CS6033 Design and Analysis of Algorithms I
Homework Assignment 6 Part 1
12.11.2022

Group Members:
rk4305 (Sai Rajeev Koppuravuri)
sg7372 (Sriharsha Gaddipati)
vt2182 (Venu Vardhan Reddy Tekula)
vt2184 (Veeravenkata Raghavendra Naveenkumar Tata)

1. (5 points) Given $n > 2$ distinct numbers, you want to find a number that is neither the minimum nor the maximum. What is the smallest number of comparisons that you need to perform?

   Ans:
   Given an array of n distinct elements (> 3):

   - Recurrence relation for computing minimum comparisons required to find minimum and maximum element of array size n is: T(n) = 2T(n/2) + 2
   - Number of comparisons required to find minimum and maximum is $= 3 * n/2 - 2$.
   - So, minimum comparisons required to return the element which is neither the maximum nor the minimum is
     $= 3 * n/2 - 2 + 1$
     $= 3 * n/2 - 1$

   For example, n $=$ 3 we need only **3** comparisons to find an element/number which is neither the maximum nor the minimum.

2. (5 points) In the algorithm SELECT, the input elements are divided into groups of 5. Show that the algorithm works in linear time if the input elements are divided into groups of 7 instead of 5.

   Ans:

   Here, we are dividing the elements into groups of 7 instead of 5. On each partitioning, the minimum number of elements that are less than (or greater than) x will be

   $4 * (\lceil \frac{1}{2} \lceil \frac{n}{7} \rceil \rceil - 2) \geq (\frac{2n}{7} - 8)$

   Let's do some partitioning mechanisms to reduce the sub-problem to the size of at most $5n/7 + 8$. This gives the following recurrence equation

   $$T(n) = \begin{cases} \mathcal{O}(1) & \text{if} \quad n < n_0 \\ T(\lceil n/7 \rceil) + T(5n/7 + 8) + \mathcal{O}(n) & \text{if} \quad n \geq n_0 \end{cases}$$

   Let's assume that $T(n) \leq cn$ and bound the non-recursive term with 'an'

   $T(n) \leq c\lceil n/7 \rceil + c(5n/7 + 8) + an$
   $T(n) \leq cn/7 + c + 5cn/7 + 8c + an$

$T(n) = 6cn/7 + 9c + an$
$T(n) = cn + (-cn/7 + 9c + an)$
$T(n) \leq cnT(n) = \mathcal{O}(n)$

The above rule applies when the constant $(-cn/7 + 9c + an) \leq 0$

$(-cn/7 + 9c + an) \leq 0$
$-c(n/7 - 9) + an \leq 0$
$c(n/7 - 9) \geq an$
$c(\frac{n-63}{7}) \geq an$
$c \geq \frac{7an}{n-63}$

So, if we consider $n_0 = 126$ and $n \leq n_0$, we can get $\frac{n}{n-63} \leq 2$

and

$c \geq \frac{7an}{n-63}$
$c \geq \frac{7*a*126}{126-63}$
$c \geq 14a$

Hence, this will run in linear time.

3. (5 points) Show how to implement a stack using two queues. Analyze the running time of the stack operations.

Ans:

```
class Stack:

    def __init__(self):

        # Two inbuilt queues
        self.q1 = queue()
        self.q2 = queue()

    def push(self, x):
        self.q2.append(x)

        while (self.q1):
            self.q2.append(self.q1.popleft())

        # swap the names of two queues
        self.q1, self.q2 = self.q2, self.q1

    def pop(self):
        # if no elements are there in q1
        if self.q1:
            self.q1.popleft()

    def top(self):
        if (self.q1):
            return self.q1[0]
        return None

    def size(self):
        return len(self.q1)
```

Time complexity for Push operation: O(N)

Time complexity for Pop operation: O(1)

4. (10 points) Given head, the head of a linked list, determine if the linked list has a cycle in it. Write a Puesdocode for the same and running time. (Note : You can not use any extra space to detect cycle in linked list.)

   Ans: Take two points, fast and slow pointing to head. Iterate slow with one step and fast with two steps till it reaches end. If there is a cycle then both fast and slow pointers would collide each other once.

```
    def detectCycle(head):
        slow_p = head
        fast_p = head
        while(slow_p and fast_p and fast_p.next):
            slow_p = slow_p.next
            fast_p = fast_p.next.next
            if slow_p == fast_p:
                return 1
        return 0
```

5. (5 points) Explain how to implement two stacks in one array A[1 : n] in such a way that neither stack overflows unless the total number of elements in both stacks together is n. The PUSH and POP operations should run in O(1) time.

   Ans: Let us take an array A[1..n], all we need to do is place one stack at the first index and the other at the last index, and every time we perform a PUSH or POP operation, we must perform an overflow check. When trying to add more pieces to any of the two stacks, stack overflow would occur if the two top indices of the stacks are nearby indexes.

```
PUSHLEFT(S,x)
    IF S.topLeft < S.topRight-1
        S.topLeft = S.topLeft + 1
        S.array[S.topLeft] = x
    ELSE
        PRINT("Overflow")


PUSHRIGHT(S,x)
    IF S.topLeft < S.topRight - 1
```

```
        S.topRight = S.topRight - 1
        S.array[S.topRight] = x
    ELSE
        PRINT("Overflow")


POPLEFT(S)
    IF S.topLeft >= 1
        val = S.array[S.topLeft]
        S. array[S.topLeft] = NIL
        S.topLeft = S.topLeft - 1
        RETURN val
    ELSE
        PRINT("Stack is empty")

POPRIGHT(S)
    IF S.topRight < n
        val = S.array[S.topRight]
        S. array[S.topRight] = NIL
        S.topRight = S.topRight + 1
        RETURN val
    ELSE
        PRINT("Stack is empty")
```

6. (10 points) Draw the binary tree rooted at index 6 that is represented by the following attributes:

index	key	left	right
1	17	8	9
2	14	NIL	NIL
3	12	NIL	NIL
4	20	10	NIL
5	33	2	NIL
6	15	1	4
7	28	NIL	NIL
8	22	NIL	NIL
9	13	3	7
10	25	NIL	5

Ans:

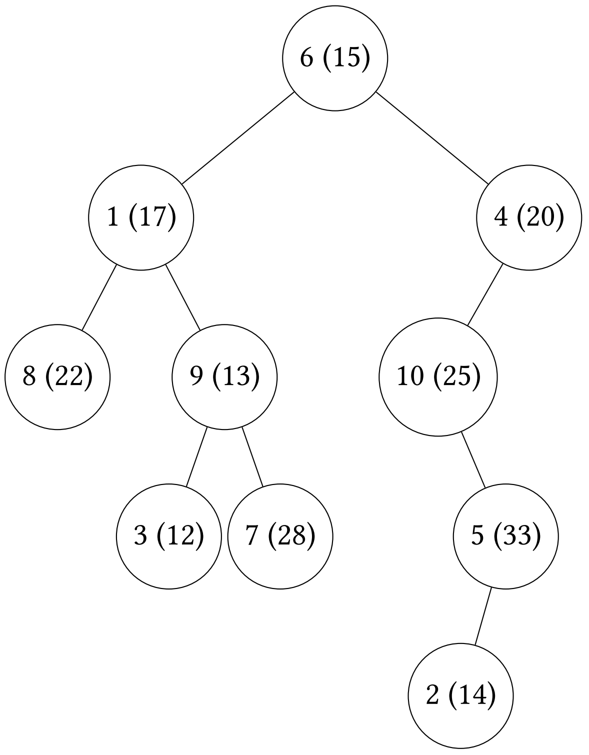
The index 6 contains 15 as key. so 15 will be root. On the left side of 15, the node at index 1 (17) will be stored. On the right side of 15, the node at index 4 (20) will be stored.

The index 1 contains 17 as key. On the left side of 17, the node at index 8 (22) will be stored. On the right side of 17, the node at index 9 (13) will be stored.

The index 4 contains 20 as key. On the left side of 20, the node at index 10 (25) will be stored. There is no element on the right side of 20.

On the right side of 25, the node at index 5 (33) will be stored. On the left side of 33, the node at index 2 (14) will be stored.

The index 9 contains 13 as key. On the left side of 13, the node at index 3 (12) will be stored. On the right side of 13, the node at index 7 (28) will be stored.



7. (10 points) You are given a string s consisting of lowercase English letters. A duplicate removal consists of choosing two adjacent and equal letters and removing them. We repeatedly make duplicate removals on s until we no longer can. Return the final string after all such duplicate removals have been made. It can be proven that the answer is unique.

   Input: s = abbaca

   Output: ca

   Write a Puesdocode for the same and running time.

   Ans:

```
def remAdjDup(s, rem):
    if len(s) == 0 or len(s) == 1:
        return s

    if s[0] == s[1]:
        rem = ord(s[0])
        while len(s) > 1 and s[0] == s[1]:
            s = s[1:]
        s = s[1:]
        return remAdjDup(s, rem)

    rem_str = remAdjDup(s[1:], rem)

    if len(rem_str) != 0 and rem_str[0] == s[0]:
        rem = ord(s[0])
        return (rem_str[1:])

    if len(rem_str) == 0 and rem == ord(s[0]):
        return rem_str

    return ([s[0]] + rem_str)

def remove(s):
    rem = 0
    s_lst = [ch for ch in s]
    res = remAdjDup(s_lst, rem)
    return ''.join(res)

s = 'abbaca'
print(remove(s))
```

Time complexity $= \mathcal{O}(n)$