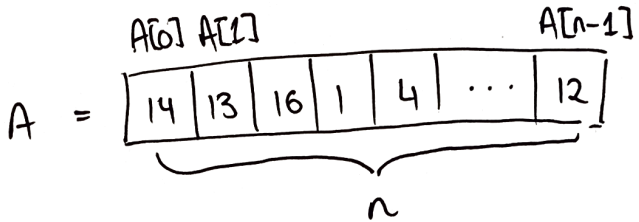
TUTORIAL WEEK 2

COMP20007 DESIGN OF ALGORITHMS

2020

ARRAYS



→ ACCESS INDEX i IN $O(1)$ TIME, E.G., print A[i] OR A[i] = 10

→ PROVIDED THERE IS ENOUGH MEMORY, $O(1)$ TO INSERT AT END

→ AN ARRAY IS <u>SORTED</u> (IN INCREASING ORDER) IF,

IF $0 \leq i < j < n$ THEN $A[i] \leq A[j]$

QUESTION 1. (i)

HOW LONG WILL THE FOLLOWING OPERATIONS TAKE USING AN UNSORTED ARRAY OF LENGTH n?

GIVE ANSWERS IN BIG-O NOTATION.

→ INSERTING A NEW ELEMENT → DELETING THE FINAL ELEMENT

→ SEARCHING FOR AN ELEMENT → DELETING A SPECIFIC ELEMENT

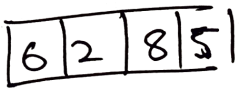
QUESTION 1. (i) SOLUTION

INSERTING A NEW ELEMENT USING AN UNSORTED ARRAY

ANSWER: O(1)

SEARCHING FOR AN ELEMENT IN AN UNSORTED ARRAY

ANSWER: WORST CASE WE HAVE TO LOOK AT EVERY ELEMENT - O(n)

Eg., search for 9 in 

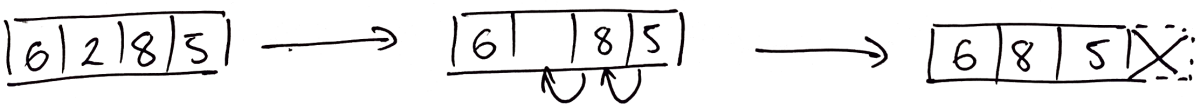
QUESTION 1. (i) SOLUTION

DELETING THE FINAL ELEMENT IN AN UNSORTED ARRAY

ANSWER: CAN JUST REDUCE LENGTH OF ARRAY — O(1)

DELETING A SPECIFIC ELEMENT IN AN UNSORTED ARRAY

ANSWER: MUST MOVE EVERYTHING OVER TO FILL THE ~~GAP~~ GAP - O(n)



QUESTION 1. (ii)

HOW LONG WILL IT TAKE TO PERFORM THE FOLLOWING OPERATIONS USING A SORTED ARRAY?

GIVE ANSWERS IN BIG-O NOTATION

→ INSERTING A NEW ELEMENT → DELETING THE FINAL ELEMENT

→ SEARCHING FOR AN ELEMENT → DELETING A SPECIFIC ELEMENT

BINARY SEARCH

BINARY SEARCH FOR x IN A SORTED ARRAY A:

- COMPARE $A[n/2]$ AND x
- IF $A[n/2] == x$ REPORT THAT x HAS BEEN FOUND
- NARROW SEARCH SPACE DEPENDING ON IF

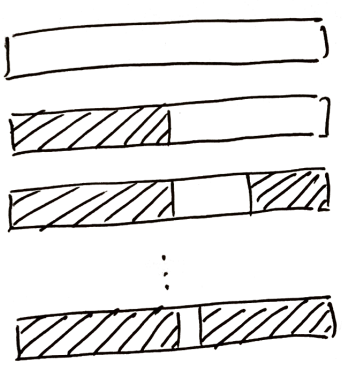
$$x < A[n/2] \quad \text{OR} \quad x > A[n/2]$$

Eg.) SEARCHING FOR 7:

SINCE $7 \neq 8$ WE HAVE EXHAUSTED THE POSITIONS 7 COULD BE IN WITHOUT FINDING IT, SO WE REPORT 7 NOT IN A

BINARY SEARCH TIME COMPLEXITY

WORST CASE IS WE NARROW DOWN SEARCH SPACE TO 1 ELEMENT



SIZE	COMPARISONS
$n = n/2^0$	1
$n/2 = n/2^1$	1
$n/4 = n/2^2$	1
⋮	⋮
$1 = n/2^k$	1

COMPARISONS = $k+1$

TO FIND k: $1 = n/2^k \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

SO, # COMPARISONS $= k+1 = \log_2 n + 1$

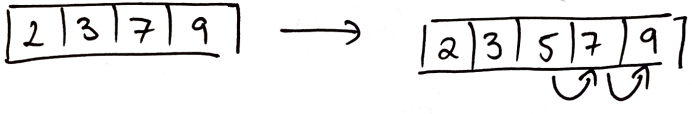
SO BINARY SEARCH IS $O(\log n)$.

QUESTION 1. (ii) SOLUTION

INSERTING AN ELEMENT IN A SORTED ARRAY

ANSWER: BINARY SEARCH FOR POSITION TO INSERT, MOVE EVERYTHING OVER THEN INSERT – $O(\log n) + O(n) = O(n)$

Eg.



SEARCHING FOR AN ELEMENT IN A SORTED ARRAY

ANSWER: BINARY SEARCH – $O(\log n)$

DELETING THE FINAL ELEMENT IN A SORTED ARRAY

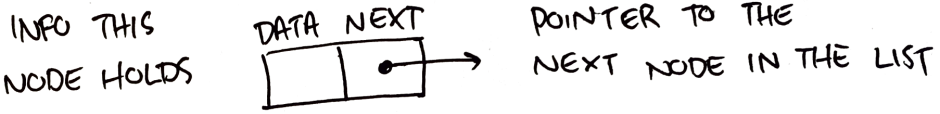
ANSWER: SAME AS UNSORTED CASE – $O(1)$

DELETING A SPECIFIC ELEMENT IN A SORTED ARRAY

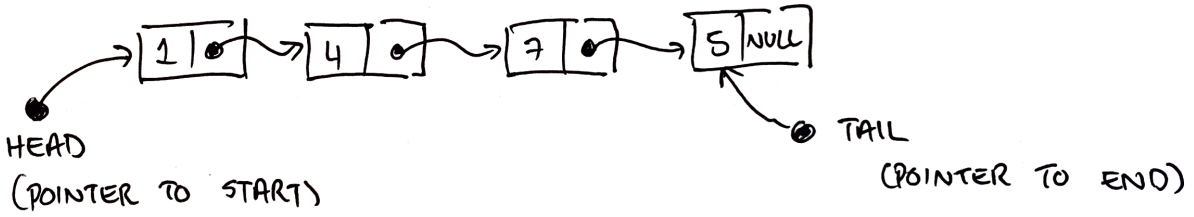
ANSWER: SAME AS UNSORTED CASE – $O(n)$

LINKED LISTS

→ LINKED LISTS ARE MADE UP OF NODES:



→ EXAMPLE OF A LINKED LIST:



→ CAN ONLY ACCESS NODES WE HAVE A POINTER TO,
EG, TO ACCESS 2ND NODE MUST DO HEAD.NEXT

LINKED LISTS EXAMPLE

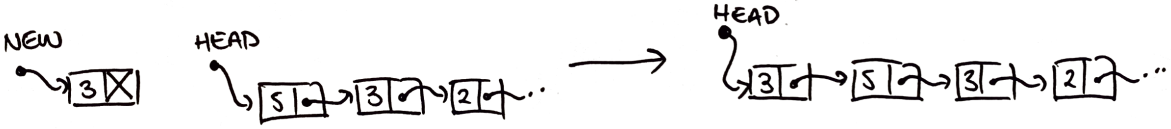
→ FINDING LENGTH OF A LINKED LIST

```
current ← head
length ← 0
while current ≠ NULL
    length ← length + 1
    current ← current.next
```

LOOKS AT EVERY NODE UNTIL WE GET TO THE END

⇒ O(n)

→ INSERTING AN ELEMENT AT THE START




```
new ← node()
new.data ← 3
new.next ← head
head ← new
```

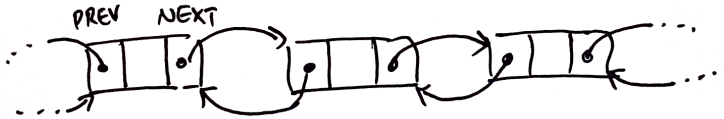
DOESN'T DEPEND ON # OF NODES IN LIST

⇒ O(1)

DOUBLY- VS. SINGLY-LINKED LISTS

→ SINGLY-LINKED: POINTERS ONLY GO FORWARD

→ DOUBLY-LINKED: POINTERS FORWARDS AND BACKWARDS



(+) CAN TRAVERSE LIST IN BOTH DIRECTIONS

(-) TAKES UP MORE MEMORY

QUESTION 2.

DESCRIBE HOW TO PERFORM THE FOLLOWING OPERATIONS, AND GIVE THEIR COST IN BIG-O NOTATION, FOR

(i) A SINGLY-LINKED LIST, AND A (ii) DOUBLY-LINKED LIST

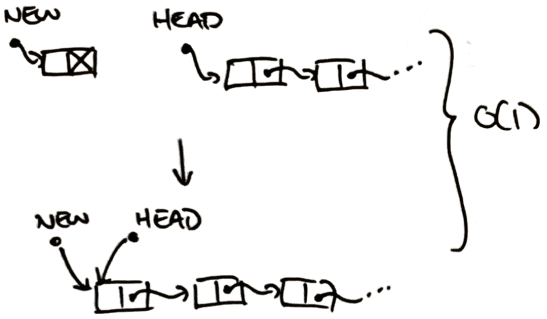
→ INSERT AT THE START → DELETE FROM THE START

→ INSERT AT THE END → DELETE FROM THE END

QUESTION 2. SOLUTIONS

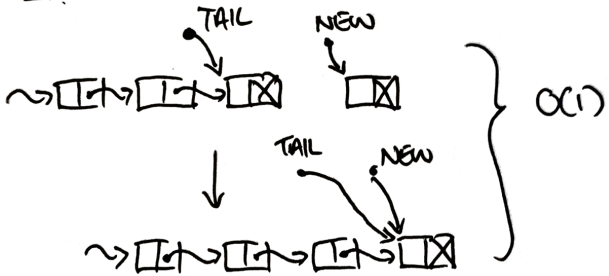
(i) SINGLY-LINKED LIST

→ INSERT AT START



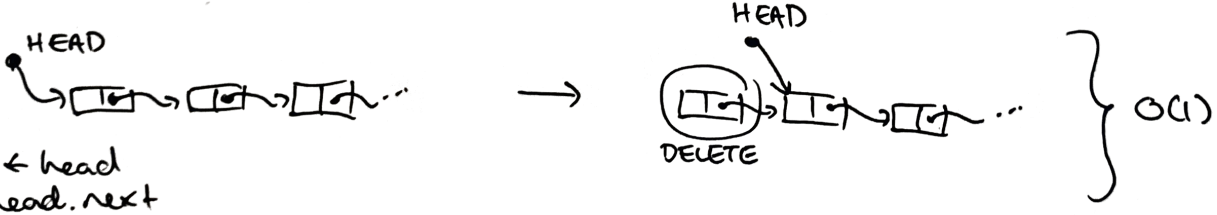
new.next ← head
head ← new

→ INSERT AT END



tail.next ← new
tail ← new

→ DELETE FROM START



old_head ← head
head ← head.next
delete old_head

QUESTION 2. SOLUTIONS

(i) SINGLY-LINKED: DELETE THE FINAL ELEMENT

↳ NEW TAIL NEEDS TO BE SECOND LAST ELEMENT, CAN'T JUST GO BACKWARDS FROM TAIL (WE DON'T HAVE THE POINTER)

```
node ← head
while node.next.next ≠ NULL
    node ← node.next
second_last ← node
```

} O(n)

↳ THEN JUST SET tail ← second_last AND DELETE OLD TAIL AS WELL AS SETTING tail.next ← NULL.

(ii) DOUBLY-LINKED LIST:

↳ ALL OPERATIONS ARE THE SAME EXCEPT FOR DELETING FROM THE END:

```
second_last ← tail.prev
old_tail ← tail
tail ← second_last
tail.next ← NULL
delete old_tail
```

} O(1) ⇒ ALL OPERATIONS ARE O(1) WITH A DOUBLY LINKED LIST.

STACKS

→ DATA STRUCTURES ARE DEFINED BY HOW THE DATA IS STORED

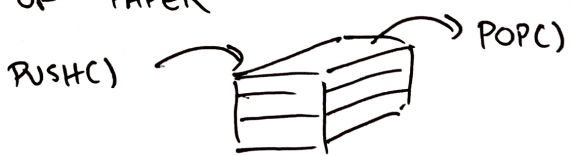
→ ABSTRACT DATA STRUCTURES ARE DEFINED BY THE OPERATIONS WE CAN PERFORM ON THEM

→ A STACK IS AN ABSTRACT DATA STRUCTURE WITH THE FOLLOWING OPERATIONS:

↪ PUSH() – ADD ELEMENT TO "TOP" OF STACK

↪ POP() – REMOVE FROM "TOP" OF STACK

Eg., STACK OF PAPER



QUESTION 3.

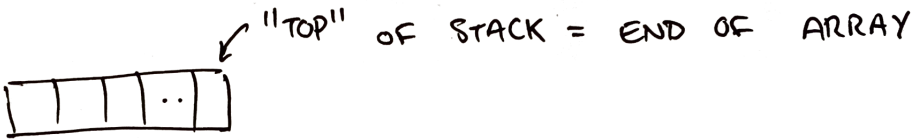
HOW WOULD YOU IMPLEMENT A STACK USING

(i) AN UNSORTED ARRAY

(ii) A SINGLY-LINKED LIST

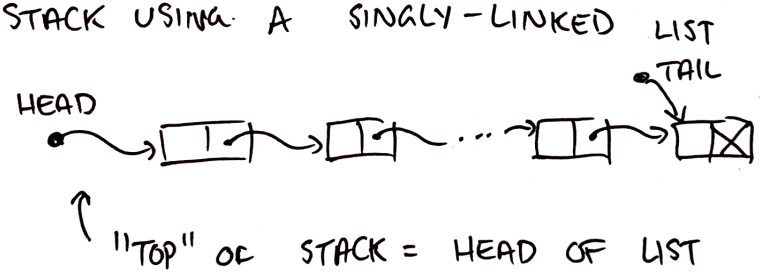
QUESTION 3. SOLUTION.

(i) STACK USING AN UNSORTED ARRAY



→ PUSH() IN O(1) TIME → POP() IN O(1) TIME

(ii) STACK USING A SINGLY-LINKED LIST



→ PUSH() IN O(1) TIME → POP() IN O(1) TIME

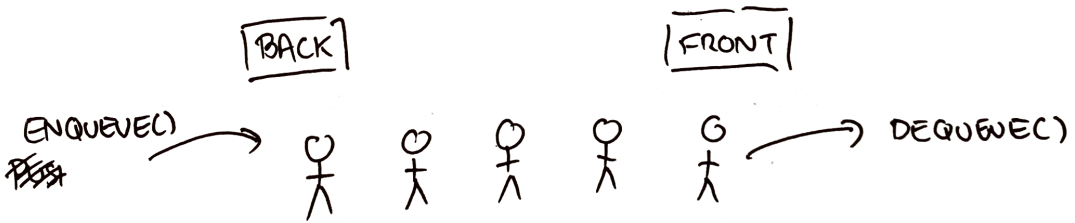
QUEUES

A QUEUE IS AN ABSTRACT DATA STRUCTURE WITH THE FOLLOWING OPERATIONS:

→ ENQUEUE() – ADD ELEMENT TO "BACK" OF QUEUE

→ DEQUEUE() – REMOVE ELEMENT FROM "FRONT" OF QUEUE

EG, QUEUE OF PEOPLE



QUESTION 4.

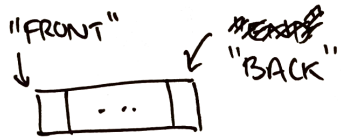
HOW WOULD YOU IMPLEMENT A QUEUE USING

(i) AN UNSORTED ARRAY

(ii) A SINGLY-LINKED LIST

QUESTION 4 SOLUTIONS

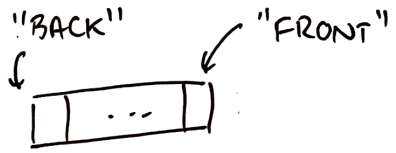
(i) QUEUE USING AN UNSORTED ARRAY



→ ENQUEUE() IN O(1)

→ DEQUEUE() TAKES O(n)

—OR—

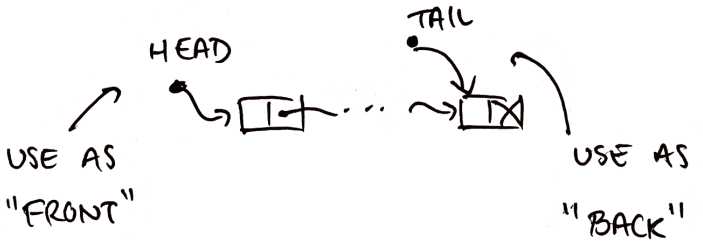


→ ENQUEUE() TAKES O(n)

→ DEQUEUE() TAKES O(1) ONLY

WE SHOULD DETERMINE WETHER WE ARE MORE LIKELY TO ENQUEUE() OR DEQUEUE() MORE AND DECIDE ACCORDINGLY

(ii) QUEUE USING A SINGLY-LINKED LIST:



→ ENQUEUE() AND DEQUEUE() BOTH RUN IN O(1) TIME