CSCE 221 Cover Page
Homework #1
Due July 11 at midnight to eCampus

First Name McKenzie Last Name Burch UIN 225005240

User Name mburch13gig-em E-mail address mburch13gig-em@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	Sara Wild	Julia Hann(PT)	
Web pages (provide URL)	https://www.symbolab.com/	https://plot.ly/	
Printed material	Textbook		
Other Sources	Lecture Notes		

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 McKenzie Burch Date July 8, 2018

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. (5 points) Write a C++ function for the Binary Search algorithm based on the second sets of the lecture sides, slide 46

```
int Binary_Search(vector<int> &v, int x) {
   int mid, low = 0;
   int high = (int) v.size()-1;
   while (low < high) {
      mid = (low+high)/2;
      if (num_comp++, v[mid] < x) low = mid+1;
      else high = mid;
   }
  if (num_comp++, x==v[low]) return low; //found
  throw Unsuccessful_Search(); //not found
}
```

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:

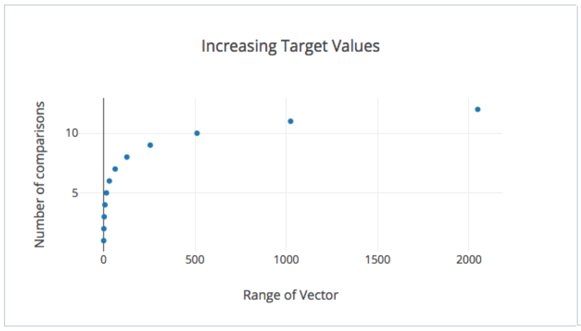
      i. consecutive increasing integers in the ranges from 1 to powers of 2, that is, to these numbers: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048.
      Select the target as the last integer in the vector.

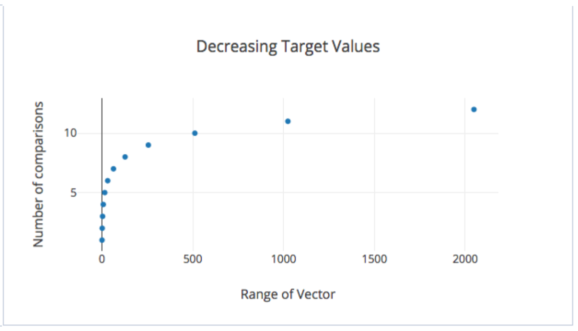
      ii. consecutive decreasing integers in the ranges from powers of 2 to 1 , that is, to these numbers: 2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1.
      Select the target as the last integer in the vector.

   (c) (10 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
[1,32]	32	6	1	6	6
[1,62]	62	7	1	7	7
[1,125]	125	8	1	8	8
[1,256]	256	9	1	9	9
[1,512]	512	10	1	10	10
[1,1024]	512	11	1	11	11
[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.





(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.

i. A formula which take n the size of a vector array and returns the number of comparisons is $k = log_2 n + 1$. This formula matches the computed output, and is valid in that the sorted vector of size $n = 2^k$, $k = 1, 2, \ldots, 11$ shows that that the target value's index value + 1 is the number of comparisons when given the range of [1,n] variables with $n = 2^k$.

(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 $2^k - 1$, $k = 1, 2, 3, \ldots, 11$. To modify the formula take the floor of $log_2 n$. So the formula for values of $2^k - 1$ is $k = \lfloor log_2 n \rfloor + 1$.

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
[1,63]	63	6	1	6	6
[1,127]	127	7	1	7	7
[1,255]	255	8	1	8	8
[1,511]	511	9	1	9	9
[1,1023]	1023	10	1	10	10
[1,2047]	2047	11	1	11	11

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

i. The Big-O of the binary search algorithm is $O(log_2 n)$, because the max number of comparisons needed to find the element at the last index of the given vector is $log_2 + 1$.

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 $n \geq n_0$.

   A: $8n\log_2 n$

   B: $2n^2$

   $8n\log_2 n = 2n^2$

   $4\log_2 n = n$

   $4 = \frac{n}{\log_2 n}$

   $n_0 = 16$

   So when $n_0 = 16$ then A is better than B for $n \geq 16$.

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?

   ```
   arrayFind has a worst-case scenario of O($n$)
   find2D calls arrayFind $n$ times which loops through $n$ times
      find2D has worst-case scenario in terms of $n$ of O($n^2$)
   Total size of A: $N = n^2$
      find2D has worst-case scenario in terms of $N$ of O($N$)
   This means that find2D is a linear-time algorithm in terms of $N$ the size of array.
   ```

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.

   (a) It is possible for a O(n^2)-time algorithm to runs faster then a O($n\log n$)-time for smaller input values because Big-O is an asymptotic notation, meaning that when looking for the worst case scenario it is assumed that there is a significally large amount of input, therefore once the input is greater than or equal to 100 then the O(n^2)-time algorithm runs too slowly.

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.
   $s \leftarrow A[0]$ 1 ASSIGNMENT
   for $i \leftarrow 1$ to $n-1$ do n-1 COMPARISONS
      $s \leftarrow s + A[i]$ 2 OPERATORS
   return $s$
   Formula: $f(n) = 2n+1$
   Big-O: $O(n)$
   ```

```
Algorithm Ex2(A):
      Input: An array A storing $n \geq 1$ integers.
      Output: The sum of the elements at even positions in A.
    $s \leftarrow A[0]$ 1 ASSIGNMENT
    for $i \leftarrow 2$ to $n-1$ by increments of 2 do (n-1)/2 COMPARISONS
      $s \leftarrow s + A[i]$ 2 OPERATORS
    return $s$
    Formula: $f(n) = n$
    Big-O: $O(n)$

    Algorithm Ex3(A):
       Input: An array A storing $n \geq 1$ integers.
       Output: The sum of the partial sums in A.
    $s \leftarrow 0$ 1 ASSIGNMENT
    for $i \leftarrow 0$  to $n-1$ do n COMPARISONS
       $s \leftarrow s + A[0]$ 2 OPERATORS
       for $j \leftarrow 1$ to $i$ do n COMPARISONS
         $s \leftarrow s + A[j]$ 2 OPERATORS
    return $s$
    Formula: $f(n) = 4n^2 + 1$
    Big-O: $O(n^2)$

    Algorithm Ex4(A):
       Input: An array A storing $n \geq 1$ integers.
       Output: The sum of the partial sums in A.
    $t \leftarrow 0$
    $s \leftarrow 0$ 1 ASSIGNMENT
    for $i \leftarrow 1$ to $n-1$ do n-1 COMPARISONS
       $s \leftarrow s + A[i]$ 2 OPERATORS
       $t \leftarrow t + s$ 1 OPERATORS
    return $t$
    Formula: $f(n) = 3n + 2$
    Big-O: $O(n)$
```

Submit to eCampus answer to all the questions above in a single pdf.
Code for question 1 will be verified during labs.