DATA STRUCTURE AND ALGORITHM KNOWLEDGE:—

- ANDKIE:—

```
Function CompressFirstBox(boxes){
  Console.log(boxes[0]);
}
```

⟹ Constant time O(1)

$$O(n+(n\times n)) = O(n+n^2) = O(2n^2) = O(n^2)$$

- Alway big O measures the worst Case

- when a program excutes it has 2 ways to remember things → The heaps and Stack

heap is where we store variable

stack is where we keep track of our function calls.

Alocate = hold

Arrays ⟹ AlgoExpert

Get index an array ⟹ O(1) ST Set an array like array[2] = 5 ⟹ O(1) ST Initialize an array ⟹ O(N) ST

Traversing array for For loop ⟹ O(N) T, O(1) S

Copy array ⟹ O(n) ST

Insert array ⟹ O(N) T, O(1) S ⟹ because of wipe out the old one

pop() ⟹ O(1) ST

shift() ⟹ O(N) ST ⟹ because you're shifting.

DYNAMIC ARRAY

Dynamic Array uses languages like python, JS

[1] Append 2 to the array

[1, 2, 3, 4, 5, 6, 7, 8]

O(1) append | O(N) O(1) append | O(N) O(1) append | O(1) | O(1)

DYNAMIC ARRAY ARE ⇒ O(1) Time

shifting half of the array

$O(N) \Rightarrow O(0.25N)$ HALF

$N + \frac{N}{2} + \frac{N}{4} + \frac{N}{6} \cdots 1$

this converges $O(2N) \Rightarrow O(N)$

INSERTION ARRAY (O(N) T)

immutable means cannot be change ⇒ Tubles

mutable means can be changed ⇒ lists, Dictionaries

10 15 21 28 36 45

[1, 2, 3, 4, 5, 6, 7, 8, 9] Sum = 45 45 - 37 = 8

[1, 2, 3, 4, 5, 6, 7, 9] Sum = 37

LINKED LIST

Algo Expert

Bits :	128	64	32	16	8	4	2	1
Binary:								
Comp :								

8 4 2 1

0 1 1 1 ⇒ 7

Different between Arrays and linked list :-

→ Arrays stored back to back in memory

→ Linked list stored in any where in memory.

insertion in Linked List is always is: O(i) Time (Constant in TS)

and O(1) Space

Insertion in array ⇒ O(n) TS

16 + 49 + 4 + 64 + 25

This is to disable → csrutil disable + enter
This is to enable → csrutil enable
csrutil status

- The beginning of linked list is called ⇒ Head
The end of linked list is called ⇒ Tale.

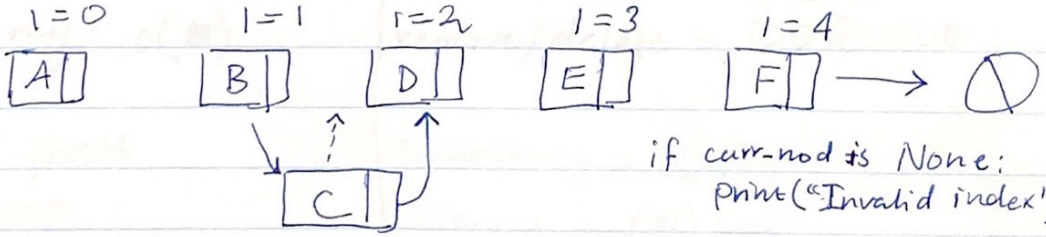
Single linked list ⇒ 3 → 1 → 4 → 2

Double linked list has two pointers ⇒ 3 ⇄ 1 ⇄ 4 ⇄ 2.
∅ null ∅ null

Single linked list that points one Node to another is called NEXT pointer in Code. (in Node class)

Double linked list that points one Node to another is calle NEXT and PREV pointer in Node class.

- Double linked list you can go back to head from Tale while Single linked list you can't go back.




```
if index = = 0:
    self.insert_head(data)
i = 0
curr_node = self.head
while (curr_node is not None) and (i < index - 1):
    curr_node = curr_node.next
    i += 1
if curr_node is None:
    print("Invalid index")
else:
    new_node = Node(data)
    new_node.next = curr_node.next
    curr_node.next = new_node.
```

inserting at the beggining or at the middle of the array is NOT constant time/operating. we have traversy the list.

STACK & Queues

Stack is like Books that are stack to each other adding and removing is gonna be taking from the top.
LAST IN FIRST OUT (LIFO)

Queues is like people paying tickets in queue.
FIRST IN FIRST OUT (FIFO) O(1)ST

- both stack an queues support in "PEEK METHOD" which runs constant ST peek method in both equevilant pop and dequeu (remove).

STACK:

O(1) time & space removing/deleting
O(1) time & space inserting

- Stacks are dynamic array
[1, 2, 3] → add append
remove → pop
- Searching takes Linear time in stack arrays. O(n)

insersation = push
deletion = pop
storing = O(N) space
- Compleation of stack:
E.G = Transform stack to a MAXSTACK or MINSTACK →
Stacks that keeps track of the largest element in it or the smallest element in it

Queue:

O(1) ST for deleting/adding.

- insersation method in queue called "ENQUEUE"
- Queue has to be implemented with linked list

insersation = ENQUEUE
remove/delete = DEQUEUE

Searching = O(N)T, O(1)S
storing = O(N) space
- Compleation of Queue:
E.G = Turn Queue to a PRIORITY Queue → which keeps track of an element with high priority.

DATA STRUCTURE STRING

- Strings stored in memory as an array of integers.
- it uses ASCII in memory A = 65 B = 66 a = 97
- ASCII has viewer than 256 characters

- Traverse string O(N) Time O(1) S
- Copying String O(N) ST
- GET (access string at given index) // O(1) ST

- C++ Strings are mutable
- Lots of languages like phython, java, JS strings are immutable
- appending String to another String ⟹ O(N) ST
- "abc" + "def" ⟹ O(N+M) Linear Time
  (N, M)

→ when dealing immutable strings

- To append string most languages requires to split( ) the string into characters and then append. E.X = [S, T, R, i, N, G ..]

- There no Set at index method when dealing with immutable strings.

ASCii = All upper case characters fall in between range
65–90 while all lower case version of those same characters
A Z are 32 more than the upper version.
For example = 65+32 = 97 → that is lowercase "a"

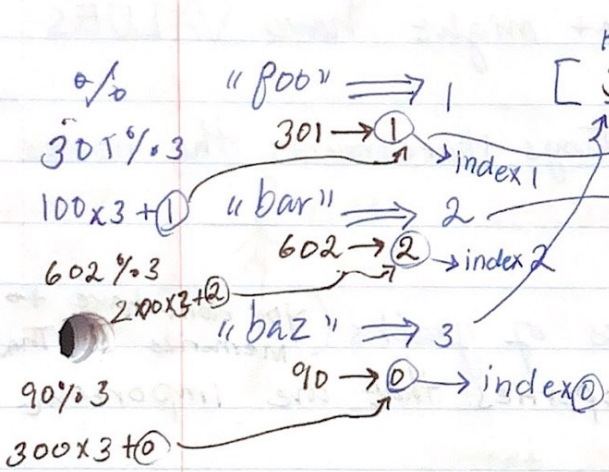
[65, 98, 100, 105, 102, 97, 116, 97, 104, ·, 77, 111, 104, 97, 109, 101, 100]
Abdifatah Mohamed.

32	16	8	4	2	1
1	0	0	0	0	1

Hash Tables

- Hash Tables = is data structures where you are able to store pairs of Keys and values. (where every Keys maps to a Values). "foo" ⇒ 1, "bar" ⇒ 2, "baz" ⇒ 3

- Insertion, Searching, deletation ⇒ O(1) Time O(N) Space.
- Hash Tables are built on top of arrays under the hood.



Hashing function ⇒ map every character to it's Ascii Character in coded value and sum up those interegs from Ascii String like ⇒ "foo"

- Transform String keys into indeces using hashing function (above) explanation.
- Use hash function to transform the Key "foo" for example into an index

• need to know very well how to traverse graph (DFS) and (BFS)

GRAPHS

GRAPHS are collections of notes that may or may not connect to each other.

- Every node of the graph is called VERTEX.
- The connections of the ~~graph~~ arrows of the graph is called EDGES.
- GRAPHS is made up of two special things Vertices (vertices) Edges
- The vertices are just nodes that might have VALUES for instance INTEGER VALUES.
- The edges are CONNESTIONS. (things that connects the nodes to one another).

- There are alot of different types of graphs (You don't have to Memorize Them)
- The key things or concepts or properties that are important when to look at any graph:

1= Connection and disconnections (whether or not the graph is connected)
2= whether or not the graph is directed (edges of graphs have direction)
3= Whether or not the graph has cycles on it (cyclic graph and acyclic graph)
example of cyclic graph → Wikipedia Links

- Storing in graph $O(V+E)$ Space (V = vertices, E = edges)
- Traversing graph → there 2 main method DEPTH FIRST SEARCH (DFS) BREATH FIRST SEARCH (BFS)
- (DFS) Traversal → means traversing the graph deeper first (go all edges in deep) and the go wide
- (BFS) Traversal → Means traversing wider befor we go to deeper.
- (DFS) and (BFS) traversal Time Complexity is $O(V+E)$ (V = Vertices, E = Edges)