DSA ASSIGNMENT 2

BT22CSH011
Devashish Athawale

Problem Statement:

Applications of Linked List - Polynomial representation and arithmetic operations

Q.1)

(a) Implement radix sort algorithm using arrays for the input list given below. Deduce the time complexity T(n) for the best, worst and average cases.
{ 136, 487, 358, 469, 570, 247, 598, 639, 205, 609 }

(b) Use linked list for implementation of Radix sort for the same elements given above. Deduce the time complexity T(n) for the best, worst and average cases.

Q.1) (a)

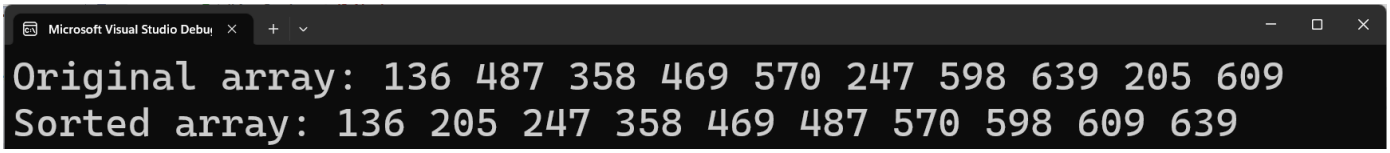
```
#include<iostream>
#include<stdlib.h>
#include<vector>

class Queue //for use in sorting
{
    public:
    Queue()
    {
     Start = NULL;
     End = NULL;
     Size = 0;
    }
    void display()
    {
     for (int i = 0; i < Size; i++)
     {
           std::cout << Start[i] << " ";
     }
    }

    void push(int data)
    {
        if (Size == 0)
        {
           Start = (int*)std::malloc(sizeof(int));
           End = Start;
           Start[0] = data;
           Size = 1;
        }
        else
        {
            int* New = (int*)std::malloc(sizeof(int) * (Size + 1));
            for (int i = 0; i < Size; i++)
            {
                New[i] = Start[i];
            }
            New[Size] = data;
            free(Start);
            Start = New;
            End = &New[Size];
            Size++;
        }
    }
```

```
    //continued
    int pop()
    {
        int data;
        if (Size >= 2)
        {
            data = Start[0];
            for (int i = 0; i < Size - 1; i++)
            {
                Start[i] = Start[i + 1];
            }
            Size--;
            End = &Start[Size - 1];
        }
        else if (Size)
        {
            data = Start[0];
            Size = 0;
            free(Start);
            Start = End = NULL;
        }
        else
        return 0;
        return data;
    }

    int isEmpty()
    {
        if (Size == 0)
        {
          return 1;
        }
        else
        {
          return 0;
        }
    }

    private:
    int Size;
    int* Start;
    int* End;
};
```

```
void countSort(int* arr, int size, int place)
{
    Queue bin[10];
    for (int i = 0; i < size; i++)
    {
          bin[(arr[i] / place) % 10].push(arr[i]);
    }
    int outputPos = 0;
    for (int i = 0; i < 10; i++)
    {
     while (!bin[i].isEmpty())
     {
          arr[outputPos] =a bin[i].pop();
          outputPos++;
     }
    }
}

int maxElement(int* arr, int size)
{
    int max = arr[0];
    for (int i = 0; i < size; i++)
    {
     if (arr[i] > max)
     {
          max = arr[i];
     }
    }
    return max;
}

void radixSort(int* arr, int size)
{
    int max = maxElement(arr, size);
    for (int place = 1; max / place > 0; place = place * 10)
    {
     countSort(arr, size, place);
    }
}
```

```
void main()
{
    int size = 10;
    int arr[] = { 136, 487, 358, 469, 570, 247, 598, 639,
    205, 609 };

    std::cout << "Original array: ";
    for (int i = 0; i < size; i++)
    {
        std::cout << arr[i] << " ";
    }

    radixSort(arr, size);
    std::cout << "\nSorted array: ";
    for (int i = 0; i < size; i++)
    {
        std::cout << arr[i] << " ";
    }
}
```

Output:



Deducing time complexity:

The radixSort() for loop runs for the number of digits in the largest number in given array (=d).

The countSort() function runs a for loops n times (where n is the number of elements in given array). Another for loop is run 10 times (b = 10; 10 is the base of the decimal number system), and the queues (bins) are popped till they are all empty (n times).

Thus, time complexity = O(d * n).

Q.1) (b)

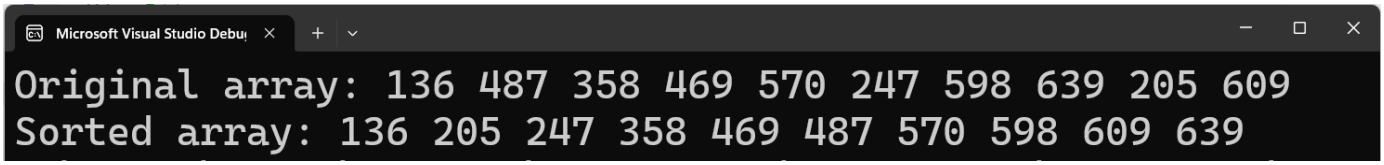
```
#include<iostream>
class LinkedList
{
     class Node
     {
          public:
          Node* nextLoc;
          int data;
          Node()
          {
             nextLoc = NULL;
             data = 0;
          }

          Node(int INData)
          {
             nextLoc = NULL;
             data = INData;
          }
     };
public:
     Node* HEAD;
     int Size;
     LinkedList()
     {
      HEAD = NULL;
      Size = 0;
     }

     void push(int data)
     {
          if (HEAD)
          {
             Node* CURRENT = HEAD;
             while (CURRENT->nextLoc)
             {
                    CURRENT = CURRENT->nextLoc;
             }
             Node* NEW = new Node(data);
             CURRENT->nextLoc = NEW;
          }
          else
          {
             HEAD = new Node(data);
          }
          Size++;
     }
```

```
int pop()
{
     int output = 0;
     if (HEAD)
     {
          Size--;
          if (HEAD->nextLoc)
          {
               Node* TEMP = HEAD;
               HEAD = HEAD->nextLoc;
               output = TEMP->data;
               delete TEMP;
          }
          else
          {
               output = HEAD->data;
               delete HEAD;
               HEAD = NULL;
          }
     }
     return output;
}

bool isEmpty()
{
  if (HEAD)
  return false;
  else
  return true;
}

int maxElement()
{
     int max = 0;
     if (HEAD)
     {
        max = HEAD->data;
        Node* TEMP = HEAD;
        while (TEMP)
        {
              if (TEMP->data > max)
              {
                    max = TEMP->data;
              }
              TEMP = TEMP->nextLoc;
        }
     }
     return max;
}
```

```
int getElement(int pos)
{
     int output = 0;
     if (HEAD)
     {
          Node* CURRENT = HEAD;
          for (int i = 0; i < pos; i++)
          {
               CURRENT = CURRENT->nextLoc;
          }
          output = CURRENT->data;
     }
     return output;
}

void display()
{
  if (HEAD)
  {
        Node* CURRENT = HEAD;
        while (CURRENT)
        {
              std::cout << CURRENT->data << " ";
              CURRENT = CURRENT->nextLoc;
        }
  }
}

void erase()
{
     while (HEAD)
     {
        Node* TEMP = HEAD;
        HEAD = HEAD->nextLoc;
        delete TEMP;
     }
     HEAD = NULL;
     Size = 0;
}

void radixSort()
{
     if (HEAD->nextLoc)
     {
        int max = maxElement();
        for (int place = 1; max / place > 0; place = place * 10)
        {
              countSort(place);
        }
     }
}
```

```
private:
    void countSort(int place)
    {
        LinkedList bins[10];

        for (int i = 0; i < Size; i++)
        {
          int element = getElement(i);
          bins[(element / place) % 10].push(element);
        }
        erase();
        for (int i = 0; i < 10; i++)
        {
          while (!bins[i].isEmpty())
          {
               push(bins[i].pop());
          }
        }
    }
};


void main()
{
    int arr[] = { 136, 487, 358, 469, 570, 247, 598, 639, 205, 609 };
    int size = 10;


    LinkedList array;
    for (int i = 0; i < size; i++)
    {
        array.push(arr[i]);
    }
    std::cout << "Original array: ";
    array.display();
    std::cout << std::endl;

    array.radixSort();
    std::cout << "Sorted array: ";
    array.display();
}
```

Output:



Deducing time complexity:

The radixSort() for loop runs for the number of digits in the largest number in given linked list (=d).

The countSort() function runs a for loops n times (where n is the number of elements in given linked list). Another for loop is run 10 times (b = 10; 10 is the base of the decimal number system), and the queues (bins) are popped till they are all empty (n times).

Thus, time complexity = O(d * n).