Lahore University of Management Sciences (LUMS)

CS202/EE202: Data Structures (Spring’16)

Homework-1

Name : *Muneeb ZIA*

Roll No: 18100266

**Question No.1:**

The running time complexity of this algorithm will be log base 3 N. big theta(log3N) where 3 is the base. It is obvious as we divide our problem into 3 halves recursively at each iteration. So if we draw a tree, it will be 3-aray. Every level will contain 3^i nodes,where i is the ith level.

**Question No.2:**

**Part a:**

The code is given below

**void check\_cycle(Node\* A)**

**{**

**Node\* temp=A;**

**Node \* temp2=A;**

**while (temp2!=NULL)**

**{**

**if (temp2->next==temp)**

**{**

**cout << "list contains cycle";**

**break;**

**}**

**temp2=temp2->next;**

**}**

**}**

**Part b:**

The code is given below, I am considering first node with position 1 and second with position 2 and so on

void move\_node(int from,int to)

{

Node\* temp=head;

Node\*temp2=head;

Node\* prev;

Node\*prev2;

for (int i=0;i<from-1;i++)

{

prev=temp;

temp=temp->next

}

for (int i=0;i<to-1;i++)

{

prev2=temp2;

temp2=temp2->next;

}

prev->next=temp->next;

prev2->next=temp;

temp->next=temp2;

}

**PART C:**

We can do this in one single traversal through linked if we first sort linked list in nlogn or logn order of time. And then after sorting only one traversal will be enough for removing the duplicates.

Code :

Assuming the linked list is sorted, the code is given below

Void remove\_Duplicates( node\* head)

{

    node\*temp = head;

node\* next;

 if (temp!=NULL)

{

    while (temp->next != NULL)

    {

       if (temp->data == temp->next->data)

       {

           next = temp->next->next;

           delete temp->next;

           temp->next = next;

       }

       else

       {

          temp = temp->next;

       }

    }

}

}

**Question No.3:**

**PART a and c:**

The element at the top of stack is minimum and the bottom is maximum, in between there is sorting;

The variables ‘SUM’ is globally declared; initialized to one;

void check\_element(int item)

{

stack1,stack2;

for i=1 to length\_stack1;

{

int a= stack1.pop();

stack2.push(a);

if(item==a)

{

stop,end the loop;

print (the item is present);

}

}

for (i=1 to stack2.length)

{

stack1.push(stack2.pop());

}

}

void delete\_element(int item)

{

SUM=SUM-item;

Average=SUM/length\_of\_stack-1;

stack1,stack2;

for i=1 to length\_stack1;

{

int a= stack1.pop()

if(item==a)

{

stop,end the loop;

}

else (if 'a' is not that item to be deleted)

{

stack2.push(a);

}

}

if (stack2.length!=0)

{

for (i=1 to stack2.length)

{

stack1.push(stack2.pop());

}

}

}

void orderinsert(int item)

{

stack1,stack2;

int initial\_length=stack1.length()

SUM=SUM+item;

Average=SUM/initial\_length+1;

if (stack1.is.empty)

insert\_at\_head or simple stack1.push(item);

else

{

for i=1 to length\_stack1;

{

int a= stack1.pop();

if(item<=a)

{

stack2.push(item);

stack2.push(a);

break;

}

else

{

stack2.push(a)

if (length of stack2==initial length of stack1)

stack1.push(item);

}

}

if (stack2.length!=0)

{

for (i=1 to stack2.length())

{

stack1.push(stack2.pop());

}

}

}

}

**PART b:**

1. worst case complexity in terms of big oh for deletion is O(N) which is when item to be deleted is the maximum item in the stack,that is item at the bottom. So there would be N comparisons and we need to pop/search until the end of stack1.
2. worst case complexity in terms of big oh for insertion is O(N) which is when item to be inserted is the maximum of all the items present in the stack. So there would be N comparisons or N pop to reach the end of stack.
3. worst case for checking the element is also O(N), when element to be checked is maximum of the stack elements. Reasoning same as above.

**Question 4:**

**Queue sortQueue(Queue queue1, Queue queue2, int size)**

**{**

**int min\_value;**

**while (queue1.is not empty)**

**{**

**int min\_value=queue1.dequeue();**

**for (int i=0;i<size;i++)**

**{**

**int num=queue1.dequeue();**

**if (num<=min\_value)**

**{**

**queue1.enqeue(min\_value);**

**min\_value=num;**

**}**

**else**

**queue1.enqueue(num);**

**}**

**queue2.enqueue(min\_value);**

**}**

**return queue2;**

**}**

**Question No.5:**

**1)False but when number of items are odd its is true;**

**2)False**

**3)True**

**4)True**

**5)True**