CSCE 221 Cover Page
Homework Assignment #1
Due Sep. 19 at midnight to CSNet

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: Aggie Honor System Office

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 Peng Li Date Sep 19 2014

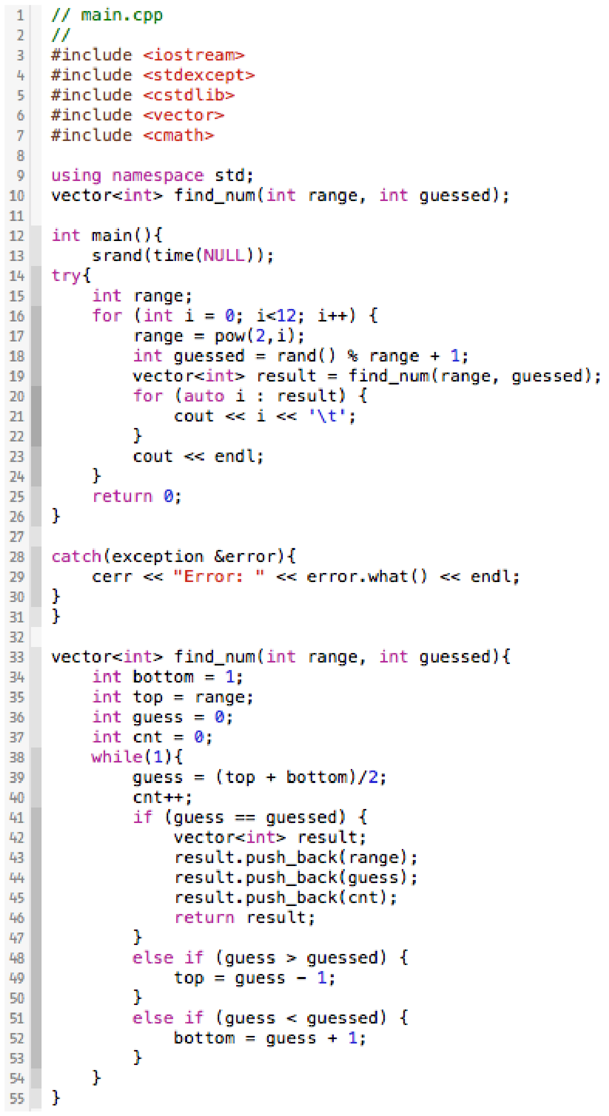
Type the solutions to the homework problems listed below using the LYX/LaTeX program, see the class webpage for more information about its installation and tutorial.

1. (50 points) There are two players. The first player selects a random number between 1 and 32 and the other one (could a computer) needs to guess this number asking a minimum number of questions. The first player responses possible answers to each question are:

   - *yes* – the number is found
   - *lower* – the number to be guessed is smaller than the number in the question
   - *higher* – the number to be guessed is greater than the number in the question

   Hint. The number of questions in this case (range $[1, 32]$) should not exceed 6.

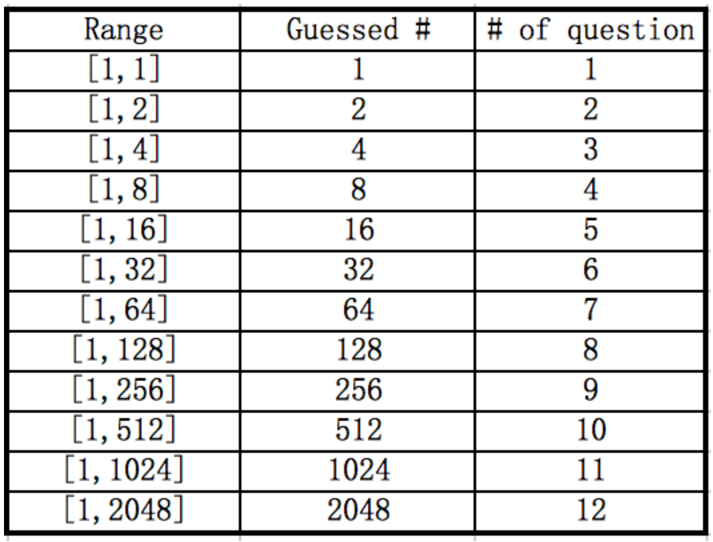
   (a) Write a C++ code for this algorithm and test it using the following input in ranges from 1 to: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048. Be sure that your program throws an exception in case of an invalid dialog entry during the computations.
   Note that the "brute force" algorithm is not accepted.



```
// main.cpp
//
#include <iostream>
#include <stdexcept>
#include <cstdlib>
#include <vector>
#include <cmath>

using namespace std;
vector<int> find_num(int range, int guessed);

int main(){
    srand(time(NULL));
try{
    int range;
    for (int i = 0; i<12; i++) {
        range = pow(2,i);
        int guessed = rand() % range + 1;
        vector<int> result = find_num(range, guessed);
        for (auto i : result) {
            cout << i << '\t';
        }
        cout << endl;
    }
    return 0;
}

catch(exception &error){
    cerr << "Error: " << error.what() << endl;
}
}

vector<int> find_num(int range, int guessed){
    int bottom = 1;
    int top = range;
    int guess = 0;
    int cnt = 0;
    while(1){
        guess = (top + bottom)/2;
        cnt++;
        if (guess == guessed) {
            vector<int> result;
            result.push_back(range);
            result.push_back(guess);
            result.push_back(cnt);
            return result;
        }
        else if (guess > guessed) {
            top = guess - 1;
        }
        else if (guess < guessed) {
            bottom = guess + 1;
        }
    }
}
```

Figure 1:

i.

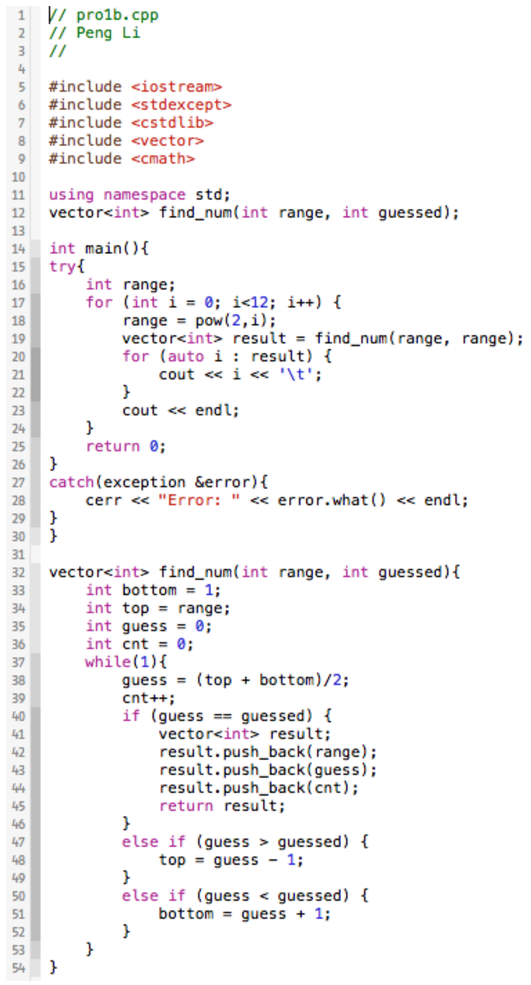


Range	Guessed #	# of question
[1, 1]	1	1
[1, 2]	2	2
[1, 4]	4	3
[1, 8]	8	4
[1, 16]	16	5
[1, 32]	32	6
[1, 64]	64	7
[1, 128]	128	8
[1, 256]	256	9
[1, 512]	512	10
[1, 1024]	1024	11
[1, 2048]	2048	12

Figure 2:

ii.

(b) Tabulate the output results in the form (range, guessed number, number of comparisons required to guess it) in a given range using an STL vector. Plot the number of questions returned by your algorithm when the number to be guessed is selected as $n = 2^k$, where $k = 1, 2, \ldots, 11$. You can use any graphical package (including a spreadsheet).



```cpp
// pro1b.cpp
// Peng Li
//

#include <iostream>
#include <stdexcept>
#include <cstdlib>
#include <vector>
#include <cmath>

using namespace std;
vector<int> find_num(int range, int guessed);

int main(){
try{
    int range;
    for (int i = 0; i<12; i++) {
        range = pow(2,i);
        vector<int> result = find_num(range, range);
        for (auto i : result) {
            cout << i << '\t';
        }
        cout << endl;
    }
    return 0;
}
catch(exception &error){
    cerr << "Error: " << error.what() << endl;
}
}

vector<int> find_num(int range, int guessed){
    int bottom = 1;
    int top = range;
    int guess = 0;
    int cnt = 0;
    while(1){
        guess = (top + bottom)/2;
        cnt++;
        if (guess == guessed) {
            vector<int> result;
            result.push_back(range);
            result.push_back(guess);
            result.push_back(cnt);
            return result;
        }
        else if (guess > guessed) {
            top = guess - 1;
        }
        else if (guess < guessed) {
            bottom = guess + 1;
        }
    }
}
```

Figure 3:

i.

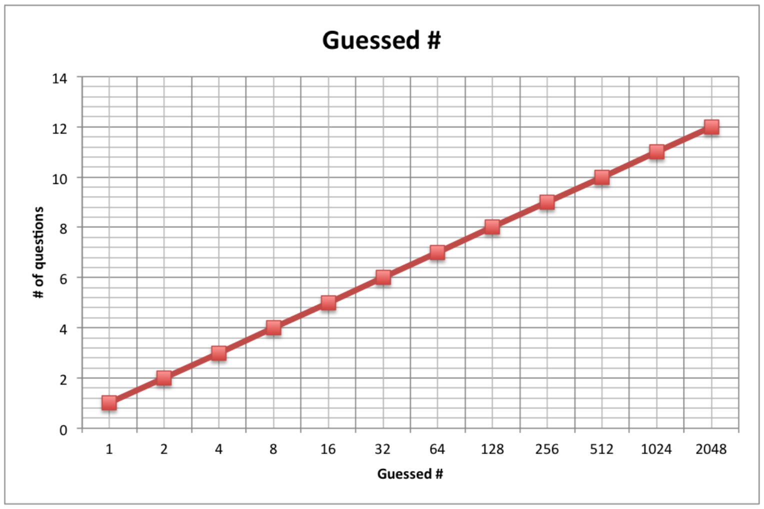


Figure 4:

(c) Provide a mathematical formula/function which takes n as an argument, where n is equal to the upper value of the testing ranges, and returns as its value the maximum number of questions for the range $[1, \ldots, n]$. Does your formula match computed output for a given input? Justify your answer.

i. The formular is max # = $\log_2(\text{n}) + 1$

ii.

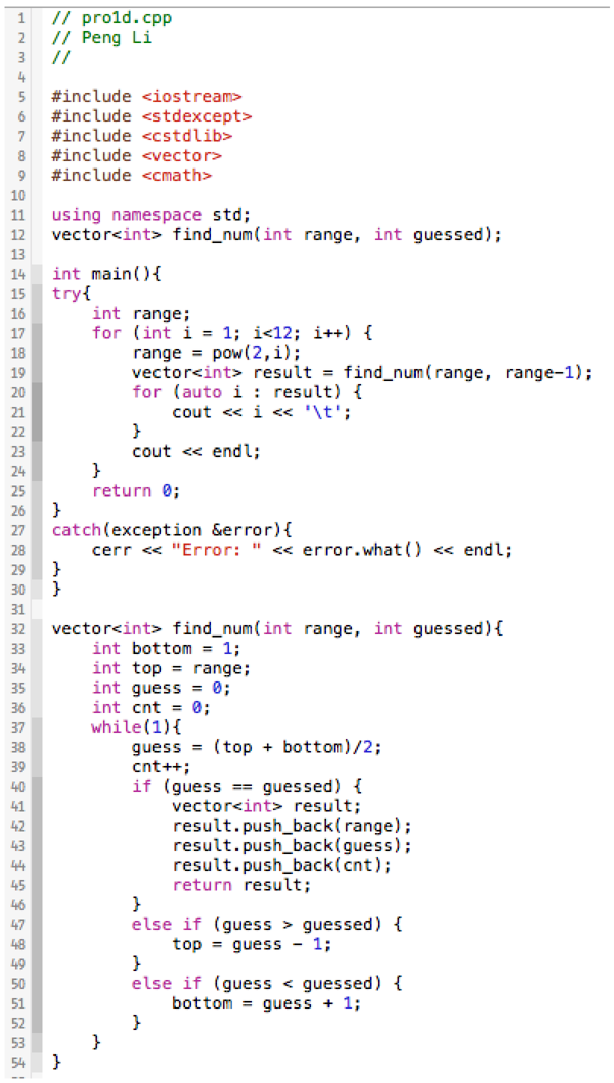
Range [1..n]	True answer n	# guesses/comparison	Result of formula in (c)
[1,1]	1	1	1
[1,2]	2	2	2
[1,4]	4	3	3
[1,8]	8	4	4
[1,16]	16	5	5
[1,32]	32	6	6
[1,64]	64	7	7
[1,128]	128	8	8
[1,256]	256	9	9
[1,512]	512	10	10
[1,1024]	1024	11	11
[1,2048]	2048	12	12

(d) How can you modify your formula/function if the number to be guessed is between 1 and N, where N is not an exact power of two? Test your program using input in ranges starting from 1 to $2^k - 1$, $k = 2, 3, \ldots 11$.

i. The fomular is # of question = $\lfloor log_2(n) \rfloor + 1$

ii.

Range [1..N]	True answer N	# guesses/comparison	Result of formula in (d)
[1,1]	1	1	1
[1,3]	3	2	2
[1,7]	7	3	3
[1,15]	15	4	4
[1,31]	31	5	5
[1,63]	63	6	6
[1,127]	127	7	7
[1,255]	255	8	8
[1,511]	511	9	9
[1,1023]	1023	10	10
[1,2047]	2047	11	11



```cpp
// pro1d.cpp
// Peng Li
//

#include <iostream>
#include <stdexcept>
#include <cstdlib>
#include <vector>
#include <cmath>

using namespace std;
vector<int> find_num(int range, int guessed);

int main(){
try{
    int range;
    for (int i = 1; i<12; i++) {
        range = pow(2,i);
        vector<int> result = find_num(range, range-1);
        for (auto i : result) {
            cout << i << '\t';
        }
        cout << endl;
    }
    return 0;
}
catch(exception &error){
    cerr << "Error: " << error.what() << endl;
}
}

vector<int> find_num(int range, int guessed){
    int bottom = 1;
    int top = range;
    int guess = 0;
    int cnt = 0;
    while(1){
        guess = (top + bottom)/2;
        cnt++;
        if (guess == guessed) {
            vector<int> result;
            result.push_back(range);
            result.push_back(guess);
            result.push_back(cnt);
            return result;
        }
        else if (guess > guessed) {
            top = guess - 1;
        }
        else if (guess < guessed) {
            bottom = guess + 1;
        }
    }
}
```

Figure 5:

iii.

(e) Use Big-O asymptotic notation to classify this algorithm.

i. $O(log_2(n))$

Submit for grading an electronic copy of your code and solutions to the questions above.

Points Distribution

(a) (b) (5 pts) # guesses in a table; (5 pts) A plot in the report; (15 pts) Program code using STL vector and exception

(c) (5 pts) A math formula of n; (5 pts) Formula results compared to # guesses

(d) (5 pts) A math formula of N; (5 pts) Program code (and the second table)

(e) (5 pts) A big-O function

Submit an electronic copy of your code and results of all your experiments for grading.

2. (10 points) **(R-4.7 p. 185)** The number of operations executed by algorithms A and B is $8n\log n$ and $2n^2$, respectively. Determine n_0 such that A is better than B for $n \geq n_0$.

   (a) $8nlogn < 2n^2$, if n $\neq 0$, $4logn < n$, $logn^4 < n$, $n^4 < 2n$, $n^3(n-2) > 0$, $n > 2$. So $n_0 = 3$. Hence for any n greater or equal to 3, A is better than B.

3. (10 points) **(R-4.21 p. 186)** Bill has an algorithm, find2D, to find an element x in an $n \times n$ array A. The algorithm find2D iterates over the rows of A, and calls the algorithm arrayFind, of Code Fragment 4.5, on each row, until x is found or it has searched all rows of A. What is the worst-case running time of find2D in terms of n? What is the worst-case running time of find2D in terms of N, where N is the total size of A? Would it be correct to say that Find2D is a linear-time algorithm? Why or why not?

   (a) Worst case is O(n^2). In the worst case, it will check every element in the matrix. The last element is the target. Worst case in terms of N is O(N). This is the linear-time algorithm since the running time is linear to the input function.

4. (10 points) **(R-4.39 p. 188)** Al and Bob are arguing about their algorithms. Al claims his $O(n\log n)$-time method is always faster than Bob's O(n^2)- time method. To settle the issue, they perform a set of experiments. To Al's dismay, they find that if $n < 100$, the O(n^2)-time algorithm runs faster, and only when $n \geq 100$ is the O($n\log n$)-time one better. Explain how this is possible.
   One possible situtation is that: Al: $f(n) = nlog(n) + 96^2 = nlog(n) + 9216 = O(nlog(n))$ Bob: $f(n) = n^2 = O(n^2)$. When n<100, say n = 99, Al = 99xlog(99)+96^2=9872.306, Bob = 99x99=9801. So Bob is better than Al. When n $\geqslant$ 100, say n=100, Al = 100xlog(100)+96^2 = 9880.385, Bob = 100 x 100 = 100,00. So Al is better than Bob.

5. (20 points) Find the running time functions and classify the algorithms presented in the exercise 4.4, p. 187.

```
Algorithm Ex1(A):
  Input: An array A storing $n \geq 1$ integers.
  Output: The sum of the elements in A.
$s \leftarrow A[0]$
for $i \leftarrow 1$ to $n-1$ do
   $s \leftarrow s + A[i]$
return s
```

$f(n) = (n-1) \times 2 + 1 = n - 1 = O(n)$

```
Algorithm Ex2(A):
  Input: An array A storing $n \geq 1$ integers.
  Output: The sum of the elements at even cells in A.
$s \leftarrow A[0]$
for $i \leftarrow 2$ to $n-1$ by increments of 2 do
   $s \leftarrow s + A[i]$
return s
```

$f(n) = \frac{n-2}{2} \times 2 + 1 = n - 1 = O(n)$

```
Algorithm Ex3(A):
   Input: An array A storing $n \geq 1$ integers.
   Output: The sum of the prefix sums in A.
$s \leftarrow 0$
for i ←0 to $n-1$ do
   $s \leftarrow s + A[0]$
   for $j \leftarrow 1$ to $i$ do
      $s \leftarrow s + A[j]$
return s
```

$f(n) = 2n + \frac{2(1-2^{n-1})}{1-2} \times n = 2n + 2^n n - 2n = 2^n n = O(n \cdot 2^n)$

```
Algorithm Ex4(A):
   Input: An array A storing $n \geq 1$ integers.
   Output: The sum of the prefix sums in A.
$s \leftarrow A[0]$
```

```
$t \leftarrow s$
for $i \leftarrow 1$ to $n-1$ do
    $s \leftarrow s + A[i]$
    $t \leftarrow t + s$
return $t$
```

$f(n) = 2 + 4(n-1) = 4n - 2 = O(n)$