CSCE 221 Cover Page
Homework #1
Due February 13 at midnight to eCampus

First Name Robert Last Name Quan

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	Peer Teacher		
People	Peer Teacher		
Web pages (provide URL)	Learn Cplusplus http://www.learncpp.com/		
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 Robert Quan Date Feb 13 2017

Type the solutions to the homework problems listed below using preferably X͟E/LaTeX word processors, see the class webpage for more information about their installation and tutorial.

1. (10 points) Write a C++ program to implement the Binary Search algorithm for searching a target element in a sorted vector. Your program should keep track of the number of comparisons used to find the target.

   (a) (5 points) To ensure the correctness of the algorithm the input data should be sorted in ascending or descending order. An exception should be thrown when an input vector is unsorted.

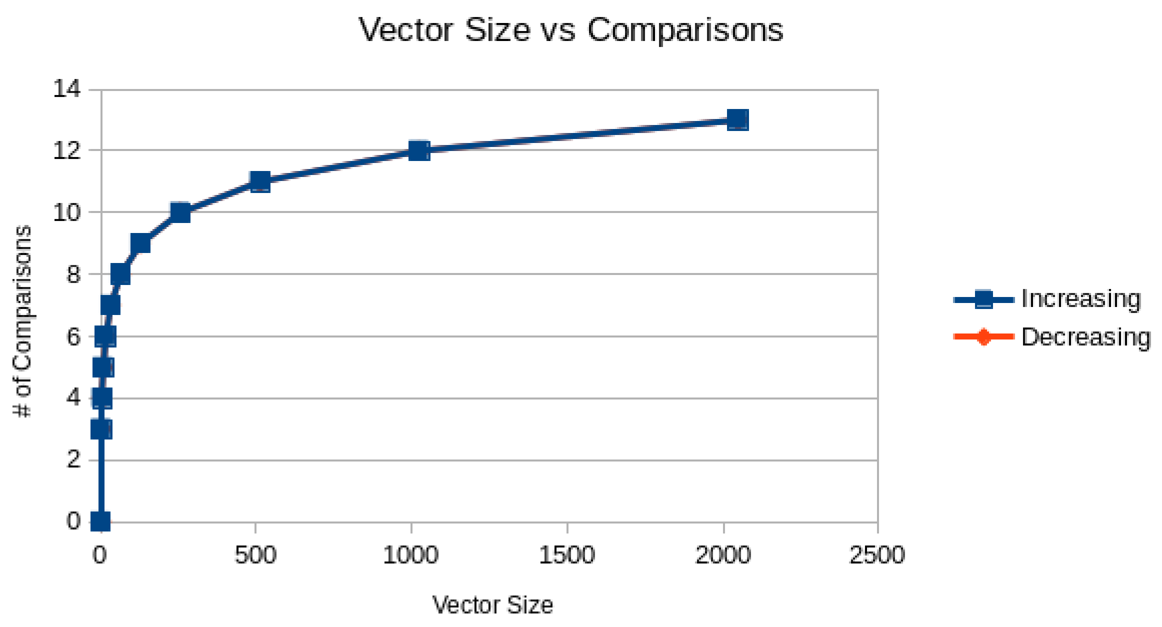
   (b) (10 points) Test your program using vectors populated with consecutive (increasing or decreasing) integers in the ranges from 1 to powers of 2, that is, to these numbers:
   1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048.
   Select the target as the last integer in the vector.

   (c) (5 points) Tabulate the number of comparisons to find the target in each range.

Range [1,n]	Target for incr. values	# comp. for incr. values	Target for decr. values	# comp. for decr. values	Result of the formula in item 5
[1,1]	1	0	1	0	0
[1,2]	2	3	1	3	1
[1,4]	4	4	1	4	2
[1,8]	8	5	1	5	3
[1,16]	16	6	1	6	4
...					
[1,2048]	2048	13	1	13	11

   I get the same number of comparisons for the increasing or decreasing values because I created two different binary search algorithms for the increasing or decreasing case.

   (d) (5 points) Plot the number of comparisons to find a target where the vector size $n = 2^k$, $k = 1, 2, \ldots, 11$ in each increasing/decreasing case. You can use any graphical package (including a spreadsheet).



Just to clarify, I got the same number of comparisons on both cases. One graph is plotted on top of the other one. This is due to the fact that I am using two different binary search algorithms that are executed whenever I have a decreasing or increasing vector.

(a) (5 points) Provide a mathematical formula/function which takes n as an argument, where n is the vector size and returns as its value the number of comparisons. Does your formula match the computed output for a given input? Justify your answer.

$$f(n) = log_2(n)$$

My comparisons number is different from the formula as numbers progress, but the overall figure does resemble the log base 2 figure. The difference is due to the fact that the integer is truncated while running though the mid value calculation. Overall the figure does follow a logarithm equation.

(a) (5 points) How can you modify your formula/function if the largest number in a vector is not an exact power of two? Test your program using input in ranges from 1 to $2^k - 1$, $k = 1, 2, 3, \ldots, 11$.

Range [1,n]	Target for incr. values	# comp. for incr. values	Target for decr. values	# comp. for decr. values	Result of the formula in item 5
[1,1]	1	0	1	0	0
[1,3]	3	3	1	3	1.58496
[1,7]	7	4	1	4	2.80735
[1,15]	15	5	1	5	3.90689
[1,31]	31	6	1	6	4.95419
...					
[1,2047]	2047	12	1	12	10.99929

To modify my function I would use a ceiling function for the integer operation in the ascending case. This way the mid division will always go the larger integer from the division, this will shorten the time for searching in the ascending case. For the Descending case I would use the floor operator to find the value faster.

(b) (5 points) Use Big-O asymptotic notation to classify this algorithm and justify your answer.

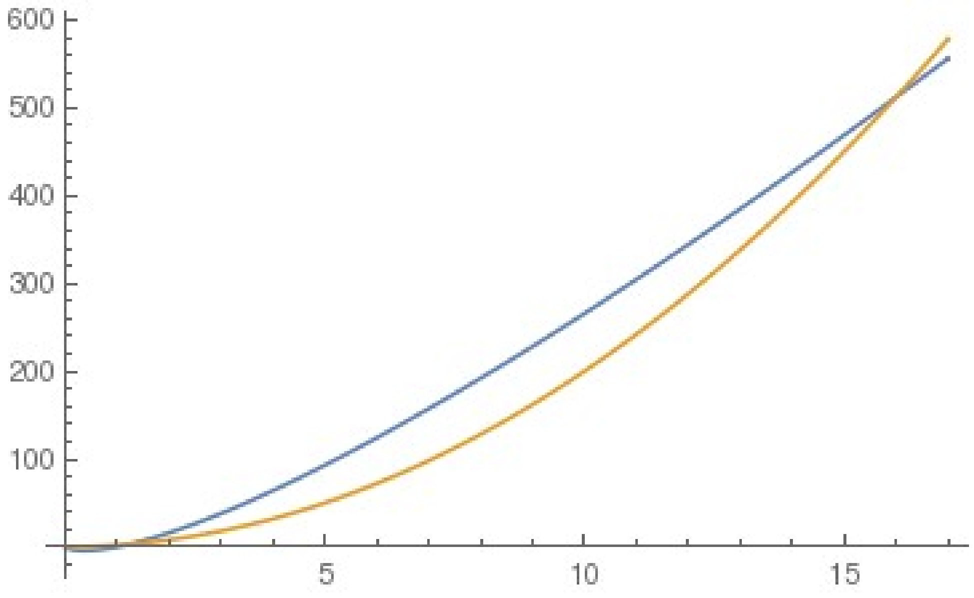
$$O(log(n))$$

It is O(log(n)) due to the fact that the algorithm is recursive. Since we take the middle point of the range and perform that operations various times. This is comparable to divide our vector into two search-able sections from which we choose where to search. Thus using logn operations.

(c) Submit to CSNet 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 $n \geq n_0$.

   By testing out different values of n, I found that the integer number that should be used is n0= 16 for algorithm A to be better than algorithm B. This is interesting due to the fact that there are two places where A can be better than B. I have posted the figure bellow:



   The Blue line is the 8nlogn and the orange line is the 2n^2 function. When solved numerically we find that the value of x has 2 x intercepts:

$$x->1.2396$$

$$x->16.$$

   But for this question, the answer will be when n0 = 16 . So any number greater than 16 will make A better than B.,

$$n>16$$

1. (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?

   The worst run time in this case is for when the value of x is not in the array, then we get a run time of

$$O(n^2)$$

   This is because we iterate over each row for a total of n rows and then each column that are each n columns. Therefore n^2. Since the total number of items is N=n^2, the worst run time would be:

$$O(N)$$

   No, it would be incorrect to say that find2D is linear-time since it depends on the input and the input in this case is n^2. So it is a quadratic time complexity.

2. (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 $n \geq 100$ then the O($n\log n$)-time one is better. Explain how this is possible.

The reason why could be due to two reasons. First, the big O notation removes the constant that could be multiplying the highest order of the function. Lets say that Al's algorithm has a constant factor of 15 multiplying his nlogn equation. (15*nlogn) That would mean that for any value of n that is smaller than 100, the n^2 function will be faster than the nlogn function. Solving this numerically for the example, we find:

$$x -> 99.5631$$

So any number bigger than 99.56 indicates that the n^2 function is worse than the 15*nlogn function.

The second reason why, is due to the fact that the Big O notation removes the lower order terms in the function. Al's algorithm could have a term of n in his function that is contributing to the number of operations that he is using with his input. For example:

$$f(n) = 13nlog(n) + 5n$$

Al just does not understand Big O notation.

1. (20 points) Find the running time functions for the algorithms below and write their classification using Big-O asymptotic notation. The running time function should provide a formula on the number of operations performed on the variable ***s.***

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

For this algorithm, line one will give one comparison by assigning the first element to s. The in line 2 there is 2 operations and it runs n-1 times, but there is no assignment on s so it does not count. In line 3 we see two operations (assignment and addition), and this runs n-1 times. In line 4 we return s, so 1 operation. The equation and the Big-O asymptotic notation is:

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

$$O(n)$$

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

For this algorithm, line one will give me one assignment operation on the variable s. In line two, there are two operations that run a total of (n-2)/2 times, but no operation on variable s. In line 3, we use 2 operations on variable s a total of (n-2)/2 times. Line 4 uses 1 operation. The equation and the Big-O asymptotic notation is:

$$f(n) = 2(n-2)/2 + 2 = n$$

$$O(n)$$

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

For this one, we see nested loop. On line 1, we use 1 operation. On line 2, we see a cost of 2 that runs n times, but does not contribute to the cost of operations on variable s. On line 3 we see 2 operations that run n times. On line 4, we see 2 operations run n(n+2)/2 times but do not contribute to s. On line 5 we see 2 operations that run n(n+1)/2 times. and on line 6 we get one operation.

$$f(n) = n^2 + 3n + 2$$

$$O(n^2)$$

```
Algorithm Ex4(A):
   Input: An array A storing $n \geq 1$ integers.
   Output: The sum of the partial sums in A.
1. $t \leftarrow 0$
2. $s \leftarrow 0$
3. for $i \leftarrow 1$ to $n-1$ do
4.    $s \leftarrow s + A[i]$
5.    $t \leftarrow t + s$
6. return $t$
```

We see on line 2 we have one operation. On line 3 we run the loop n-1 times. On line 4 we have two operations. On line 5 we notice that no operation is being performed on our varibale s! We do see two operations on the variable t though.

$$f(n) = 2n - 1$$

$$O(n)$$