CSCE 221 Cover Page
Homework Assignment #2
Due March 25 at 23:59 pm to eCampus

First Name: Pratik Last Name: Patel UIN: 527004337

User Name: p.pratik99 E-mail address: p.pratik99@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources			
People			
Web pages (provide URL)			
Printed material			
Other Sources			

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

Your Name: Pratik Patel Date: 03/23/2019

Homework 2

due March 25 at 11:59 pm to eCampus.

1. (15 points) Describe in pseudo code how to implement the stack ADT using two queues.

   (a) Write a C++ function that implements your solution. You can use the C++ STL queue container.

```
#include <iostream>
#include <queue>

void push(T elem, queue<T> queue)
{
    queue.push_back(elem);
}

T pop(queue<T> queue)
{
    queue queue2;

    while(queue.size() > 1)
        queue2.push_back(queue.pop_front());

    T temp = queue.pop_front();

    while(queue2.isEmpty())
        queue.push_back(queue2.pop_front());

    return temp;
}
```

   (b) What is the running time of the push and pop functions in this case?

   push() – $O(1)$

   pop() – $O(n)$

   Because it has to put the first n-1 elements in the second queue that takes n operations. Therefore, the time complexity is $O(n)$

2. (15 points) Write a recursive function in C++ that counts the number of nodes in a singly linked list.

```
int DoublyLinkedList::countNodes(DListNode& n)
{
    if(n.next == nullptr)
        return 0;
    else if(n.next != nullptr)
        return 1 + countNodes(*(n.next));
}
```

(a) Test your function using different singly linked lists. Include the code and screenshots with testing cases.

```
#include "DoublyLinkedList.h"
#include <iostream>
using namespace std;
int main () {
  // Construct a linked list with header & trailer
  cout << "Create a new list" << endl;
  DoublyLinkedList dll;
  cout << "list: " << dll << endl << endl;

  // Insert 10 nodes at back with value 10,20,30,..,100
  cout << "Insert 10 nodes at back with value 10,20,30,..,100" << endl;
  for (int i=10;i<=100;i+=10) {
    dll.insertLast(i);
  }
  cout << "list: " << dll << endl << endl;

  //testing for countNodes function
  cout << "Number of Nodes in list 1 is" << endl;
  cout << dll.countNodes(*(dll.getFirst())) << endl;
```

```
Pratiks-MacBook-Pro:DoublyLinkedList pratik$ ./run-dll
Create a new list
list:


Insert 10 nodes at back with value 10,20,30,..,100
list: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100


Number of Nodes in list 1 is
10
```

(b) Write a recurrence relation that represents your algorithm.

$$T(n) = T(n-1) + 1$$
$$T(0) = 1$$

(c) Solve the relation using the iterating or a recursive tree method to obtain the running time of the algorithm in Big-O notation.

$$T(n) = T(n-1) + 1$$
$$T(n) = T(n-2) + 2$$
$$T(n) = T(n-3) + 3$$
$$T(n) = T(n-k) + k$$
$$For\ a\ maximum\ k = n,$$
$$T(n) = T(0) + n$$

$$T(n) = n + 1$$
$$T(n) = O(n)$$

3. (15 points) Write a C++ recursive function that finds the maximum value in an array of integers without using any loops.

   (a) Test your function using different input arrays. Include the code and screenshots with testing cases.

```cpp
#include <iostream>

using namespace std;

int findMax(int[], int, int, int);

int main()
{
    int temp[5] = {12, 13, 5, 14, 7};

    cout << "Below is an array:" << endl;

    for(int i = 0; i < 5; ++i)
    {
        cout << temp[i] << " " ;
    }
    cout << endl << endl;

    cout << "The maximum number returned by the function is " << findMax(temp, 0, temp[0], 5) << endl;
    return 0;
}

int findMax(int arr[], int index, int max, int size)
{
    if(arr[index] > max)
    {
        if(index == size - 1)
            return arr[index];
        else
            return findMax(arr, ++index, arr[index-1], size);
    }

    else
    {
        if(index == size - 1)
            return max;
        else
            return findMax(arr, ++index, max, size);
    }
}
```

```
Pratiks-MacBook-Pro:HW#2 pratik$ ./a.out
Below is an array:
12 13 5 14 7

The maximum number returned by the function is 14
Pratiks-MacBook-Pro:HW#2 pratik$
```

   (b) Write a recurrence relation that represents your algorithm. Solve the relation and obtain the running time of the algorithm in Big-O notation.

$$T(n) = T(n-1) + 1$$
$$T(0) = 1$$

$$T(n) = T(n-1) + 1$$
$$T(n) = T(n-2) + 2$$
$$T(n) = T(n-3) + 3$$
$$T(n) = T(n-k) + k$$
$$For\ a\ maximum\ k = n,$$
$$T(n) = T(0) + n$$
$$T(n) = n + 1$$
$$T(n) = O(n)$$

4. (15 points) What is the best, worst and average running time of quick sort algorithm? Provide arrangement of the input and the selection of the pivot point at every case. Provide a recursive relation and solution for each case.

- **Best:** Big-Oh for quick sort for the best case is $O(nlogn)$. The best case occurs when the pivot point is in the middle of the sequence. That makes both sides equal in terms of size. Therefore, each recursive call will divide the sequence in half.

  $$T(0) = 1$$
  $$T(n) = 2T(n/2) + n$$

  $$T(n/2) = 4T(n/4) + n/2$$
  $$T(n/4) = 8T(n/8) + n/4$$
  $$T(n) = O(nlogn)$$

- **Average:** Big-Oh for average case is also $O(nlogn)$. The average case occurs when the pivot point is between first and last element(exclusive) and not the middle point. This situation divides the sequence differently every time. Hence, we take average of each test case.

  The recursive relation is same as best case.

- **Worst:** The worst case appears when the pivot is the first or the last element in the sequence. If this happens then the next call contains n-1 elements, making it a function of running time of $O(n^2)$.

  $$T(n) = T(n-1) + n$$
  $$T(0) = 1$$

  $$T(n) = T(n-2) + n - 1 + n$$
  $$T(n) = T(n-3) + n - 2 + n - 1 + n$$
  $$T(n) = T(n-k) + (n-k) + \ldots + (n-1) + n$$
  $$For\ a\ maximum\ k = n,$$
  $$T(n) = T(0) + (n(n+1)/2)$$
  $$T(n) = \left(\frac{n^2+n}{2}\right) + 1$$
  $$T(n) = O(n^2)$$

5. (10 points) What is the best, worst and average running time of merge sort algorithm? Use two methods for solving a recurrence relation for the average case to justify your answer.

Best, Average, Worst all have $O(nlogn)$ running time for merge sort.

- Iterative Method:

$$T(0) = 1$$
$$T(n) = 2T(n/2) + n$$

$$T(n/2) = 4T(n/4) + n/2$$
$$T(n/4) = 8T(n/8) + n/4$$
$$T(n) = O(nlogn)$$

- Master Theorem:

For this case Master theorem can be written as below

$$T(n) = 2T(n/2) + \Theta(n)$$

We have $a = 2, b = 2, f(n) = \theta(n)$. Therefore, we have case 2 of master theorem. Hence,

$$T(n) = O(nlogn)$$

6. (10 points) R-10.17 p. 493
For the following statements about red-black trees, provide a justification for each true statement and a counterexample for each false one.

(a) A subtree of a red-black tree is itself a red-black tree.

False — a subtree with a red root is not a red-black tree.

(b) The sibling of an external node is either external or it is red.

True - The depth property says that all external nodes must have the same black depth, that is, the same number of black ancestors.

(c) There is a unique (2,4) tree associated with a given red-black tree.

True – Because every node in a red-black tree is represented uniquely in a (2, 4) tree.

(d) There is a unique red-black tree associated with a given (2,4) tree.

False – There can be two different red-black trees associated with one (2, 4) tree.

7. (10 points) R-10.19 p. 493
Consider a tree T storing 100,000 entries. What is the worst-case height of T in the following cases?

(a) T is an AVL tree.

$$\lceil 2(log_2 100000) + 1 \rceil = 34$$

(b) T is a (2,4) tree.

$$\lceil log_2 1000000 + 1 \rceil = 17$$

(c) T is a red-black tree.

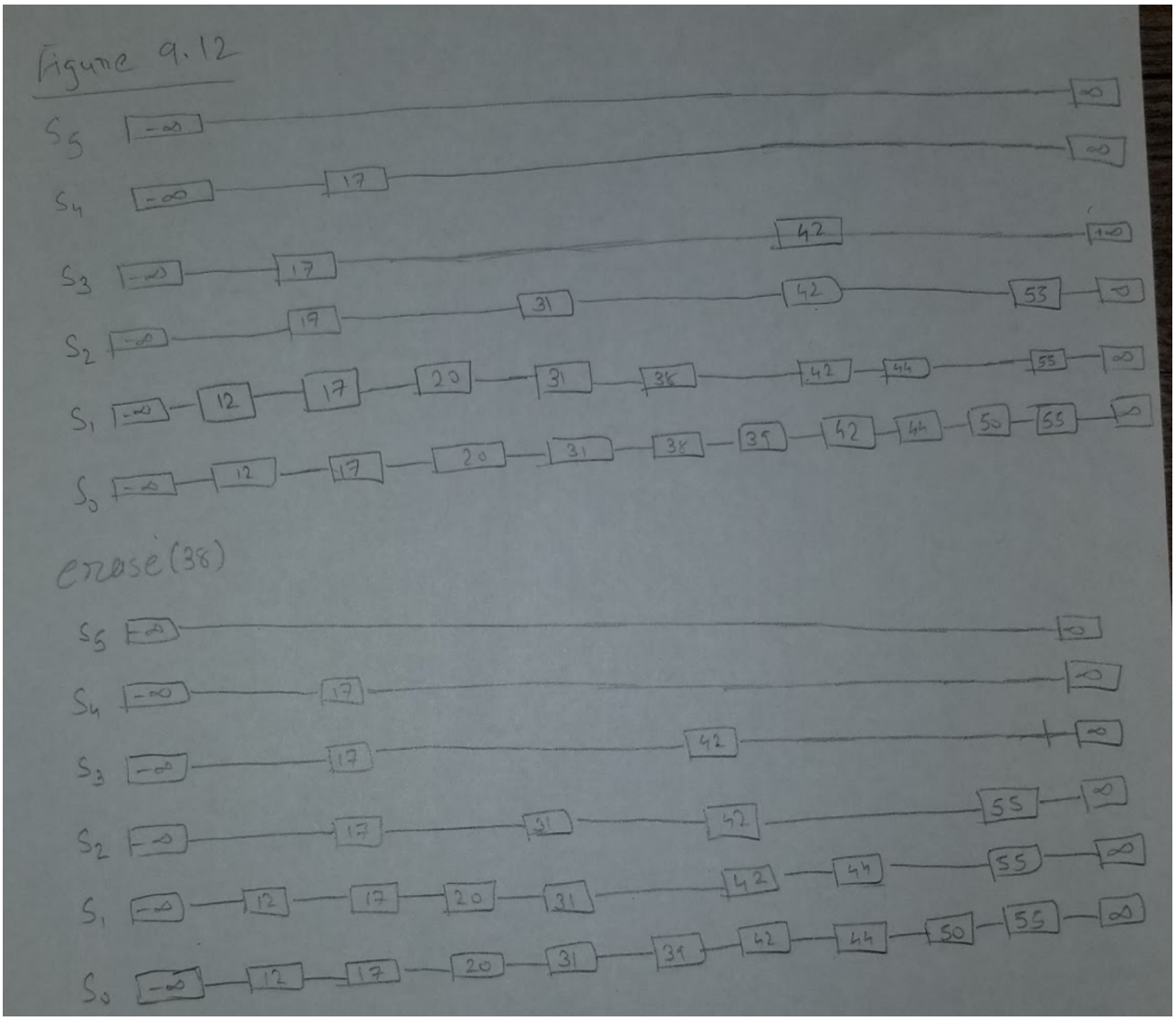
$$\lceil 2(log_2 100000) + 1 \rceil = 34$$

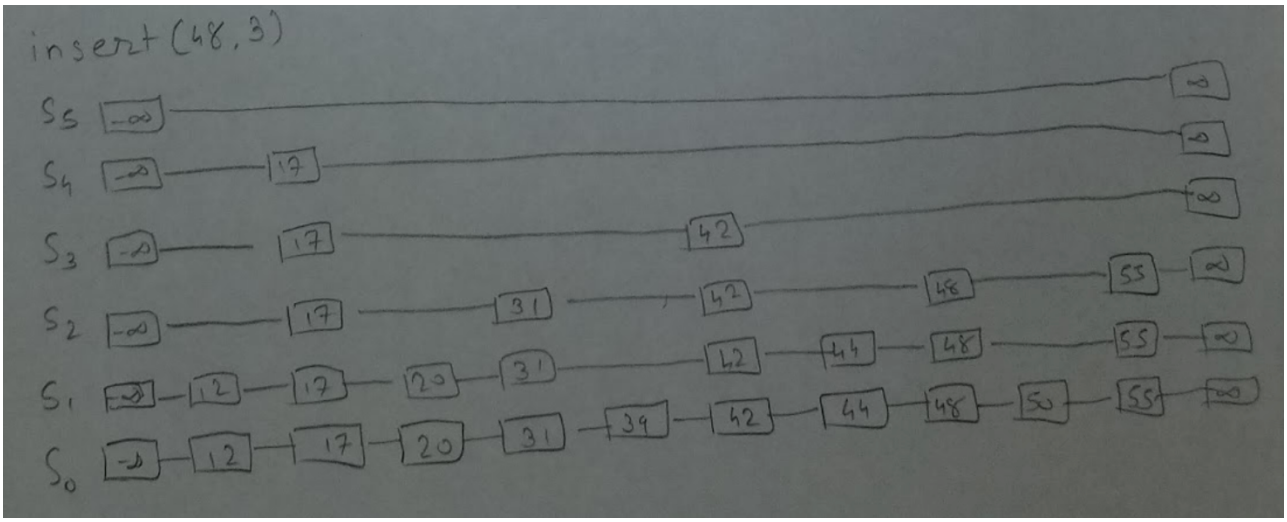
(d) T is a binary search tree.

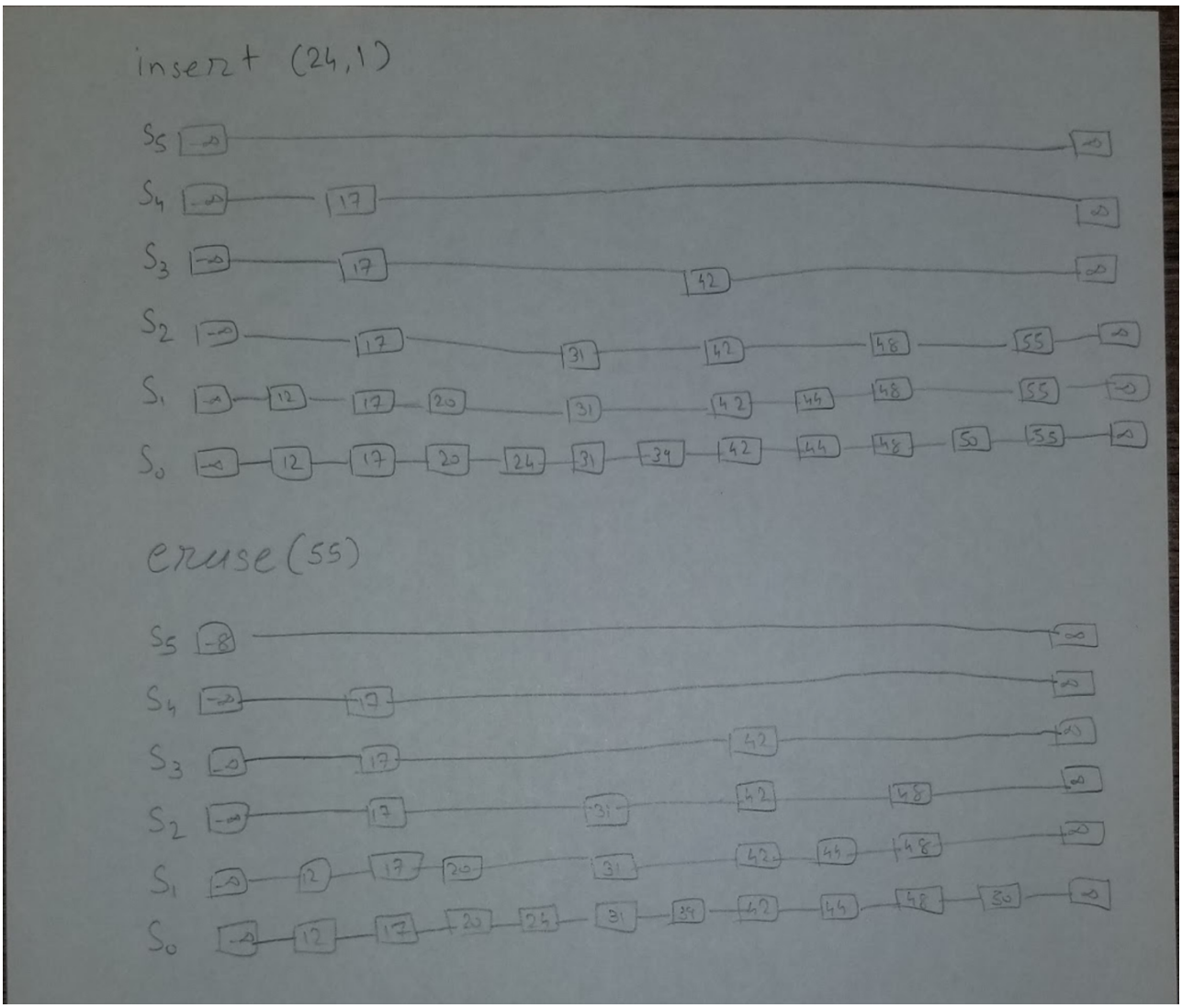
$$n = 1000000$$

8. (10 points) R-9.16 p. 418

Draw an example skip list that results from performing the following series of operations on the skip list shown in Figure 9.12: `erase(38)`, `insert(48,x)`, `insert(24,y)`, `erase(55)`. Record your coin flips, as well.




insert (48, 3)
S5 -∞ ∞
S4 -∞ 17 ∞
S3 -∞ 17 42 ∞
S2 -∞ 17 31 42 48 55 ∞
S1 -∞ 12 17 20 31 42 44 48 55 ∞
S0 -∞ 12 17 20 31 34 42 44 48 50 55 ∞


insert (24, 1)
S5 -∞ ∞
S4 -∞ 17 ∞
S3 -∞ 17 42 ∞
S2 -∞ 17 31 42 48 55 ∞
S1 -∞ 12 17 20 31 42 44 48 55 ∞
S0 -∞ 12 17 20 24 31 34 42 44 48 50 55 ∞
erase (55)
S5 -∞ ∞
S4 -∞ 17 ∞
S3 -∞ 17 42 ∞
S2 -∞ 17 31 42 48 ∞
S1 -∞ 12 17 20 31 42 44 48 ∞
S0 -∞ 12 17 20 24 31 34 42 44 48 50 ∞