == tail)
                {
                    tail = current->prv;
                    tail->nxt = NULL;
                }
                else
                {
                    current->prv->nxt = current->nxt;
                    current->nxt->prv = current->prv;
                }
            }
            current = current->nxt;
        }
    }
};


int main()
{
    DoublyLinkedList dl;
    dl.InsertAtHead(7);
    dl.InsertAtHead(0);
    dl.InsertAtHead(6);
    dl.InsertAtHead(0);
```

```
    dl.InsertAtHead(3);

    dl.Traverse();

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

    dl.deleteZero();
    dl.Traverse();

    return 0;
}
```