Searching Algorithms

Chapter No.	24
Status	Completed

▼ Linear Search

- Linear search is a searching algorithm that iterates through an array until the search value is found or the end of the array is reached
- O(n)

▼ Process

1. Iterate through the array and compare the element in the array with the search value
2. Continue to iterate through the array until the element matches the search value or the end of the array is reached and there are no more elements to iterate through

```
def linearsearch(lis, searchvalue):
    found = False

    for i in range(len(lis)):
        if lis[i] == searchvalue:
            found = True
            break

    if found:
        print("Item found at index:", i)
    else:
        print("Item not in list")
```

▼ Binary Search

- Binary search is a searching algorithm searches a sorted array by repeatedly dividing the search interval in half, until the middle element of a search interval matches the search value or the search interval is empty
- O(log n)

▼ Process

1. Compare the middle element of the array with the search value.
2. If this element matches the search value, the search value is found.
3. Else, if the search value is larger than the middle element of the array, narrow the search interval to the portion of the array that that comes after the middle element. If the search value is smaller than the middle element of the array, narrow the search interval to the portion of the array that comes before the middle element.
4. Repeat this process of dividing the search interval, until the middle element of a search interval matches the search value, or the search interval is empty.

```
def binarysearch(lis, searchvalue):
    L = 0
    R = len(lis)
    found = False
    search_failed = False

    while not found and not search_failed:
        middle = (L+R)//2

        if lis[middle] == searchvalue:
            found = True

        else:
            if L >= R:
                search_failed = True
            else:
                if searchvalue < lis[middle]:
                    R = middle-1
                else:
                    L = middle+1

    if found:
        print("Item found at index:", middle)
    else:
        print("Item not in list")
```

▼ Hash Tables

▼ A hash table is a data structure that stores data by mapping keys to values

- Each piece data can be separated into a key and a record, such that every piece of data has a unique key

▼ Hashing is the process of calculating the address (index) of a piece of data from its key using a hash function

- Ideally, a hash function should be a one-one function so that every unique key value gives a different address
- However, this is not always possible due to limitations, such as the speed of the hash function

▼ A collision is created when two different keys hash to the same address

▼ 3 Ways to Handle Collisions

▼ Chaining

- Create a linked list for collisions with the start pointer at the hashed address

▼ Closed Hashing

- Using overflow areas
- All collisions are stored in a separate area reserved for overflows

▼ Open Hashing

- Using neighbouring slots
- A nearby address with an empty bucket is found by performing a linear search from the hashed address and the new record is stored there

▼ Process of Storing/Retrieving Data in a Hash Table

1. The address at which the record of the piece of data belongs at is calculated by passing the key of the piece of data through the hash function (hashing)
2. This address in the hash table is checked
3. The record is stored at/retrieved from this address