CSCE 221 Cover Page
Homework #1
Due September 18 at midnight to CSNet

First Name: Mitesh Last Name: Patel UIN: 124002210

User Name: patel221881 E-mail address: patel221881@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			
People	TA	Peer teachers HRBB 129	
Web pages (provide URL)	Urls provided below table	Urls provided below table	
Printed material	Data structures and Algorithms Chapter 4.		
Other Sources			

URLs:
http://stackoverflow.com/questions/3255/big-o-how-do-you-calculate-approximate-it
http://stackoverflow.com/questions/4999650/checking-to-see-if-user-didnt-input-anything-in-cin
http://stackoverflow.com/questions/5307074/how-to-compare-two-strings-are-equal-in-value-what-is-the-best-method
http://cs.stackexchange.com/questions/192/how-to-come-up-with-the-runtime-of-algorithms
http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Weiss/L03-BigOh.htm
piazza.com

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 Mitesh Patel Date 9-16-15

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 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

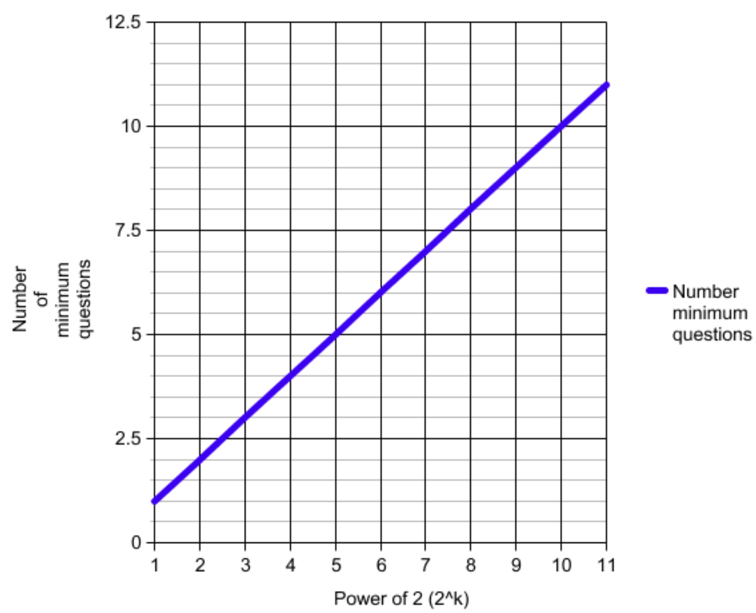
   Hint. The number of questions in this case (range $[1, 32]$) should not exceed 6.

   (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.

   (b) Your program must allow the user to set a target number, so that you can do controlled testing. It would be good if you also had a mode where the human's and computer's roles were switched, with the computer generating a (random) target number and the user trying to guess it.

   (c) For the report, you need to measure how many guesses the program 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).



      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. Yes, the formula matches the computed output for the given input. The minimum will always be for example 2^k the minimum will be what ever k+1 is. For example lets say k is 5 the minimum will be 6 for that range (1-32). The formula can be seen in the table below, as mentioned k is the exponent in respect to the maximum number. Hint: $2^0 = 1$, k = 0+1 = 2. $2^1 = 2$ k = 1+1 = 2 ii.

Range [1..n]	True answer n	# guesses/comparison	Result of formula in (c)
[1,1]	1	1	f(k) = k+1
[1,2]	2	2	f(k) = k+1
[1,4]	4	3	f(k) = k+1
[1,8]	8	4	f(k) = k+1
[1,16]	16	5	f(k) = k+1
...			
[1,2048]	2048	12	f(k) = k+1

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

Range [1..N]	True answer N	# guesses/comparison	Result of formula in (d)
[1,1]	1	1	f(k) = k-1
[1,3]	3	2	f(k) = k-1
[1,7]	7	2	f(k) = k-1
[1,15]	15	3	f(k) = k-1
[1,31]	31	4	f(k) = k-1
...			
[1,2047]	2047	9	f(k) = k-1

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

Big - O notation for this algorithm would be O(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 for loop I wrote executed n times, and the assignment I had in the block therefore executed n times with the operation being the assignment operator. Also for the index I had to use two more subtracting operators. There for our running time function would be 2n+3.

   The length of my string depended on what the user entered and how many letters it contained. For example I told the user to input these 4 palindromes:

   racecar - length of 7.

   radar - length of 5.

   madam - length of 5.

   boob - length of 4.

   The big-O notation from my algorithm would be O(n) deprived from the above function stated.

   The electronic copy has been submitted on csnet with the name "palindrome.cpp".

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(r:integer)

   Input: vector V

   for i <— r+1 to V.size() do

   V[i-1] <— v[i] then // slide elements over

   size <— size - 1

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.
   This is possible because for the $O(n^2)$ algorithm it is effective for small inputs n$\leq$ 100. If it is above 100, then the algorthm's efficiency decreases, because the run time will increase since it is n^2. When n $\geq$ 100 O(nlogn) is better because the big-O notation ignore lower order terms. In conclusion Bob's algorithms has a better lower bound than Al's algorithm because of the facts stated in the question.

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 ≥ 1 integers.
  Output: The sum of the elements at even cells in A.
s ← A[0] executed 1 time(s)
for i ← 2 to n − 1 by increments of 2 do executed n/2 - 1 time(s)
   s ← s + A[i] executed n/2 time(s)
return s executed 1 time(s)

Therefore: f(n) = n/2 + n/2 - 1 + 1 + 1 = n + 1
Big -O Notation: O(n)
```

```
Algorithm Ex2(A):
   Input: An array A storing n ≥ 1 integers.
   Output: The sum of the prefix sums in A.
s ← 0 executed 1 time(s)
```

```
for i ←0 to n−1 do executed n time(s)
    s ← s + A[0] executed n time(s)
    for j ← 1 to n do executed n-1 time(s)
        s ← s + A[j] executed n-1 time(s)
return s executed 1 time(s)
```

Therefore: f(n) = 2+ 2n(2n-2) = 4n^2 - 4n + 2
Big -O Notation: O(n^2)

```
Algorithm Ex2(A):
    Input: An array A storing n ≥ 1 integers.
    Output: The sum of the prefix sums in A.
s ← 0 executed 1 time(s)
for i ←0 to n−1 do executed n time(s)
    s ← s + A[0] executed n time(s)
    for j ← 1 to i do executed n(n+1)/2 time(s)
        s ← s + A[j]
return  s executed 1 time(s)
```

Therefore: f(n) = (n+n+1+1)((n^2+n)/2) = (2n+2)[(n^2+n)/2]
Big -O Notation: O(n^2)