CSCE 221 Cover Page

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 O ce https://aggiehonor.tamu.edu/

Name	Mualla Argin
UIN	728003004
Email address	Margin25@tamu.edu

Cite your sources using the table below. Interactions with TAs and resources presented in lecture do not have to be cited.

People	1. None
Webpages	https://www.youtube.com/watch?v=UGaOXaXAM5M
Printed Materials	1. None
Other Sources	1. None

Homework 2

Due March 25 at 11:59 PM

Typeset your solutions to the homework problems preferably in LaTeXor LyX. See the class webpage for information about their in-stallation and tutorials.

1. (15 points) Provided two sorted lists, l1 and l2, write a function in C++ to e ciently compute l1 \ l2 using only the basic STL list operations. The lists may be empty or contain a di erent number of elements e.g jl1j 6= jl2j. You may assume l1 and l2 will not contain duplicate elements.

Examples (all set members are list node):

- f1; 2; 3; 4g \ f2; 3g = f2; 3g
- ; \ f2; 3g = ;
- f2; 9; 14g \ f1; 7; 15g = ;

(a) Complete the function below. Do not use any routines from the algorithm header le.

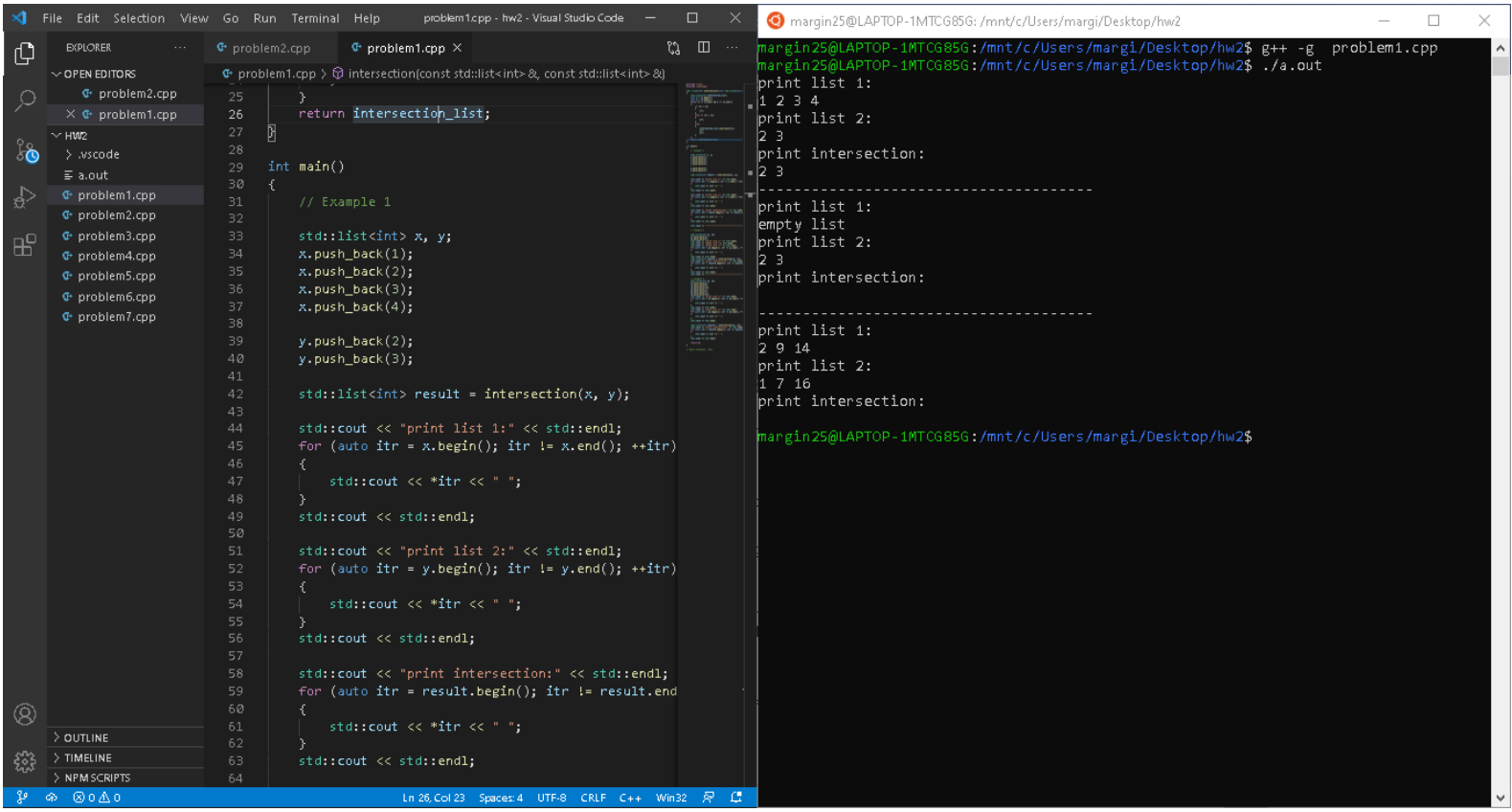
```
1 #include<list>
2
3 std::list<int>intersection(const std::list<int>&l1,
      const std::list<int>& l2) f
```

```
std::list<int> intersection_list;
    auto x = l1.begin();
    auto y = l2.begin();
    while (x != l1.end() && y != l2.end())
    {
        if (*x < *y)
        {
            x++;
        }
        else if (*x > *y)
        {
            y++;
        }
        else
        {
            intersection_list.push_back(*x);
4           x++;
```

```
            y++;
        }
    }
    return intersection_list;
```

5 g

(b) Verify that your implementation works properly by writing two test



cases. Provide screenshot(s) with the results of your testing.

(c) What is the running time of your algorithm? Provide a big-O bound. Justify.

The running time of my algorithm is O(n) because there are no nested for loops or while loops and the algorithm goes through my non-nested while loop.

2. (15 points) Write a C++ recursive function that counts the number of nodes in a singly linked list. Do not modify the list.

Examples:

- count_nodes((2) ! (4) ! (3) ! nullptr) = 3

- count_nodes(nullptr) = 0

(a) Complete the function below:

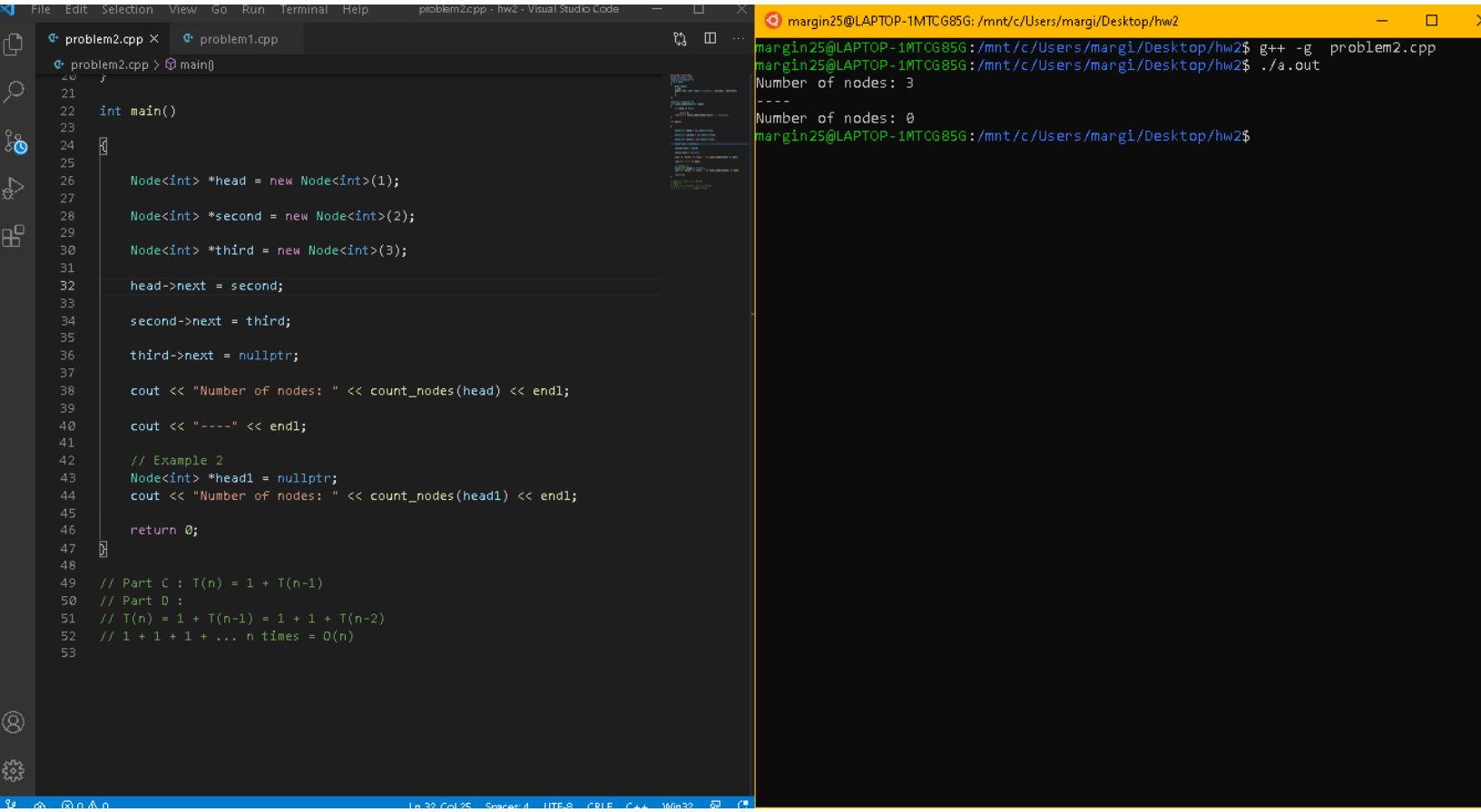
```
template<typename T>
struct Node {
    Node * next;
    T obj;

    Node (T obj, Node * next= nullptr)
        : obj(obj),next(next)
    {
    }
};
```

```
template <typename T>
int count_nodes(Node<T> *head)
{
    if (head == NULL)

        return 0;
    return 1 + count_nodes(head->next); // recursive
}
```

(b) Verify that your implementation works properly by writing two test cases for the function you completed in part (a). Provide screenshot(s) with the results of your testing.



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

```
T(n) = 1 + T(n-1)
```

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

```
2      // T(n) = 1 + T(n-1) = 1 + 1 + T(n-2)
3      // 1 + 1 + 1 + ... n times = O(n)
```

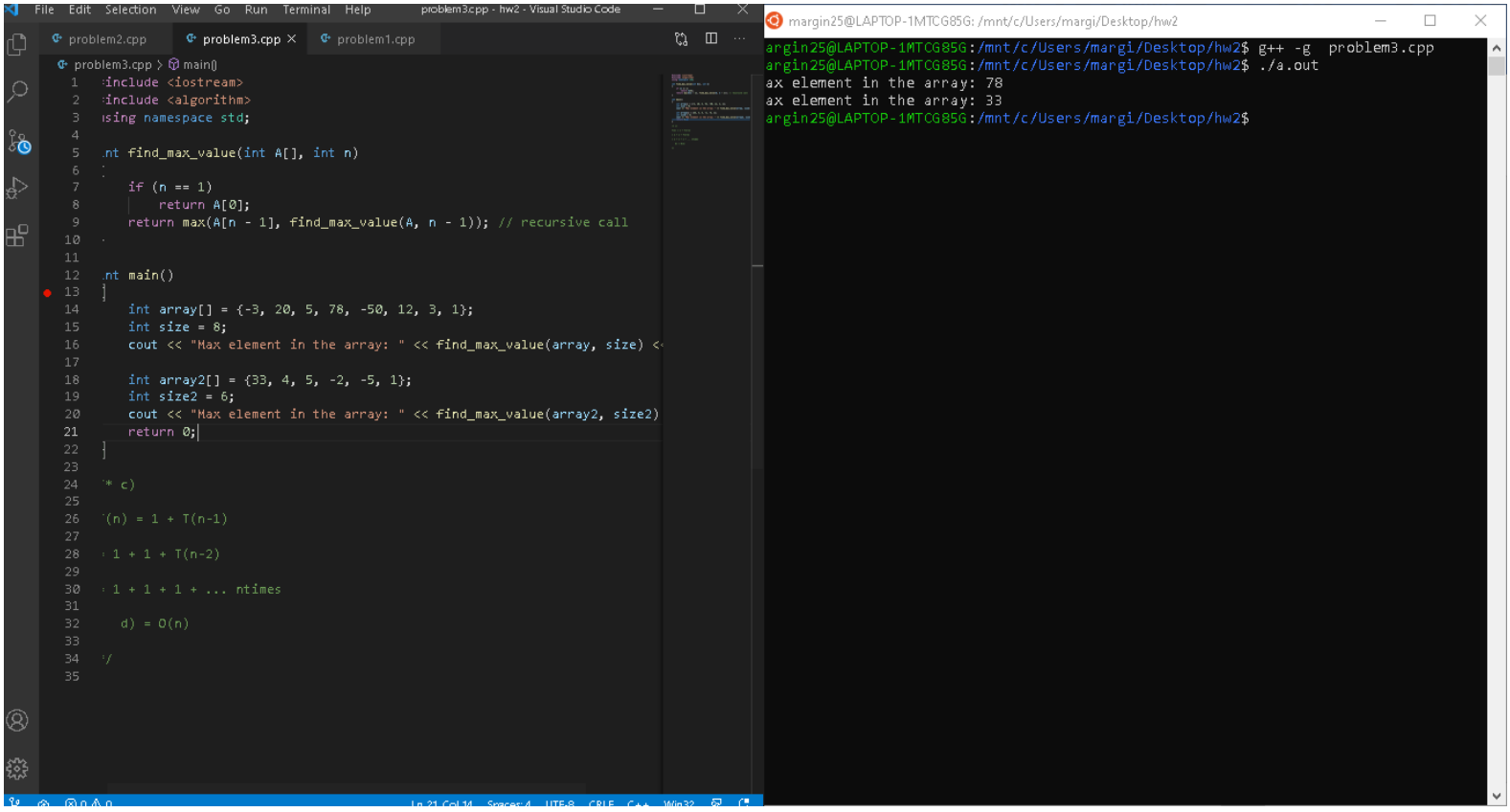
3. (15 points) Write a C++ recursive function that nds the maximum value in an array (or vector) of integers without using any loops. You may assume the array will always contain at least one integer. Do not modify the array.

(a) Complete the function below:

```
#include <iostream>
#include <algorithm>
using namespace std;
```

```cpp
int find_max_value(int A[], int n)
{
    if (n == 1)
        return A[0];
    return max(A[n - 1], find_max_value(A, n - 1)); // recursive call
}
```

(b) Verify that your implementation works properly by writing two test



cases. Provide screenshot(s) with the results of the tests.

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

```
//  T(n-1) + 1
```

(d) Solve the recurrence relation and obtain the running time of the algorithm in Big-O notation. Show your process.

```
// d) T(n) = T(n-1) +1
// if T(n-1) = T(n-2) + 1 then
// T(n) = T(n-1) +1 is also (T(n-2)+1) + 1
// (T(n-2)+1) + 1 can be rewritten as (T(n-3)+1) + 1 + 1
// T(n) = T(n-k) + 1 +...+1
// if we simplify this we get 1+1+1+...n times therefore
// the big O notation is O(n)
```

4. (15 points) What is the best, worst and average running time of quick sort algorithm?

(a) Provide recurrence relations. For the average case, you may assume that quick sort partitions the input into two halves proportional to c and 1 c on each iteration.

best case : T(n)= T(n/2) + T(n/2) + O(n) = 2 T(n/2) + O(n)

T(1) = 0

Avg case: T(n) = T(cn) + T((1-c)n) + n
T(1) = 0

worst case : T(n)=T(n-1)+n

(b) Solve each recurrence relation you provided in part (a)

Best case:
T(n) = 2 T(n/2) + O(n)
T(n/2) = 2T(n/4) + O(n/2)
= 2(2T(n/4) + O(n/2)) + O(n)
= 4T(n/4) + 2O(n)
T(n/4) = 2 T(n/8) + O(n/4)
= 4(2T(n/8) + O(n/4)) + 2O(n)
= 8T(n/8) + 3O(n)
Step k = 2^k * T(n/2^k) + k*O(n)
n= 2^k * T(n/2^k) + k * O(n)
= n * T(1) + log_2(n) * O(n)
= 0 + log_2(n) * O(n)
= O(n*log_2(n))
= 2^(log_2(n)) = n^(log_2(2)) = n

Average Case:
See attached at end of doc.

Worst case:
T(n) = T(n-1) + n

= T(n-2) + (n-1) + n
= T(n-3) + (n-2) + (n-1) + n
Step k : = T(n-k+1) + (n-k+2) +..+(n-1) + n
Step n : T(1) + 2 + 3 +…+(n-1) + n
= ((n(n+1)) / 2) -1 = O(n^2)

(c) Provide an arrangement of the input array which results in each case. Assume the rst item is always chosen as the pivot for each iteration.

Best/Average Case:
Array = [50,70,35,90,20]
The pivot is the first element: 50. All elements smaller than 50 will be placed on it's left and elements greater than 50 will pe placed in it's right.
Array = [20,35,50,90,70]
We accomplish this by swapping the first num less than 50 and the number greater than 50.
The array is partionioned into two sections : 20,35 and 90,70. We repeat the above process until the array is sorted and we get [20,35,50,70,90].

Worst Case:
We are given an already sorted array: [20,35,50,70,90].
The pivot is the first element: 20. Since the array is already sorted we will do the traversal and partionining we did above in the best/average case. However, this process will be very inefficient since the array is already sorted.

5. (15 points) Write a C++ function that counts the total number of nodes with two children in a binary tree (do not count nodes with one or none child). You can use a STL container if you need to use an additional data structure to solve this problem.

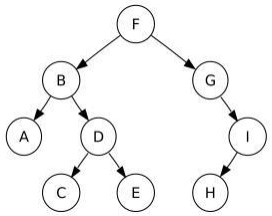


Figure 1: Calling count filled nodes on the root node F returns 3

(a) Complete the function below. The function will be called with the root node (e.g. count filled nodes(root)). The tree may be empty. Do not modify the tree.

```
template <typename T>
struct Node
{
```

```
    Node<T> *left, *right;
    T obj;

    Node(T obj, Node<T> *left = nullptr, Node<T> *right = nullptr) : obj(obj), le
ft(left), right(right)
    {
    }
};

template <typename T>
int count_filled_nodes(const Node<T> *node)
{
    if (node == nullptr)
    {
        return 0;
    }
    int counter = count_filled_nodes(node->left) + count_filled_node(node-
>right);
    if ((node->left != nullptr) && (node->right != nullptr))
    {
        counter++:
    }
    return counter;
}
```

(b) Use big-O notation to classify your algorithm. Show how you arrived at your answer.

O(n) : The recursive function traverses through all the nodes on the binary tree once therefore the big O notation is O(n).

6. (15 points) For the following statements about red-black trees, provide a justi cation 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 red.

True, all external black nodes have the same number of black ancestors/black depth.

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

True, a node with no red children is considered a 2 node tree and a node with two red children is a 4 node tree.

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

False, A 3-node has two distinct possible representations in a red-black tree.

7. (10 points) Modify this skip list after performing the following series of operations: erase(38), insert(48,x), insert(24, y), erase(42). Provided the recorded coin ips for x and y. Provide a record of your work for partial credit.

ATTACHED BELOW

Question 4b avg case

Thursday, March 25, 2021 11:22 PM

Solving the recurrence relation

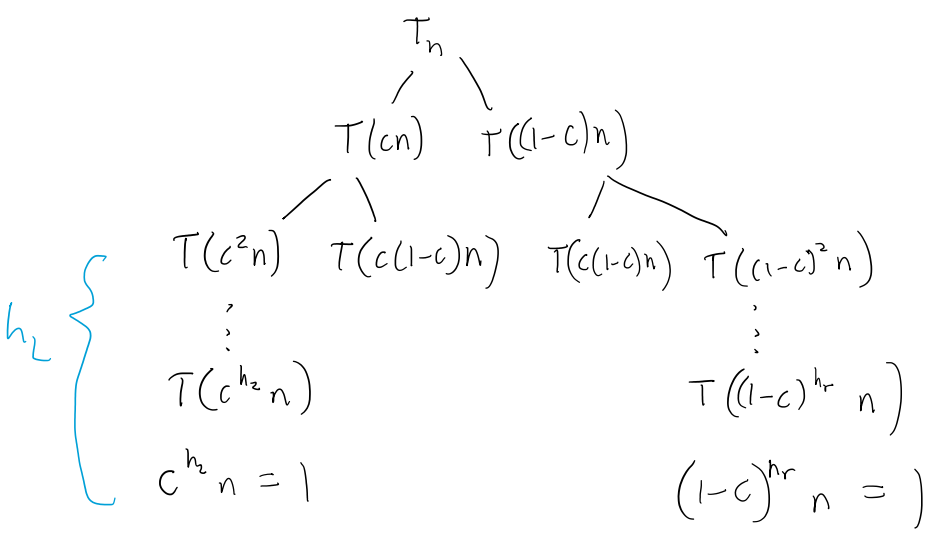
$T(n) = T(cn) + T((1-c)n) + n$

$T(1) = 0$

Find: h_L = height of left subtree

h_r = height of right subtree

* $h_r > h_L$



Solve for h_L

$c^{h_L} = \frac{1}{n}$

$h_L = \log_c n$

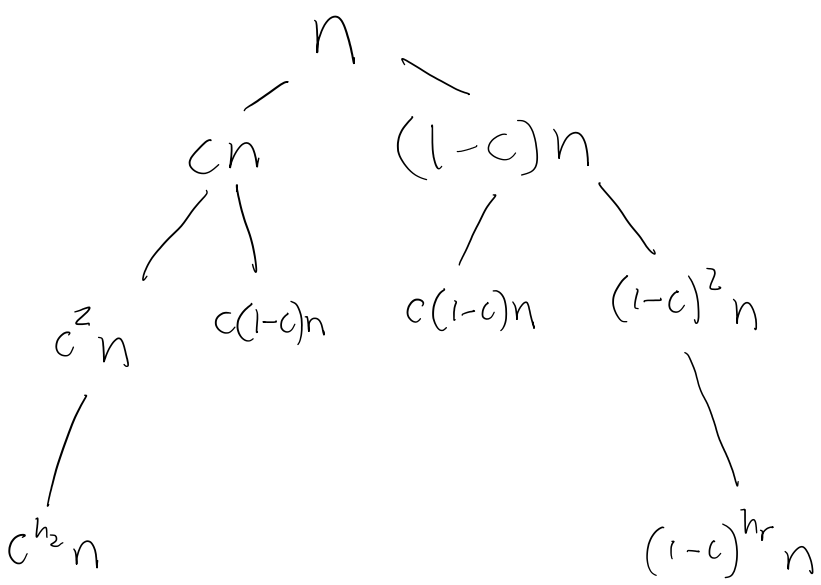
$h_L = \log_{1/c} n$

Solve for h_R

$(1-c)^{h_r} = \frac{1}{n}$

$h_r = -\log_{(1-c)} n$

$h_r = \log_{1/1-c}$



$c^2n + 2c(1-c)n + (1-c)^2n$

$= c^2n + 2cn - 2c^2n + n - 2cn + c^2n = n$

$h_r \cdot n = n\log_{1/(1-c)} n$

$= n\log_2 n \cdot \frac{1}{\log_2 1/1-c}$

$= \partial(n\log_2 n)$

Question 7

Thursday, March 25, 2021 11:30 PM

Question 7

a) erase (38)

Start @ negative infinity. $-\infty < 38$, therefore go down and right and the next value we get is 17. $17 < 38$. Go down and right the next value we get is 38. $38 = 38$ and that is the value we want to erase. The key node is now 38. Go down the column and change the pointers for previous & next, the current column is now removed

$-\infty$ — — — — $+\infty$

$-\infty$ — 17 — — $+\infty$

$-\infty$ — 17 — 42 $+\infty$

$-\infty$ — 17 — 42 $+\infty$

$-\infty$ 12 17 — 42 $+\infty$

$-\infty$ 12 17 20 42 $+\infty$

b) insert (48, a)

a represents the number of heads we get. I landed on tails on my 1st attempt therefore 48 will only be on the bottom row of the colum. We start with negative infinity, $-\infty < 48$ so we move down the column over to 17. $17 < 48$, so we move down to 42. $42 < 48$ and we set 42 to key node bc it is the greatest value in the skip list that < 48. We move onto the next column. $\infty > 48$ and it continues to be until the last list.

c) insert (24, b)

'b' here signifies the # of tails we get. I land on head first attemp. We take the 24 at the bottom of the column. $-\infty < 24$ so we go down and over to 17. The next values are $+\infty$, 48, 42. All are > 24. Therefore: 17 is the key node. As we go down the column, we see that 20 is the largest in the skip list thats < 24. 20 becomes the key node. The next value is 42 which is > 20. So we know that we can insert the new column between 20 and 42.

d) erase (42)

We start at $-\infty$, We know that $-\infty < 42$, so we set that as the key node. We then go down and over to 17. $17 < 42$ so it becomes keynod. We then go down & over to 42. $42 = 42$ so now 42 is the key node. The next value is $+\infty$ and that is > 42. Therefore 42 stays as key node. When we alter the prev & next pointers while looking @ the 42 column we see that the whole column of 42 gets deleted.