# CSCE 221 Cover Page

## Homework #1

### Due February 10 at midnight to Canvas

First Name: Mualla Last Name: Argin UIN: 728003004

User Name: margin25 E-mail address: margin25@gmail.com

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 | PT Jason Jia | | |
| Web pages (provide URL)`` | https://www.geeksforgeeks.org/binary-search/ | | |
| Printed material | Not Applicable | | |
| Other Sources | Not Applicable | | |

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: Mualla Argin Date : 2/10/2021

**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.

   - What do you consider as an operation for each algorithm?

   O(n) solution: Since there is only one loop for the OddProduct() function the time complexity is O(n). O(n^2) solution: Since there is two loops for the OddProduct() function the time complexity is O(n^2).

   - Are the best and worst cases for both the algorithms the same in terms of Big-O notation? Justify your answer.

   O(n) Best Case: The best case scenario is when the first two values of the vector are odd. In that case, we don't check the rest of the vector so the time complexity is O(1).

   O(n^2) Best Case: The best case scenario is when the first pair of numbers multiplied has an odd product. In that case the time complexity is O(1).

   O(n) Worst Case: The worst case scenario is when there are no odd numbers or the final two values of the vector are odd. In that case, the time complexity is O(n).

   O(n^2) Worst Case: The worst case scenario is when no two pair products are odd. This causes us to go all the way through the two nested loops therefore the time complexity is O(n^2).

   According to my observations above the best cases of the algorithms are the same : O(1). While the worst cases are different : O(n) vs O(n^2).

   - 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.

O(n) Best Case Example: input : 4 , [5, 7, 2, 4] Since the first two numbers of the vector are odd we get an odd product. This results in us returning true and a constant time O(1).

O(n^2) Best Case Example: input : 4 , [5, 7, 2, 4] Since the first pair is 5 and 7 and the product of 5 * 7 = 35, The function returns true. Therefore, the time complexity is O(1) . This is because The first pair we check has a odd product.

O(n) Worst Case Example: input : 6 , [0,2,4,6,8,10] When we loop through the vector we find no odd numbers. Therefore we return false and since we go through the one for loop in its entirety the time complexity is O(n).

O(n^2) Worst Case Example: input : 6 , [0,2,4,6,8,10] If we go through the two nested loops in this case we won't find any pairs of numbers that result in an odd product. This is because all numbers in the vector are even. Since we go through the two nested loops in their entirety, we get a time complexity of O(n^2).

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

// O(n) solution:
bool OddProduct(int n, vector<int> nums)
{
    // odd x odd = odd
    // odd x even = even
    // even x even = even
    // therefore we need at least 2 odd numbers
    // in the vector to get an odd product
    int counter = 0;
    for (int i = 0; i < n; i++)
    {
        if (nums.at(i) % 2 == 1)
        {
            counter++;
        }
        if (counter >= 2)
        {
            return true;
        }
    }
    return false;
}

// O(n^2) solution:
bool OddProduct(int n, vector<int> nums)
{
    int counter = 0;
    int final_product = 0;
    for (int x = 0; x < n; x++)
    {
        for (int y = x + 1; y < n; y++)
        {
            final_product = nums.at(x) * nums.at(y);
            if (final_product % 2 == 1)
            {
                return true;
            }
        }
    }
    return false;
}
```

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.

(b) (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 : 4
ii. the target x is the last element of the vector v : 4
iii. the target x is in the middle of the vector v : 4

What is your conclusion from the testing for $n$ = 16? 4

From testing n=16, I can conclude that there are 4 comparisons.

(c) (10 points) Test your program using vectors of size $n = 2^k$ where $k = 0, 1, 2, \ldots, 11$ 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.

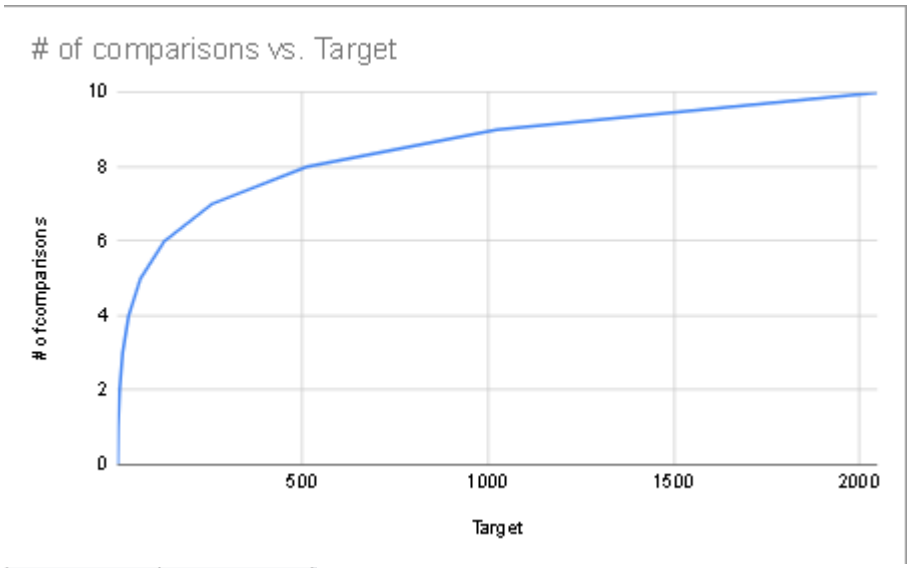
| Range [1,*n*] | Target | # comp. |
|---|---|---|
| [1, 1] | 1 | 0 |
| [1, 2] | 2 | 1 |
| [1, 4] | 4 | 2 |
| [1, 8] | 8 | 3 |
| [1, 16] | 16 | 4 |
| [1, 32] | 32 | 5 |
| [1, 64] | 64 | 6 |
| [1,128] | 128 | 7 |
| [1,256] | 256 | 8 |
| [1,512] | 512 | 9 |
| [1,1024] | 1024 | 10 |
| [1, 2048] | 2048 | 11 |

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


(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 any input? Justify your answer.

A mathematical formula to represent the relationship between number of comparisons and vector size is $\log_2(n)$ = # of comparisons where n is the vector size. The formula does match the computed output for any input. I tested it with the values above to check if this was true.

(f) (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 $n = 2^k - 1$, $k = 1, 2, \ldots, 11$ and plot the number of comparisons vs. the size of the vector.




| Target | # of comparisons |
|---|---|
| 1 | 0 |
| 2 | 1 |
| 4 | 2 |
| 8 | 3 |
| 16 | 4 |
| 32 | 5 |
| 64 | 6 |
| 128 | 7 |
| 256 | 8 |
| 512 | 9 |
| 1024 | 10 |
| 2048 | 11 |

(g) (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.

The number of comparisons in the experiment above would be the same for a vector of strings or doubles because vectors of strings and vector of doubles have integer representations and for this reason the same binary search algorithm can be implemented on those vectors.

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

Observing the trend in the charts above we can conclude that the Big-O asymptotic notation that classifies the binary search algorithm is $O(n) = \log_2(n)$ .

(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.

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 ≥ 1 integers.
    Output: The sum of the elements in A.
s ← A[0]
for i ← 1 to n − 1 do
      s ← s + A[i]
end for return
s
```

There is only one loop in the algorithm which runs a total of n-1 number of times. All other statements are constant time. Therefore, the time complexity of this algorithm will be O(n), the algorithm is linear. The # of operation performed on variable is 1 + n-1 = n.

```
Algorithm Ex2(A):
    Input: An array A storing n ≥ 1 integers.
    Output: The sum of the elements at even positions in A.
s ← A[0]
for i ← 2 to n − 1 by increments of 2 do
    s ← s + A[i]
end for
return s
```

The only for loop is repeated (n-2)/2 times to add even numbers (divided by 2 bc of the +2). All other statements are constant time. In Big O notation time complexity = O((n - 2) / 2)  = O(n / 2), the algorithm is linear.

```
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]
      end for
end for
return s
```

In this algorithm , the outer for loop is repeated (n-1) times and inner for loop is repeated a maximum number of (n-1) times. All other statements are constant time.  In Big O notation time complexity = (n - 1) * (n - 1)  = O(n^2) , the algorithm is quadratic.

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

the only for loop is repeated (n - 1) times repeated. All other statements are constant time. In Big O notation time complexity = O(n - 1) = O(n) , the algorithm is linear.

```
Algorithm Ex5(A, B):
    Input: Arrays A and B storing n ≥ 1 integers.
    Output: The number of elements in B equal to the partial sums in A.
c ← 0 //counter
for i ← 0 to n − 1 do
    s ← 0 //partial sum
    for j ← 0 to n − 1 do
        s ← s + A[0]
        for k ← 1 to j do
            s ← s + A[k]
        end for
    end for
    if B[i] = s then
        c ← c + 1
    end if
end for
return c
```

In this algorithm , the outer for loop is repeated (n) times and inner for loop is repeated a maximum number of m times where m is (1+2+...n). Therefore the time function T(n) = n(1 + 2 +…+ n) simplifies to n3 / 2 + n2 / 2 = O(n^3). In Big O notation time complexity = O(n^3), the algorithm is cubic.

Question 1.CPP

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

// O(n) solution:
bool OddProduct(int n, vector<int> nums)
{
    // odd x odd = odd
    // odd x even = even
    // even x even = even
    // therefore we need at least 2 odd numbers
    // in the vector to get an odd product
    int counter = 0;
    for (int i = 0; i < n; i++)
    {
        if (nums.at(i) % 2 == 1)
        {
            counter++;
        }
        if (counter >= 2)
        {
            return true;
        }
    }
    return false;
}

// O(n^2) solution:
bool OddProduct(int n, vector<int> nums)
{
    int counter = 0;
    int final_product = 0;
    for (int x = 0; x < n; x++)
    {
        for (int y = x + 1; y < n; y++)
        {
            final_product = nums.at(x) * nums.at(y);
            if (final_product % 2 == 1)
            {
                return true;
            }
        }
    }
    return false;
}
// || What do you consider as an operation for each algorithm? ||

/* O(n) solution: Since there is only one loop for the OddProduct() function the
time complexity is O(n).
```

```
   O(n^2) solution: Since there is two loops for the OddProduct() function the ti
me complexity is O(n^2). */

// | Are the best and worst cases for both the algorithms the same in terms of Bi
g-O notation? Justify your answer.|

/*  O(n) Best Case:
    The best case scenario is when the first two values of the vector are odd.
    In that case, we don't check the rest of the vector so the time complexity is
 O(1).

    O(n^2) Best Case:
    The best case scenario is when the first pair of numbers multiplied has an od
d product.
    In that case the time complexity is O(1).

    O(n) Worst Case:
    The worst case scenario is when there are no odd numbers or
    the final two values of the vector are odd. In that case, the time complexity
 is O(n).

    O(n^2) Worst Case:
    The worst case scenario is when no two pair products are odd.
    This causes us to go all the way through the two nested loops
    therefore the time complexity is O(n^2).

    According to my observations above the best cases of the algorithms are the s
ame : O(1).
    While the worst cases are different : O(n) vs O(n^2).
*/

// | Describe the situations of getting the best and worst cases, give the sample
s of the input for eachcase and check if your running time functions match the nu
mber of operations.|

/*
    O(n) Best Case Example:
    input : 4 , [5, 7, 2, 4]
    Since the first two numbers of the vector are odd we get an odd product.
    This results in us returning true and a constant time O(1).

    O(n^2) Best Case Example:
    input : 4 , [5, 7, 2, 4]
    Since the first pair is 5 and 7 and the product of 5 * 7 = 35,
    The function returns true. Therefore, the time complexity is O(1) .
    This is because The first pair we check has a odd product.

    O(n) Worst Case Example:
    input : 6 , [0,2,4,6,8,10]
    When we loop through the vector we find no odd numbers.
    Therefore we return false and since we go through the one for
```

```
    loop in its entirety the time complexisty is O(n)

    O(n^2) Worst Case Example:
    input : 6 , [0,2,4,6,8,10]
    If we go through the two nested loops in this case we won't
    find any pairs of numbers that result in an odd product.
    This is because all numbers in the vector are even.
    Since we go through the two nested loops in their entirety,
    we get a time complexity of O(n^2).
```

Question 2 CPP

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

// global variables
int num_comp = 0;
bool sorted = true;

//Implementation of Binary search given in assignment

template <typename T>
T BinarySearch(vector<T> &v, T x)
{
    int mid, low = 0;
    int high = (int)v.size() - 1;
    while (low < high)
    {
        mid = (low + high) / 2;
        if (v[mid] < x)
        {
            low = mid + 1;
            num_comp++;
        }
        else
        {
            high = mid;
            num_comp++;
        }
    }
    if (x == v[low])
        return low;               //OK: found
    throw("Unsuccessful_Search"); //exception: not found
}

int main()
{

    //creating vector
```

```
    vector<int> v;
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    v.push_back(4);
    v.push_back(5);
    v.push_back(6);
    v.push_back(7);
    v.push_back(8);
    v.push_back(9);
    v.push_back(10);
    v.push_back(11);
    v.push_back(12);
    v.push_back(13);
    v.push_back(14);
    v.push_back(15);
    v.push_back(16);

    //checking if vector is sorted before calling binary search function [Part B]
    for (int x = 0; x < (v.size() - 1); x++)
    {
        if (v.at(x) > v.at(x + 1))
        {
            sorted = false;
            throw("the input vector is unsorted"); //exception: the vector is uns
orted
        }
    }
    num_comp = 0;
    BinarySearch(v, 1);
    cout << " the value of num_comp in the cases when the target x is the first e
lement of the vector v: " << num_comp << endl;
    num_comp = 0;
    BinarySearch(v, 8);
    cout << " the value of num_comp in the cases when the target x is the last el
ement of the vector v : " << num_comp << endl;
    num_comp = 0;
    BinarySearch(v, 16);
    cout << " the value of num_comp in the cases when the target x is the middle
element of the vector v :" << num_comp << endl;

    vector<int> v2;
    v2.push_back(1);
    num_comp = 0;
    BinarySearch(v2, 1);
    cout << num_comp << endl;

    vector<int> v3;
    v3.push_back(1);
    v3.push_back(2);
    num_comp = 0;
```

```
    BinarySearch(v3, 2);
    cout << num_comp << endl;

    vector<int> v4;
    v4.push_back(1);
    v4.push_back(2);
    v4.push_back(3);
    v4.push_back(4);
    num_comp = 0;
    BinarySearch(v4, 4);
    cout << num_comp << endl;

    vector<int> v5;
    for (int i = 1; i <= 8; i++)
    {
        v5.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v5, 8);
    cout << num_comp << endl;

    vector<int> v6;
    for (int i = 1; i <= 16; i++)
    {
        v6.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v6, 16);
    cout << num_comp << endl;

    vector<int> v7;
    for (int i = 1; i <= 32; i++)
    {
        v7.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v7, 32);
    cout << num_comp << endl;

    vector<int> v8;
    for (int i = 1; i <= 64; i++)
    {
        v8.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v8, 64);
    cout << num_comp << endl;

    vector<int> v9;
    for (int i = 1; i <= 128; i++)
    {
```

```
    v9.push_back(i);
}
num_comp = 0;
BinarySearch(v9, 128);
cout << num_comp << endl;

vector<int> v10;
for (int i = 1; i <= 256; i++)
{
    v10.push_back(i);
}
num_comp = 0;
BinarySearch(v10, 256);
cout << num_comp << endl;

vector<int> v11;
for (int i = 1; i <= 512; i++)
{
    v11.push_back(i);
}
num_comp = 0;
BinarySearch(v11, 512);
cout << num_comp << endl;

vector<int> v12;
for (int i = 1; i <= 512; i++)
{
    v12.push_back(i);
}
num_comp = 0;
BinarySearch(v12, 512);
cout << num_comp << endl;

vector<int> v13;
for (int i = 1; i <= 1024; i++)
{
    v13.push_back(i);
}
num_comp = 0;
BinarySearch(v13, 1024);
cout << num_comp << endl;

vector<int> v14;
for (int i = 1; i <= 2048; i++)
{
    v14.push_back(i);
}
num_comp = 0;
BinarySearch(v14, 2048);
cout << num_comp << endl;
```

```
    cout << "||||||||||||||||||||" << endl;
    vector<int> v15;
    v15.push_back(1);
    num_comp = 0;
    BinarySearch(v15, 1);
    cout << num_comp << endl;

    vector<int> v16;
    v16.push_back(1);
    v16.push_back(2);
    v16.push_back(3);
    num_comp = 0;
    BinarySearch(v16, 3);
    cout << num_comp << endl;

    vector<int> v17;
    for (int i = 1; i <= 7; i++)
    {
        v17.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v17, 7);
    cout << num_comp << endl;

    vector<int> v18;
    for (int i = 1; i <= 15; i++)
    {
        v18.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v18, 15);
    cout << num_comp << endl;

    vector<int> v19;
    for (int i = 1; i <= 31; i++)
    {
        v19.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v19, 31);
    cout << num_comp << endl;

    vector<int> v20;
    for (int i = 1; i <= 63; i++)
    {
        v20.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v20, 63);
    cout << num_comp << endl;
```

```
    vector<int> v21;
    for (int i = 1; i <= 127; i++)
    {
        v21.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v21, 127);
    cout << num_comp << endl;

    vector<int> v22;
    for (int i = 1; i <= 255; i++)
    {
        v22.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v22, 255);
    cout << num_comp << endl;

    vector<int> v23;
    for (int i = 1; i <= 511; i++)
    {
        v23.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v23, 511);
    cout << num_comp << endl;

    vector<int> v24;
    for (int i = 1; i <= 1023; i++)
    {
        v24.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v24, 1023);
    cout << num_comp << endl;

    vector<int> v25;
    for (int i = 1; i <= 2047; i++)
    {
        v25.push_back(i);
    }
    num_comp = 0;
    BinarySearch(v25, 2047);
    cout << num_comp << endl;
}
```