Al-Gore-Rhythmic Ascent
Week 2 : Theory Questions

Raghav Sharma(24b1010)

June 9, 2025

Compare and contrast arrays vs linked lists in terms of memory usage, access time, and insertion/deletion operations.

Property	Arrays	Linked-List
Memory Usage	$O(n)$	$O(n)$ (but more than arrays(because it has to store the pointer along with value))
Access Time	$O(1)$	$O(n)$
Insertion	$O(n)$ (because copying of all previous elements is required)	$O(1)$
Deletion	$O(n)$ (again copying is required)	$O(n)$ (finding the specific element depends on then length of list)

Explain the trade-offs between different collision resolution techniques in hash tables.
Collision Handling can be done through *Open Addressing* and *Separate Chaining.*

Separate Chaining is where we use a linked list to store the key-value pairs hashing to the same index. It works well when all the buckets have similar number of key value pairs. It takes more space but could be implemented for any Load Factor(α). This technique becomes inefficient when there are way too many key value pairs in a few buckets, in which case operating(searching and deleting) in these buckets takes more time. Although if we wish to insert a lot and delete or search key-value pairs rarely separate chaining works pretty solidly.

The other technique Open Adressing works only if $\alpha < 1$ i.e. Loadfactor(α) is less than 1. This is the biggest drawback of Open Adressing. In Open Adressing we place Key-Value pairs whose bucket is filled in another bucket. This is perfect for saving memory and when we have datasets where many buckets could be empty. This is in general very effective but even here if some bucket has way too many key-value pairs, then it impacts the efficiency of our operations on the key-value pairs of this bucket negatively. It is also simpler to implement than Separate Chaining, but its biggest advantage remains its effective use of memory.

When would you choose Quick Sort over Merge Sort, and vice versa?
We should use Quicksort if

- We want to finish our work in less memory

- We want to know how the elements changed indices(for some reason) i.e. to which index did the element with ith indice go.

We Should use Mergesort if

- We want stable sorting.
- We are working with already sorted/reverse sorted or mostly sorted/reverse sorted data
- We have to sort in $O(nlog(n))$ only.

What makes a sorting algorithm "stable" and why does it matter?
If an algorithm preserves the order of "equal" elements, that is it is equal.
For example let's say we have two 2's in an array in the order 2 2', if this order always remains in the sorted array then the sorting algorithm is a stable one.
It doesn't really matter in case of the usual data types like int,char,string,bool,etc. but when a datatype stores more than one value, and we wish to sort that datatype by preferencing one value over the other then we must use a stable sorting algorithm.
To demonstrate this lets take an example of an array of pairs $\{\{5,6\},\{5,4\},\{4,9\}\}$ and we wish to sort this array in such a way that we sort the pairs by their first value and if it's equal then we sort them by their second value.
This could be easily done by a stable algorithm as we will simply first sort by second value to obtain $\{\{5,4\},\{5,6\},\{4,9\}\}$ and then sort by first value which would give us $\{\{4,9\},\{5,4\},\{5,6\}\}$, on the other hand if we were to use an unstable algorithm then we may end up with either $\{\{4,9\},\{5,4\},\{5,6\}\}$ or $\{\{4,9\},\{5,6\},\{5,4\}\}$.
Hence stable or unstable sort depend on the datatype we are working with. Note that if all the elements are distinct then both stable and unstable sort produce same results.