CSCE 221 Cover Page

Homework 1

Due February 10 at midnight to eCampus

First Name: Cristian   Last Name: Avalos   UIN: 627003137

Username: avalos672918   E-mail address: avalos672918@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	TA		
Web pages (provide URL)	https://www.tutorialspoint.com/cplusplus/cpp$_exceptions_handling.htm$		
Printed material			
Other Sources	CSCE 222 notes		

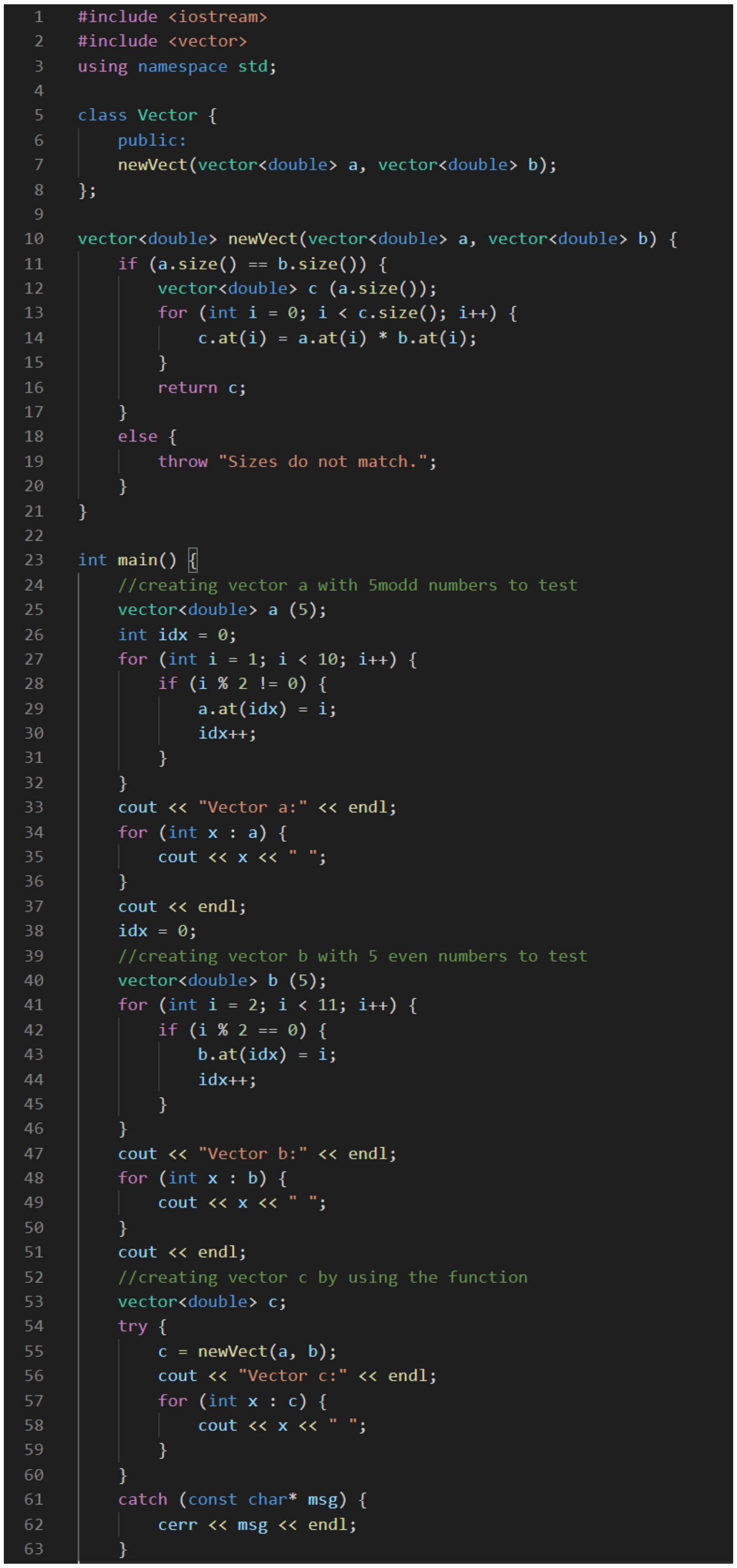
I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

Cristian Avalos   February 10, 2020

1. (10 points) Use the STL class vector<double> to write a C++ function that takes two vectors, a and b, of the same size and returns a vector c such that $c[i] = a[i] \cdot b[i]$. How many scalar multiplications are used to create elements of the vector c of size n? Provide a formula on the number of scalar multiplications in terms of n. What is the classification of this algorithm in terms of the Big-O notation?

   n scalar multiplications are used to create the elements of c because of the sizes of the input vectors being n.
   The formula for the scalar multiplications in terms of n is $f(n) = n$.
   The Big-O of this notation is $O(n)$.
   (code provided in next page)



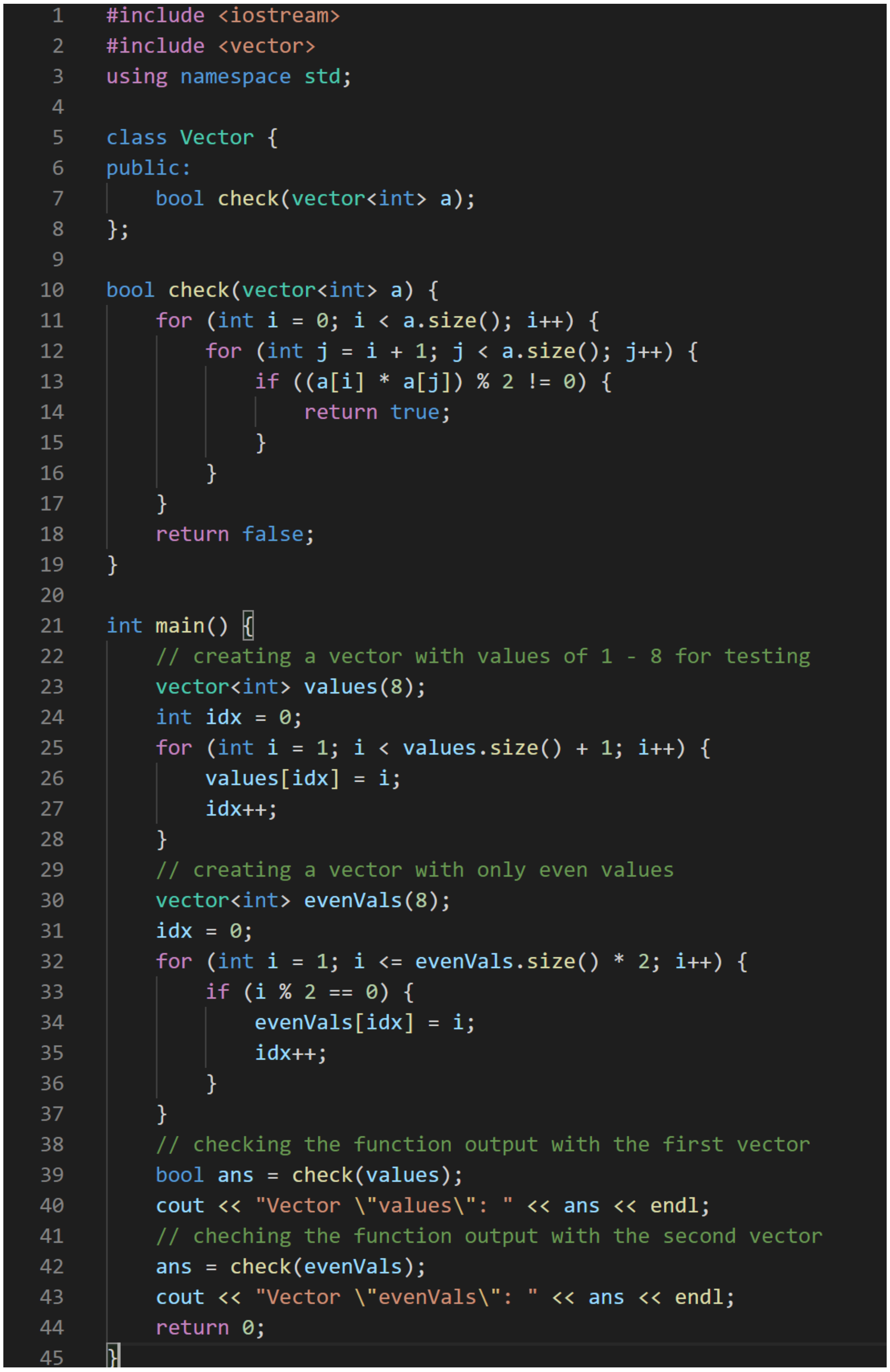
```cpp
#include <iostream>
#include <vector>
using namespace std;

class Vector {
    public:
    newVect(vector<double> a, vector<double> b);
};

vector<double> newVect(vector<double> a, vector<double> b) {
    if (a.size() == b.size()) {
        vector<double> c (a.size());
        for (int i = 0; i < c.size(); i++) {
            c.at(i) = a.at(i) * b.at(i);
        }
        return c;
    }
    else {
        throw "Sizes do not match.";
    }
}

int main() {
    //creating vector a with 5modd numbers to test
    vector<double> a (5);
    int idx = 0;
    for (int i = 1; i < 10; i++) {
        if (i % 2 != 0) {
            a.at(idx) = i;
            idx++;
        }
    }
    cout << "Vector a:" << endl;
    for (int x : a) {
        cout << x << " ";
    }
    cout << endl;
    idx = 0;
    //creating vector b with 5 even numbers to test
    vector<double> b (5);
    for (int i = 2; i < 11; i++) {
        if (i % 2 == 0) {
            b.at(idx) = i;
            idx++;
        }
    }
    cout << "Vector b:" << endl;
    for (int x : b) {
        cout << x << " ";
    }
    cout << endl;
    //creating vector c by using the function
    vector<double> c;
    try {
        c = newVect(a, b);
        cout << "Vector c:" << endl;
        for (int x : c) {
            cout << x << " ";
        }
    }
    catch (const char* msg) {
        cerr << msg << endl;
    }
```

Figure 1: My code for problem 1

2. (10 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 a formula on the number of scalar multiplications in terms of n, the size of the vector, to solve the problem in the best and worst cases. Describe the situations of getting the best and worst cases, give the samples of the input at each case and check if your formula works. What is the classification of the algorithm in the best and worst cases in terms of the Big-O notation?

   The formula on the number of scalar multiplications in terms of n is $f(n) = \frac{n(n-1)}{2}$.
   For the best case is if the product is odd in the first iteration, this will be $O(1)$.
   The worse case is if there are no odd products in the vector and the algorithms searches everything, which is $O(n^2)$.

   (code provided in next page)



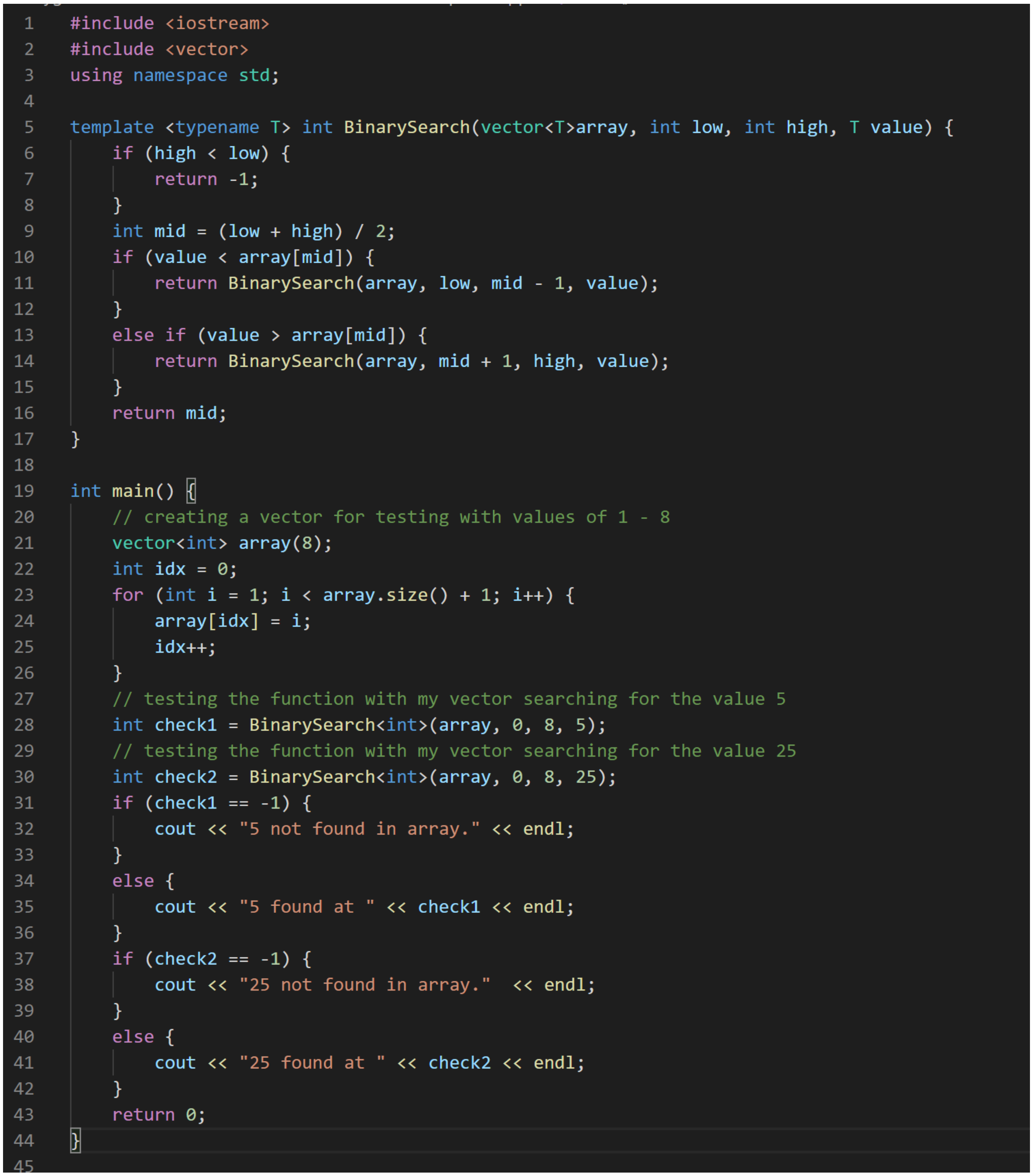
```cpp
#include <iostream>
#include <vector>
using namespace std;

class Vector {
public:
    bool check(vector<int> a);
};

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

int main() {
    // creating a vector with values of 1 - 8 for testing
    vector<int> values(8);
    int idx = 0;
    for (int i = 1; i < values.size() + 1; i++) {
        values[idx] = i;
        idx++;
    }
    // creating a vector with only even values
    vector<int> evenVals(8);
    idx = 0;
    for (int i = 1; i <= evenVals.size() * 2; i++) {
        if (i % 2 == 0) {
            evenVals[idx] = i;
            idx++;
        }
    }
    // checking the function output with the first vector
    bool ans = check(values);
    cout << "Vector \"values\": " << ans << endl;
    // cheching the function output with the second vector
    ans = check(evenVals);
    cout << "Vector \"evenVals\": " << ans << endl;
    return 0;
}
```

Figure 2: My code for problem 2

3. (20 points) Write a templated C++ function called *BinarySearch* which searches for a target x of any numeric type T, and test it using a sorted vector of type T. Provide the formulas on the number of comparisons in terms of n, the length of the vector, when searching for a target in the best and worst cases. Describe the situations of getting the best and worst cases. What is the classification of the algorithm in the best and worst cases in terms of the Big-O notation?

   The formula on the number of comparisons of n is $F(n) = F(\frac{n}{2}) + c$. The length of my vector is 8. The best case will be when the element that I am searching for is the middle element, the Big-O is O(1). The worst case is when there is no such target. So, it is O(log(n)).

   (code provided in next page)



```cpp
#include <iostream>
#include <vector>
using namespace std;

template <typename T> int BinarySearch(vector<T>array, int low, int high, T value) {
    if (high < low) {
        return -1;
    }
    int mid = (low + high) / 2;
    if (value < array[mid]) {
        return BinarySearch(array, low, mid - 1, value);
    }
    else if (value > array[mid]) {
        return BinarySearch(array, mid + 1, high, value);
    }
    return mid;
}

int main() {
    // creating a vector for testing with values of 1 - 8
    vector<int> array(8);
    int idx = 0;
    for (int i = 1; i < array.size() + 1; i++) {
        array[idx] = i;
        idx++;
    }
    // testing the function with my vector searching for the value 5
    int check1 = BinarySearch<int>(array, 0, 8, 5);
    // testing the function with my vector searching for the value 25
    int check2 = BinarySearch<int>(array, 0, 8, 25);
    if (check1 == -1) {
        cout << "5 not found in array." << endl;
    }
    else {
        cout << "5 found at " << check1 << endl;
    }
    if (check2 == -1) {
        cout << "25 not found in array."  << endl;
    }
    else {
        cout << "25 found at " << check2 << endl;
    }
    return 0;
}
```

Figure 3: My code for problem 3

4. (10 points) 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 faster than B for $n \geq n_0$.

   At $n_o = 16$ is when both of the graphs intersects. After $n_0 > 16$, A becomes faster than B.

5. (10 points) Two friends are arguing about their algorithms. The first one claims his $O(n \log n)$-time algorithm is always faster than the second friend's $O(n^2)$-time algorithm. To settle the issue, they perform a set of experiments. Show that it is possible to find an input $n < 100$ that the $O(n^2)$-time algorithm runs faster, and only when $n \geq 100$ that the $O(n \log n)$4-time algorithm is faster.

   let C be a constant to prove this problem and $n = 100$. We get:
   $C \cdot 100 \log (100) = 100^2$
   $C \log (100) = 100$
   $C = \frac{100}{log(100)}$
   Therefore, for any n values between 0 and 100, $C \cdot n \log (n)$ will be greater than n^2.

6. (10 points) An algorithm takes 0.5ms for an input of size 100. Assuming that lower-order terms are negligible, how long will it take for an input of size 500 if the running time is:

   (a) linear, $O(n)$?

   5 times, $0.5 \cdot 5 = 2.5$ms

   (b) cubic, $O(n^3)$?

   $f(5n) = (5n)^3$
   $= 125n^2$
   running an input size 5 times the original takes $125 \cdot 0.5 = 62.5$ms

7. (10 points) An algorithm takes 0.5ms for an input of size 100. Assuming that lower-order terms are negligible, how large a problem can be solved in 1 minute if the running time is:

   (a) $O(n \log n)$?

   If $c \cdot 100 \log(100) = 0.5$
   Then $c = \frac{0.5}{664} \approx 0.00075$
   For 1 minute:
   $c \cdot n \log(n) = 60{,}000$
   $n\log(n) = \frac{60{,}000}{0.00075} \approx 80{,}000{,}000$
   n$\approx 3.7 \cdot 10^6$

   (b) quadratic, $O(n^2)$?

   If $c \cdot 100^2 = 0.05$
   Then $c = c \cdot 10^{-5}$
   $c \cdot n^2 = 60{,}000$
   $n^2 = 1.2 \cdot 10^9$
   $n = 34641$

8. (20 points) Find running time functions for the algorithms below and write their classification using Big-O asymptotic notation. A running time function should provide a formula on the number of operations performed on the variable s. Note that array indices start from 0.

   For this part, I will say that +, -, *, /, =, and returns count as an operation.

```
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
    $s \leftarrow s + A[i]$
end for
return $s$
$f(s) = (2(s-1)+1)+2$
$O(s)$
```

```
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
    $s \leftarrow s + A[i]$
end for
return $4s$
$f(s) = (2log_2(s)) + 1) + 3$
$O(log_2(s))$
```

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

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