John Gill

Dr. Tala Talaei Koei

CS 5008

20 June 2024

Assignment 2

**Question 1**: What is a linked list, and how does it differ from an array? What are the advantages and disadvantages of using a linked list over an array?

Answer: A linked list is similar to an array in that it is a data structure consisting of multiple elements. Though, unlike an array, linked lists have nodes at each element that point to the next element in the sequence. Linked lists have dynamic memory allocation while arrays have static memory allocation, meaning a linked list can grow or shrink in size during execution while an array’s size cannot change. Linked lists also have an access time of O(n) where an array’s access time is O(1), but, those time complexities change for insertion and deletion: a linked list has a time complexity of O(1) for insertion and deletion, and arrays have a time complexity of O(n) for those functions. The dynamic size and lower time complexity for insertion/deletion make linked lists better alternatives to arrays, but their higher memory usage and slower element access times give arrays an advantage.  
**Question 2**: Describe the working principle of the Bucket Sort algorithm. Under what conditions is Bucket Sort most efficient?

Answer: Bucket Sort is a sorting algorithm that operates by dividing elements into multiple “buckets” and sorting them individually. It creates the buckets, distributes the elements into the different buckets, sorts each bucket, then concatenates all the sorted buckets back into the final sorted array. Bucket Sort is most efficient when the input is uniformly distributed for the range, and when the number of buckets is proportional to the number of elements.   
**Question 3**: What is a circular linked list? How does it differ from a linear linked list in terms of  
structure and applications? Also, discuss a scenario where you would choose to use a doubly linked list instead of other data structures in a C program.

Answer: A circular linked list is one in which the node of the final element points back to the first element, forming a circle without a NULL value at the end, unlike a linear linked list. Circular linked lists are preferable to linear linked lists when we need to cyclically iterate over the elements in a scenario that calls for endless scheduling. A doubly-linked list will have a NULL value at the end of the list in the same way a linear linked list has, but unlike a linear linked list, it has two nodes at each element: one pointing to the next element, and one pointing to the previous element. Doubly linked lists are preferable to other data structures in undo operations, like those found in text editors, or in navigation systems where forward and backward movement is necessary.   
**Question 4**: Discuss the time complexity of various operations (e.g., push, pop, peek) in a stack  
implemented using an array versus a linked list?

Answer: Push, pop, and peek operations have similar time complexities for arrays and linked lists. Push operations for both arrays and linked lists have a time complexity of O(1), though it can be O(n) for linked lists if the element needs to be added to the beginning of the list. Push can also have a time complexity of O(n) for arrays if the array is full, since it will need to be resized. Pop operations have a time complexity of O(1) for linked lists and arrays. The time complexity for the pop operation never changes for arrays, but it can change for linked lists if the element is not at the head of the list. In that case, the time complexity for the pop operation of an element in a linked list is O(n), where n is the number of elements in the list. And the peek operation is always O(1) for both linked lists an arrays, since peek is always returning the head element of whatever list or array we’re working with, and the head element never changes.   
**Question 5**: Can you describe an algorithm that uses a queue as an auxiliary data structure? Provide an example.

Answer: An algorithm that uses a queue as an auxiliary data structure is Breadth-First Search (BFS). BFS is an algorithm that searches graphs data structures by starting at a given node, then exploring its neighbors at the current depth before moving to the nodes at the next depth. As an example, we can initialize an empty queue and enqueue the starting node. Once the starting node is marked as visited, and while the queue is not empty, we can dequeue the current node, process it, then enqueue all the adjacent, unvisited nodes and mark them as visited, then print out their values as we move through them.  
**Question 6**: Write a C program to reverse a singly linked list. Provide both iterative and recursive solutions. Ensure that you implement the necessary functions to create, display, and free the linked list.

Answer: Code and screenshots attached in the .zip file.