CSCE 221 Cover Page
Homework #1
Due Sept. 23 by midnight to CSNet

First Name Alexander **Last Name** Kaiser **UIN** 924007333

User Name ALKYAYZZZ **E-mail address** alkyayzzz@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	Data Structures and Algorithms in C++ M.T. Goodrich, R. Tamassia and D. Mount		
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 Alexander Kaiser Date 9/8/16

Type the solutions to the homework problems listed below using preferably LYX/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	1	1	1	1
[1,4]	4	2	1	2	2
[1,8]	8	3	1	3	3
[1,16]	16	4	1	4	4
[1,32]	32	5	1	5	5
[1,64]	64	6	1	6	6
[1,128]	128	7	1	7	7
[1,256]	256	8	1	8	8
[1,512]	512	9	1	9	9
[1,1024]	1024	10	1	10	10
[1,2048]	2048	11	1	11	11

   (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).

   See attatched .xlsx file.

   (e) (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.

   Comparisons = 1.4427ln(n) - 2E-15
   The formula matches the computed output for the given input. All values output the same value

   in the table above.

   (f) (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	1	1	1	1.585
[1,7]	7	2	1	2	2.807
[1,15]	15	3	1	3	3.901
[1,31]	31	4	1	4	4.954
[1,63]	63	5	1	5	5.977
[1,127]	127	6	1	6	6.989
[1,255]	255	7	1	7	7.994
[1,511]	511	8	1	8	8.997
[1,1023]	1023	9	1	9	9.999
[1,2047]	2047	10	1	10	10.999

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

The case for an unsuccessful search yeilds $\lfloor log_2 n \rfloor + 1$ whereas the worst case for a successful search yeilds $\lfloor log_2 n \rfloor$ therefore the average case is $\lfloor log_2 n \rfloor / 2$ or $O(log_2 n)$.

(h) 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 $n \geq n_0$

   $8nlogn = 2n^2$
   $4 = \frac{n}{logn}$
   $n = 16$
   $\frac{16}{log_2 16} = \frac{16}{log_2 2^4}$
   $4 = 4$
   $n \geq 17$

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?

   The worst case scenario of $find2D$ is n^2 since it is searching a $n \times n$ array. Thus the big-O for the worst case scenario is $O(n^2)$. In terms of N, the big-O would change to $O(N)$ since $N = n^2$ in terms of the parameters of the question. It is safe to assume that the function is linear because the algorithm searches each row individually.

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

   $O(nlogn)$ is asymptotically better than $O(n^2)$ since $nlogn = O(n^2)$. However, asymptotic analysis ignores smaller values. The less aspytotically better algorithm always runs faster at lower values of n where $n \geq n_0$. In this case n_0 is equal to 100.

5. (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.
   $s \leftarrow A[0]$
   for $i \leftarrow 1$ to $n-1$ **do**
   $s \leftarrow s + A[i]$
   return s

   $n - 1 + 1$
   $O(n)$

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

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

```
Algorithm Ex3(A):
    Input: An array A storing n ≥ 1 integers.
    Output: The sum of the partial sums in A.
s ← 0
for i ← 0  to n − 1 do
    s ← s + A[0]
    for j ← 1 to i do
        s ← s + A[j]
return s
```

$(n-1)(n-1)+1$
$O(n^2)$

```
Algorithm Ex4(A):
    Input: An array A storing n ≥ 1 integers.
    Output: The sum of the partial sums in A.
t ← 0
s ← 0
for i ← 1 to n − 1 do
    s ← s + A[i]
    t ← t + s
return t
```

$(n-1)+(n-1)+1+1$
$O(n)$