# CSCE 221 Cover Page

Homework Assignment #2

First Name Peng Last Name Li UIN 822003660

User Name abc E-mail address billlipeng@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 on Aggie Honor System Office website: `http://aggiehonor.tamu.edu/`

| | | | | |
|---|---|---|---|---|
| Type of sources | | | | |
| People | | | | |
| Web pages (provide URL) | | http://goo.gl/9vygMr | | |
| Printed material | CSCE221 Slide | | | |
| 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 Peng Li Date Oct 17, 2014

**Homework 2**

**due October 17 at 11:59 pm.**

1. (20 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.

```
int cnt(ListNode* node){
    if (node == NULL)
        return 0;
    else
        return 1+cnt(node->next);
    }
```

```
NP(n) = NP(n-1) + 1
NP(n-1) = NP(n-2) + 1
...
NP(1) = 1   =>
NP(n) = n = O(n)
```

2. (20 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.

```
int find_max(int A[] , int size, int max) {
    if(size==1){
        if(A[0]>max)  return A[0];
        else    return max;
    }
    else if(size>1){
        if(A[size-1]>max)
            return find_max(A , size-1, A[size-1]);
        else
            return find_max(A , size-1, max);
    }
    else{
        cout << "range error"<<endl;
        return -1;
    }
}
```

```
NP(n) = NP(n-1) + 1
NP(n-1) = NP(n-2) + 1
...
NP(1) = 1   =>
NP(n) = n = O(n)
```

3. (15 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 "gohangasalamiimalasagnahog" are palindromes. Justify your answer. Use Big-O notation to represent the efficiency of your algorithm.

Suppose the length of the string is n. Use two stacks, one size is n, the other one size is ceiling of n/2. Push the whole string into a stack of char. This costs O(n) times. Next pop n/2 characters from the stack and push them into the second stack. This costs O(n/2). Finally pop characters from two stacks at the same time and compare them. This costs O(n). Note if n is odd, the middle character should be handled specially by pop itself first. Totally the complexity is O(n).

4. (15 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?

```
void push(T obj){

    Q1.enqueue(obj);
}
T pop(){
    queue Q2;
    while(!Q1.size()>1){ // copy n-1 elements from Q1 to Q2
        Q2.enqueue(Q1.dequeue());
    }
    T obj= Q1.dequeue();  // pop last element in Q1
    while(!Q2.isEmpty()){ // copy Q2 to Q1
        Q1.enqueue(Q2.dequeue());
    }
    return obj;
}
```

Push:
Enqueue all elments in queue1. Running time is O(1).

Pop:
Create a new queue Q2. While size of Q1 is greater than 1, pipe dequeued items from Q1 into Q2.
Dequeue and return the last item of Q1, then copy Q2 to Q1. Two copys take 2n operations to dequeue and 2n operation to enqueue. So running time is f(n) = 4n = O(n).

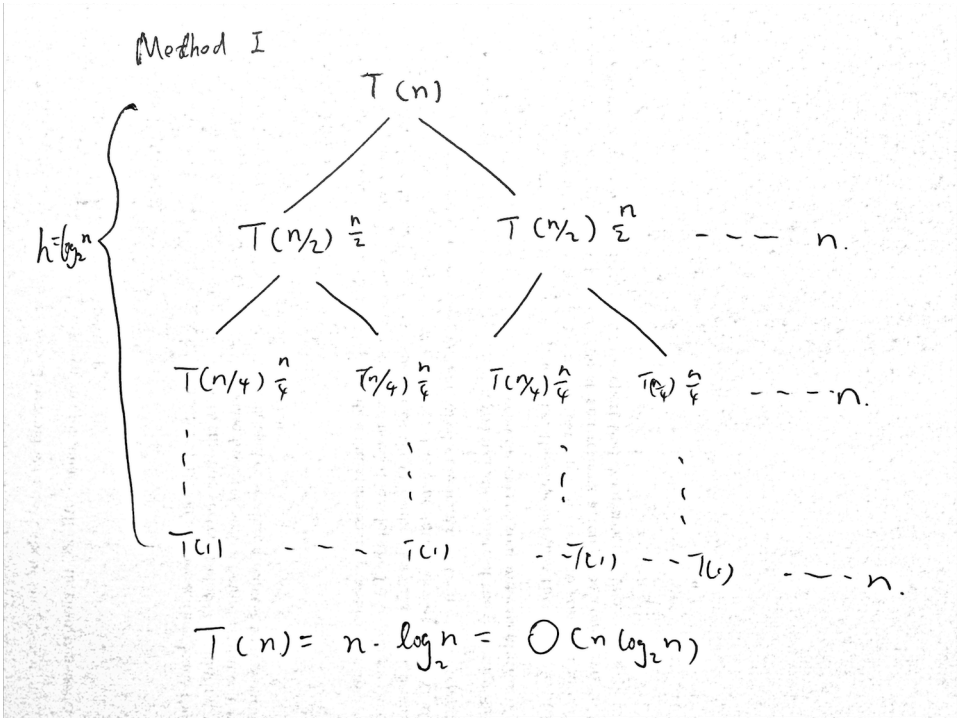
5. (15 points) What is the best, worst and average running time of quick sort algorithm? Provide arrangement of the input and the selection of the pivot point at every case. Provide a recursive relation and solution for each case.

```
Suppose the size of the data is n.

Best(The pivot is the middle of all the data, thus same number
of elements on each sides of the pivot):
QS(n) = QS(n/2) + QS(n/2) + n
QS(n) = 2 * QS(n/2) + n
QS(1) = 0
====>
QS(n) = O(n*log(n))



Worst(when the pivot is the minimum or maximum element in the
data set):

QS(n) = QS(n-1) + QS(1) + n
QS(n) = QS(n-2) + (n-1) + n
QS(n) = QS(n-3) + (n-2) + (n-1) + n
QS(n) = 0
====>
QS(n) = n(n+1)/2 = O(n^2)


Average(suppose the left side of pivot has å*n elements and
right side of pivot has (1-å)*m elements):
QS(n) = QS(å*n) + QS((1-å)*n) + n      å<0.5
====>
QS(n) = O(n*log(n))
```

6. (15 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.

There is no best case or worst case for the merge sort. Using merge sort need increase memory because it needs a duplicate copy of the array being sorted.

Method I:




Method II:

```
T(n) = 2 T(n/2) + θ(n)  if n>1.
T(1) = θ(1).

According to the Master Method:
T(n) = aT(n/b) + f(n)
Then T(n) can be bounded asymptotically as follows:
T (n) = θ(n^(log b (a))*log(n)) if f (n) = θ(n^(log b (a)) )

We have a = 2, b = 2, f(n) = θ(n). Then log2 = 1 and n^1 = n.
So the merge sort requires T(n) = nlog(n) = O(nlog(n))
comparisons.
```