CSCE 221 Cover Page

Homework #1

Due February 10 at midnight to Canvas

First Name Annemarie Last Name Bell UIN 529007879

User Name aeb178 E-mail address aeb178@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	The Big-O notation	graphing	
People			
Web pages (provide URL)	https://rob-bell.net	https://www.desmos.com	
Printed material			
Other Sources			

*Full big-o notation website (so it would fit the box) https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/#:~:text=Big%20O%20notation%20is%20used,on%20disk)%20by%20an%20algorithm.

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 Annemarie Bell Date 02/09/2021

Homework 1 Objectives:

1. Developing the C++ programming skills by using
   (a) templated dynamic arrays and STL vectors.
   (b) exceptions for reporting the logical errors or unsuccessful operations.
   (c) tests for checking correctness of a program.
2. Comparing theory with a computation experiment in order to classify algorithm.
3. Preparing reports/documents using the professional software LyX or LaTeX.
4. Understanding the definition of the big-O asymptotic notation.
5. Classifying algorithms based on pseudocode.

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

1. (25 points) Use the STL class `vector<int>` to write a C++ function that returns true if there are two elements of the vector for which their product is odd, and returns false otherwise. Provide two algorithms for solving this problem with the efficiency of $O(n)$ for the first one and $O(n^2)$ for the second one where n is the size of the vector.

   Justify your answer by writing the running time functions in terms of n for both the algorithms.

```
bool oddProduct(vector<int> v){
        for (unsigned int i = 1; i < v.size(); i++){
                for (unsigned int j = 0; j < i; j++){
                        if((v[i] * v[j]) % 2 == 1){
                                return true;
                        }
                }
        }
        return false;
}
```

   - What do you consider as an operation for each algorithm?
     - I consider the number of times that the algorithm would have to mutiply the two vectors as the operation for the algorithms.
   - Are the best and worst cases for both the algorithms the same in terms of Big-O notation? Justify your answer.
     (a) The best possible case would be O(1) = 1, which would just be the mutiplication only needing to go through once, before the algohim returns true
     (b) The worst possible case would be O(n^2) = $\frac{n(n+1)}{2}$, in which the mutiplication would have to irrate through all the loops to find a product that is false.
   - Describe the situations of getting the best and worst cases, give the samples of the input for each case and check if your running time functions match the number of operations.
     - Best Case: the vector holds the first two ints as odd numbers and don't contain any version of an even number, example vector: <1,3,2,5,6,7,8>
     - Worst Case: the vector holds only even numbers, the loop will never return false, example vector: <2,4,6,8,2,2,6>

   -

2. (50 points) The binary search algorithm problem.

(a) (5 points) Implement a templated C++ function for the binary search algorithm based on the set of the lecture slides *"Analysis of Algorithms"*.

```
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; //OK: found
   throw Unsuccessful_Search(); //exception: not found
}
```

Be sure that before calling `Binary_Search` elements of the vector `v` are arranged in increasing order. The function should also keep track of the number of comparisons used to find the target `x`. The (global) variable `num_comp` keeps the number of comparisons and initially should be set to zero.

```
int main(){
        vector <int> v{1,2,3,4,5,9,25,26,29,50,51,56,57,67,70,71};
        Binary_Search(v, 1); //target is going to be the first element
        Binary_Search(v,71); //target is the last element
        Binary_Search(v,26); //target is the middle of the vector
        return 0;
}
```

(a) (10 points) Test your algorithm for correctness using a vector of data with 16 elements sorted in ascending order. An exception should be thrown when the input vector is unsorted or the search is unsuccessful.
What is the value of `num_comp` in the cases when

i. the target `x` is the first element of the vector `v`
   A. first element was found with a num_comp of 4
ii. the target `x` is the last element of the vector `v`
   A. last element was found with a num_comp of 5
iii. the target `x` is in the middle of the vector `v`
   A. when the target element is in the middle the element is found with a num_comp of 1

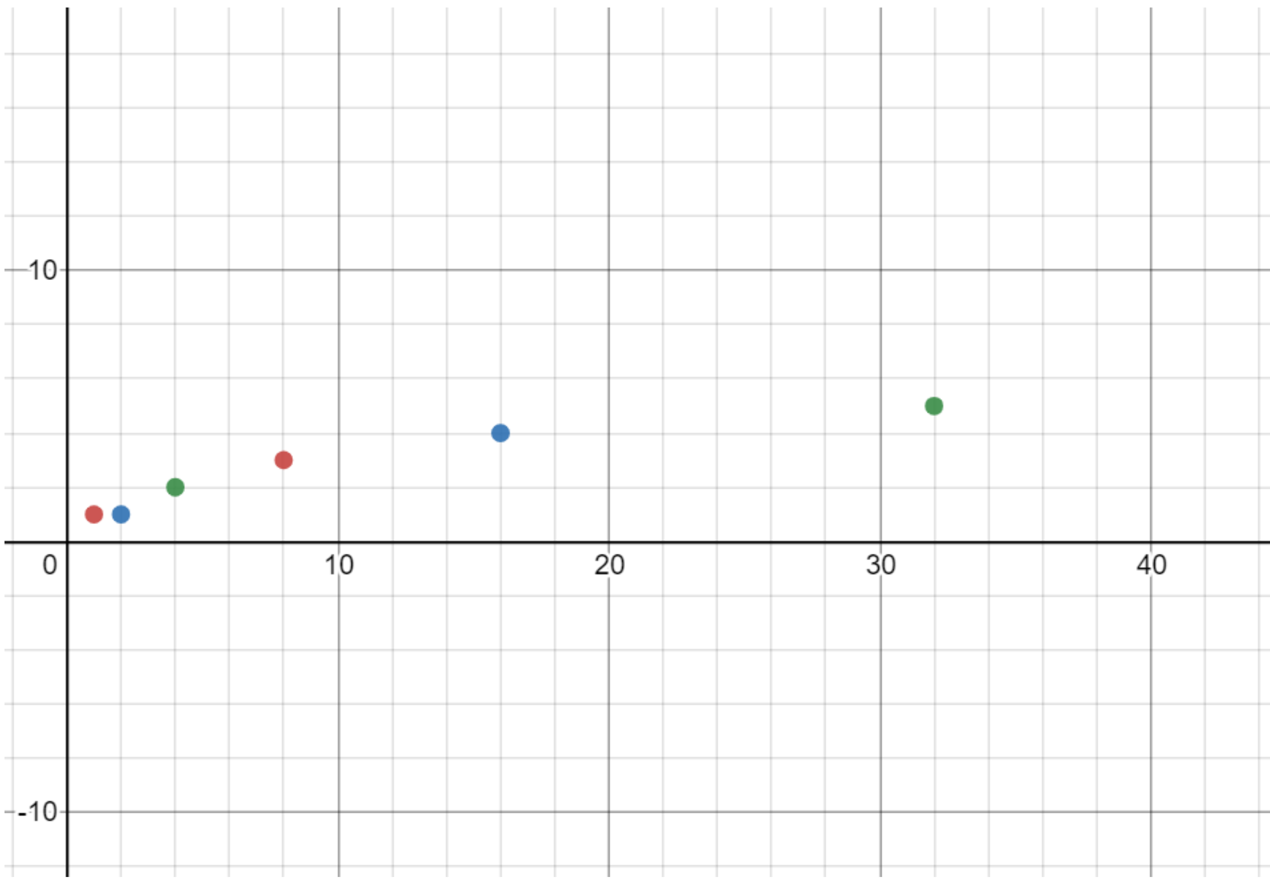
What is your conclusion from the testing for $n = 16$?

i. Selection sort takes a big O approach of n^2 where if the int that is being used then the best case happens however it could take longer and longer through every itteration.

(b) (10 points) Test your program using vectors of size $n = 2^k$ where $k = 0, 1, 2, \dots, 11$ populated with consecutive increasing integers in these ranges: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048. Select the target as the last element in the vector. Record the value of `num_comp` for each vector size in the table below.

Range [1,n]	Target	# comp.
[1, 1]	1	1
[1, 2]	2	1
[1, 4]	4	2
[1, 8]	8	3
[1, 16]	16	4
...		...
[1, 2048]	2048	11

(c) (5 points) Plot the number of comparisons for the vector size $n = 2^k$, $k = 1, 2, \dots, 11$. You can use a spreadsheet or any graphical package.

i. 

(d) (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 any input? Justify your answer.

$$log_2 n$$

To solve for what n would be I took the equation, $n = 2^k$and solved for K to find what the big O would be in this situation, it happens to match the way that the graph follows the line, if n is the number of ints in the vector.

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

Range [1,n]	Target	# comp.
[1, 1]	1	1
[1, 3]	3	1
[1, 7]	7	3
[1, 15]	15	4
[1, 31]	31	5
...		
[1, 2047]	2047	11

(f) (5 points) Do you think the number of comparisons in the experiment above are the same for a vector of strings or a vector of doubles? Justify your answer

i. **I believe that the numbers would reamin the same if they were doubles or strings because the number of items in the vector would remain the same even through the type of the item has changed, it is not the type of the object in the vector it compares but rather the amount of items in the vector it compares.**

(h) (5 points) Use the Big-O asymptotic notation to classify binary search algorithm and justify your answer.

   i. $O(log_2(n))$ **is the Big-O notation that classifies the binary search algorithm because no matter the amount of items in the search the search always runs** $n = 2^k$**where n is equivalent to the number of items in the vector.**

(i) (Bonus question—10 points) Read the sections 1.6.3 and 1.6.4 from the textbook and modify the algorithm using a functional object to compare vector elements. How can you modify the binary search algorithm to handle the vector of decreasing elements? What will be the value of `num_comp`? Repeat the search experiment for the smallest number in the integer arrays. Tabulate the results and write a conclusion of the experiment with your justification.

(j)

3. (25 points) Find running time functions for the algorithms below and write their classification using Big-O asymptotic notation in terms of n. A running time function should provide a formula on the number of arithmetic operations and assignments performed on the variables s, t, or c (the return value). Note that array indices start from 0.

```
Algorithm Ex1(A):
  Input: An array A storing $n \geq 1$ integers.
  Output: The sum of the elements in A.
$s \leftarrow A[0]$   //Assignment- 1
for $i \leftarrow 1$ to $n-1$ do      //(n-1)loops
   $s \leftarrow s + A[i]$ //assignment and add (2)
end for
return $s$ //return, opperation
```

(a) F(n) = 1 + 2(n-1) + 1 = 2n. **O(2n)**

```
Algorithm Ex2(A):
  Input: An array A storing $n \geq 1$ integers.
  Output: The sum of the elements at even positions in A.
$s \leftarrow A[0]$ //one opp, assignment
for $i \leftarrow 2$ to $n-1$ by increments of 2 do
//interations = log2(n-2)
  $s \leftarrow s + A[i]$ //add and assignment
end for
return $s$ //return opp
```

F(n) = 1 + 2(log2(n-2))+1 = 2+ 2log2(n-2). **O(2log2(n-2)).**

```
Algorithm Ex3(A):
   Input: An array A storing $n \geq 1$ integers.
   Output: The sum of the partial sums in A.
$s \leftarrow 0$ //one oppp
   for $i \leftarrow 0$  to $n-1$ do //starting at zero to n-1 loop: n
   $s \leftarrow s + A[0]$ // 2 opps; add and assign
   for $j \leftarrow 1$ to $i$ do //going through i so 2n
     $s \leftarrow s + A[j]$ // 2 opp add and assign
   end for
end for
return $s$ //+1
```

F(n)= 1+2(n)+2(2n)+1 = 2 + 6(n); **O(6n)**

```
Algorithm Ex4(A):
    Input: An array A storing $n \geq 1$ integers.
    Output: The sum of the partial sums in A.
$t \leftarrow A[0]$ //one op
$s \leftarrow A[0]$ // one op
for $i \leftarrow 1$ to $n-1$ do //loop: n-1
    $s \leftarrow s + A[i]$ // 2 opp add and assign
    $t \leftarrow t + s$ // 2 opp add and assign
end for
return $t$ // one opp return
F(n) = 1+1+2(2(n-1)+ 1 = 3 + 4n -4 = 4n -1. O(4n)
```

```
Algorithm Ex5(A, B):
    Input: Arrays A and B storing $n \geq 1$ integers.
    Output: The number of elements in B equal to the partial sums in A.
$c \leftarrow 0$ //counter  //one opp
for $i \leftarrow 0$ to $n-1$ do //loop: n
    $s \leftarrow 0$ //partial sum //one op
    for $j \leftarrow 0$ to $n-1$ do //loop: 2n
        $s \leftarrow s + A[0]$ // 2 opp add and assign
        for $k \leftarrow 1$ to $j$ do // loop: 4n
            $s \leftarrow s + A[k]$ //2 opp
        end for
    end for
    if $B[i] = s$ then // 1 assign
        $c \leftarrow c + 1$ // 2 opp
    end if
end for
return $c$ //+1
F(n) = 1 + 4(n) + 2 (2(n)) + 2(4(n))+ 1 = 2 + 16n. O(16n)
```