CSCE 221 Cover Page
Homework #1
Due February 14 at midnight to CSNet

First Name: Chris Last Name: Comeaux UIN622006681

User Name cmc236 E-mail address: cmc236@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. According to the University Regulations, Section 42, scholastic dishonesty are including: acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion read more: Aggie Honor System Office

Type of sources			
Web Page	http://www.chegg.com/homework-help/questions-and-answers/write-function-pseudo-code-takes-input-object-vector-type-removes-element-rank-k-constant–q6608397?trackid=4639cf7b&strackid=2b8a5eef&ii=1		
Web pages (provide URL)	https://www.topcoder.com/community/data-science/data-science-tutorials/binary-search/ https://en.wikipedia.org/wiki/Binary_search_algorithm		
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 Date

Type the solutions to the homework problems listed below using preferably 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 upper bound (like 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) You should implement in C++ the first interactive version of the problem. Your program must allow the user to guess (with input from keyboard) the solution kept by the computer, and test it with a given upper bound (32) from keyboard and .Note that the "brute force" method is not accepted.

```
// Homework Problem 1.A
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>

using namespace std;

int main()
{
        int max, guess;
        int num_trials=1, max_trials;

        //ask for the range of numbers
        cout <<"Enter the upperbound for the range you wish to guess from.\n";
        cin >> max;

        //set up random number
        srand(time(0));
        int number = rand()%max +1;

        //find the max number of guesses
        max_trials = floor(log2(max)+1);
        cout<< "It should not take more than " <<max_trials <<" guess(es).\n\n";

        //set up first prompt
        cout <<"First guess: ";
        while(cin>>guess && guess != number) //continue loop while reciving
        {
//input and the guess is not correct
                if(guess > number)
                        cout <<"Lower\n\n";
                else
                        cout <<"Higher\n\n";
                // if user guesses too many times throw fail
                if(++num_trials > max_trials)
        {
            cout<<"It should not have taken you that many guesses.\n"
             <<"You Lose...\n";
            return 1;
        }
        cout <<"Next guess: "; //prompt again
    }

    cout <<"Yes\n\n";
    cout <<"It took you " <<num_trials <<" guesses.\n";
return 0;
}
```

(b) You should implement in C++ the second non-interactive version of the problem to guess the solution with your own algorithm, and test it using the following upper bound from 1 to: 1, 2, 4, 8, 16, 32, 64, 128, 256, Be sure that your program throws an exception in case of an invalid dialog entry during the computations.

```
// Homework1 Problem 1.B
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>

using namespace std;

int search(int, int);

int main()
{
    try
    {
        int max, max_trials;

        //ask for the range of numbers
        cout <<"Enter the upperbound for the range"
                 <<"you wish the computer to guess from.\n";
        cin >> max;

        //input validadtion
        if(max<=0)
            throw 1;

        //set up random number
        srand(time(0));
        int number = rand()%max +1;

        //find the max number of guesses
        max_trials = floor(log2(max)+1);
        cout<< "It should not take more than " <<max_trials
            <<" guess(es).\n";

        int num_trials = search(max, number);

        cout <<"Yes\n";
```

```
                cout <<"It took the computer " <<num_trials
                         <<" to guess the correct number\n";
        }
        catch(int)
        {
                cerr << "Exception: Please enter a resonable upperbound\n";
        }

return 0;
}

int search(int max, int number)
{
        int trials = 1;
        int low = 1;
        int mid = floor((max+low)/2.0); //midpoint of the data for first guess

        cout<<"First guess : "<<mid <<endl;

        while(mid!=number) // continue to search until the
        {
                if(mid > number) // number is in lower section
                {
                        max = mid-1;
                        cout <<"Lower\n";
                }
                else if(mid < number) //number is in upper section
                {
                        low = mid+1;
                        cout <<"Higher\n";
                }
                mid = floor((max+low)/2.0);
                cout <<"Next Guess: " <<mid <<endl;
                ++trials;
        }
        return trials;
}
```

(c) Your third version of the program in C++ must allow the user to set a target number, so that you can do controlled testing.

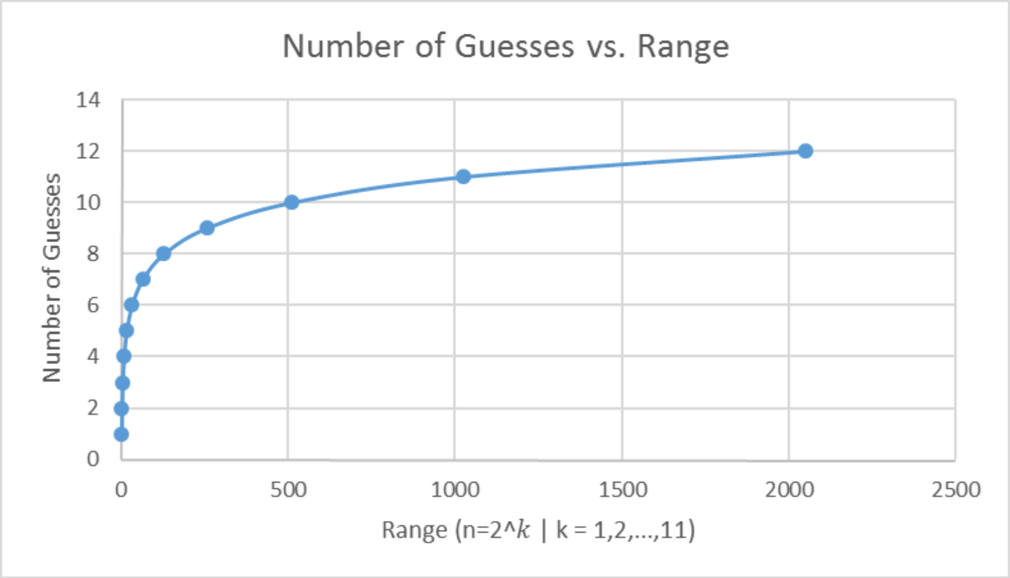
```
// Homework1 Problem 1.C
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>

using namespace std;

int search(int, int);

int main()
{
        try
        {
                int max, max_trials, number;

                //ask for the range of numbers
                cout <<"Enter the upperbound for the range"
                          <<"you wish the computer to guess from.\n";
                cin >> max;

                //input validadtion
                if(max<=0)
                        throw 1;

                //set up target number
                cout <<"Enter a target value between [1," <<max <<"] \n";
                cin >> number;

                //input validation
                if(number<1 || number>max)
                        throw 'e';

                //find the max number of guesses
                max_trials = floor(log2(max)+1);
                cout<< "It should not take more than " <<max_trials
                        <<" guess(es).\n";

                int num_trials = search(max, number);

                cout <<"Yes\n";
                cout <<"It took the computer " <<num_trials
                          <<" guess(es)to guess the correct number\n";
        }
        catch(int)
        {
                cerr << "Exception: Please enter a resonable upperbound\n";
        }
        catch(char)
        {
                cerr << "Exception: Please enter a number within the range"
                     <<"[1,upperbound]\n";
        }

return 0;
}

int search(int max, int number)
{
        int trials = 1;
        int low = 1;
        int mid = floor((max+low)/2.0); //midpoint of the data for first guess

        cout<<"First guess : "<<mid <<endl;

        while(mid!=number) // continue to search until the
        {
                if(mid > number) // number is in lower section
                {
                        max = mid-1;
                        cout <<"Lower\n";
                }
                else if(mid < number) //number is in upper section
                {
                        low = mid+1;
                        cout <<"Higher\n";
                }
                mid = floor((max+low)/2.0);
                cout <<"Next Guess: " <<mid <<endl;
                ++trials;
        }
        return trials;
}
```

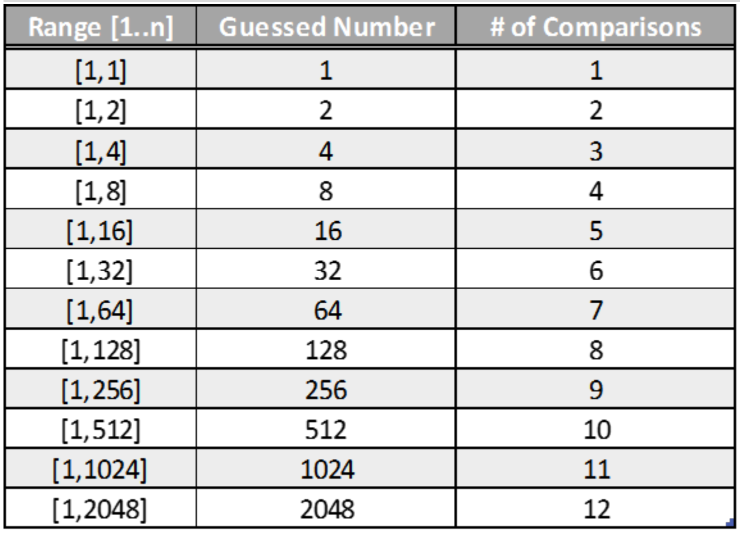
(d) For the report, you need to measure how many guesses the program 2 and 3 takes to find the numbers 2^n and $2^n - 1$ as sample input, not as the only valid input.

i. 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).

```
//Homework 1_D
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <vector>

using namespace std;

int search(int, int);

int main()
{
        try
        {
                int max, max_trials, number;

                //ask for the range of numbers
                cout <<"Enter the upperbound for the range"
                         <<"you wish the computer to guess from.\n";
                cin >> max;

                //input validadtion
                if(max<=0)
                        throw 1;

                //set up target number
                cout <<"Enter a target value between [1," <<max <<"] \n";
                cin >> number;

                //input validation
                if(number<1 || number>max)
                        throw 'e';

                //find the max number of guesses
                max_trials = floor(log2(max)+1);
                cout<< "It should not take more than " <<max_trials
                        <<" guess(es).\n";

                int num_trials = search(max, number);

                cout <<"Yes\n";
                cout <<"It took the computer " <<num_trials
                         <<" guess(es)to guess the correct number\n";
        }
        catch(int)
        {
                cerr << "Exception: Please enter a resonable upperbound\n";
        }
        catch(char)
        {
                cerr << "Exception: Please enter a number within the range"
                         <<" [1,upperbound]\n";
        }

return 0;
}

int search(int max, int number)
{
        int trials = 1;
        int low = 1;
        int mid = floor((max+low)/2.0); //midpoint of the data for first guess
        int guesses = 0;
        int range = max;// remember value to be out in resullts vector

        /*SEARCH*/
        cout<<"First guess : "<<mid <<endl;
        while(mid!=number) // continue to search until the
        {
                if(mid > number) // number is in lower section
                {
                        max = mid-1;
                        cout <<"Lower\n";
                }
                else if(mid < number) //number is in upper section
                {
                        low = mid+1;
                        cout <<"Higher\n";
                }
                mid = floor((max+low)/2.0);
                cout <<"Next Guess: " <<mid <<endl;
                ++trials;
        }

        //Tabulate results in STL vector
        //(range, guessed number, number of comparisons required to guess it)
        vector<int> results;
        results.push_back(range); //range
        results.push_back(number); //number trying to be guessed
        results.push_back(trials); //number of camparisons
```

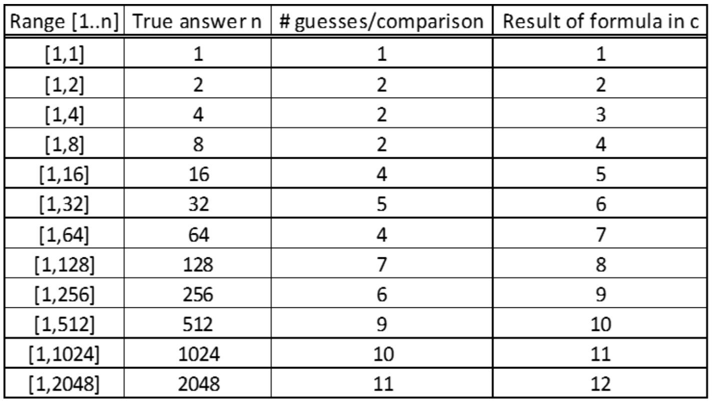
```
            return trials;
    }
```





Range [1..n]	Guessed Number	# of Comparisons
[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

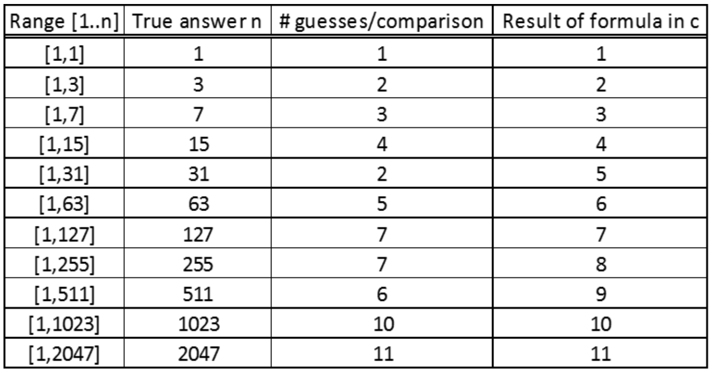
i. Provide a mathematical formula/function which takes n as an argument, where n is equal to the upper value of the testing ranges, and returns as its value the maximum number of questions for the range $[1, \ldots, n]$. Does your formula match computed output for a given input? Justify your answer.
Formula: $log_2(n) + 1$ No, the formula only outputs the max number of guesses. Depending on the input and number, it takes the algorithm different number of guesses every time.



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

i. 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 $2^k - 1$, $k = 2, 3, \ldots 11$.

Modified Formula: $\lfloor (log_2(N)) + 1 \rfloor$



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

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

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. (15 points) Write a C++ function using the STL string which can determine if a string s is a palindrome, that is, it is equal to its reverse. For example, “racecar” and “gohangasalamiimalasagnahog” are palindromes. Provide 7 test cases, including: the empty string, 4 string which are palindromes and two string which are not palindromes. Write the running time function in terms of n, the length of the string, and its big-O notation to represent the efficiency of your algorithm. Submit an electronic copy of your code and results of all your experiments for grading.

The running time of the is_palindrome is f(n) = 3(n/2)+3 which is O(n)

```
//Homework 2
#include <iostream>
#include <string>
using namespace std;

bool is_palindrome(string, int);


int main()
{
        /*TEST CASES*/
        //empty string
        string str1;

        //Palidromes
        string str2 = "racecar";
        string str3 = "deified";
        string str4 = "detartrated";
        string str5 = "gohangasalamiimalasagnahog";

        //Not Palidromes
        string str6 = "palidromes";
        string str7 = "homework";

        //Test each case
        if(is_palindrome(str1, str1.size()))
                cout <<"The empty string is a palidrome\n";
        else
                cout <<"The empty string is not a palidrome\n";

        if(is_palindrome(str2, str2.size()))
                cout <<str2 <<" is a palidrome\n";
        else
                cout <<str2 <<" is not a palidrome\n";

        if(is_palindrome(str3, str3.size()))
                cout <<str3 <<" is a palidrome\n";
        else
                cout <<str3 <<" is not a palidrome\n";

        if(is_palindrome(str4, str4.size()))
                cout <<str4 <<" is a palidrome\n";
        else
                cout <<str4 <<" is not a palidrome\n";

        if(is_palindrome(str5, str5.size()))
                cout <<str5 <<" is a palidrome\n";
        else
```

```
                cout <<str5 <<" is not a palidrome\n";

        if(is_palindrome(str6, str6.size()))
                cout <<str6 <<" is a palidrome\n";
        else
                cout <<str6 <<" is not a palidrome\n";

        if(is_palindrome(str7, str7.size()))
                cout <<str7 <<" is a palidrome\n";
        else
                cout <<str7 <<" is not a palidrome\n";

return 0;
}

bool is_palindrome(string str, int size)
{
        int begin = 0;
        int end = size -1;

        while (str[begin] == str[end]) // keep looping while the letters match up
        {
                begin++;
                end--;
                if (begin >= end) // if all letters match up return true
                        return true;
        }
        return false; //otherwise drop out of loop and return false
}
```

3. (10 points) Write a function (in pseudo code) which takes as an input an object of `vector` type and removes an element at the rank `k` in the constant time, O(1). Assume that the order of elements does not matter.

```
Algorithm remove_at_rank(Vector V, integer k)
    Input: vector and index of element that is to be removed
    Output: vector that has element k removed
//switch back element with element at k
t←V[k]
V[k]←V[size-1]
V[size-1]←t
remove back element        //which is now V[k]
```

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.

If you were to examine the graph of $n^2$and nlog(n) you would see that they would intersect around n = 100. Also, you would see that for n < 100, $n^2$is in fact below nlog(n). Another way to explain it is through a direct proof. Take n to be 99. Then n^2= 9801 and nlog(n) = 9872.306 which would means Bob's algorithm is faster then Al's when n = 99.

5. (15 points) Find the running time functions and classify the algorithms using Big-O asymptotic notation 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 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
```

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

```
Algorithm Ex2(A):
    Input: An array A storing $n \geq 1$ integers.
    Output: The sum of the prefix sums in A.
$s \leftarrow 0$
for i $\leftarrow 0$ to $n-1$ do
    $s \leftarrow s + A[0]$
    for $j \leftarrow 1$ to $n$ do
        $s \leftarrow s + A[j]$
return s
```

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

```
Algorithm Ex2(A):
    Input: An array A storing $n \geq 1$ integers.
    Output: The sum of the prefix sums in A.
$s \leftarrow 0$
for i $\leftarrow 0$ to $n-1$ do
    $s \leftarrow s + A[0]$
    for $j \leftarrow 1$ to $i$ do
        $s \leftarrow s + A[j]$
return s
```

$f(n) = 2n + n(\frac{2(1-2^{n-1})}{1-2}) = 2n + 2^n n - 2n = O(n2^n)$