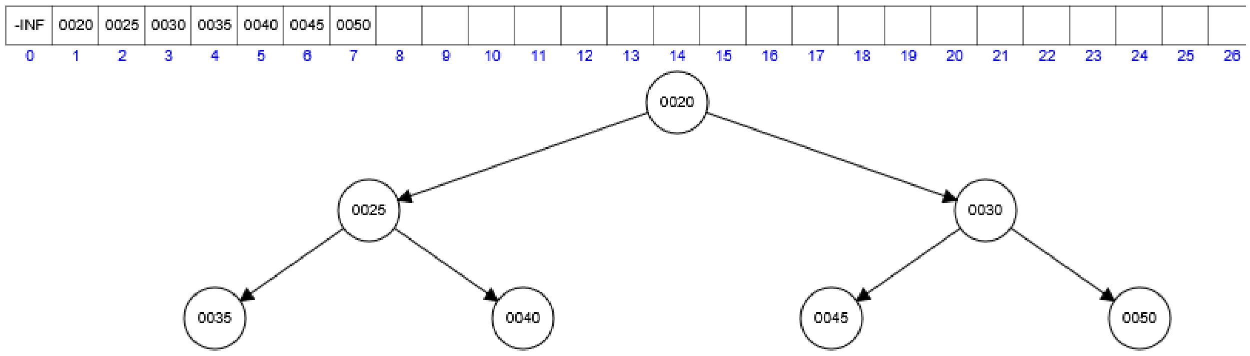
Ali Abdullah Ahmad
20031246

Assignment 3
CS 600

Q1. (No. 5.7.11)
Is there a heap T storing seven distinct elements such that a preorder traversal of T yields the elements of T in sorted order? How about an inorder traversal? How about a postorder traversal?

Ans.

Let us consider a Min – Heap Tree having 7 distinct elements 20, 25, 30, 35, 40, 45, 50.



The Preorder Traversal here is: 20, 25, 30, 35, 40, 45, 50
The Sequence for a Preorder Traversal is Root, Left Child, Right Child

The Inorder Traversal here is: 30, 25, 35, 20, 45, 40, 50
The Sequence for a Inorder Traversal is Left Child, Root, Right Child
In an inorder Traversal, parent is always listed after its left child. The parent is always the smallest in the tree. So, inorder traversal of a heap is not sorted.

The Postorder Traversal here is: 30, 35, 25, 45, 50, 40, 20
The Sequence for a Postorder Traversal is Left Child, Right Child, Root
In a Postorder Traversal, the left child traverse first, after which right child traverses and at the end, the parent traverses. So, the postorder traversal also does not give sorted order for the Tree T elements.

Q2. (No. 5.7.24)
Let T be a heap storing n keys. Give an efficient algorithm for reporting all the keys In T that are smaller than or equal to a given query key x(which is not necessarily in T). For example, given the heap of Figure 5.4.1 and query key x=7, the algorithm should report 4, 5, 6, 7. Note that the keys do not need to be reported in sorted order. Ideally, Your algorithm should run in O(k)time, where k is the number of keys reported.
Ans.

From the figure 5.4.1 we can see that this heap is a min-heap tree wherein the value of every internal nodes is less than or equal to all of its children's node. We will perform pre order traversal and check if the current node key is less than x if yes then we will print that key and recursively traverse the left and the right sub tree. If not, we will come out of the recursion.

```
Algorithm: reportKeys(Node node):
Input: A min-heap tree T with n nodes
Ouput: Printing all the keys in the tree whose key value is <= x

if(node == null or node.key > x) then

        return;

if(node.key <= x)

        Print(node.key)

reportKeys(node.left);
reportKeys(node.right);
return;
```

Intitally we will call the function reportKeys() by passing the root node as a parameter.
The time complexity of the above algorithm is O(k) where k is the number of keys reported and the space complexity is O(n).

Q3. (No. 5.7.28)
In a discrete event simulation, a physical system, such as a galaxy or solar system, is modeled as it changes over time based on simulated forces. The objects being modeled define events that are scheduled to occur in the future. Each event, e, is associated with time, te, in the future. To move the simulation forward, the event, e, that has smallest time, te, in the future needs to be processed. To process such an event, e is removed from the set of pending events, and a set of physical forces are calculated for this event, which can optionally create a constant number of new events for the future, each with its own event time. Describe a way to support events processing in a discrete event simulation in O(log n)time, where n is the number of events in the system.

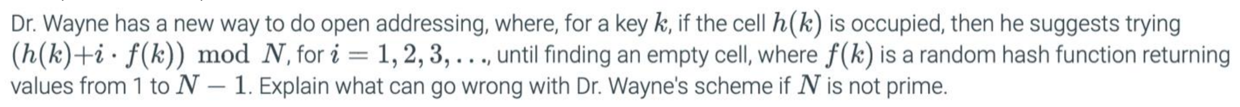
Ans.

We need to design a system that processes events in O(logn) time. Given a set of future events, each with a processing time te, we must efficiently select the event with the smallest te.
A linear search to find the minimum event takes O(n), which is inefficient. Instead, we use a **min-heap**, which allows us to access the minimum element in O(1) and insert events in O(logn).
Implementation Steps:

1. Use an array to implement the min-heap, initializing it with a maximum size and a size variable set to 0.
2. Set a[0] to negative infinity for reference.
3. To insert a new event:
    - Increment size and insert the value at a[size].
    - If this insertion violates the min-heap property, perform a **heap-up** operation to restore order.
4. To remove an event:
    - Retrieve the smallest event from a[1].
    - Replace a[1] with the last element a[size], decrement size, and perform a **heap-down** operation to maintain the heap structure.

Using this approach, we ensure efficient event processing with O(logn) complexity for both insertion and removal operations.

Q4. (No. 6.7.13)



Dr. Wayne has a new way to do open addressing, where, for a key k, if the cell $h(k)$ is occupied, then he suggests trying $(h(k)+i \cdot f(k)) \bmod N$, for $i = 1, 2, 3, \ldots$, until finding an empty cell, where $f(k)$ is a random hash function returning values from 1 to $N-1$. Explain what can go wrong with Dr. Wayne's scheme if N is not prime.

Ans.

Dr. Wayne's open addressing strategy uses probing to resolve collisions when inserting a key k into a hash table. If the initial hash position h(k) is occupied, the algorithm searches for the next available position using:
(h(k)+i·f(k)) mod N
where i=1,2, 3,…, and f(k) returns a random number between 1 and N−1.

Issue with Poor Hash Function Choices
If f(k) generates a fixed value, such as 5, and N is not carefully chosen, certain patterns can cause clustering. For example, if N=10 and f(k) = 5, then probing cycles through the same few locations, leading to poor distribution and increased collisions.

Solution: Using Prime Numbers
To avoid such clustering, N should be a **large prime number**, ensuring a uniform distribution of keys and reducing patterns in collisions. Additionally, f(k) should never evaluate to zero, which can be achieved using:
f(k)=q−(k mod q)
where q is a prime number less than N. This ensures that the entire hash table space is used efficiently, improving search and insertion performance.

Q5. (No. 6.7.17)
Suppose you are working in the information technology department for a large hospital. The people working at the front office are complaining that the software to discharge patients is taking too long to run. Indeed, on most days around noon there are long lines of people waiting to leave the hospital because of this slow software. After you ask if there are similar long lines of people waiting to be admitted to the hospital, you learn that the admissions process is quite fast in comparison. After studying the software for admissions and discharges, you notice that the set of patients currently admitted in the hospital is being maintained as a linked list, with new patients simply being added to the end of this list when they are admitted to the hospital. Describe a modification to this software that can allow both admissions and discharges to go fast. Characterize the running times of your solution for both admissions and discharges.

Ans.

The current system uses a **linked list** to manage patient admissions and discharges. While **insertion** is efficient at O(1), **removal** is slow at O(n) due to the need for a linear search. To optimize this, we introduce a **hash table** for faster lookups.
Improved Approach:

1. **Admission**:
   - Check if the patient exists in the **hash table**.
   - If not, add the patient to both the **linked list** and **hash table**, storing the linked list index as the hash value.
2. **Discharge**:
   - Retrieve the patient's index from the **hash table** in O(1).
   - If the patient is at the last index, remove them directly.
   - Otherwise, swap their record with the last patient in the list, update the hash table, and remove the last entry.

This optimization reduces removal time to **O(1)**, making patient management much more efficient.

Q6. (No. 6.7.25)
A popular tool for visualizing the themes in a speech is to draw a word cluster diagram, where the unique words from the speech are drawn in a group, with each word's size being in proportion to the number of times it is used in the speech. Given a speech containing n total words, describe an efficient method for counting the number of times each word is used in that speech. You may assume that you have a parser that returns the n words in a given speech as a sequence of character strings in O(n) time. What is the running time of your method?

Ans.

We can use hash table to count the frequency of the words used in the speech.
Our hash table will contain key as the word and the value will represent the number of times it is used in the speech.
Steps:
1. Declare a hashtable
2. Run a for loop to iterate over all the words.
3. Check if the given word is present in the hashtable. If yes then increment the hash table value for a particular word by 1. If not then add a new word in the hash table and set its value equal to 1.
4. Once all the words are iterate just print the hashtable.

The Time Complexity for the above Algorithm is O(n) because we are iterating over all the word one by once. The space Complexity is also O(n)