CS 302
Data Structures and Algorithm Analysis

Indiana State University

Final Exam

Name: Tristian [illegible]

Question	Points	Score
Problem 1	15	
Problem 2	7	
Problem 3	18	
Problem 4	9	
Problem 5	6	
Problem 6	90	
Total:	145	

Problem 1. (15 points) For each of the following statements, circle at most one answer indicating whether you think that statement is **true** or **false**. A statement is true (T) if it is **always** true, while it is false if there is at least one counterexample that negates it.

1. $f(n) = n^3 + 2$ is $\mathrm{O}\left(n^4\right)$.
   A. True.
   B. False.

   True

2. If $f(n)$ is $\mathrm{O}\left(n^2\right)$ and $g(n)$ is $\mathrm{O}\left(n^2\right)$, then $\frac{f(n)}{g(n)}$ is $\mathrm{O}\left(1\right)$.
   A. True.
   B. False.

   False.

   $5n$ is $O(n^2)$ $\frac{n^2}{5n} = \frac{n}{5}$.
   n^2 is $O(n^2)$
   $\frac{n}{5}$ is not $O(1)$.

3. $n^2 + 700n$ is $\mathrm{O}\left(n^2\right)$.
   A. True.
   B. False.

   True.

4. $n^2 + 700n$ is $\mathrm{O}\left(n^3\right)$.
   A. True.
   B. False.

   True

5. When implementing a queue, the asymptotic runtime of the `enqueue` operation is always the same regardless of the implementation of the queue.
   A. True.
   B. False.

   False.

Problem 2. (7 points) Using the definition of the Big-Oh notation, prove or disprove the following statement:

$$n^4 - 7 \times n^3 + 4 \times n^2 - 3 \text{ is } \mathrm{O}\left(n^4\right) .$$

$n^4 - 7n^3 + 4n^2 - 3$ is $O(n^4)$

$n^4 = n^4$ there exists a constant
$7n^3 \leq 7n^4$ for each term of f
$4n^2 \leq 4n^4$ that can be multiplied
$3 \leq 3n^4$ with n^4 where $c \cdot n^4$
is greater than the term
of f.

Problem 3. For each of the following code snippets, give exact formulas and Big-Theta notation of the number of times the statement `sum++` executes. Your answers should be simplified closed-form functions of n. Please **show your work** and give explanations when calculating the formulas for the number of executions. However, you do not need to prove any Big-Theta statement, you should only state it directly.

In other words, you should give an argument (either in the form of a counting table, or a clear argument) when stating the counting formula. Then you can state the Big-Theta asymptotic runtime directly.

(a) (9 points) In the following code snippet, you may assume that n is a multiple of 2.

```
1 for(int i = n; i >= 5; i--) {
2   for(int k = 0; k < n; k = k + 2) {
3     sum++;
4   }
5 }
```

n=6

i	k
6	0, 2, 4
5	0, 2, 4

for { # runs n-4 times.
  for { # runs n/2 times
    sum ++
  } $\frac{(n-4)\cdot n}{2} = \frac{n^2-4n}{2}$ $\Theta(n^2)$
}

n=10

i	k
10	0, 2, 4, 6, 8
9	0, 2, 4, 6, 8
8	0, 2, 4, 6, 8
7	0, 2, 4, 6, 8
6	0, 2, 4, 6, 8
5	0, 2, 4, 6, 8

(b) (9 points) In the following code snippet, you may assume that n is a power of 2.

```
1 for(int i = 2; i <= n; i = i * 2) {
2   for(int k = 1; k <= i; k++) {
3     sum++;
4   }
5 }
```

Hint: You may find the summation $\sum_{k=0}^{m} 2^k = 2^{m+1} - 1$ useful.

$n = 2^m$

$n = 2^m$ a power of 2.

$\sum_{k=1}^{m} 2k = 2^{m+1} - 2 = 2 \cdot 2^m - 2$

substitute n....

$= 2n - 2 \rightarrow 2(8) - 2 = 14$

exact: $2n - 2$

$\Theta(n)$

n=8

i	k
2	1, 2
4	1, 2, 3, 4
8	1, 2, 3, 4..., 8

n=8
sum=14

Problem 4. (9 points) We are interested in implementing a new data structured called `SortedList` that is simply a sorted array of elements. A `SortedList` needs to support three methods:

1. `insert`: Inserts an element into the list.
2. `find_min`: Finds the minimum element in the list.
3. `print`: Prints the list elements in sorted order.

Your task is to decide on the implementation of the `SortedList` data structure. What really matters is that the list should *appear* to be sorted. In other words, it doesn't matter if the elements stored in the data structure are not sorted, as long as when `print` is called, they appear in sorted order. This means that one possibility is to sort the list only before printing.

In what follows, you may assume that n represents the number of elements in the `SortedList`. Additionally, if a binary search tree is involved, you may use `H` to represent the height of the tree.

Consider the following three possible implementations:

1. A binary search tree with nodes compared by value: This is a simple BST where each node contains an element of the list and the elements are compared by value before being added to the list.
2. A **sorted** linked list with both head and tail pointers: This implementation uses a linked list. However, the linked list **always** maintains sorted order, so every insert operation will find the right spot for the element and will insert it there. In other words, the linked list is always sorted.
3. An **unsorted** growable array: This array grows dynamically by doubling its capacity every time it runs out of space. However, the array does not maintain any ordering constraints on its elements; new elements are always added at the end! When `print` is called, the growable array is then sorted using **insertion sort** to show the correct representation.

In the table below, for each one of the implementations and corresponding operation, give the worst-case Big-Oh runtime. However, if the worst-case and the amortized runtimes are different, then please list out both.

could be O(n) if BST is like a list...

Implementation	`insert`	`find_min`	`print`
A binary search tree with nodes compared by value.	$\log_2(n)$ (assuming tree is balanced).	O(height)	$O(n)$
A **sorted** linked list with both head and tail pointers.	$O(n)$	$O(1)$	$\Theta(n)$
An **unsorted** growable array that uses the doubling strategy to grow its capacity.	worst $O(n)$ Amort. $O(1)$	$O(n)$	$O(n^2)$

Problem 5. You are hired to help design the backend of a new movie streaming service in Terre Haute, called *THFlix*. For each one of the design problems below, give the most appropriate data structure to use along with its Big-Theta runtime for the operation in question. Please provide a brief explanation (one or two sentences) for each design decision you make.

Note: If the runtime of a data structure is implementation dependent, then please list the implementation next to the runtime stated. For example, insert into an array if $O(1)$ if implemented as a linked list but amortized $O(1)$ and worst cast $O(n)$ if implemented as a growable array.

(a) (3 points) Once users start browsing the platform, *THFlix* would like to give them the opportunity to go back to titles they have just viewed. Any time the user presses the back button, you should be able to pull up the last title they have viewed and show it to them. If they press the button again, then go back to the one before that, and so on.

- Suggested data structure: linked list
- Big-theta runtime of getting the last title viewed: $\Theta(1)$.

the list grows as they watch. so the most recently watched is the tail of the list. each new node points to the last one. so as you go back its always $O(1)$

(b) (3 points) In order to make profit, *THFlix* plans to always collect users' viewing history so it can later sell it to advertisers (because why not?). However, they don't have much space to store all of the viewing history, so they will limit it to one week.

During the week, every title that the user views is added the history. At the end of each week, *THFlix* will remove titles from a user's viewing history in the order of earliest viewing. In other words, if on Sunday, *THFlix* starts removing titles from the history data structure, then titles viewed on Monday come out first, then Tuesday, then Wednesday, and so on.

- Suggested data structure: linked list.
- Big-theta runtime of removing a title from the history: $\Theta(1)$.

The oldest movie watched is at the head of the list. so removing the oldest one is $\Theta(1)$.

Problem 6. (90 points) Please see the `readme.md` file in the finalexam code directory. You may open that directory directly in Visual Studio Code and it will autoload all of the tests for you.