CSCE 221 Cover Page
Homework #1
Due July 12 at midnight to eCampus

First Name: Jonathan Last Name: Westerfield UIN: 224005649

User Name: jgwesterfield E-mail address: jgwesterfield

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	Peer Helpdesk	Lecture Slides	
People	Lauren Kleckner		
Web pages (provide URL)			
Printed material			
Other Sources		class lecture slides	

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 Jonathan Westerfield Date 7/12/17

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 one C++ function for the Binary Search algorithm based on the pseudocode in the textbook on page 396. to search a target element in a sorted, ascending or descending, order vector. Your function should also 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:

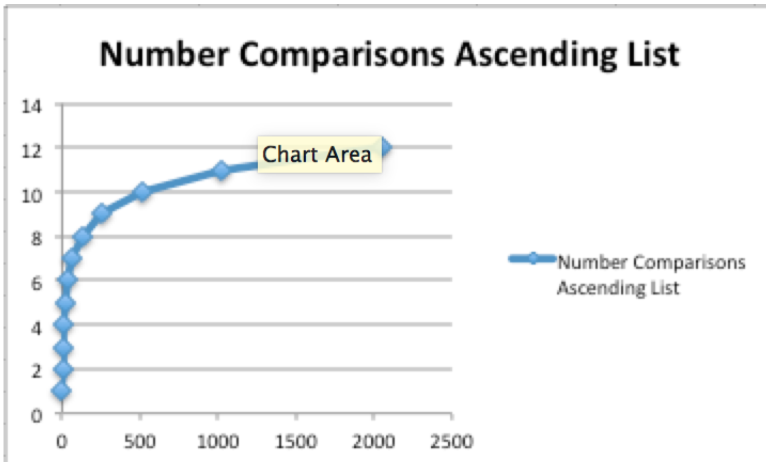
      i. consecutive increasing integers in the ranges from 1 to powers of 2, that is, to these numbers: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048.
      Select the target as the last integer in the vector.

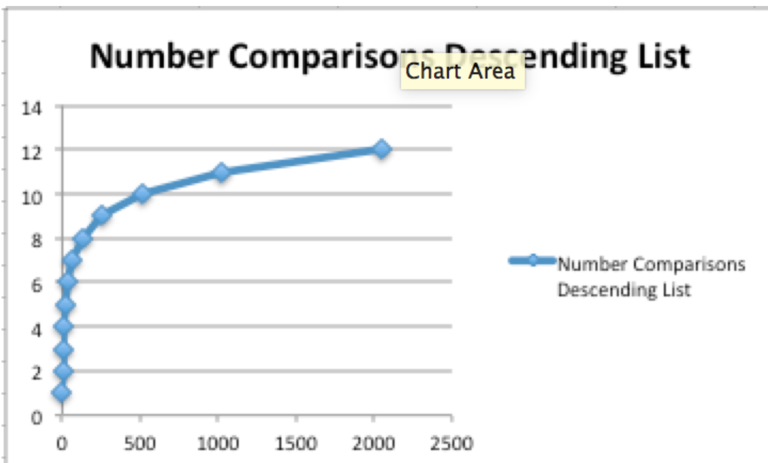
      ii. consecutive decreasing integers in the ranges from powers of 2 to 1 , that is, to these numbers: 2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1.
      Select the target as the last integer in the vector.

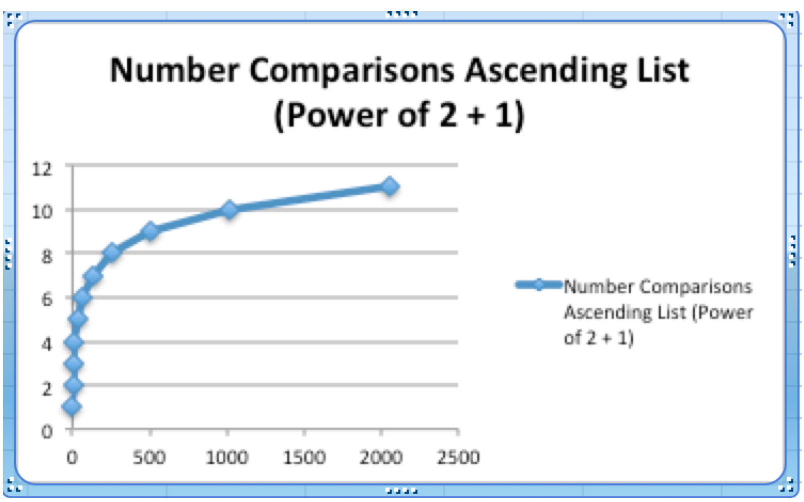
   (c) (5 points) Tabulate the number of comparisons to find the target in each range.

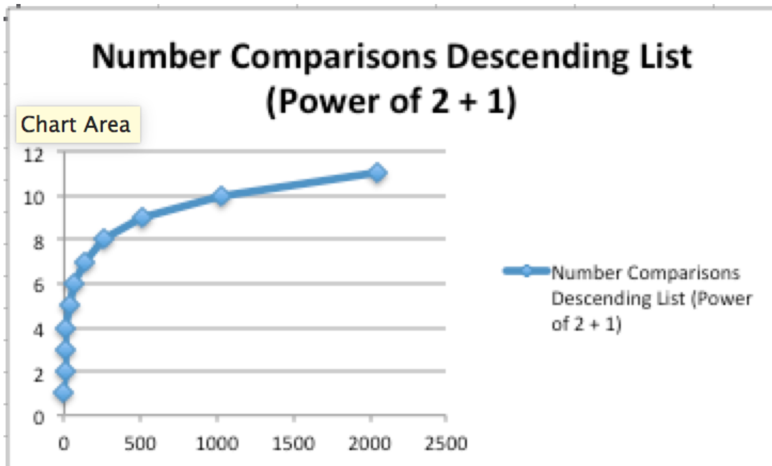
Range [1,n]	Target for incr. values	# comp. for incr. values	Target for decr. values	# comp. for decr. values	Result of the formula in item 5 (e)
[1,1]	1	1	1	1	1
[1,2]	2	2	1	2	2
[1,4]	4	3	1	3	3
[1,8]	8	4	1	4	4
[1,16]	16	5	1	5	5
...					
[1,2048]	2048	12	1	12	12

(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). Include graphs for each case.









(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.

The formula is: $log_2(n) + 1$. This answer to the function based on $n$is equal to the number of comparisons of the binary search function based on the size of the vector. If we were to plug in $n = 2048$, this would equal 12, which is the number of comparisons the binary search function makes.

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

The modified function for when the largest number in a vector is not an exact power of 2 is: $log_2(n)$. This function matches the output of the binary search algorithm.

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 (e)
[1,1]	1	1	1	1	1
[1,3]	3	2	1	2	2
[1,7]	7	3	1	3	3
[1,15]	15	4	1	4	4
[1,31]	31	5	1	5	5
...					
[1,2047]	2047	11	1	11	11

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

The Big-O notation for this function is $O(log_2(n))$ or $O(logn)$. This matches the formula we obtained and it also matches the function of the graph we created.

(a) Submit to CSNet an electronic copy of your code, testing results of all your experiments, and answer to the questions above for grading.

(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 $n \geq n_0$.

$$8nlogn = 2n^2$$

$$logn = \frac{n}{4}$$

$$n = 2^{\frac{n}{4}}$$

$$n = 16$$

n_0 such that A is better than B for $n \geq n_0$ must be 17 since $n = 16$.

(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 running time of find2D in terms of n is $O(n^2)$. The worst case running time of find2D in terms of N is $O(N^2)$. It would be incorrect to say that find2D is a linear time algorithm because its run time is $O(N^2)$, a quadratic function.

(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.

For a very small set of n, a $O(n^2)$ algorithm is actually faster than a $O(n\log n)$ algorithm. However, at the numbers that Al and Bob are dealing with, the most likely culprit is the fact that "Big O" notation hides a lot of details about the runtime of the algorithm. Even though Bob's algorithm is $O(n\log n)$, it is possible the "Big O" notation is hiding other factors of the algorithm. It is entirely possible that Bob's function has many, many more terms in his algorithm but $nlogn$ is simply the biggest factor. This would explain why even though Al's algorith is better at term 99, in which the number of operations would be $99^2 = 9801$, despite Bob's function at term 100 is $100log_2 100 = 664.385$. At 100, just $nlogn$ is clearly smaller than Al's algorithm, therefore, there must be other terms in the equation for Bob's algorithm. This could mean that Bob's algorithm looks similar to this - $nlogn + n + n/2 + x$ ($x$is any arbitrary positive number) where $n$would be smaller than $nlogn$ and therefore would be left out.There is also a possibly that Al's algorithm is a $\frac{n^2}{x}$ algorithm where x is any arbitrary positive number. This would also have changed the actual run time of the algorithm to match Bob's algorithm.

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 ≥ 1 integers.
  Output: The sum of the elements in A.
s ← A[0] // 1 operation: assignment
for i ← 1 to n − 1 do // n-1 iterations
   s ← s + A[i] // 2 operations: assignment, addition
return s // 1 operation
```

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

$$= O(n)$$

Algorithm Ex2(A):
Input: An array A storing $n \geq 1$ integers.
Output: The sum of the elements at even positions in A.

```
s ← A[0] // 1 operation: assignment
for i ← 2 to n − 1 by increments of 2 do // n-2 iterations
  s ← s + A[i] // 2 operations: assignment, addition
return s // 1 operation
```

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

$$= O(n)$$

Algorithm Ex3(A):
Input: An array A storing $n \geq 1$ integers.
Output: The sum of the partial sums in A.

```
s ← 0 // 1 operation
for i ← 0  to n − 1 do // n iterations
   s ← s + A[0] // 2 operations: assignment, addition
   for j ← 1 to i do // (n − 1) iterations
     s ← s + A[j] // 2 operations: assignment, addition
return s // 1 operation
```

$$F(n) = 1 + n(2(2n) + 2) + 1 = 1 + n(4n + 2) + 1 = 2 + n(4n + 2) + 1 = n^2 + 2n + 2$$

$$= O(n^2)$$

Algorithm Ex4(A)
Input: An array A storing $n \geq 1$ integers.
Output: The sum of the partial sums in A.

```
t ← 0 // 1 operation: assignment
s ← 0 // 1 operation: assignment
for i ← 1 to n − 1 do // n-1 iterations
   s ← s + A[i] // 2 operations: assignment, addition
   t ← t + s // 2 operations: assignment, addition
return t // nothing since deals with t not s
```

$$F(n) = 2 + 4(n - 1) = 2 + 4n - 4 = 4n - 2$$

$$= O(n)$$