22/02/2025

240500 1032 _ Danny IDUKUNDA TWESE

Q1. What is data structure, classify Data structures as primitive DS and non-primitive DS by providing 4 example for each:

Answer

| Primitive | Non-primitive |
|---|---|
| → General type of data structure which are normally predefined | → They are derived from primitive data structure |
| → ex: * int * float * char * boolean | → ex: * linear ✓ array ✓ Queue ✓ stack ✓ linked list * non-linear ✓ tree ✓ Graphs |

Data structure - refer to way of organizing data for proper access so as to be accessed efficiently.

(b). Describe storage presentation of primitive data structure versus non-primitive.

Answer

| Primitive | non-primitive |
|---|---|
| → Their memory are predefined | → They are defined on go / execution (runtime) |
| ex: int (4 bytes) | ex: linked lists (10 elements 10 nodes ↳ 4x4x10 = 80 bytes |

c) Describe criteria for selecting good algorithm

**Answer:**

Good algorithm should be;

(i) Correctness (correctly ~~the~~ handle the problem).

(ii). ~~Efficiency~~

it should have minimized

- ✓ time complexity and
- ✓ space complexity.

(iii) Finiteness

↳ It should have the start and the end.

(iv) Clarity and simplicity

↳ It should be clear to which problem it solves and simple to use.

(v) Generality

↳ It should be applicable in broad scenarios with broad range of inputs.

(vi). Robustness

↳ It should be able to handle the unexpected errors

(vii). Maintainability

↳ it should be easy to modify and edit as well as reuse.

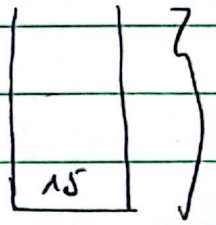
Henceforth, Good algorithm should be;

- ✓ correct
- ✓ efficient
- ✓ Finite
- ✓ Can clear and simple
- ✓ General
- ✓ Robust and
- ✓ have ability to be maintained.

240500 1032. Danny IDUKUNDAYWTJE

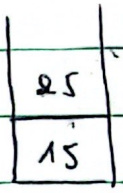
Q2. Given empty stack

(i). Push (15)



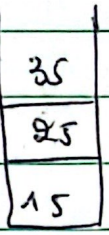

result stack after push (15)

(ii). Push (25)



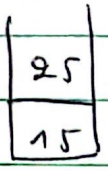

stack after push 25

(iii). Push (35)



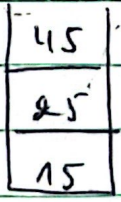

stack after push 35

(iv). Pop() (removes the element (top element)
result:



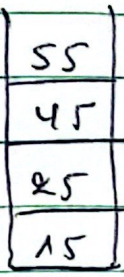

result after pop()

(v). Push (45)




after push (45)

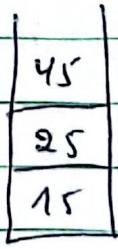
(vi). Push (55)



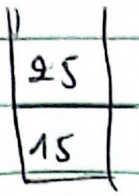

after pushes

(vii). Peek() → returns top element which is 55

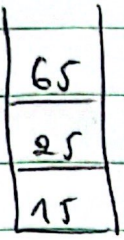
(viii). Pop() → removes top element
resulting stack



(ix). pop() → removes top element (45)

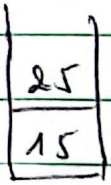



(x). Push 65 (results)




(xi). peek() - returns top element
⇒ 65

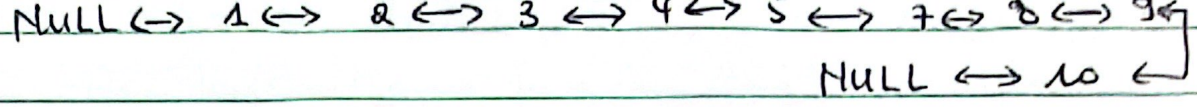
(xii). Pop() → removes top element
resulting stack:




Final element of resulting stack are 25 and 15.

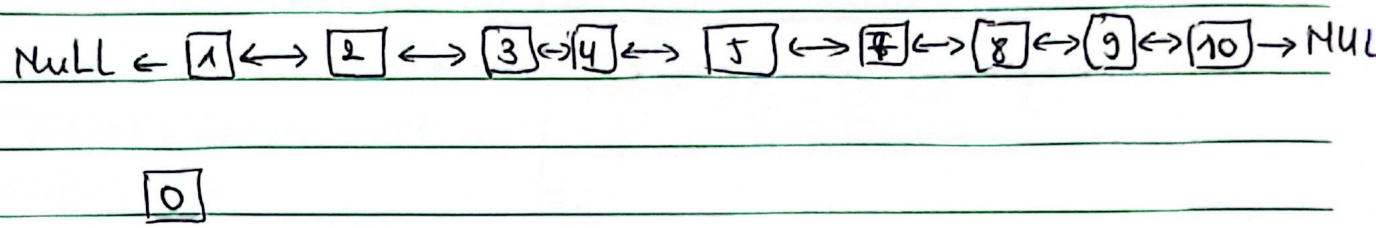
2405001032 - Danny IDULMD ATWEJE

Q3. For given list :



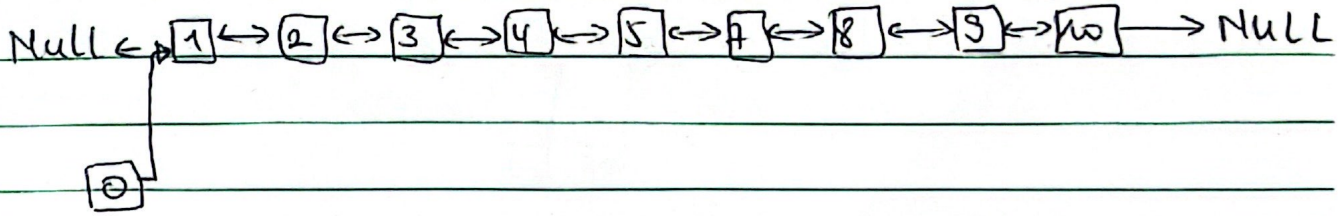

a) Insert value 0 before 1 :

i. Allocate new node



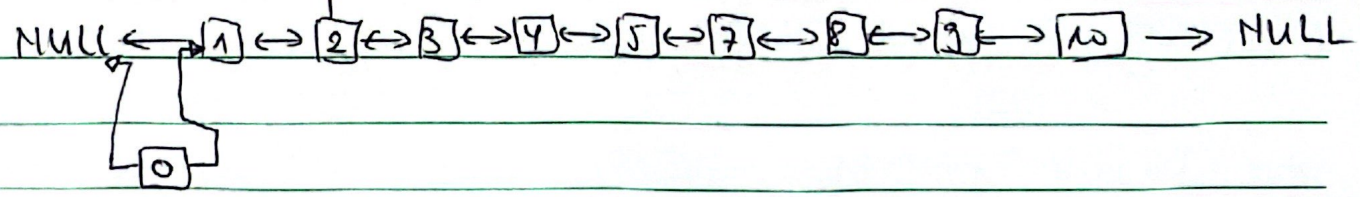

ii. newNode.next = RightNode

0.next = 1



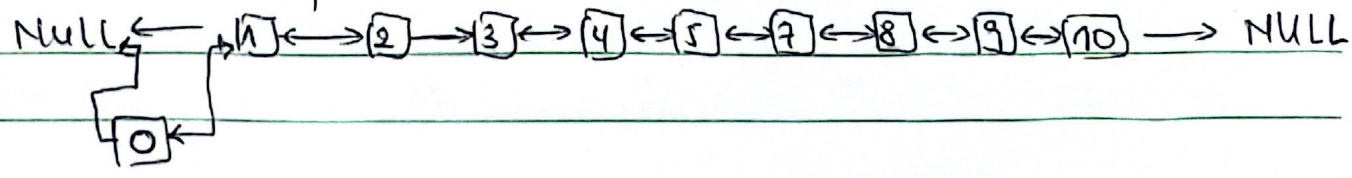

iii. newNode.previous = NULL

0.previous = NULL

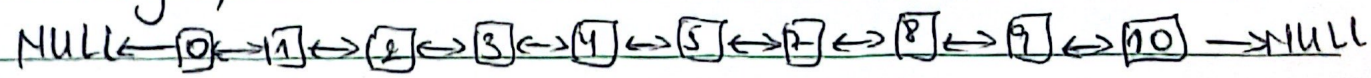



iv. RightNode.previous = newNode

1.previous = 0



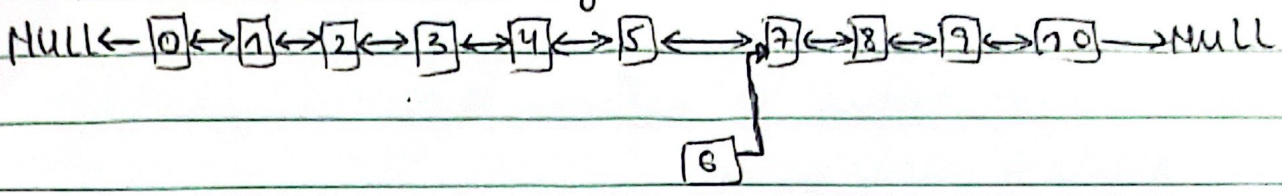

Finally ;



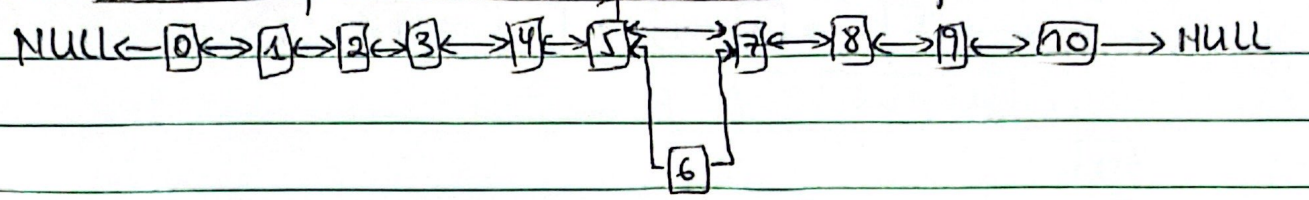
(b). Insert 6 after 5

(*) Allocate newNode: [6]

(*). NewNode.next = RightNode (6.next = 7)

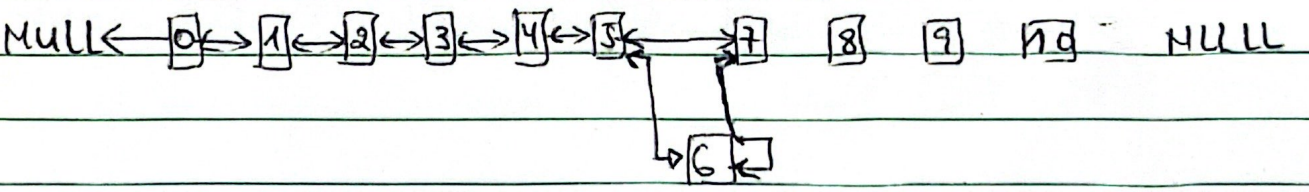



(*). newNode.previous = leftNode (6.previous = 5)

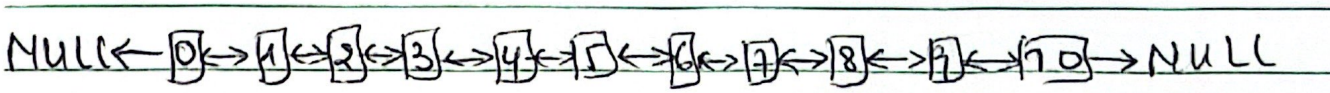



(*). leftnode.next = newNode (5.next = 6)

(*). rightnode.previous = newNode (7.previous = 6)




Finally;




(c). Insert 13 after 10

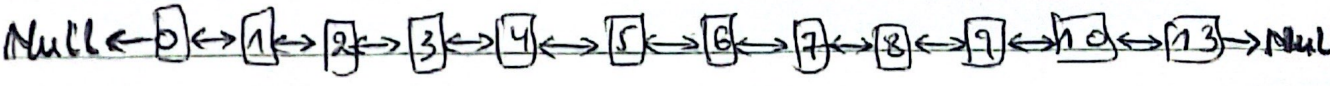
(*). allocate newNode [13]

(*). newNode.next = Null (13.next = NULL

(*). newNode.previous = leftNode (13.previous = 10)

(*). leftnode.next = newNode (10.next = 13)

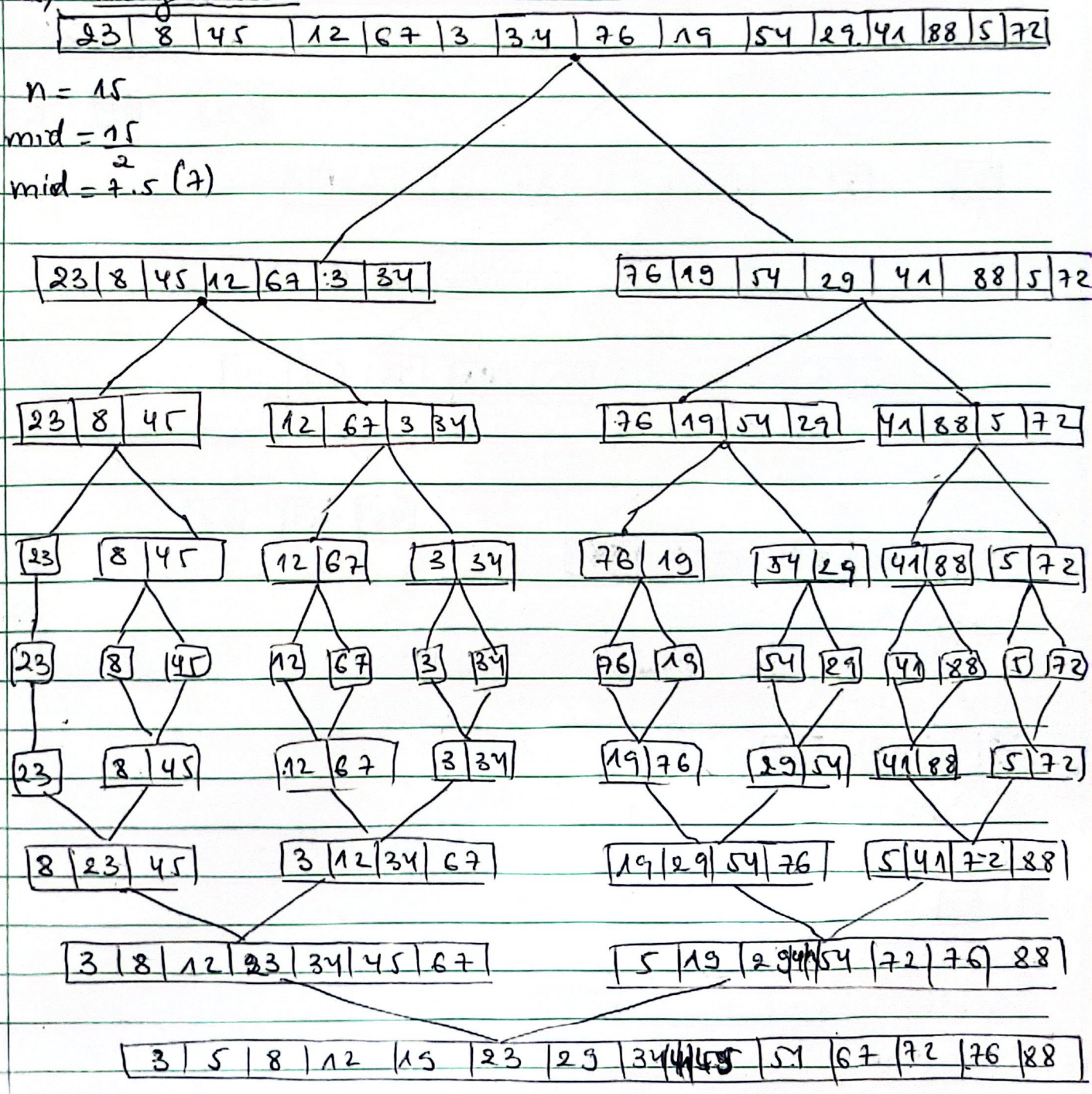
Finally:



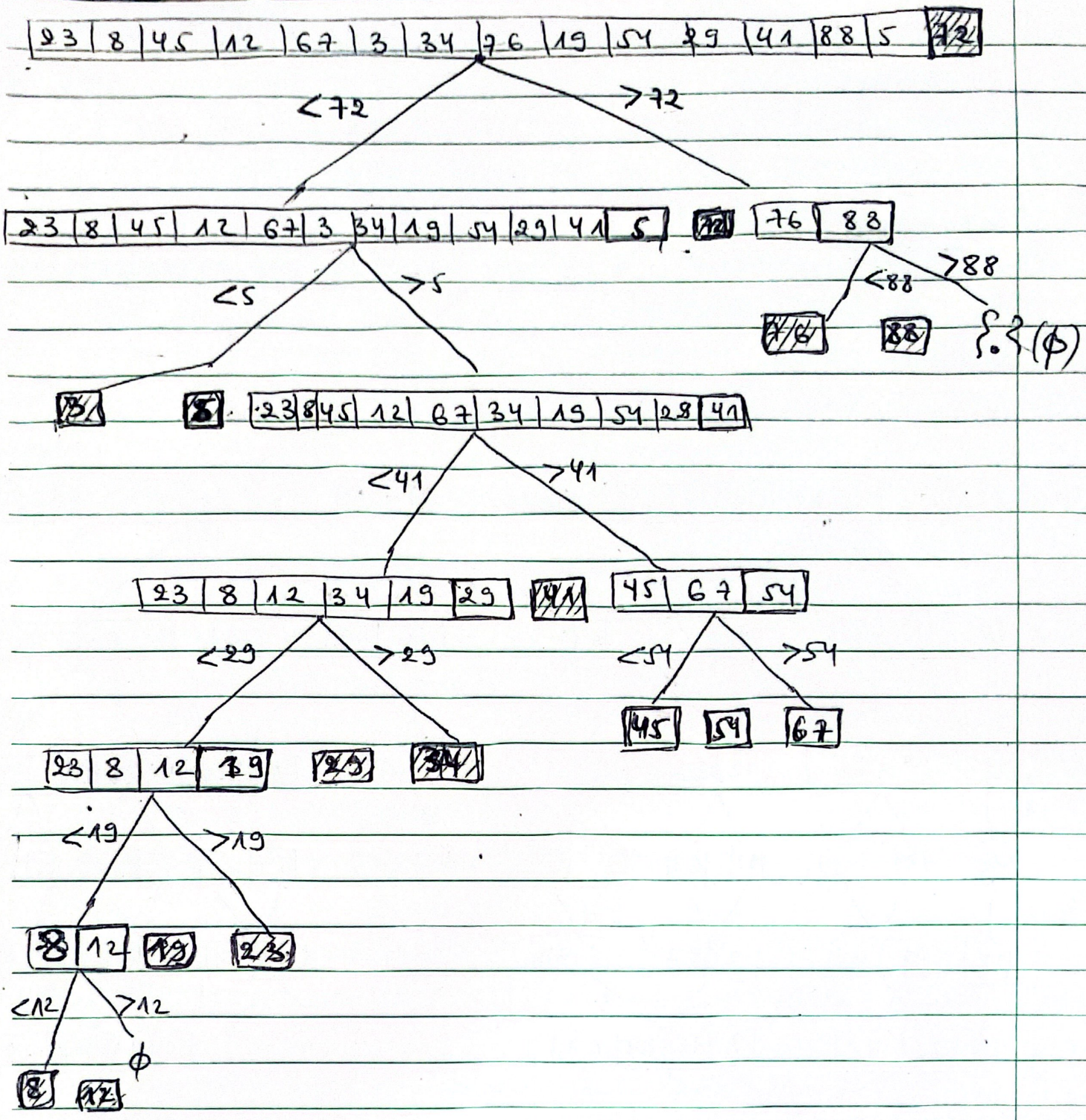
240500 1032 - Danny IBN KUMDAMWESE

Q4. array = {23, 8, 45, 12, 67, 3, 34, 76, 19, 54, 29, 41, 88, 5, 72}

a) merge sort




b. Quick ~~merg~~ sort



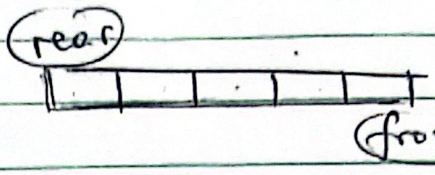

Finally, sorted list

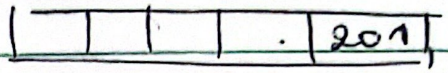
| 3 | 5 | 8 | 12 | 19 | 23 | 29 | 34 | 41 | 45 | 54 | 67 | 72 | 76 | 88 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

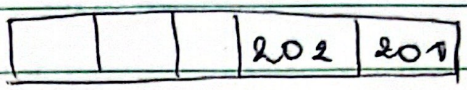
| Comparison : | merge sort | quick sort |
|---|---|---|
| Time complexity | $O(n \log n) \Rightarrow O(15 \log 15)$ | $O(n \log n) = O(15 \log 15)$ |
| space complexity | $O(n) \Rightarrow O(15)$ | $O(1)$ |

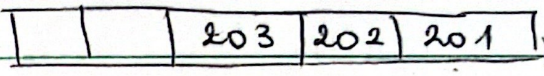
2405001032 - Danny IOUICWHDATWEJE

Q5.

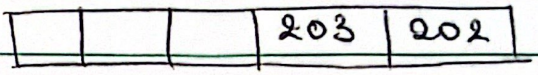
Empty queue 

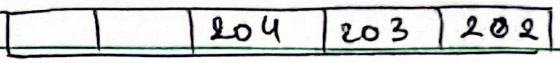

(i). enqueue 201 ⇒ 

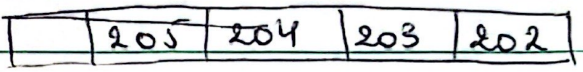

(ii). enqueue 202 ⇒ 


(iii). enqueue 203 ⇒ 


(v). dequeue () ⇒ removes element at front (201)

⇒ 

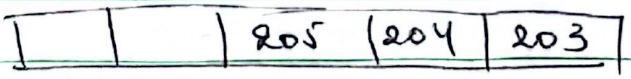

(vi). enqueue 204 ⇒ 

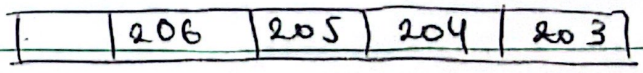

(vii). enqueue 205 ⇒ 

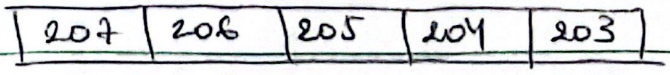

(viii) peek() ⇒ returns element at front

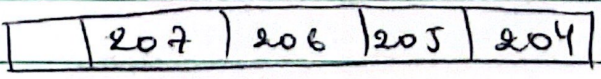
⇒ 202

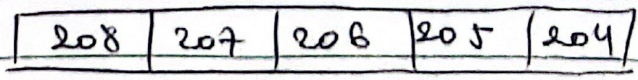
(ix). dequeue ⇒ remove element from front. (202)




(x). enqueue 206 ⇒ 


(xi). enqueue 207 ⇒ 


(xii). dequeue ⇒ 


(xiii) enqueue (208) ⇒ 

Note, once attempt made on adding 203 (enqueue 208) on full queue
It will return overflow error since queue will be full.

240500 1032 - Danny IDUKUNDA TWEJE

① Design:

Algorithm to check if a string is a palindrome (ignoring spaces, punctuation and case sensitivity).

Soln:

(i). Input s (string) // Enter a string to check)

(ii). Initialize len = 0, flag = 0

(iii). Iterate through the string: (traverse)

WHILE (s[len] != NULL):

len++

(iv). i=0, j = len - 1

(v). While (i < (len/2) + 1):

if (s[i] == s[j])

flag = 0

else:

flag = 1

i++, j--

(vi). If (flag == 0):

print key is palindrome

else:

print key is not palindrome

(vii). End.

② check "Madam In Eden, I'm Adam"

len = 23