CSCI 3333 Practice Midterm #1

- Do not start until instructed to do so.
- Write your UTRGV ID only in the space provided at the top of this page.
- The midterm is closed - no books, notes, computers, cell phones, calculators, etc.
- The time allotted for the exam is 70 minutes.
- There are 7 questions worth 28 points total; each problem is worth 4 points.
- *These questions are somewhat more challenging than the real midterm questions.*

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1. Select whether $\circ$ should be $O$, $\Omega$, or $\Theta$ (select $\Theta$ if possible):

\[
\begin{align*}
    n^2 &= \circ(n^3) & \square \Theta & \square O & \square \Omega \\
    n \log_2(n) &= \circ(5n) & \square \Theta & \square O & \square \Omega \\
    n &= \circ(\log(n)) & \square \Theta & \square O & \square \Omega \\
    10n \log_2(n) &= \circ(n \log(n)) & \square \Theta & \square O & \square \Omega
\end{align*}
\]

Problem 2. For each scenario, pick whether a dynamic-array-based list, i.e. vector, or a doubly-linked-list-based list, i.e. list, is faster.

Accessing elements in the middle of the list. \square vector \square list

Adding many elements to both the front and back of the list. \square vector \square list

Adding elements to the back of the list. \square vector \square list

Removing elements from the middle of the list using an iterator. \square vector \square list
Problem 3. What does the following program print? Draw a box around your solution.

```cpp
#include <vector>
#include <iostream>

using namespace std;

template <typename T>
void add(T &x, T y)
{
    T tmp = x + y;
    x = tmp;
    y = tmp;
}

void add(vector<int> &V, int y)
{
    for (vector<int>::iterator i = V.begin(); i != V.end(); ++i)
        add(*i, y);
}

int main()
{
    vector<int> A;
    A.push_back(1);
    A.push_back(2);
    A.push_back(3);
    A.push_back(4);
    add(A, 7);
    for (int x : A)
        cout << x << endl;
}
Problem 4. Consider the following function that returns whether an element \( x \) is in a float array \( A \) of length \( n \):

```c
bool search(float x, float* A, int n)
{
    if (n < 1)
        return false;
    if (A[0] == x)
        return true;
    return search(x, &(A[1]), n-1);
}
```

Fill in the blanks with answers based on the above function \texttt{search}:

A worst-case input \( x \) and length-6 array \( A \) are ________________________________.

The recurrence relation \( f(n) = \) ________________________________ with \( f(1) = \) ________ describes the worst-case running time of \texttt{search}.

Using repeated substitution once on the recurrence relation gives \( f(n) = \) ________________.

A closed form for the running time of \texttt{search} is \( f(n) = \) ________________.

Problem 5. Determine the truth of the following statements about the asymptotic running times of sorting algorithms.

Every algorithm has the same worst-case and best-case running times.  \( \square \) True  \( \square \) False

An algorithm can have a worst-case running time of \( O(\log(n)) \)  \( \square \) True  \( \square \) False

For every input, mergesort is as fast or faster than quicksort.  \( \square \) True  \( \square \) False

Quicksort runs in \( O(n) \) time for some inputs.  \( \square \) True  \( \square \) False
Problem 6. Suppose you’re given a function that performs binary search on a sorted array $A$ and has the following prototype:

```c
bool search(float x, float* A, int n)
```

Write an C++ function named `triple` that returns whether a sorted array $A$ has any three (not necessarily distinct) elements that sum to 0, has the following prototype:

```c
bool triple(float* A, int n)
```

and has a worst-case running time less than $\Theta(n^3)$. Give the worst-case asymptotic running time of your function. Provide arguments for both upper and lower bounds for this running time.
Problem 7. Complete the following implementation of counting sort.

```c
void sort(char* A, int n)
{
    int H[____];

    for (int i = 0; i < _____; ++i)
    {
        H[i] = _____;
    }

    for (int i = 0; i < n; ++i)
    {
        H[A[i]] += _____;
    }

    int i = 0;
    int h = 0;
    while (h < _____)
    {
        if (H[h] == _____)
        {
            ++h;
        }
        else
        {
            A[i] = _____;
            ++i;
            H[h] = H[h] _____ 1;
        }
    }
}
```