Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 4 REPORT

OZAN ŞELTE
161044061

Course Assistant: Ayşe Şerbetçi Turan

1 QUESTION 1

1.1 Part A

```
public static LinkedListO<Integer> q1_iterative(LinkedListO<Integer> L) {
    if(null == L) return null;
    NodeO<Integer> head = L.head;
    if(null == head || null == head.data) return null;
    int maxLen = 0, len = 1, count = 1, maxFirstIndex = 0, firstIndex = 0;
    int last = head.data;
    head = head.next;
    while(null != head) {
        if(last <= head.data) {
            len++;
        }
        else {
            if(len > maxLen) {
                maxLen = len;
                maxFirstIndex = firstIndex;
            }
            firstIndex = count;
            len = 1;
        }
        count++;
        last = head.data;
        head = head.next;
    }
    if(len > maxLen) {
        maxLen = len;
        maxFirstIndex = firstIndex;
    }
    LinkedListO<Integer> S = new LinkedListO<Integer>();
    head = L.head;
    for(int i = 0; i < maxFirstIndex; i++) {
        head = head.next;
    }
    for(int i = 0; i < maxLen; i++) {
        S.add(head.data);
        head = head.next;
    }
    return S;
}
```

There is one while-loop to traverse all the numbers which are θ(n) and then there are two for-loops to iterate to the first item of the sublist and one loop to copy sublist. All the for-loop is θ(n). But in the last for-loop, the add method is θ(n) for LinkedLists. All the others are constant.

T(n) = n^2 + 2n + f(n) -> θ(n^2)

1.2 Part B

Recursive function with LinkedList parameter is not possible. We should give as parameter the head node or use a helper function. I have choosen the second one.

```
public static LinkedListO<Integer> q1_recursive(LinkedListO<Integer> L) {
    if(null == L) return null;
    NodeO<Integer> head = L.head;
    int index = recursion(head, 0, 0, 0);
    LinkedListO<Integer> S = new LinkedListO<Integer>();
    for(int i = 0; i < index; i++) {
        head = head.next;
    }
    int last = head.data;
    while(null != head && last <= head.data) {
        last = head.data;
        S.add(last);
        head = head.next;
    }
    return S;
}

public static int recursion(NodeO<Integer> N, int maxFirstIndex, int maxLen, int index) {
    NodeO<Integer> head = N;
    if(null == head || null == head.data) return -1;
    int len = 1, count = 1, last = head.data;
    head = head.next;
    while(null != head) {
        if(last <= head.data) {
            len++;
        }
        else {
            if(len > maxLen) {
                maxLen = len;
                maxFirstIndex = index;
            }
            maxFirstIndex = recursion(head, maxFirstIndex, maxLen, index + count);
            break;
        }
        count++;
        last = head.data;
        head = head.next;
    }
    if(len > maxLen) {
        maxFirstIndex += index;
    }
    return maxFirstIndex;
}
```

For *q1_recursive* method, the complexity is Trecursion(n) and two seperate loops which are θ(n) and θ(n^2) so:

T(n) = Trecursion(n) + n^2 + n +contants

For *recursion* method, a => 1 and b > 1, **T(n) = aT(n/b) + f(n)**

n = size of numbers
a = sorted sublists number
n/a = size of each subproblem. (To apply Master Theorem, it is assumed that all subproblems are essentially the same size. a = b)
f(n) = the cost of dividing the problem and merging solutions. (n/a)
c = T(f(n)) = n/a

T(n) = aT(n/a) + n/a

a is not constant number. Master Theorem cannot be apply to this equation. But if we try:

T(n) = aT(n/a) + n/a log_b(a) = log_a(a) log_b(a) = 1
f(n) = θ(n^c * log^k(n)) -> T(n) = θ(n^c * log^(k+1)(n))
c = 1, k = 0 c = log_b(a)
T(n) = θ(n * log(n))

If we use induction, T(n) = aT(n/a) + n/a, and we guess T(n) <= cnlogn

T(n) <= a * (c * (n/a) * log(n/a)) + n/a
T(n) <= cn*logn – cn*loga + n/a
T(n) <= cn*logn

T(n) for *q1_recursive* method, T(n) = Trecursion(n) + n^2 + n + constants

T(n) = nlogn + n^2 + n + constants -> θ(n^2)

2 QUESTION 2

```
int[] arr = DEFINED;
int x = DEFINED;

int idxL = 0, idxR = arr.length;

while(idxR >= idxL) {
   int sum = arr[idxL] + arr[idxR];
   if(sum > x) idxR--;
   else if(sum < x) idxL++;
   else break;
}
```

idxL and idxR are the special indexes. T(n) = n -> θ(n)

3 QUESTION 3

This code has three inner for-loops.

T(n) = (2 * n) * n * (n / 3) = 2 * n * n * n / 3
T(n) = n^3 / 3 and T(n) = n / 3 = O(log_3(n))
O(n^2 * log(n))