CSCE 221 Cover Page
Homework Assignment #2
Due March 27 at 23:59 pm to eCampus

First Name Jialu Last Name Zhao UIN 426000712

User Name zhaojialu123 E-mail address zhaojialu123@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	web sources	
People		
Web pages (provide URL)	https://www.cise.ufl.edu/class/cot3100fa07/quicksort_analysis.pdf	http://www.cs.princeto
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 Jialu Zhao Date 03/16/2017

Homework 2

due March 27 at 11:59 pm to eCampus.

1.(15 points) Write a recursive function that counts the number of nodes in a singly linked list. Write a recurrence relation that represents your algorithm. Solve the relation and obtain the running time of the algorithm in Big-O.

recursive function:

```
int count(node* Iter)
{
if(Iter == NULL)
return 0;
return 1+count(Iter->next);
}
```

recurrence relations:

$$F_n = F_{n-1} + 1$$

initial value:

$$F_0 = 0$$

solve the relation:

$$F_n = F_{n-1} + 1 = F_{n-2} + 1 + 1 = F_{n-3} + 1 + 1 + 1 = ... = F_0 + 1 + 1 + ... + 1 = F_0 + n = 0 + n = n$$

running time:

$$O(n)$$

2.(15 points) Write a recursive function that finds the maximum value in an array of int values without using any loops. Write a recurrence relation that represents your algorithm. Solve the relation and obtain the running time of the algorithm in Big-O.

recursive function:

```
int findMaximum(int array[], int low, int high)
{
if(low == high)
return array[low];
return max(array[low],findMaximum(array,low+1,high));
}
```

note:

low = 0;

high = the size of array - 1;

recurrence relations:

$$F_n = F_{n-1} + 1;$$

00

Note:F_n represents the times of do max function

initial value:

$$F_0 = 0$$

solve the relation:

$$F_n = F_{n-1} + 1 = F_{n-2} + 1 + 1 = F_{n-3} + 1 + 1 + 1 = ... = F_0 + 1 + 1 + ... + 1 = F_0 + n = 0 + n = n$$

running time:

$$O(n)$$

3.(10 points) What data structure is most suitable to determine if a string s is a palindrome, that is, it is equal to its reverse. For example, "racecar" and "gohan gasalamiimalasagnahog" are palindromes. Justify your answer. Use Big-O notation to represent the efficiency of your algorithm.

Stack is most suitable to determine if a string s is a palindrome.

Algorithm:

```
step 1:push every characters in the string to stack -- O(n)
step 2:pop every elements on the stack to a new string -- -- O(n)
step3: if new string is same with the old string, this string is palindrome -- O(n)
```

So the efficiency of this Algorithm:

$$O(n)$$

4.(10 points) Describe how to implement the stack ADT using two queues. What is the running time of the push and pop functions in this case?

Solution:

Use two queues Q1 and Q2, where Q1 stores elements and Q2 is used for auxiliary.

```
bool isEmpty() { return Q1.isEmpty();} --- O(1)
int size() { return Q1.size();} --- O(1)
T top()  --- O(n)
{
while(!Q1.isEmpty())
Q2.enqueue(Q1.dequeue()); // use another queue to help change the sequence
T elem = Q2.first();
while(!Q2.isEmpty())
Q1.enqueue(Q2.dequeue()); // change the sequence back
return elem;
}
T pop()  --- O(n)
{
while(!Q1.isEmpty())
Q2.enqueue(Q1.dequeue()); // use another queue to help change the sequence
T elem = Q2.dequeue();
while(!Q2.isEmpty())
Q1.enqueue(Q2.dequeue()); // change the sequence back
return elem;
}
void push(T elem) { Q1.enqueue(elem);}  --- O(1)
```

5.(10 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.

(a)Best-case analysis:

$$O(nlogn)$$

The best case of quicksort occurs when the pivot we pick happens to divide the array into two exactly equal parts, in every step.Thus we have k=n/2 and n-k=n/2 for the original array of size n.

$$T(n) = 2T(\frac{n}{2}) + an$$

$$= 2(2T(\frac{n}{4}) + a\frac{n}{2}) + an$$

$$= 4T(\frac{n}{4}) + 2an$$

$$= 2^2T(\frac{n}{2^2}) + 2an$$

.....

$$= 2^iT(\frac{n}{2^i}) + ian$$

this recurrence will stop when i = logn:

$$= nT(1) + anlogn$$

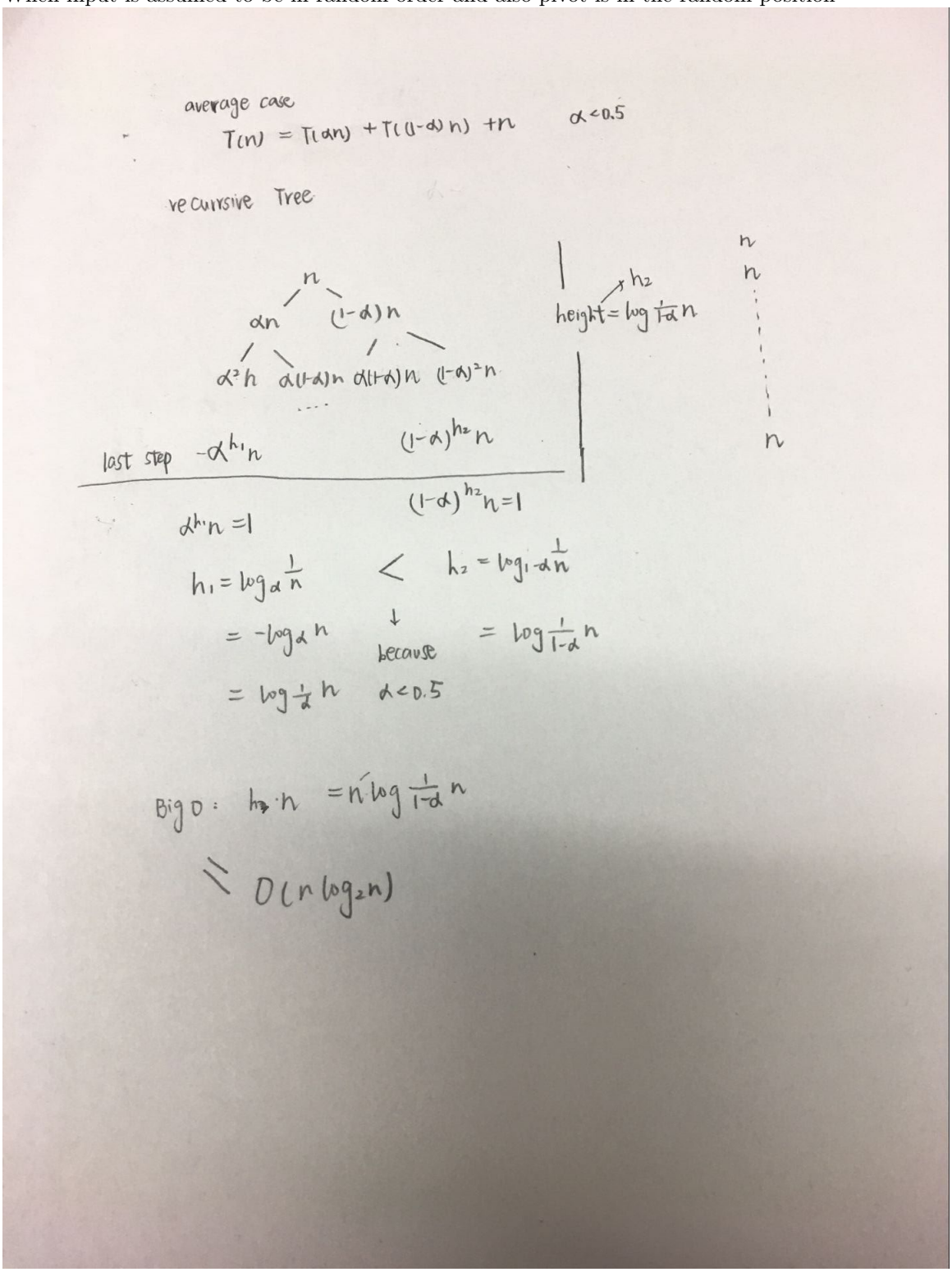
then we got the result:

$$O(nlogn)$$

(b)Average-case analysis

$O(nlogn)$

When input is assumed to be in random order and also pivot is in the random position



(c)Worst-case analysis

$O(n^2)$

When the pivot happened to be the least element of the array, so that we had k=1 and n-k=n-1. In such case, we have:

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

Now let us anaysis the time complexity of quick sort in such a case:

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

$$= T(1) + (T(n-2) + T(1) + a(n-1)) + an$$

$$= 2T(1) + T(n-2) + an + a(n-1)$$

$$= 3T(1) + T(n-3) + an + a(n-1) + a(n-2)$$

.....

$$= iT(1) + T(n-i) + a(n + (n-1) + (n-2) + ...(n-i+1))$$

this recurrence will stop when i = n:

$$= nT(1) + T(0) + a(n + (n-1) + (n-2) + ... + 2 + 1)$$

$$= nT(1) + T(0) + a\frac{n(n+1)}{2}$$

which we can got the result:

$$O(n^2)$$

6.(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.

The merge sort requires $O(nlogn)$ comparisons, where n is the length of an array to be sorted. There are no best-case or worst-case for the merge sort. Because the merge sort needs a duplicate copy of the array being sorted therefore in many situcations it can be impractical.

two methods:

(a)Iterativing method:

$$T(n) = 2T(\frac{n}{2}) + an$$

$$= 2(2T(\frac{n}{4}) + a\frac{n}{2}) + an$$

$$= 4T(\frac{n}{4}) + 2an$$

$$= 2^2T(\frac{n}{2^2}) + 2an$$

.....

$$= 2^iT(\frac{n}{2^i}) + ian$$

this recurrence will stop when i = logn:

$$= nT(1) + anlogn$$

then we got the result:

$$O(nlogn)$$

(b)Master Theorem

$$T(n) = 2T(\frac{n}{2}) + an$$

$$a = 2, b = 2, f(n) = an;$$

according to master theorem:

$$T(n) = a(n/b) + f(n)$$

$$f(n) = \Theta(nlog_2 2) = \Theta(n)$$

so we can got the result:

$$O(nlogn)$$

7.(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:

counterexample:

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:

Proof: For this question, we consider from the opposite way: we assume that there is an external node has a black internal sibling and let h be the black depth of the external node. Then the black depth of the external node's black, internal sibling would be at least h+1 which violate the depth rule of a red-black tree.

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

True:

Proof:

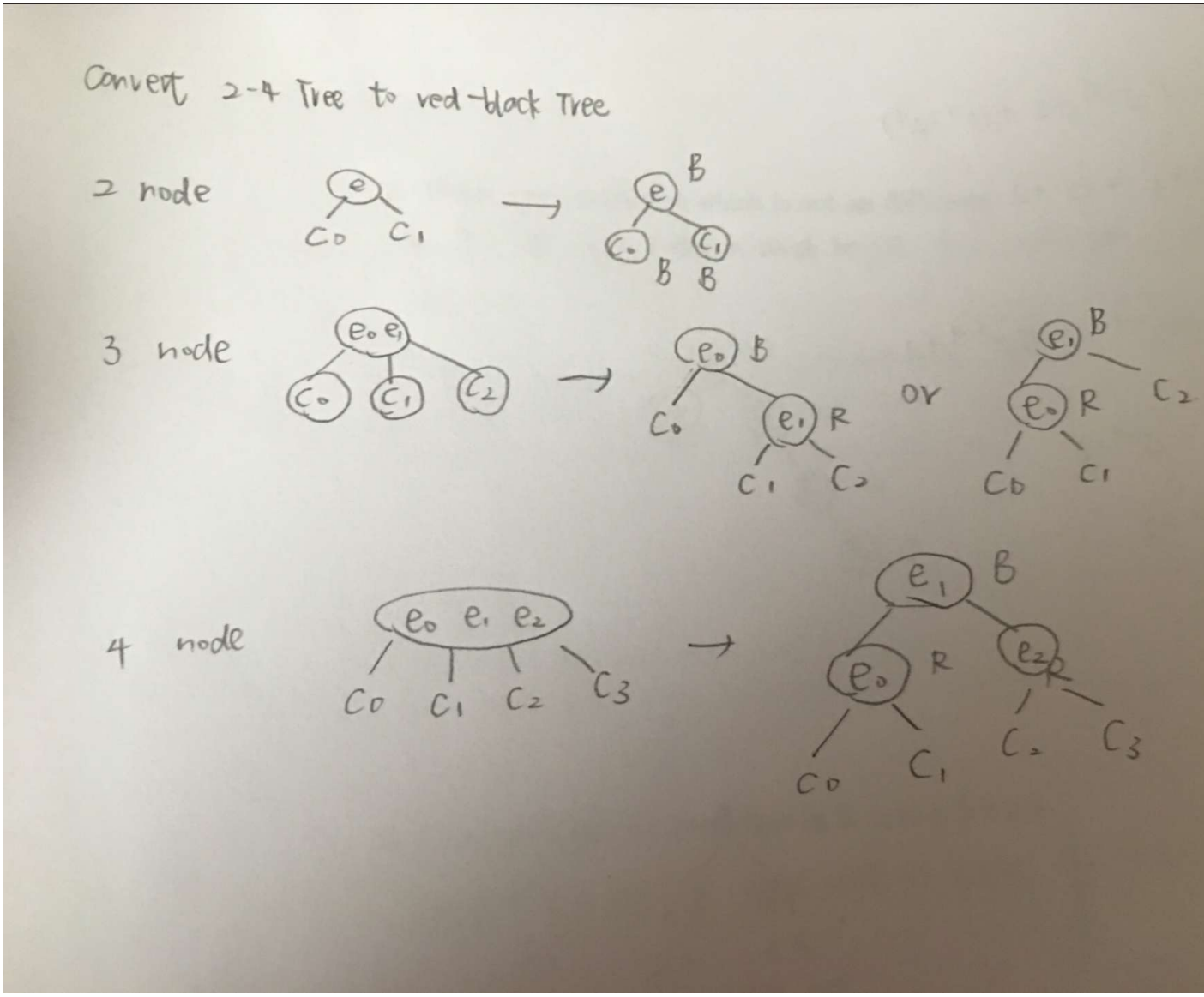
The given one means that given a red-black tree T, there is an unique 2-4 tree T' associated with T. Every node in the red-black tree with two red children is uniquely represented as a 4 -node in T', each node with one red child is uniquely represented as a 3-node in T', and a node with no red children is uniquely represented as a 2-node in T'.

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

False:

counter example:

A 3-node has two different possible representations in a red-black tree.



8.(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.

AVL tree:

$$h < 1.44log(n+1)$$

so the worst case height of T is 24.

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

(2,4) tree:

$$h <= log(n+1)$$

so the worst case height of T is 17.

(c)T is a red-black tree.
red-black tree:

$$h <= 2log(n+1)$$

so the worst case height of T is 33.

(d)T is a binary search tree.
T is a binary search tree.
The worst case is linear so 100,000.

9.(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.