Sample Problems and Solutions
Data Structures
State University of New York at Stony Brook
Instructor: Prof. Pramod Ganapathi

[Algorithm analysis.]

1. [15 points] What is the order of complexity of the following block(s) of code? Show detailed steps to prove your answer.

```
CodeBlock-1($n$)
1. $count \leftarrow 0$
2. for $i \leftarrow n; i > 0; i \leftarrow i/4$ do
3.     for $j \leftarrow 0; j < i; j \leftarrow j + 1$ do
4.         $count \leftarrow count + 1$
```

```
CodeBlock-2($n$)
1. $count \leftarrow 0; i \leftarrow 1$
2. while $i \leq n$ do
3.     $j \leftarrow n$
4.     while $j \geq 1$ do
5.         $count \leftarrow count + 1$; $j \leftarrow j/3$
6.     $i \leftarrow i \times 2$
```

Solution:

Time for CodeBlock-1:

$= \sum_{i \in \left\{\frac{n}{4^0}, \frac{n}{4^1}, \frac{n}{4^2}, \ldots, \frac{n}{4^{\lfloor \log_4 n \rfloor}}\right\}} \sum_{j \in \{0,1,2,\ldots,i\}} 1$

$= \sum_{i \in \left\{\frac{n}{4^0}, \frac{n}{4^1}, \frac{n}{4^2}, \ldots, \frac{n}{4^{\lfloor \log_4 n \rfloor}}\right\}} (i+1)$

$= \sum_{i \in \left\{\frac{n}{4^0}, \frac{n}{4^1}, \frac{n}{4^2}, \ldots, \frac{n}{4^{\lfloor \log_4 n \rfloor}}\right\}} i + \sum_{i \in \left\{\frac{n}{4^0}, \frac{n}{4^1}, \frac{n}{4^3}, \ldots, \frac{n}{4^{\lfloor \log_4 n \rfloor}}\right\}} 1$

$= \left(\frac{n}{4^0} + \frac{n}{4^1} + \frac{n}{4^2} + \cdots + \frac{n}{4^{\lfloor \log_4 n \rfloor}}\right) + \log_4 n$

$= \Theta(n) + \Theta(\log n)$ (using geometric series sum)

$= \Theta(n)$

Time for CodeBlock-2:

$= \sum_{i \in \left\{2^0, 2^1, 2^2, \ldots, 2^{\lfloor \log n \rfloor}\right\}} \sum_{j \in \left\{\frac{n}{3^0}, \frac{n}{3^1}, \frac{n}{3^2}, \ldots, \frac{n}{3^{\lfloor \log_3 n \rfloor}}\right\}} 1$

$= \sum_{i \in \left\{2^0, 2^1, 2^2, \ldots, 2^{\lfloor \log n \rfloor}\right\}} 1 \times \sum_{j \in \left\{\frac{n}{3^0}, \frac{n}{3^1}, \frac{n}{3^2}, \ldots, \frac{n}{3^{\lfloor \log_3 n \rfloor}}\right\}} 1$ (independent sums)

$= (\lfloor \log_2 n \rfloor + 1) \times (\lfloor \log_3 n \rfloor + 1)$

$= \Theta(\log n) \times \Theta(\log n)$

$= \Theta\left(\log^2 n\right)$

2. [10 points] Write the following functions $f(n)$ in the form $f(n) \in \Theta(g(n))$, where

$g(n)$ is the simplified function. (i) $f(n) = n^{1/\log n}$, (ii) $f(n) = \log\sqrt{4^n}$, (iii) $f(n) = 2^{\log_4 n}$, (iv) $f(n) = 7^{\log_{49\times 49} n}$, and (v) $f(n) = (\log\log n)^{1/\log\log\log n}$

Solution:
Please add the reasoning for individual steps

1. $f(n) = n^{1/\log n} = (2^n)^{1/\log n} = 2$ (because $n = 2^{\log_2 n}$)
   $= \Theta(1)$
2. $f(n) = \log\sqrt{4^n} = \log 4^{\frac{n}{2}} = \log(2^2)^{\frac{n}{2}} = \log 2^n$
   $= \Theta(n)$
3. $f(n) = 2^{\log_4 n} = 2^{\frac{\log n}{\log 4}} = 2^{\frac{\log n}{2}} = (2^{\log n})^{1/2} = n^{1/2} = \sqrt{n}$
   $= \Theta(\sqrt{n})$
4. $f(n) = 7^{\log_{49\times 49} n} = 7^{\log_{7^4} n} = n^{\log_{7^4} 7} = n^{\frac{\log_7 7}{\log_7 7^4}} = n^{\frac{\log_7 7}{4\log_7 7}} = n^{\frac{1}{4}}$
   $= \Theta\left(n^{1/4}\right)$
5. $f(n) = (\log\log n)^{1/\log\log\log n} = \left(2^{\log\log\log n}\right)^{1/\log\log\log n} = 2$
   $= \Theta(1)$

3. [10 points] Rank the following functions by order of growth. That is, arrange the functions in asymptotically nondecreasing order using $\prec$ and $\asymp$ relations. Give reasons for your answers. Assume that the log function has base 2 unless explicitly mentioned otherwise.

   $2^{\sqrt{2\log n}}$, $(\log n)!$, $\log^2 n$, $(\log n)^{\log n}$, $\log(n!)$

4. [20 points] Assume that you want to implement the following ADT using several different data structures. Simply mention the worst-case time complexity for different methods using different data structures given in an empty table. Provide the complexities using $\Theta()$ notation wherever feasible and use $\mathcal{O}$ notation in the remaining places. Give the amortized complexity for dynamic array implementations and average case complexity for hash table implementations assuming that the load factor is small and the hash function distributes keys uniformly. The symbols $\star$ represents amortized case complexity and $\dagger$ represents average/expected case complexity.

   Solution: See Tables 1 and 2.

Operation	AdjacencyList	AdjacencyMatrix
CheckEdgeExistence(u, v)	$\mathcal{O}(\text{degree}(u))$	$\Theta(\|1\|)$
GetNeighbors(v)	$\mathcal{O}(\text{degree}(u))$	$\Theta(\|V\|)$
Degree(v)	$\Theta(1)$	$\mathcal{O}(\|V\|)$
Traversal (BFS/DFS)	$\Theta(\|V\| + \|E\|)$	$\Theta(\|V\|^2)$
ListAllEdges	$\Theta(\|V\| + \|E\|)$	$\Theta(\|V\|^2)$
Space	$\Theta(\|V\| + \|E\|)$	$\Theta(\|V\|^2)$

Table 1: Time complexities for different operations on graphs.

Data Structure	Operation 1	Operation 2	Operation 3	Operation 4
Basic Data Structures	AddFirst	AddLast	RemoveFirst	RemoveLast
DynamicArray	$\Theta(n)$	$\Theta(1)^\star$	$\Theta(n)$	$\Theta(1)^\star$
SinglyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$
CircularlySinglyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$
DoublyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Stack ADT	Push	Pop	Top	
DynamicArray	$\Theta(1)^\star$	$\Theta(1)^\star$	$\Theta(1)$	
SinglyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	
CircularlySinglyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	
DoublyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	
Queue ADT	Enqueue	Dequeue	Front	
CircularDynamicArray	$\Theta(1)^\star$	$\Theta(1)^\star$	$\Theta(1)$	
SinglyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	
CircularlySinglyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	
DoublyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	
Deque ADT	AddFirst	AddLast	RemoveFirst	RemoveLast
CircularDynamicArray	$\Theta(1)^\star$	$\Theta(1)^\star$	$\Theta(1)^\star$	$\Theta(1)^\star$
SinglyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$
CircularlySinglyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$
DoublyLinkedList	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Sorted Set ADT	Add	Remove	Search	SortedOrder
DynamicArray	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\Theta(n \log n)$
SortedDynamicArray	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\Theta(n)$
BinarySearchTree	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\Theta(n)$
(2-4)-Tree	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\Theta(n)$
B-Tree	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\Theta(n)$
RedBlack-Tree	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\Theta(n)$
AVL-Tree	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\Theta(n)$
Set ADT	Add	Remove	Search	
DynamicArray	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	
SortedDynamicArray	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	
HashTable-Chaining-List	$\Theta(1)^\dagger$	$\Theta(1)^\dagger$	$\Theta(1)^\dagger$	
HashTable-LinearProbing	$\Theta(1)^\dagger$	$\Theta(1)^\dagger$	$\Theta(1)^\dagger$	
HashTable-QuadraticProbing	$\Theta(1)^\dagger$	$\Theta(1)^\dagger$	$\Theta(1)^\dagger$	
PriorityQueue ADT	Add	RemoveMin	Minimum	
SinglyLinkedList	$\Theta(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	
SortedSinglyLinkedList	$\mathcal{O}(n)$	$\Theta(1)$	$\Theta(1)$	
MinHeap-DynamicArray	$\mathcal{O}(\log n)^\star$	$\mathcal{O}(\log n)^\star$	$\Theta(1)$	
MinHeap-LinkedTree	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\Theta(1)$	

Table 2: Time complexities for different operations on various data structures.

[Arrays and lists.]

1. [10 points] Given a string $S[1\dots n]$, write an algorithm code to check whether the given string is a palindrome or not. A palindrome is a string that reads the same backward as forward. E.g. madam, malayalam, and racecar. Give the algorithm's time and space complexity.

```
IsPalindrome($S[1\dots n]$)
1. Write the pseudocode
```

Solution:

```
IsPalindrome($S[1\dots n]$)
1. for $i \leftarrow 1$ to $\lfloor n/2 \rfloor$ do
2.     if $S[i] \neq S[n-i+1]$ then
3.         return false
4. return true
```

This algorithm compares the characters at the beginning and end of the string, moving towards the center. If any characters don't match, it returns false. If it completes the loop, the string is a palindrome.

$\boxed{\text{Time} = \mathcal{O}(n), \text{Space} = \mathcal{O}(1)}$

2. [10 points] Given an array $a[1\dots n]$ of non-negative integers, write a recursive algorithm code to compute the greatest common divisor GCD$(a[1\dots n])$ of these n numbers. Assume that $gcd(x,y)$ computes the greatest common divisor of non-negative integers x and y.

```
GCD($a[1\dots n]$)
1. Write the pseudocode
```

Solution:

```
GCD($a[1\dots n]$)
1. if $n = 1$ then return $a[1]$
2. else return $gcd(a[n], \text{GCD}(a[1\dots(n-1)]))$
```

The recursive algorithm computes the GCD of the last element and the GCD of the rest of the array. The base case is when there's only one element, in which case the GCD is the element itself. The time complexity is $\mathcal{O}(n \cdot \log M)$, where M is the maximum value in the array, due to the recursive nature of the GCD function.

$\boxed{\text{Time} = \mathcal{O}(n \log M), \text{Space} = \mathcal{O}(n)}$

[Stacks, queues, and deques.]

1. [10 points] Fill Table 3 with the contents of the data structures after executing different operations starting from the initial configuration. Note that when you remove an element from a data structure, you return the removed element to the caller. You do not need to worry about the data overflow problem. The top element in the stack, the first element in the queue, and the first element in the deque are highlighted using a box.

   Solution: See Table 4.

Operations	Stack S	Queue Q	Deque D
Initially	$1, 4, \boxed{8}$	$\boxed{7}, 3, 6$	$\boxed{9}, 5, 2$
D.AddLast(S.Pop() + Q.Dequeue())			
Q.Enqueue(D.RemoveFirst() − S.Pop())			
S.Push(D.RemoveLast() − Q.Dequeue())			
D.AddFirst(S.Pop() − Q.Dequeue())			
D.AddFirst(D.RemoveFirst() + D.RemoveLast())			

Table 3: Fill in the table with contents of the data structures.

Operations	Stack S	Queue Q	Deque D
Initially	$1, 4, \boxed{8}$	$\boxed{7}, 3, 6$	$\boxed{9}, 5, 2$
D.AddLast(S.Pop() + Q.Dequeue())	1, 4	3, 6	9, 5, 2, 15
Q.Enqueue(D.RemoveFirst() − S.Pop())	1	3, 6, 5	5, 2, 15
S.Push(D.RemoveLast() − Q.Dequeue())	1, 12	6, 5	5, 2
D.AddFirst(S.Pop() − Q.Dequeue())	1	5	6, 5, 2
D.AddFirst(D.RemoveFirst() + D.RemoveLast())	1	5	8, 5

Table 4: Table filled with contents of the data structures.

[Trees and priority queues.]

1. Draw each of the following evolving data structures twice that result from the following operations in that order into an initially empty data structure.

   (a) [10 points] binary search tree
   insert [100, 10, 30, 50, 70, 40, 90, 60], draw tree, delete [30], insert [20, 30], delete [70], insert [80, 70], draw tree.

   (b) [10 points] (2,4) tree
   insert [100, 10, 30, 50, 70, 40], draw tree, insert [90, 60, 20, 80], draw tree.

   (c) [10 points] min heap (tree representation)
   insert [100, 10, 30, 50, 70, 40, 90, 60], draw tree, deletemin, deletemin, insert [20, 80], deletemin, draw tree.

[Hash tables.]

1. Use the hash function $\textsc{HashCode}(key) = (2 \times key + 5)$ to fill a 11-entry hash table. Insert $[5, 16, 20, 39, 11, 94, 23, 88, 13, 44, 12]$, draw table, delete $[20, 44, 39]$, insert

$[98, 14]$, draw table. Assume collisions are handled by:

(a) [5 points] separate chaining $\rightarrow$ linked list

Bucket	0	1	2	3	4	5	6	7	8	9	10
Keys											

Bucket	0	1	2	3	4	5	6	7	8	9	10
Keys											

(b) [5 points] open addressing $\rightarrow$ linear probing

Bucket	0	1	2	3	4	5	6	7	8	9	10
Keys											

Bucket	0	1	2	3	4	5	6	7	8	9	10
Keys											

(c) [5 points] open addressing $\rightarrow$ quadratic probing

Bucket	0	1	2	3	4	5	6	7	8	9	10
Keys											

Bucket	0	1	2	3	4	5	6	7	8	9	10
Keys											

[Graphs.]

1. Consider a directed weighted graph as shown in Figure 1. Assume that the traversals are considered in increasing order and all adjacency lists are given in increasing order.

   Write the following graph representations:

   (a) [5 points] Adjacency-matrix representation

   (b) [5 points] Adjacency-list representation

   Show the ordering of vertices produced by the following algorithms.

   (a) [5 points] Depth-first search

   (b) [5 points] Breadth-first search

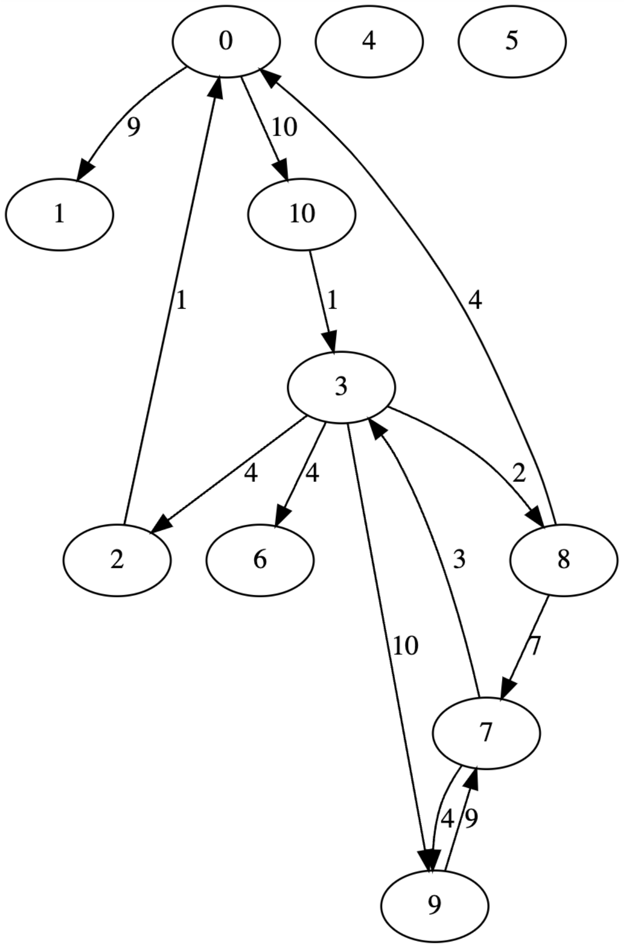

Figure 1: A directed weighted graph.