— Intro. to Stack —

- **Data Structures :-**

  Data structure is an arrangement of data in a computer's memory or sometimes on a disk.

  Ex:- Stacks
  Queues
  Linked lists
  Trees

- **Algorithms :-**

  Algorithms manipulate the data in these structures in various ways.

  Ex:- Searching and sorting algorithms

- **Usage of Data Structures :-**

  * Real world data storage
  * Real world modeling
    - queue, can model customers waiting in line
    - graphs, can represent airline routes between cities
  * Programmers Tools
    - Stacks, queues are used to facilitate some other operations.

- **An algorithm should be :-**

  - correct
  - unambiguous
  - give the correct solution for all cases
  - simple
  - terminate

Scanned by TapScanner

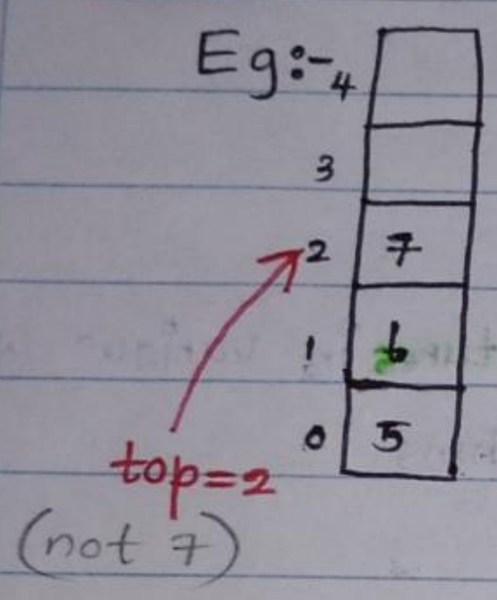
Array is a data structure
With the array data structure we can learn about stack data stru

Stacks :-

* Stack is a linear data structure.
* Stack is going to be implemented using an array.

* Main difference between stack and array is, stack have a variable called "top".
* In "top" variable, we store a value of the index of a top most value.

Eg:-



Stack 1 කි කියනේනෙකි array 1 කි වගේම data structure 1 කි. But මේකේ ප්‍රධාන වෙනස තමා stack 1 කට variable 1 කි කියනවා "top" කියලා. top variable 1 කේ අපි store කරගන්නේ top most value 1 කේ index 1 කේ අගය.

අපි stack 1 කි සහ array 1 කි වෙනස්කරලා අඳුරගන්නේ මේ top variable 1 නි. Data Structure 1 කේ top variable 1 කි mark කරලා තියෙනවනම් ඒක stack 1 කි වෙනවා.

* Allows access to only one data item; the last item inserted.

stack 1 කදී හැමතිස්සෙම අපිට access දෙන්නේ අන්තිම data item 1 කට විතරයි. (top most value 1 ට විතරයි)

* Stack 1 ක හැම insert, delete 1 කම වෙන්නේ මේ top variable point 1 නිසා.

* If you remove this item, then you can access the next-to-last item inserted.

top most value 1 [redacted] remove කරනකොට කියන -last item 1 top most value 1 වෙනවා. එතකොට ඒකේ index අගය තමා top variable 1 වෙන්නේ.

RATHNA

Scanned by TapScanner

** අපිට stack 1 කේ තියන කොන value 1 කි වුණාත් delete කරන්න ඕන නම් ඒක හැමතිස්සෙම පටන් ගන්න ඕන top most value 1 කේ ඉදලා. එකනිසා ඒ අතුල value 1 delete කරන්න බෑ. top most 1 කේ ඉදලා delete කරනි එන්න ඕන.

Insert 1 කි එහෙමයි. top most 1 කේ ඉදලා තමා උඩට insert කරන්න පුළුවන්. එතකොට top variable 1 change වෙනවා.

*** Everything that we have to do inside a stack has to be done from the "top". We can't access other values as our wish.

** In a stack all insertions and deletions are made at one end (Top). Insertions and deletions are restricted from the Middle and at the end of a stack.

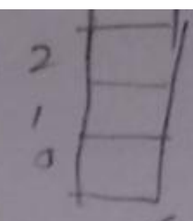
** Adding an item is called Push. (inserting)

** Removing an item is called Pop. (deleting)

** Elements are romoved from a Stack in the reverse order of that in which the elements were inserted into the Stack.
(insertion order 1 කේ reverse order 10 තමා අපි stack 1 කදි values delete කරනිකේ.)

** The elements are inserted and removed according to the Last-In-First-Out (LIFO) principle.
(stack 1 කදි LIFO principle 1 අනුව තමා elements insert & remove වෙනිකේ. Last item that we have inserted to the stack it will be the first item that we take out)

Scanned by TapScanner



← Top = -1 (we assume

- Application of Stacks :-

* The stack operations are built into the microprocessor.

Stack - Implementation

* Stack implementation using an array. (Stack වල implementation 1 කරනිනේ arrays use කරලා.)

```
//create stack class
class Stack X {
        private int maxSize; //size of Stackarray
        private double[] stackArray; //create array
        private int top; //top of the stack

//constructor
public StackX (int s) {
        maxSize = s; //set array size
        stackArray = new double [maxSize]; //set array
        top = -1; //no items
    }
    - - - .
    - - - .
}
```

මෙතන parameter 1 විදියට object create කරද්දි array size 1 pass කරනවා.

stack 1 ක values මුකුත් නැතිකම (initially) top 1 ගේ අගය -1 යි.

* pushomethod 1 දී මුලින්ම වෙන්නේ top value 1, 1 කින් increment වෙලා 'b'ට දැක්කේ නි new top value 1 තියන index 1ට pass කරන(insert කරන) value 1 assign වෙනවා.

```
top ++;
stackArray [top] = j;
```

* pop()method 1 දී මුලින්ම top value 1 return කරලා 'b'ට දැක්කේ top value 1, 1 කින් decrement වෙන්න ඕන.

```
return stackArray [top];
top--;
```

මේක syntactically correct නෑ. මොකද return statement 1 නි method 1 ක කව statements තියෙන්න ඕන. මේ දෙක ඒක line 1 නිම

Scanned by TapScanner

**** StackArray[++top] = j;

push - Implementation [push() method]

කියනකොට මුලින් top value 1 increment වෙලා තියෙන්නේ ඒ new index 1ට insert කරපු value 1 assign වෙනවා.

```
class StackX
{
    private int maxSize.

public void push (double j){
```

main 1 දී push() method 1 call කරනකොට pass කරන insert කරන්න ඕන value 1 තමා මෙතන parameter 1 විදිහට ගන්නේ.

top value 1 උපරිම (අන්තිම) index value 1ට සමාන වෙනවා කියන්නේ stack 1 full කියන එක.

```
    if (top==maxSize-1)
        System.out.println("Stack is full");
    else
        stackArray[++top] = j;
```

// first we have to increment top value by one. Then assign insert value to that new index (top value's index).

```
}
```

pop - implementation

*pop method 1ට parameters මුකුත් pass කරන්නේ නෑ. මොකද always pop() method 1 apply වෙන්නේ top variable 1ට ∴ වෙනම කිසිදු parameters pass කරන්න ඕන වෙන්නේ නෑ.

```
public double pop() {
```

return value type 1 double නිසා (array 1 double) pop() 1ගේ return type 1 වෙනුවෙන් double මයි.

```
    if (top==-1)
        return -99;
    else
        return stackArray[top--];
}
```

Stack 1 empty ද කියලා check කරන්න ඕන. මොකද stack empty නම් pop() කරන්න value 1 කි නැහැ.

මුලින්ම top value 1 return වෙලා 'ම තියෙන්නේ top 1, 1 කින් decrement වෙනවා

peek - implementation

මේකෙන් return type 1 double මොකද top value 1 return කරන්නේ

```
public double peek(){
    if (top==-1)
        return -99;
    else
        return stackArray[top];
}
```

*මේකෙදිත් වෙන්නේ pop() කරනකොට වගේම දෙයක්. වෙනස තමා value 1 return කරලා top 1 decrement වෙලා return කරන value 1 remove වෙන්නේ නෑ.

RATHNA

Scanned by TapScanner

Question – Using the implemented StackX class. Write a program to create a stack with maximum size 10 and insert the following items to the stack.
30 80 100 25

06

Creating a stack

Delete all the items from the stack and display the deleted items.

```
class StackApp {
    public static void main (String[] args) {
        StackX theStack = new StackX(10); // create a stack with max size 10
        theStack.push(30);
        theStack.push(80);
        theStack.push(100);
        theStack.push(25);
        while (!theStack.isEmpty()) {
            double val = theStack.pop();
            System.out.println(val);
            System.out.print(" ");
        }
    }
}
```

size 1 constructor 1 ට pass කරනවා

// push values to stack array

stack 1 empty වෙනකම් all item pop() (remove) කරනවා

// delete all items and display deleted items.

---

* අපි stack 1 full ද empty ද බලන්න if conditions දාන්න වගේම isFull(), isEmpty() methods හරහාත් මගෙම කරන්න පුළුවනි. ඒකට stack class 1 ඇතුලෙම ඒ methods implement කරන්න ඕන.

isFull() method

මේක boolean method 1 කි. එතකොට මේකෙන් return වෙන්නේ true or false කියන values 2 කෙනෙක් විතරයි.

```
public boolean isFull()
{
    if (top == maxSize-1)
        return true;
    else
        return false;
}
```

```
public boolean isFull()
{
    return (top == maxSize-1);
}
```

* if-else නොදා අවශ්‍ය නම් මේ විදියට one line method 1 ලියන්න පුළුවනි.

Scanned by TapScanner

Date

No

isEmpty() method

```
public boolean isEmpty()
{
    return (top==-1);
}
```

* මේකෙන් boolean method 2 කි. මේකෙදිත් මේ විදිහට one line or if-else දාලා or ලියන්න පුළුවන්.

Scanned by TapScanner

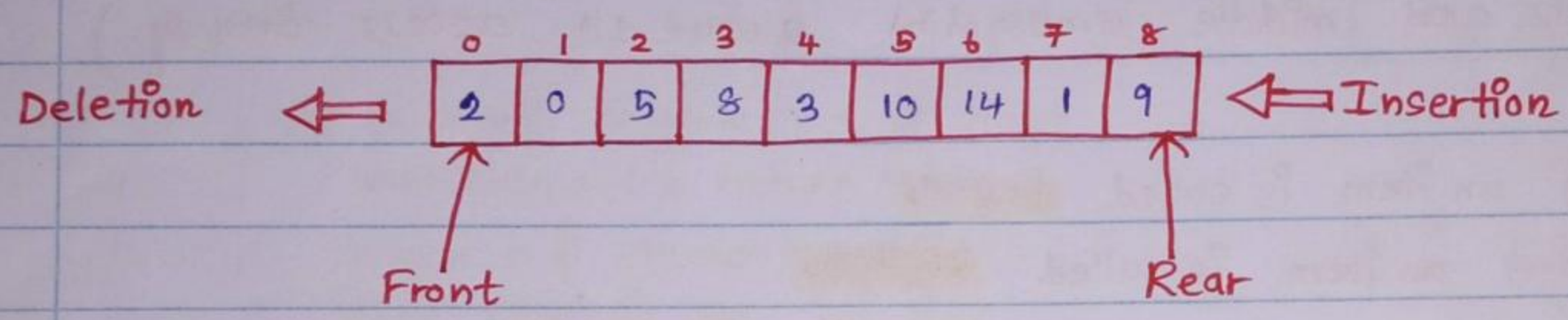
Date: 01

2

- Queues -

* Queue data structure එකේ වෙන්නෙත් real life queue එක වගෙමයි.
* The first item inserted is the first item to be removed.

* මේක වැඩ කරන්නේ FIFO method එක අනුව.
* Same insertion order will be same removing order.



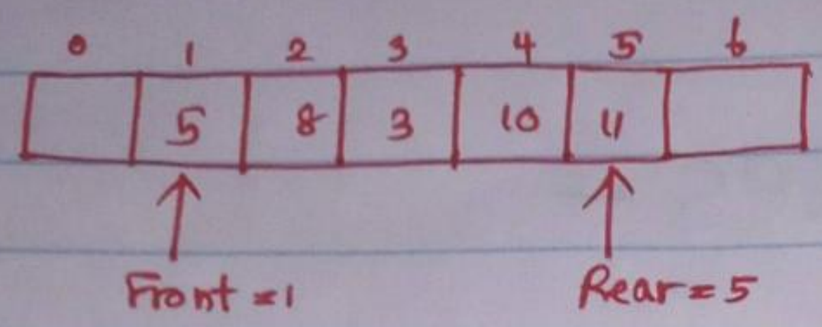
* Stack එක වගේ මේකෙත් implementation කරන්නේ array එක use කරලා.
* But මේකේ integer variables දෙකක් තියෙනවා front and rear කියලා. Stack වල තියෙන්නේ 1 integer variable එක විතරයි.
* Front එකේ තමා values remove/deletion වෙන්නේ.
* Rear එකේ තමා values insert කරන්නේ.

* කලින්ම මේ array එකේ values delete/insert වීම අනුව front එකයි rear එකයි වෙනස් වෙනවා.
  - Front - index value of the first element which is inserted in the queue.
  - Rear - index value of the last element which is inserted in the queue.

** Array එකේ ගන්නම හැමතිස්සෙම 0th index එක front වෙන්න ඕන නෑ. 1st element එක තියෙන ඕනම index value එකේ front එක වෙන්න පුළුවන්. Rear එකේ කතෙමයි. array එකේ අන්තිම index එක rear එක නෙවෙයි හැමතිස්සෙම. last element එක තියෙන index එක තමා rear වෙන්නේ. rear is not end of the array.

Scanned by TapScanner

Eg:-



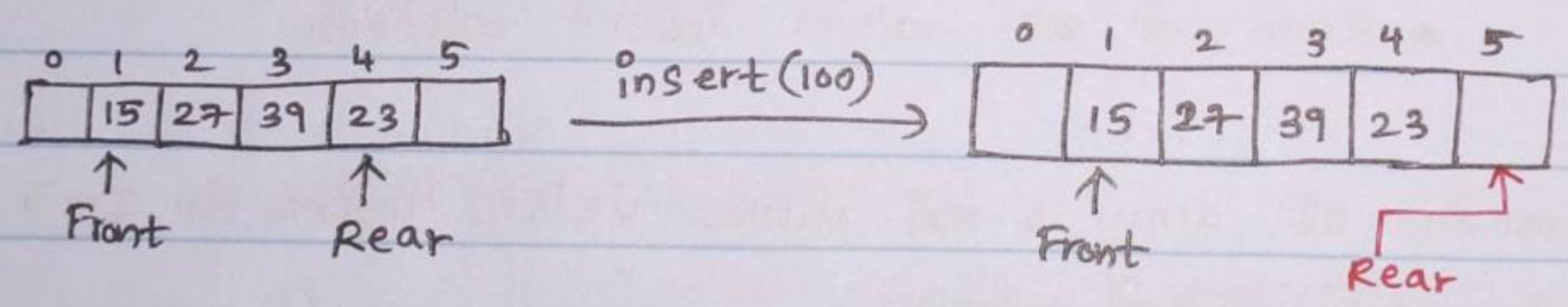
- In queue all insertions are made at the Rear end and deletions are made at the Front end.

- Insertions and deletions are restricted from the Middle of the queue.
  (දැනට queue 1 කට access කරන්න පුළුවන් front 1 or rear 1 න් විතරයි. queue 1ේ middle කොටසින් queue 1ට access කරන්න බැ.)

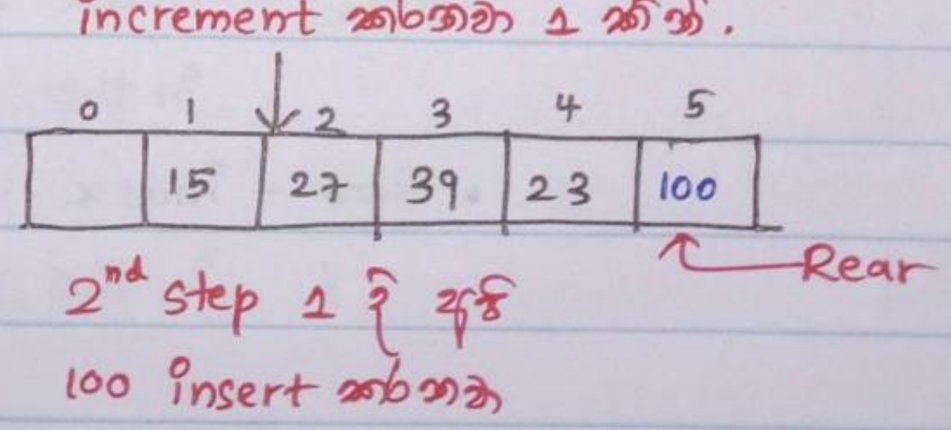
- Adding an item is called insert.
- Removing an item is called remove.

- The elements are inserted and removed according to the First-in-First-Out (FIFO) priciple.

Queue - Insert

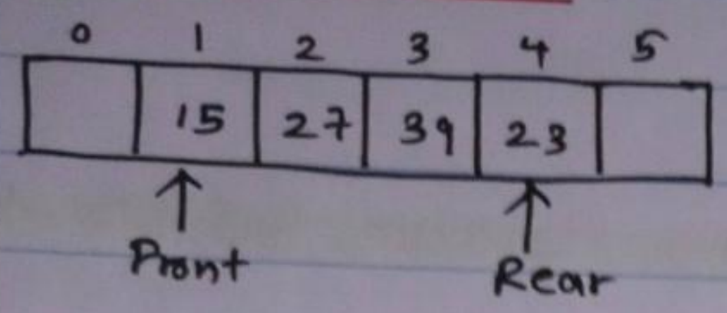


1st step 1 වෙනුවෙන් rear value 1 increment කරනවා 1 කින්.

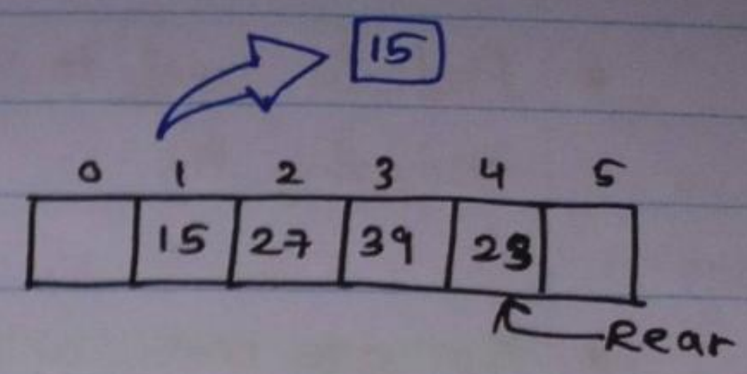


Scanned by TapScanner

Queue – Remove



remove()

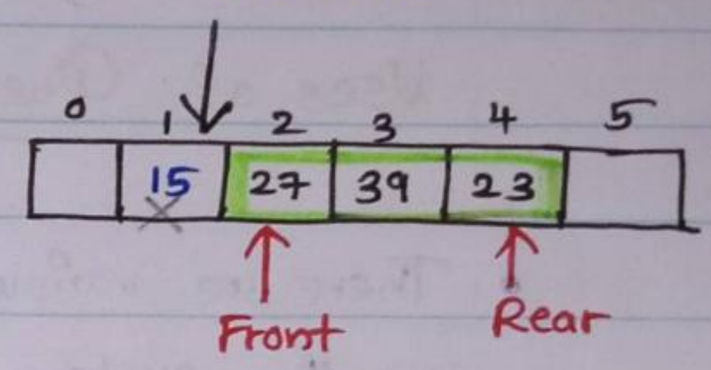


* remove method 1 ට parameters pass කරන්න ඕනෙ නෑ.
කොහෙ තියෙනවද remove always වෙන්නේ front එකෙන්.
remove() method will definitely remove the 1st element.

① st – check whether where the front is

② nd – Then return the value in front

return 15

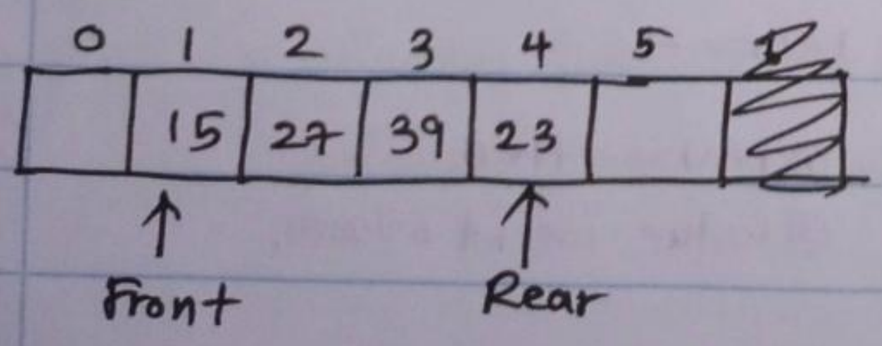


③ rd – Then increment the front value by one and it will be new front.

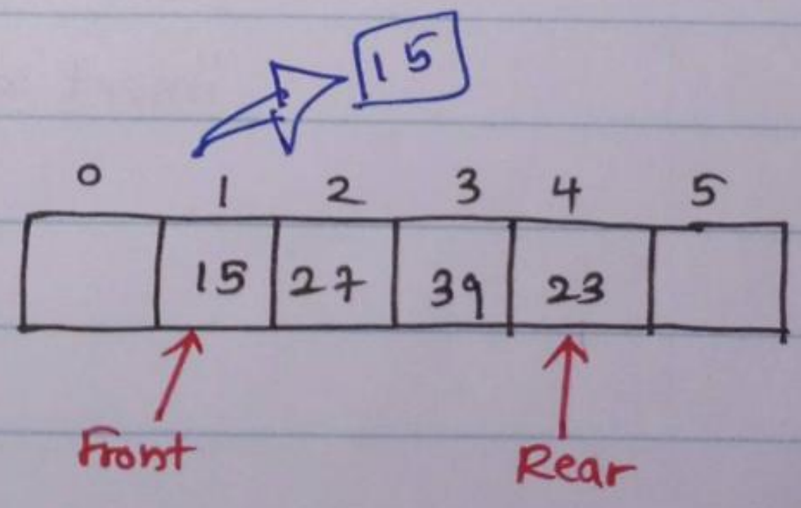
** මෙකෙනදි remove() 1 කි front element 1 return කරනව return 1 දි element 1 delete වීමක් වෙනවනේ නෑ. ඉදි කිව (skip) කොටලා තමයි කිරන්නේ කියලා. එකෙන් අපි new scope 1 කියන්නේ new front → rear element set 1.

Queue – peekFront

* stack 1 කෙ තිබුණේ peek() method. But මෙකේ තියන්නේ peekFront() කොහොද මේකෙදි අපිට Integer variable 2 කි තියනවා (front, rear). peek 1 කි කරන්නේ value read කරනවා වගේ වැඩක්. එතකොට හැමතැනම අපිට අවශ්‍ය වෙන්නේ frontvalue තියවන්න තියා මේකෙදි method 1 peekFront() කියලා අරගෙන තියනවා.



peekFront()



return 15

මෙකකදී front 1 ට pointing value 1 return වෙනවා. But Queue remains the same.

Scanned by TapScanner

** Front & rear සමග index 0 point වෙනවනේ තුල්වනේ

Date: 04

* Peek is used to read the value from the front of the Queue without removing it.

* You can peek only the Front item, all other items are invisible to the queue user.

Question 01

Uses of Queue

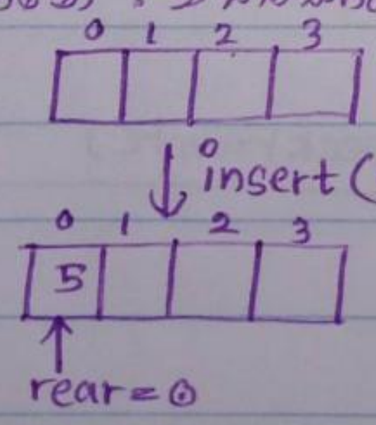
- There are various queues quietly doing their job in a computer's operating system.
  - * printer queue
  - * stores keystroke data as you type at the keyboard.
  - * pipeline.

Implementation

* Queue implementation කරනකොට array 1 කි use කරලා

* Queue 1 කේ implementation වලදී front , rear වල initial values විදිහට ගන්නේ front = 0 සහ rear = -1 කියලා

(1) rear = -1 ගන්නේ insert වෙනකොට rear 1 හරහා නිසා, එතකොට මුලින්ම වෙන්නේ rear value 1 , 1 කින් increment වෙලා new rear තියන index 1 ට තමා insert කරන value 1 insert කරන්නේ . එතකොට මුලිම empty array 1 කදී,



① rear = -1+1 = 0
② value insert වෙනවා.

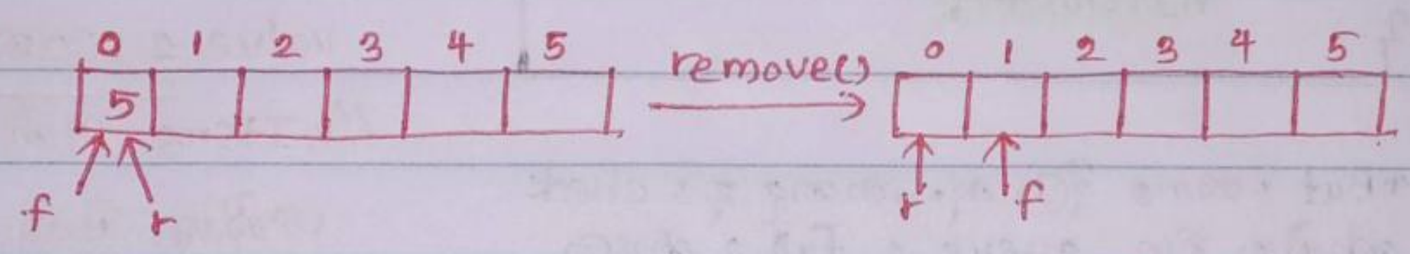
∴ initially rear value 1 = -1 කියලා තමා implement කරන්නේ.

Scanned by TapScanner

(2) front=0 ගන්නේ front එකෙන් කරන්නේ remove() නිසා. remove එකේ 1st step එකේ විදිහට අපි check කරනවා front එක කොහෙද කියලා. ඊට පස්සේ තමා 2nd step එකේ විදිහට front value එක return කරන්නේ. එතකොට initially front=0 නිසා තමා, පස්සේ values insert කරලා remove කරනකොට 1st element එකේ ඉඳලා values remove වෙන්නේ. front = -1 කියලා තිබිලා නම් array එකේ -1 index එකක් නැති නිසා එතකදී error එකක් එනවා. ∴ තමයි initially front=0 කියලා ගන්නේ.

* Queue implementation වලදී variable 1 ක් ගන්නවා nItems කියලා. ඒකෙන් තමා queue array 1 empty ද කියලා check කරන්නේ. නොකද front, rear 2 use කරලා queue 1 empty ද කියලා check කරන්න අමාරුයි. නොකද front=0, rear=-1 වුණොත් විතරක් නෙවෙයි queue 1 empty වෙන්නේ. ඒ වගේ අවස්ථා හොඳක් තියනවා.

Eg:-



වෙනකදී front=1, rear=0

∴ rear, front හොයලා queue 1 empty ද කියලා බලන්න බෑ. ∴ nItems variable 1 use කරනවා.

```
//create Queue class
class QueueX
{
    private int []queArray;
    private int maxSize;
    private int front;
    private int rear;
    private int nItems;
```

Scanned by TapScanner

```
// constructor
public QueueX(int s)
{
    maxSize = s;
    queArray = new int[maxSize];
    front = 0;
    rear = -1;
    nItems = 0;
}
```

// Initial values assigning

insert() method

```
public void insert(int j)
{
    queArray[++rear] = j;
    nItems++;
}
```

//insert 1 දී මුලින්ම rear 1 increment කරනවා. '0' වන්නේ new rear index 1ට insert කරන value 1 දානවා.

//nItems ශූන්‍ය 1 කින් increment වෙනවා

*But මෙතෙම දීලා තිබ්බොත් අපි check කරන්න ඕන queue 1 full ද කියලා values insert කරන්න කලින්.

```
public void insert(int j)
{
    if (rear == maxSize-1)
        System.out.println("Queue is full");
    else
    {
        queArray[++rear] = j;
        nItems++;
    }
}
```

Scanned by TapScanner

remove() method

```
public int remove()
{
    if (nItems==0)
    {
        System.out.println ("Queue is empty");
        return -99;
    }
    else
    {
        nItems--;
        return queArray[front++];
    }
}
```

// return type 1 int නිසා ඇතිව int value 1ක් return වෙනවා විතර.

// return statement 1 තමා ඇතිතිමමම තියෙනවා විතර.

∴ nItems decrement 1 මුලින් කරනවා විතර.

මෙතනදී මුලින්ම value 1 return වෙලා 'b'ට පස්සේ තමා front 1 increment වෙන්නේ.

peekFront() method

```
public int peekFront()
{
    if(nItems==0)
    {
        System.out.println ("Queue is empty");
        return -99;
    }
    else
        return queArray[front];
}
```

// මේකෙදී වෙන්නෙක් pop() 1ක් වෙනවේමයි. But මෙතනදී nItems අඩු වීමක් හෝ front 1 increment වීමක් වෙන්නෑ. මොකද peekFront() 1ක් කරන්නේ front value 1 read කරන එක විතරයි.

* Stack 1 දී වගේ if conditions වෙනුවට isFull() , isEmpty() method 2ක් implement කරලා use කරන්න Queue 1ක් ඉල්ලමු.

Scanned by TapScanner

isFull() method

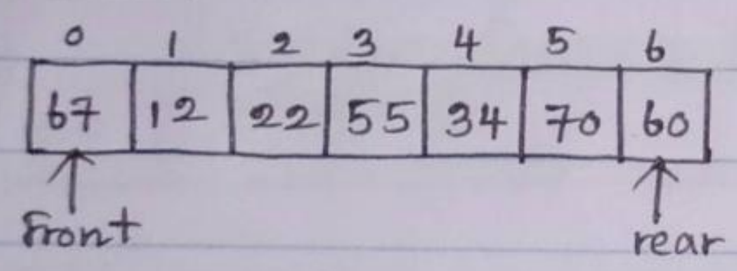
```
public boolean isFull()
{
    return (rear== maxSize-1);
}
```

//මෙතන if-else කන්න ඕනෙ නැතුව පුළුවන් stack වල වගේ.

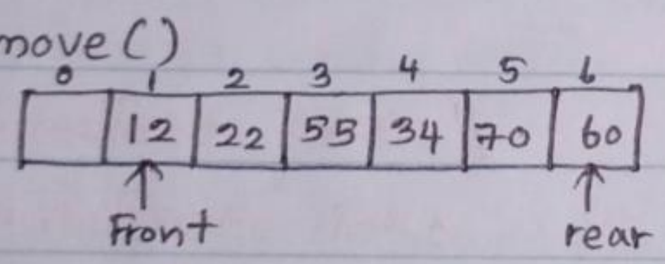
isEmpty() method

```
public boolean isEmpty()
{
    return (nItems==0);
}
```

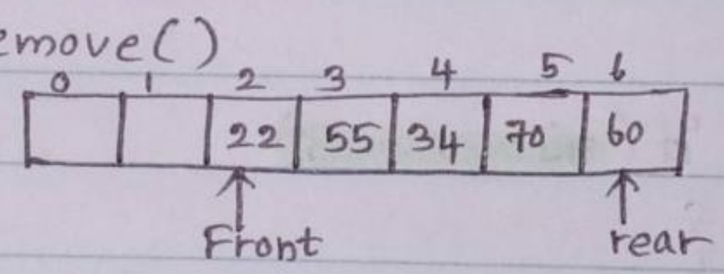
Question 3



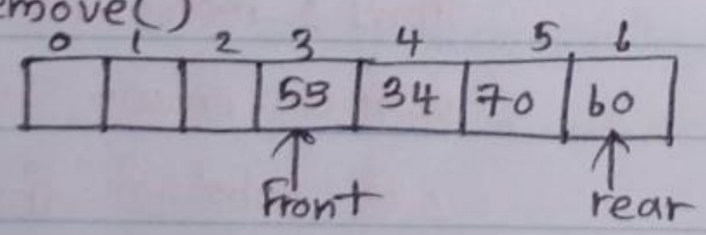
(i) remove()



(ii) remove()



(iii) remove()



(iv) Insert(88)

මෙතනදී array 2 full වුණේ නැතත් insert() කරන්න බෑ. මොකද insertion 2 වෙන්නේ rear 2 කි. rear 2 තියන්නේ last index 2 කේ තියා වෙ linear queue 2 කේ ඉඩ තිබුණත් new values insert කරන්න බෑ.

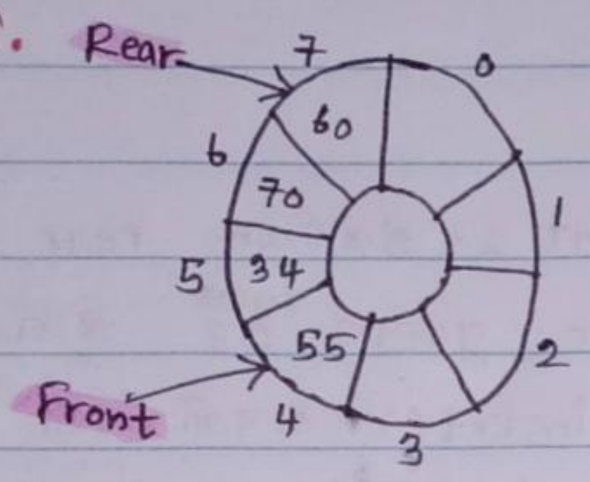
∴ මේ disadvantage 2 මගහරින්න Circular Queues use කරන්න පුළුවන්.

Scanned by TapScanner

Date: 09

Circular Queue

* Circular queue also called as ring buffers.
* Circular queue වලදී rear 1 last index 1 කේ තිබෙන අවුලක් නැ. මොකද empty indexes තියෙනවනම් last index 1 කේ ඉඳලා 0th index 1ට rear 1 අරන් යනවා queue 1 full වෙනකම් values insert කරනවා පුළුවන්.



* Circular queue 1 execute වෙන්නේ linear queue වගේ නෙවෙයි. ඒක නිසා circular queue 1 කේ implementation එකක් linear queue 1ට වඩා එකක් වෙනස්. අපි linear queue 1 කේ implementation ම එකක් modify කරලා circular queue 1 කේ implementation 1 හදාගන්න පුළුවන්.

Implementation

insert() method

```
public void insert(int j)
{
    if ( nItems=maxSize )
        System.out.println("Queue is full");
    else{
        if (rear==maxSize-1)
            rear = -1;
        queArray[++rear] = j;
        nItems++;
    }
}
```

මෙතෙදී circular queue 1 full නියන්නේ rear 1 last index 1ට එහාට නෙවෙයි. All indexes වලට values තියනවා. ∴ මෙතනදී condition 1 වෙනස් වෙනවා.

Circular queue 1දී array 1 full නැතිනම් rear 1 last index 1 කේ තියනකොට ඉන්නවනම් 0th index 1ට එනවා normal linear queue 1 දී වගේ නෙ. linear 1 දී වගේ නම් අවස්ථාවටනම් ++rear කියලා දැම්මොත් වෙන්නේ rear 1 කේ අගය last index 1 ගමන් වඩා වැඩිවෙන එක. එනෙම වෙන්න බැරි නිසා last index 1 දී rear 1 කියන අවස්ථාවට code 1 කේ එකින එක ආපහු rear 1 0th index 1ට යනවා කියලා.

Scanned by TapScanner

* මේ කියන්නේ rear අගය 1 වෙලා initial අවස්ථාවේ ඉඳලා increment වෙනවා ඕන.
* එක්කම තමා circular implementation 1 කේ new if 1 කේ add කරලා rear value 1 initial අවස්ථාවට අරන් යන්නේ.

remove() method

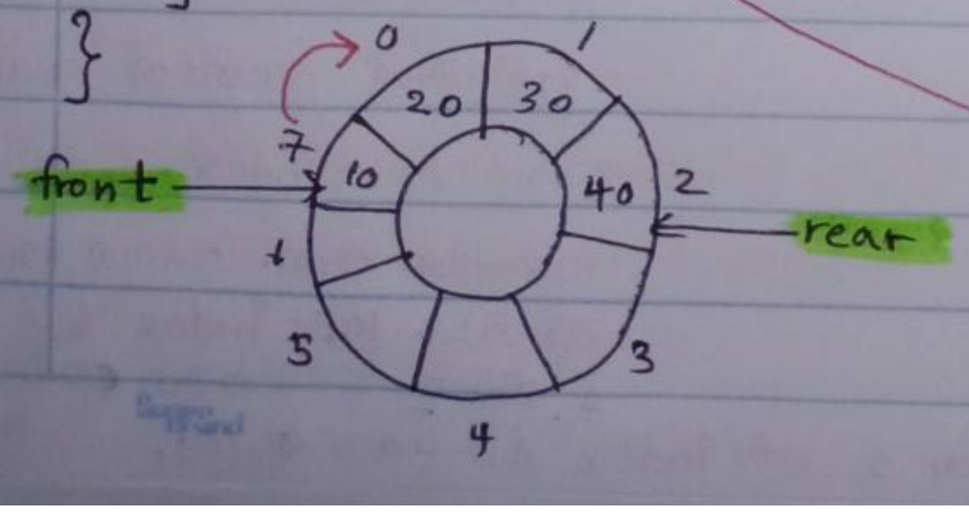
* remove() 1 දී use වෙන්නේ front 1. එතකොට rear 1 වගේම front 1 කේ last index 1ට අවම circular queue 1 දී එක අගය 0th index 1ට පොදුවෙන්න ඕන. එතකොට insert() 1 කේ වගේ මේකෙත් if එකක් දාලා front 1 0th index 1ට පොදුකරනවා.

```
public int remove()
{
    if (nItems==0)
    {
        S.o.p("Queue is empty");
        return -99;
    }
    else
    {
        nItems--;
        int temp = queArray[front ++];
        if (front==maxSize)
            front = 0;
        return temp;
    }
}
```

මේකෙදී මුලින්ම return sttmnt 1 වෙනුවට වෙනවා. return 1ට පස්සේ තව sttments තියෙන්න බැරි නිසා මුලින්ම return කරන value 1 temp variable 1 කට දාගන්නවා.

මේකේදී මුළුමත් front 1 last index 1ට පස්සේ index 1 කේ අගය ගන්නකොත් මේ අවස්ථාවේදී front 1 කේ අගය = 0 කරන්න if 1 කේ use කරනවා.

එතකොට last index 1 කේ තියෙන්නේ අන්තිම front 1 එකේ 0th index 1ට.



Finally temp 1ට දාගන්න remove කරන්න අවශ්‍ය value 1 return කරනවා

Date: 11

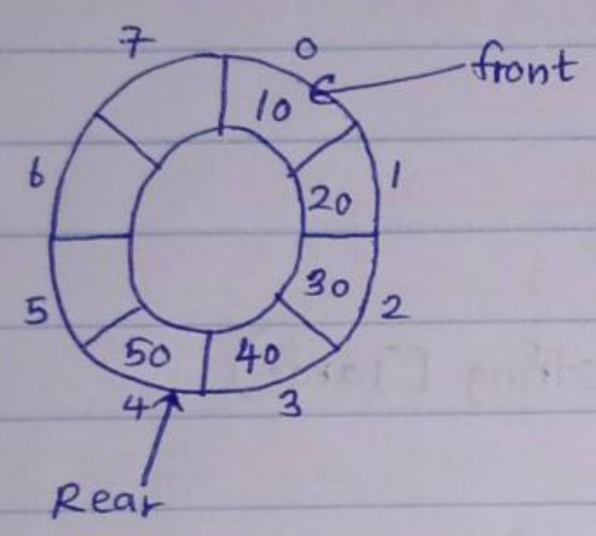
isFull() method

```
public boolean isFull()
{
    return (nItems == maxSize);
}
```

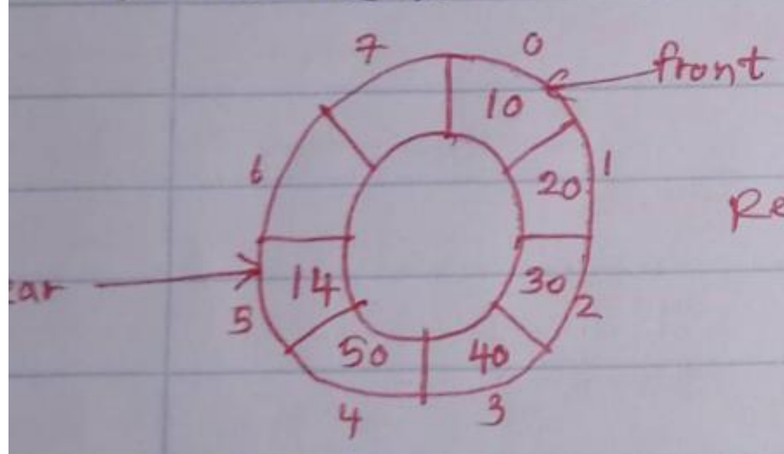
isEmpty() method

```
public boolean isEmpty()
{
    return (nItems == 0);
}
```

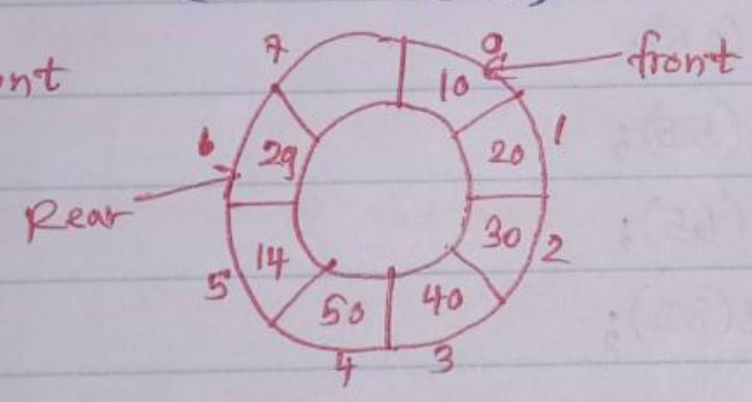
Question 4



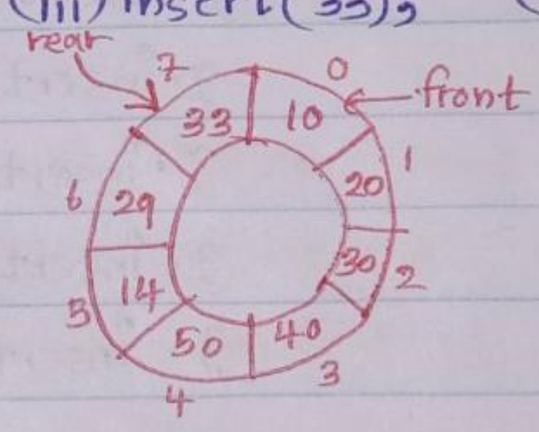
(i) insert(14);



(ii) insert(29);



(iii) insert(33);



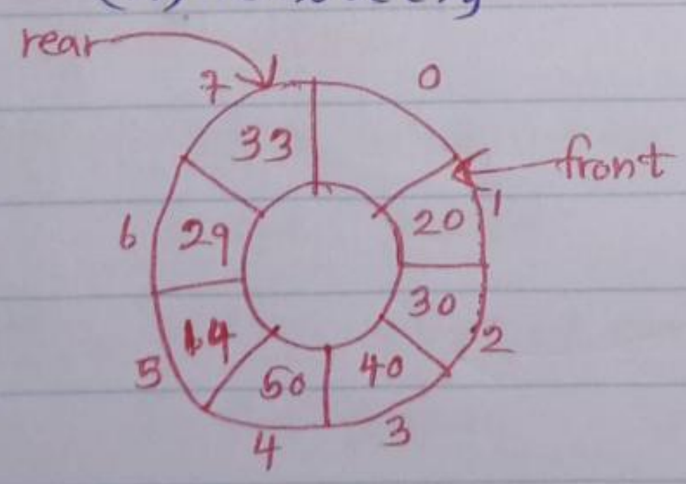
(iv) insert(88);

X

(v) peekFront();

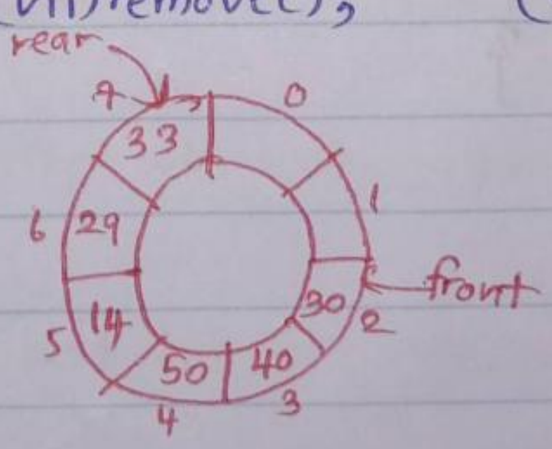
return(10);

(vi) remove();



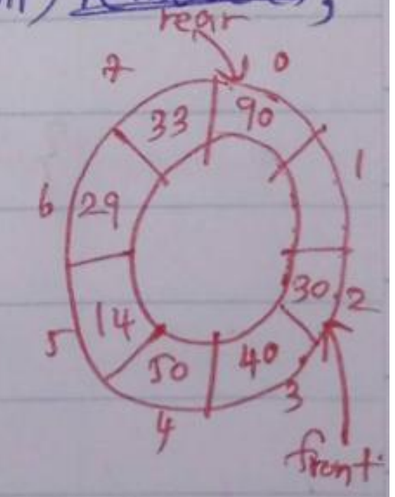
return(10);

(vii) remove();

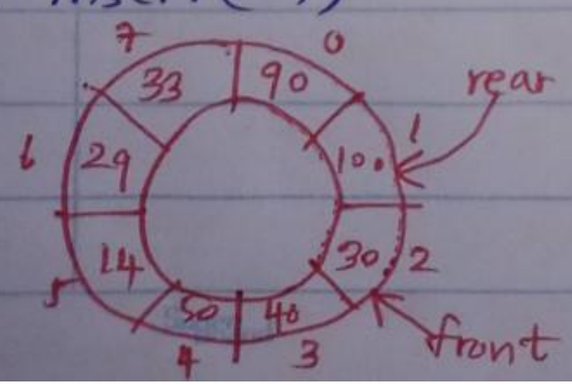


return(20);

(viii) ~~remove();~~ insert(90);



(ix) insert(100);



(x) peekfront();

return(30);

Scanned by TapScanner

Question (6)

Creating a Queue

- Using the implemented QueueX class, Write a program to create a queue with max size 10 and insert the following items.

  10 25 55 65 85

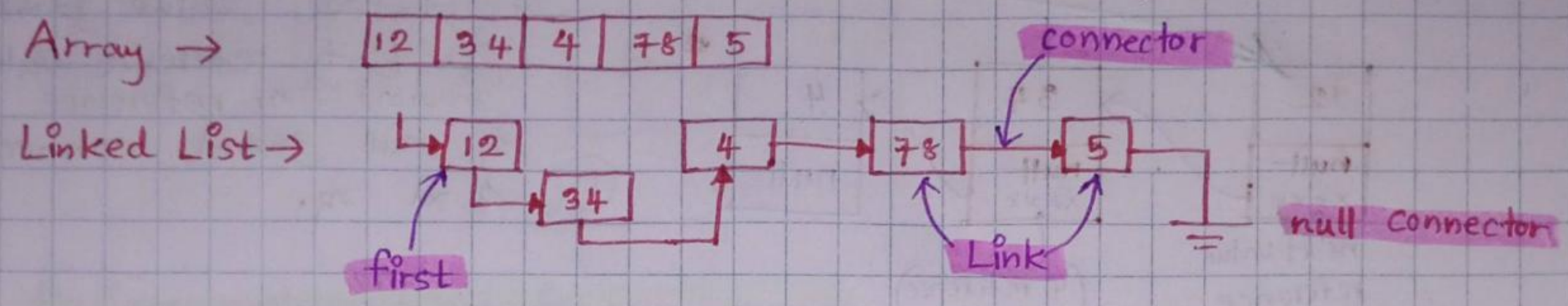
- Delete all the items from the queue and display the deleted items.

```
//main app

class QueueApp
{
    public static void main (string [] args)
    {
        QueueX q = new QueueX(10);
        q.insert(10);
        q.insert(25);
        q.insert(55);
        q.insert(65);
        q.insert(85);

        while (!q.isEmpty())
            System.out.println (q.remove());
    }
}
```


Scanned by TapScanner

- Linked Lists -

* Arrays වලට තියෙන්නේ වැඩිපුරම use වෙන data structure 1 නම් linked list කියන්නේ.



* Array 1 කි සහ linked list 1 කි ගතිතම linked list 1 කේ තියන වෙනස තමා, linked list වලට array 1 කේ වගේ indexes නෑ.
* Linked list 1 කේ values connect වෙනිකේ arrow connectors හරහා
* Array 1 කම වගේ pre defined size එකක් නෑ. අපිට connector 1 කි පදලා new element 1 කි linked list 1ට කිව වෙලාවක connect කරන්න පුළුවන්.
* Linked list 1 කේ data items directly access කරන්නන් බෑ.
* Linked list size is not fixed. ∴ Anytime we can create a link and add value to it.
* Linked list 1 කේ Last value 1 අඳුනගන්නේ "null connector" 1 කි. value 1 කම null connector 1 connect වෙලා තියෙනවනම් කිව තමා Last value 1 වෙන්නේ.
* Linked list 1 කේ,
  මුලින්ම තියන link 1 "first" කියලා හඳුන්වනවා.
* Link වලට direct accessing නැති නිසා අපිට අවශ්‍ය data 1 හොයන මුල ඉඳන් ඔක්කොම links by pass කරන් යන්න ඕන.

* අපිට Link 1 කි section 2 කට බෙදන්න පුළුවන්.

Data ← Link 1 කේ තියෙන value 1

Next ← Connector 1 (මේ next 1 කි තියෙන්නේ next link 1 කියන එක. Link 1 කියන්නෙ value 1 නෙමෙයි. value 1 කියන්නෙ Link 1 ගේ 1 section 1ක විතරයි)

මේ next connector 1 pointer 1 කි වගේ වෙනකොට කලින් next value එකේ reference 1 ට connect කරන එක

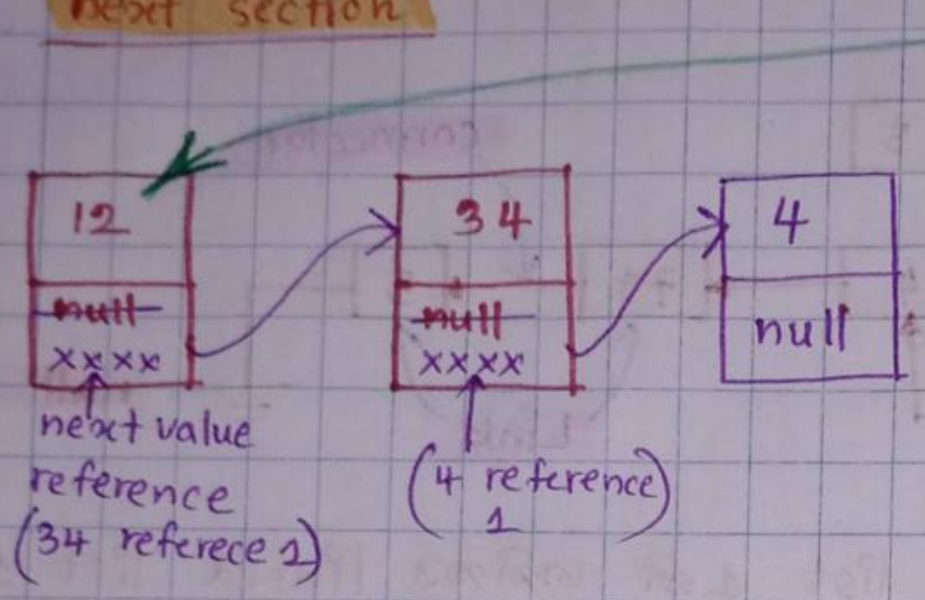
∴ Link and values is too different things.

Scanned by TapScanner

** Stack & [illegible] implement කරන්න finals PP වලට [illegible] tutorials වල [illegible] **

* ∴ අපි 12 next 1 34 විදිහට කියනවා නෑ. මොකද 12 , 34 කියන values.
* first link 1 ගේ next 1 2nd Link 1 කියලා මේ විදිහට තමා කියන්න ඕ

next section



මේ values null or garbage value or whatever value ඉන්නෙ ඉඳලත් නෑ. But reference values ඒ කියන්නෙ මින. reference value 2 null ඉන්නකොට connectivity 1 ඒක නෑ.

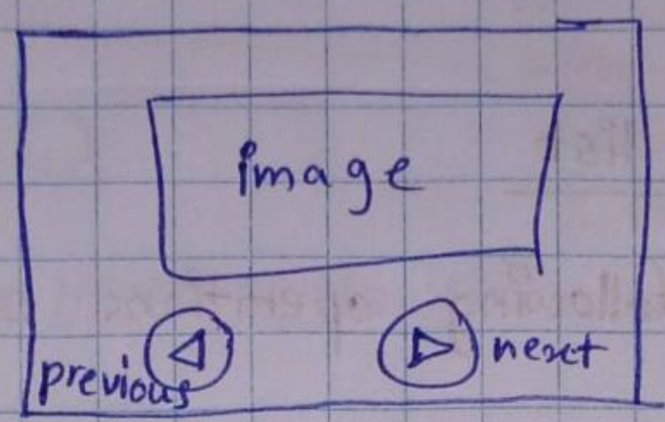
* linked list 1ේ තියෙන්නේ මුලින්ම ඉන link 1 විතරයි තමා ඒක හරි ගන්න ඕන.
* මේ විදිහට මුලින්ම 12,34 කියන values 2 කියන link 2 කියනවා ඒවා දෙකේම next 1 වෙන්නේ null. මොකද ඒ දෙකම connect වෙලා නැති නිසා ඒවට connectors නැතා.
* මේ දෙක connect කරන්න නම් අපි කරන්න ඕන 12 කියන link 1ේ next 1ට 34 කියන link 1ේ reference 1 දෙන ඕන. කියනකොට 12 next 1ේ ඉඳලා 34 value 1ට connector 1ේ තියෙනවා.
* මේ විදිහට අපි link connect කරලා linked list 1ේ හදන්න පුළුවන්.
* 4 value 1 කියන link 1 තමා දැනට list 1ේ. list value 1 වෙන්නේ ∴ ඒකේ next 1 null වෙනවා (null connector)

* Each link object contains a reference to the next link in the list.

Scanned by TapScanner

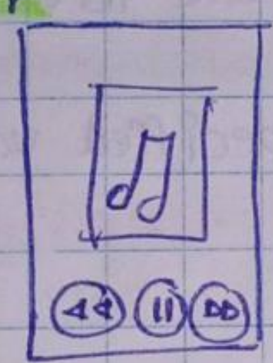
Applications of linked list in real world

* Image viewer - previous and next images are linked, hece can be accessed by next and previous button.



* Previous and next page in web browser

* Music Player



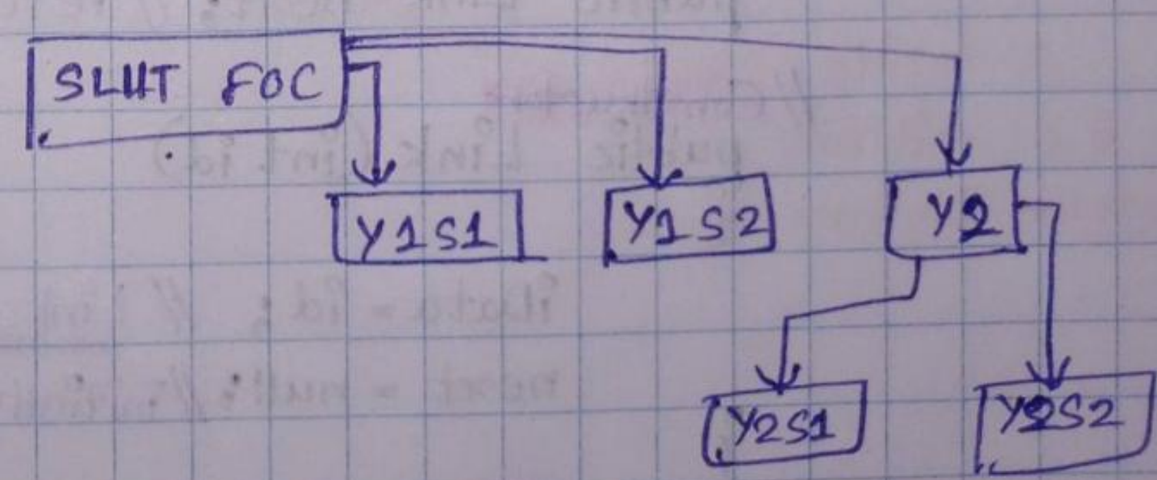
Applications of linked list in computer science

* Implementation of stacks and queues.
  (stack and queues implement කරන්න arrays වගේම linked lists ත් භාවිතා කරනවා)

* Implementation of graphs : Adjacency list

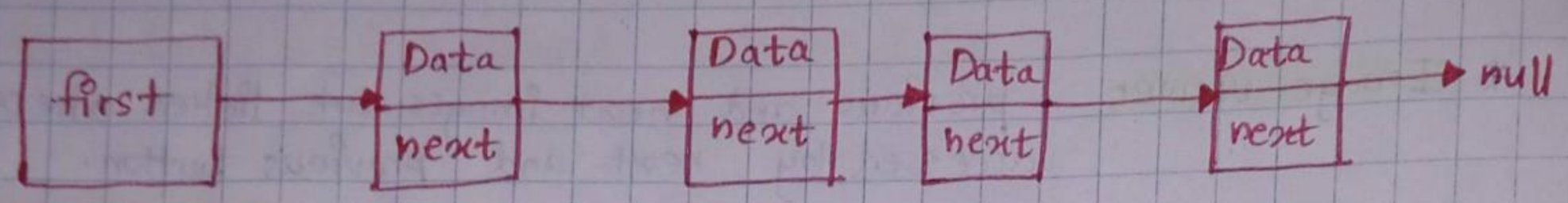
* Dynamic memory allocation

* Maintaining directory names



Scanned by TapScanner

linked list



Operations in linked list

* Mainly the following operations can be performed on a linked list.

- **Find** - Find a link with a specified key value
- **Insert** - Insert links anywhere in the list.
- **Delete** - Delete a link with the specified value.

Implementation

* Linked list implementation වලදී classes 2 කි තිබෙනවා, Link class 1 සහ Linked list class 1.

class Link

* මෙහෙදී Link class 1 Linked list class 1 කි සමාන නිසා attributes හි public වෙනවා.
* Link 1 ක section 2යි. ඒ දෙකට තමා attributes තිබෙන්නෙ මෙහෙදී

```
class Link
{
    public int iData; // Data value 1ට
    public Link next; // reference 1 store කරනවා (reference 1 කි තව link 1 කි point කරන නිසා type 1 Link)
    //Constructor
    public Link (int id)
    {
        iData = id; // Link 1 කෙනෙක්. කි data value 1 parameter 1 කි විදිහට අරන් assign
        next = null; //initially next value 1 null (දැනට Link 1ට connect කරලා නැති නිසා)
    }
    //Display Link method
    public void displayLink()
    {
        s.o.p (iData);
    }
}
```

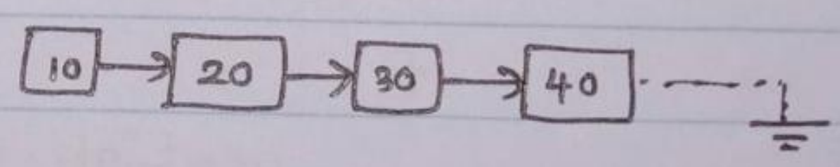
Scanned by TapScanner

Class LinkList

```
class LinkList
{
    private Link first;

    public LinkList()
    {
        first = null;
    }

    public boolean isEmpty()
    {
        return (first ==null);
    }
    public void displayList()
    {
        Link current = first;
        while (current != null)
        {
            current.displayLink();
            current= current.next;
        }
    }
}
```

~~public boolean find()~~

find 30;



find method

```
public boolean find (int id)
{
  while (current != null)
  {
      if (current.iData == id)
            return true;
      else
          current = current.next;
  }
```

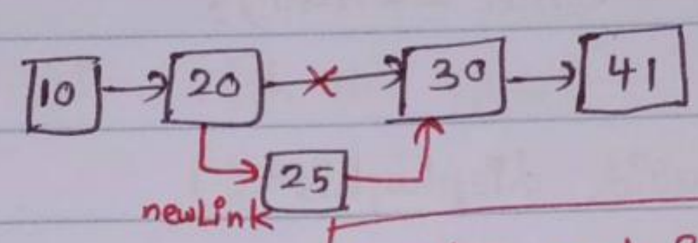
തരുന്നത് ഇവിടെ 30

Date:.....

```
return false;

}
```

insertFirst method

```
public void insertFirst (int id)
{
    Link newLink =new Link(id);
    newLink .next = first;
    first =newLink;
}
```



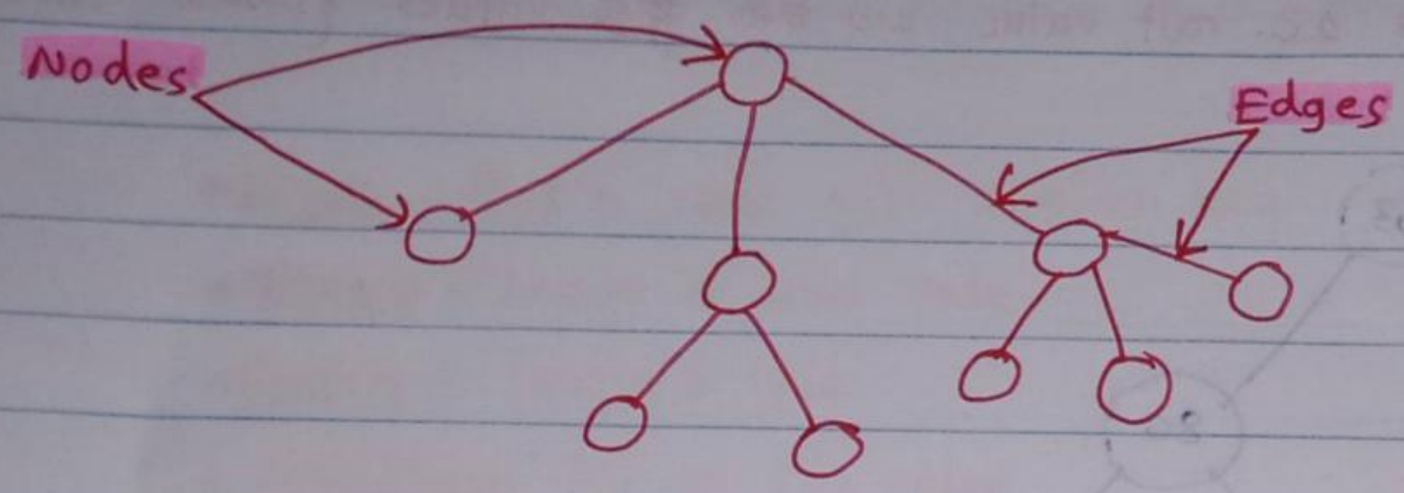
* value ඉදි insert කරන්න වද. linked list ඉදලි. කරන්නේ නම් value ඉ link කරන්න.

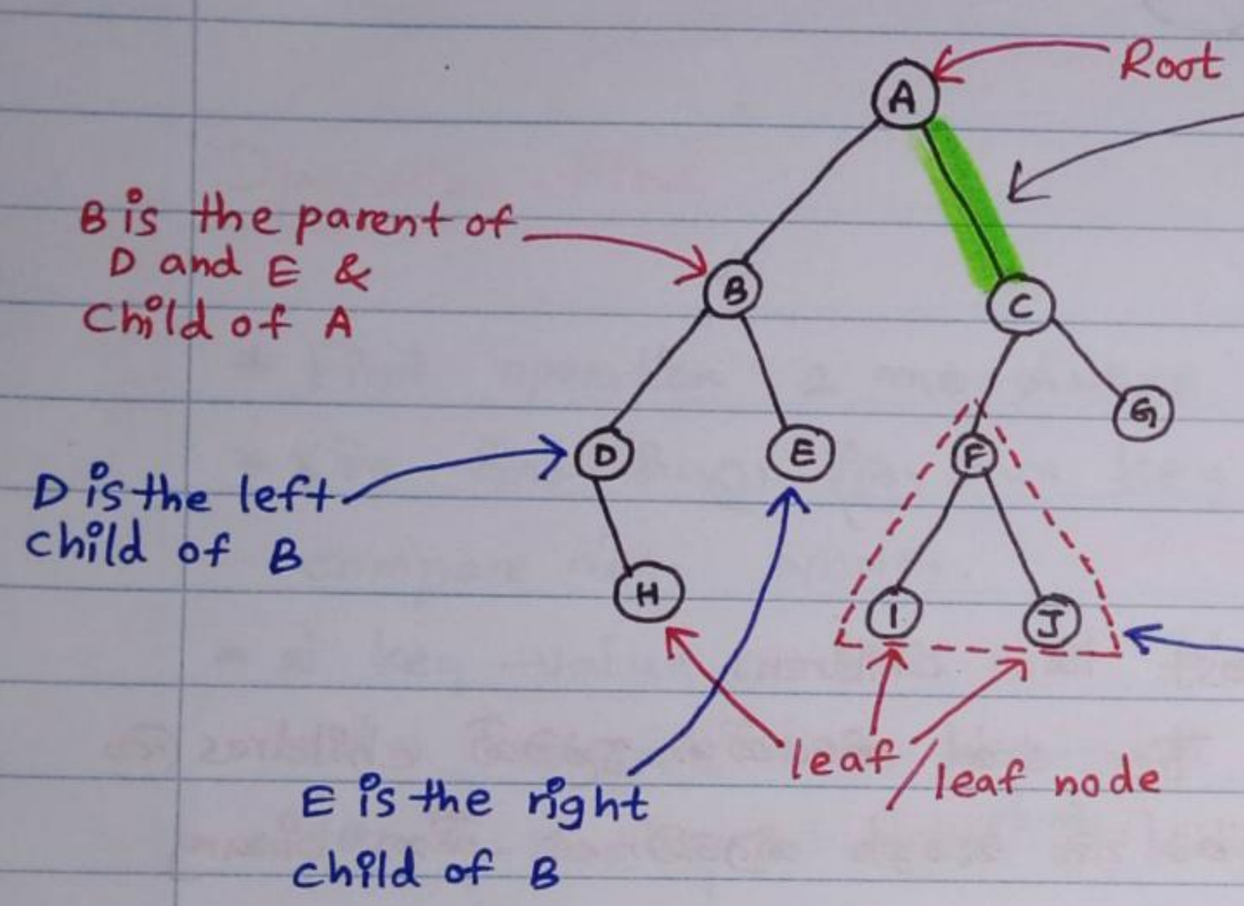
insertMiddle method

```
public void insertMiddle (int key, int id)
{
    Link newLink = new Link (id);
    Link current =first;
    while (current != null)
    {
        if (current.iData ==key)
        {
            newLink.next = current.next;
            current.next = newLink;
        }
        else
            current = current.next;
    }
}
```

Scanned by TapScanner

- Tree Data Structure -





Path :-
Node 1 ක ඉඳලා තවත් node 1 කට edges මට්ටම ගියාම තියෙන path 1 කි කියනවා

F is child of C & parent of I and J

A subtree with F as its root.

Any node which consists of its children and it's children's children and so on is called a sub tree.

Key value of a node

- Tree 1 කේ each node 1 කටම object කියනවා information contain කරලා කියන.

∴ unique data item 1 කි key value 1 කි විදියට තමයි කරන්නෙ කි අදාල node 1 අඳුනගන්න.

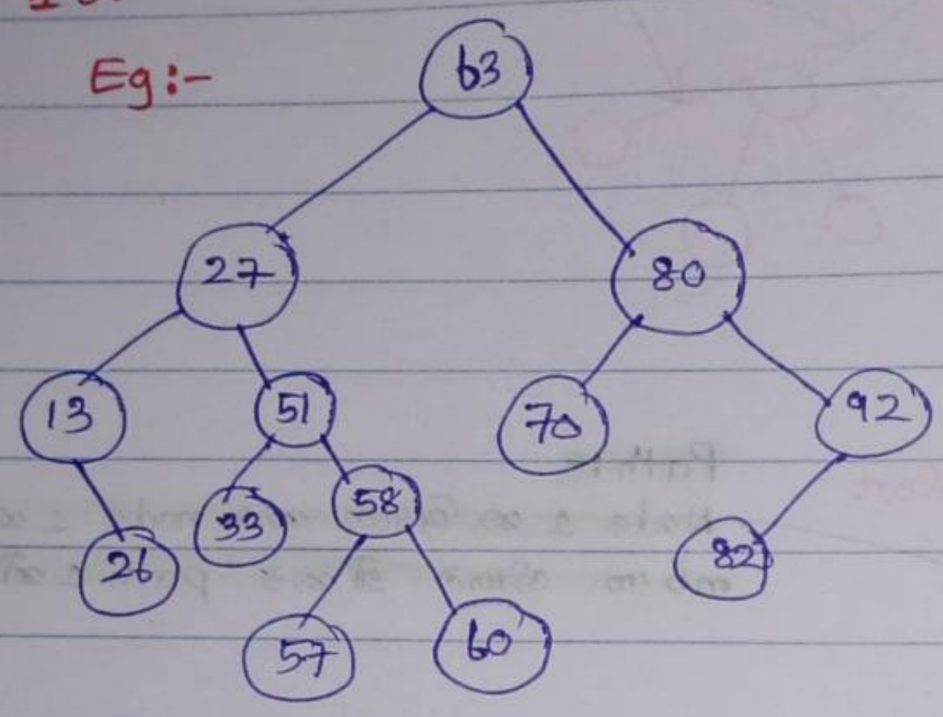
මේ key value 1 items search කරන or other operations perform කරන්න පොදුගන්න පුළුවනි.

Eg:- person object - social security number
car parts object - part number

Scanned by TapScanner

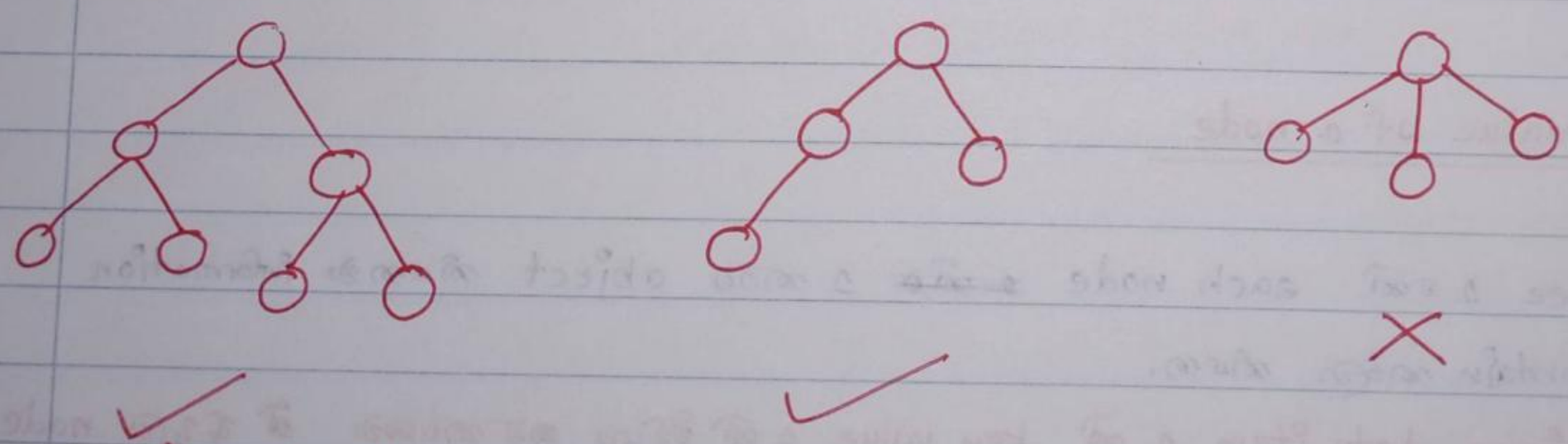
* මේ tree 1ට data add කරද්දි අපි මුලින්ම add කරන්නේ root value 1 .'ට වඩා root value 1ට වඩා අඩු values කලකියාවෙ දානේ right side 1ට. root value 1ට වඩා අඩු values දානේ left side 1ට.

Eg:-



Binary Search Tree

* වෙනදා tree has at most two children.
මේ කියන්නේ Binary Search Tree 1ක තියෙන්න පුළුවන් childres ලා 2 කි කෙනේ 'ර්ට අඩුවෙන්. 'ර්ට වඩා වැඩිනම් තිබෙන්නේ නම් binary search tree 1 කි නෙවෙයි.



Binary Search Tree

Operations of Binary Search Tree

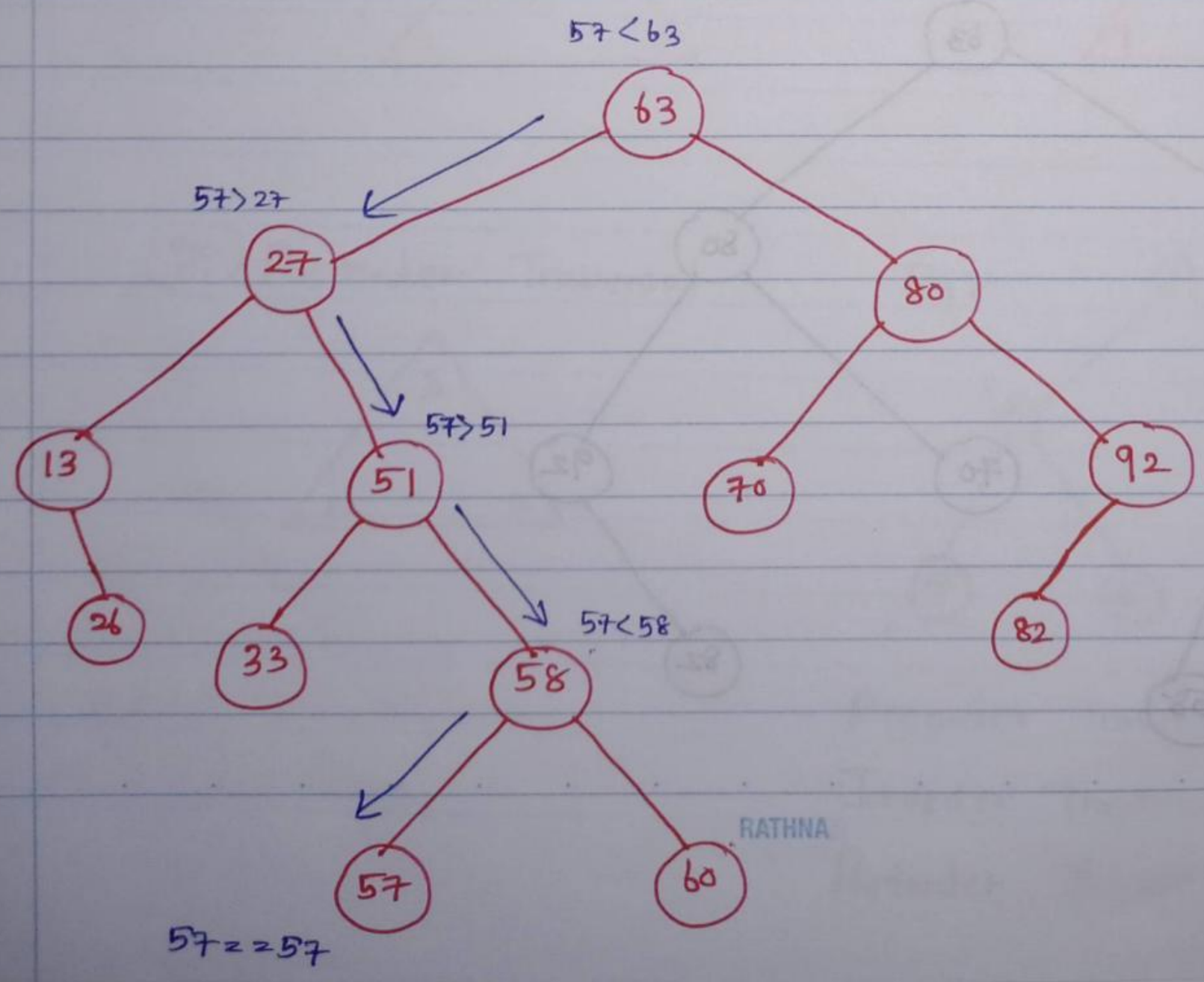
There are four main operations perform in a Binary search tree.

* Find – find a node with a given key
* Insert – Insert a new node
* Delete – Delete a node
* Traverse – visit all the nodes

Operation – Find

* Find operation ද මුල සිටගෙන සවස් වෙන root ද්‍රව්‍ය.
* දැන් find කිරීමට දීලා තියන key value ද්‍රව්‍ය දැනි වුණොත් root value සමඟ සමාන compare කරලා නැවතෙනවා.
* මේ key-value ද්‍රව්‍ය less නම් 'b'ට සවරයි root ද්‍රව්‍ය left child සමඟ compare කරනවා. higher නම් right child ද්‍රව්‍ය compare කරනවා.
* මේ විදිහට repeat කරලා key value ද්‍රව්‍ය හමුවෙනකන්.

Eg:- Find value 57

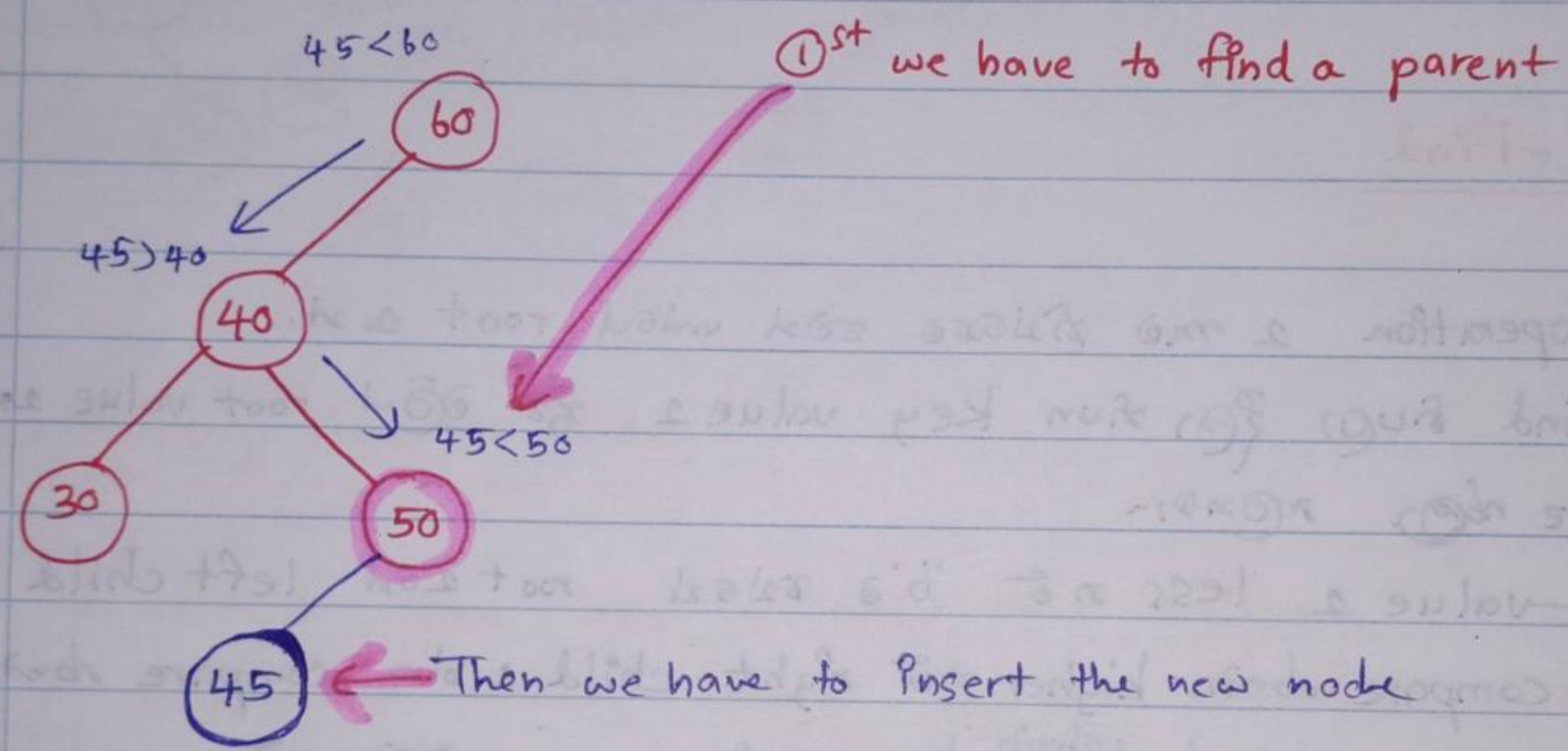


Scanned by TapScanner

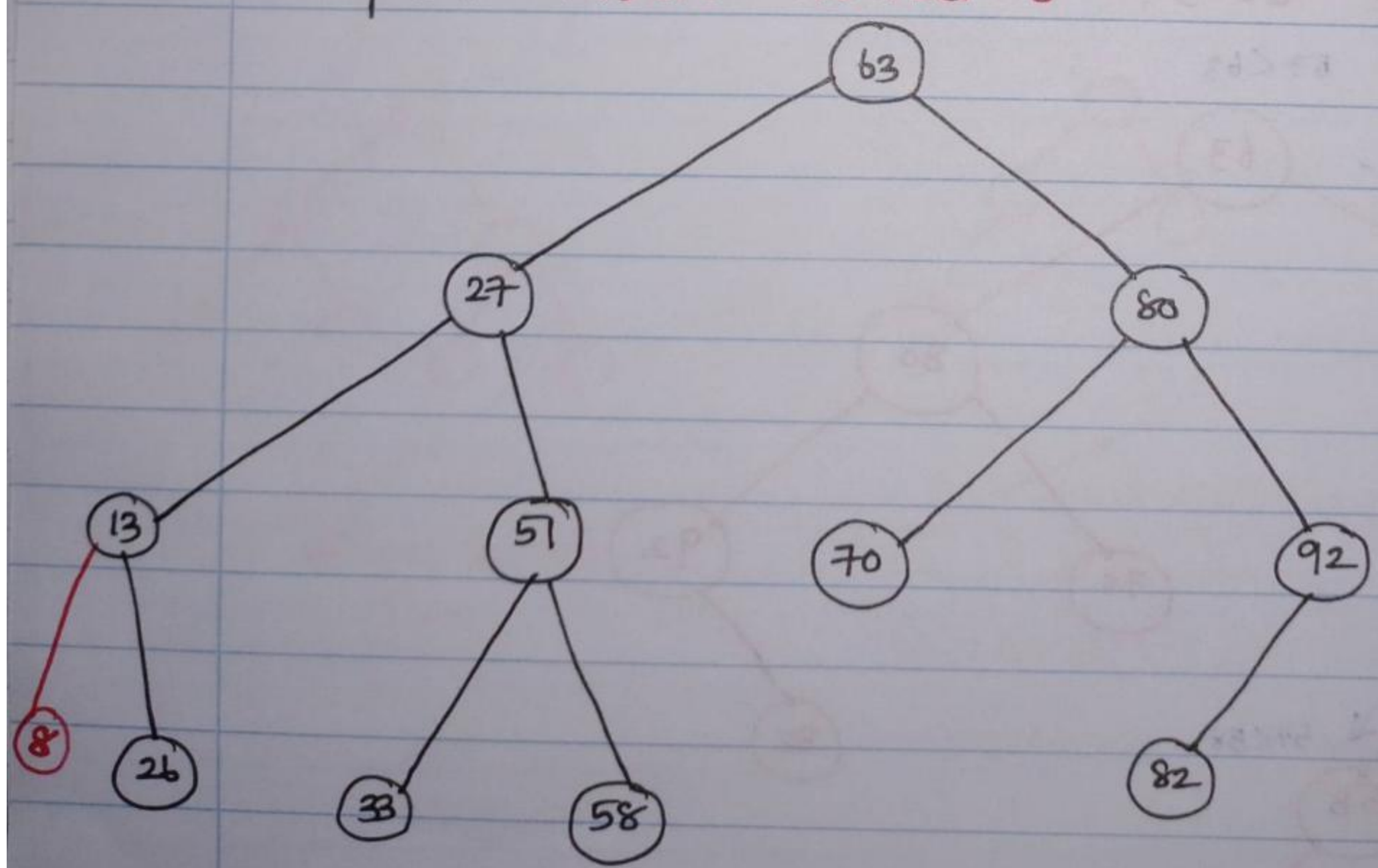
Operations - Insert

- Insert දැමීමේ new node එක create කරනවා.
- ඊට පස්සේ new node එක insert කරන්න place එක තෝරගන්නවා. (parent)
- parent තෝරගත්තාට පස්සේ එතනට left or right child කියලා (value එක අනුව) new node එක connect කරනවා.

Egr Insert value 45



Example :- insert number 8



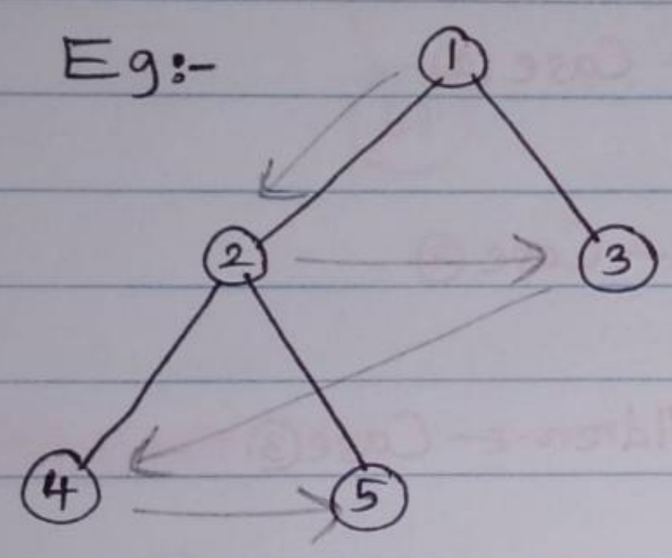
Scanned by TapScanner

BFS vs DFS for Binary Tree

Tree is typically traversed in two ways.

(01) Breadth First Traversal (Level Order Traversal)

Eg:-



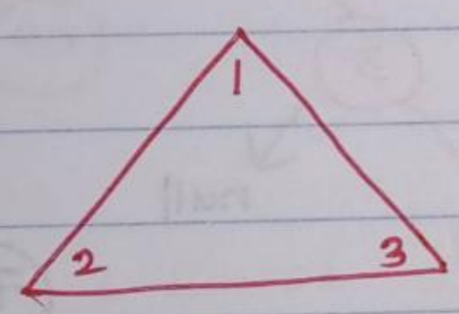
Breadth First Traversal:

1 2 3 4 5

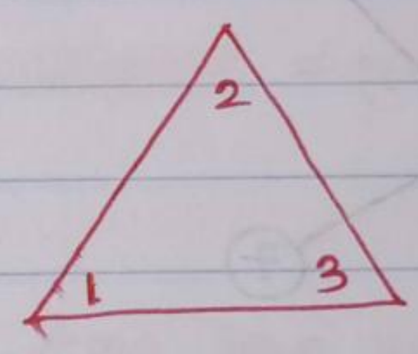
* මෙහෙදී values බිඳ දමමින් යනවා

(02) Depth First Traversals:

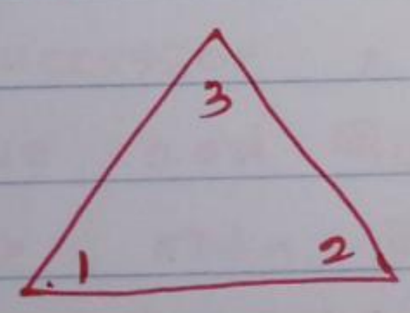
(i) Preorder Traversal



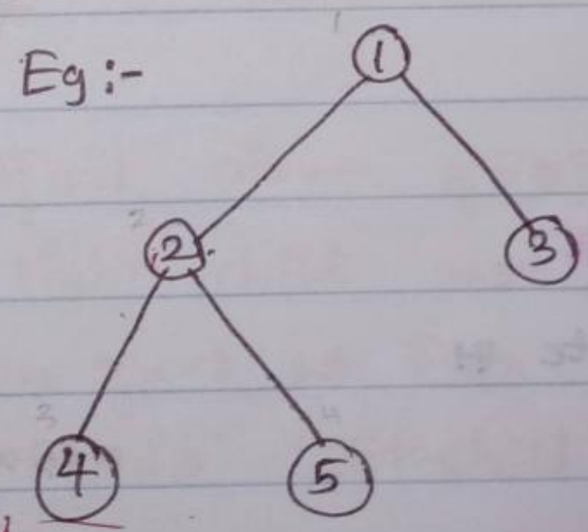
(ii) Inorder Traversal



(iii) Postorder Traversal



Eg:-



Preorder Traversal : 1 2 4 5 3

Inorder Traversal: 4 2 5 1 3

Postorder Traversal: 4 5 2 3 1

Scanned by TapScanner

Operations – Delete

(01) මුලින්ම delete කරන්න ඕන node 1 හොයාගන්න ඕන.

(02) ඊට පස්සේ අපි බලනවා cases 3 න් කියන්නා.

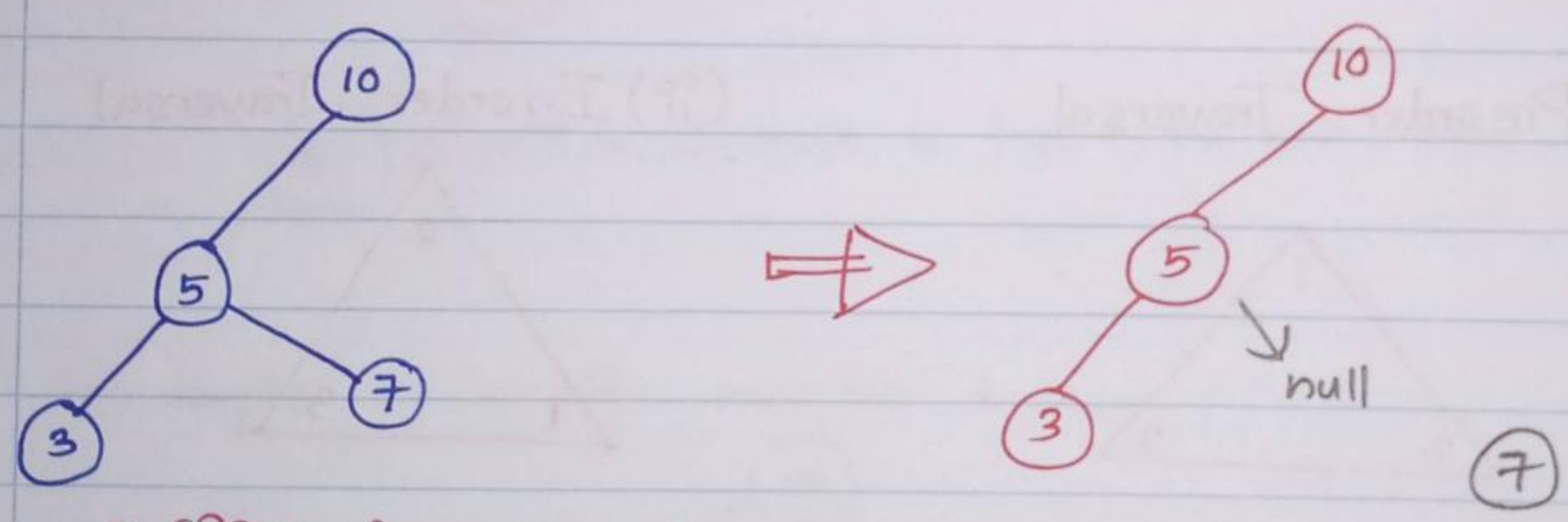
(I) Node to be deleted is a leaf ← Case (1)

(II) Node to be deleted has one child ← Case (2)

(III) Node to be deleted has two children. ← Case (3)

Case 01

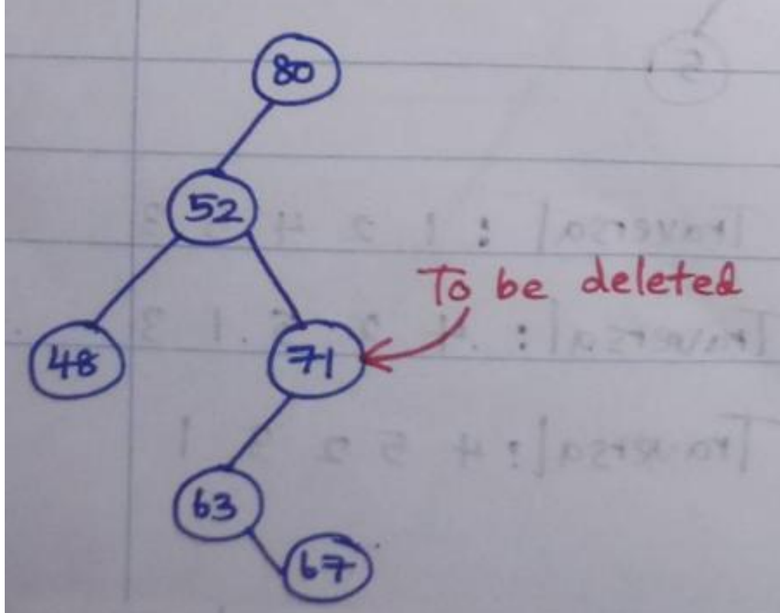
Eg:- Delete 7



* මෙතෙක් කියනකොට අපිට තේරුණේ නෑ.
හොඳින්ම 7 delete කරන්න අවශ්‍යයි.

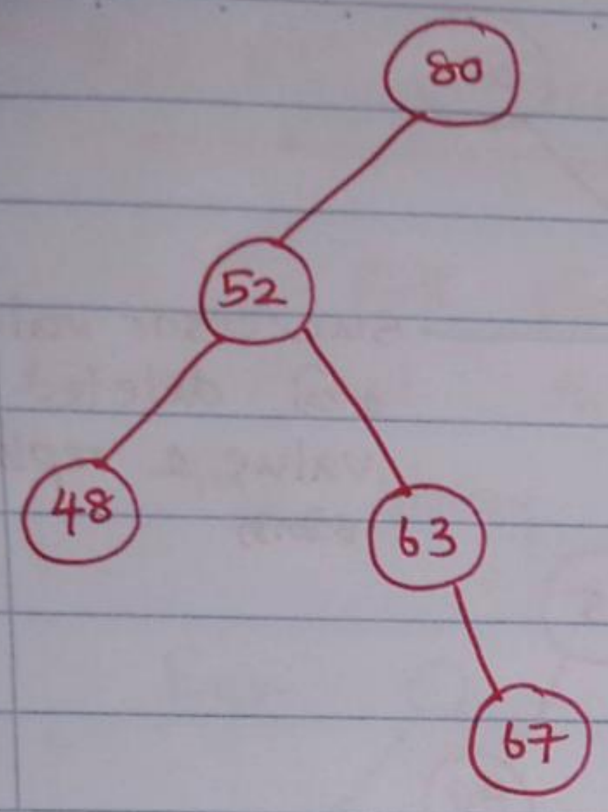
Case 02

Eg :- Delete 71

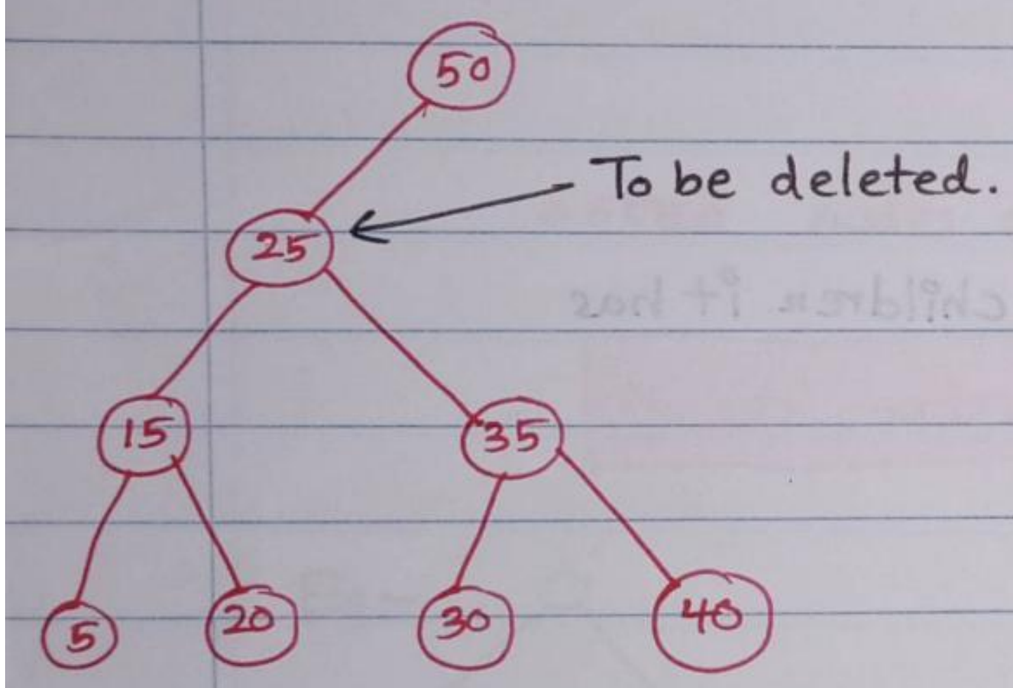


* මෙතනදී අපිට 71 delete කරන්න ඕන. But it has one child. ∴ මෙතනදී 71 delete කරනකොට එයාගේ child value 1 ක් (63) 71 replace වෙනවා.

Scanned by TapScanner



Case 03



* මේ විදිහට children ලා දෙන්නෙක් ඉන්නකොට මුල් value 1 අනතුරු delete කරන්න ඕනෑ.

මුලින්ම මේ value 1 replace කරන්න පුළුවන් value 1 තෝරගන්න ඕන.

එතකොට "successor" කියලා

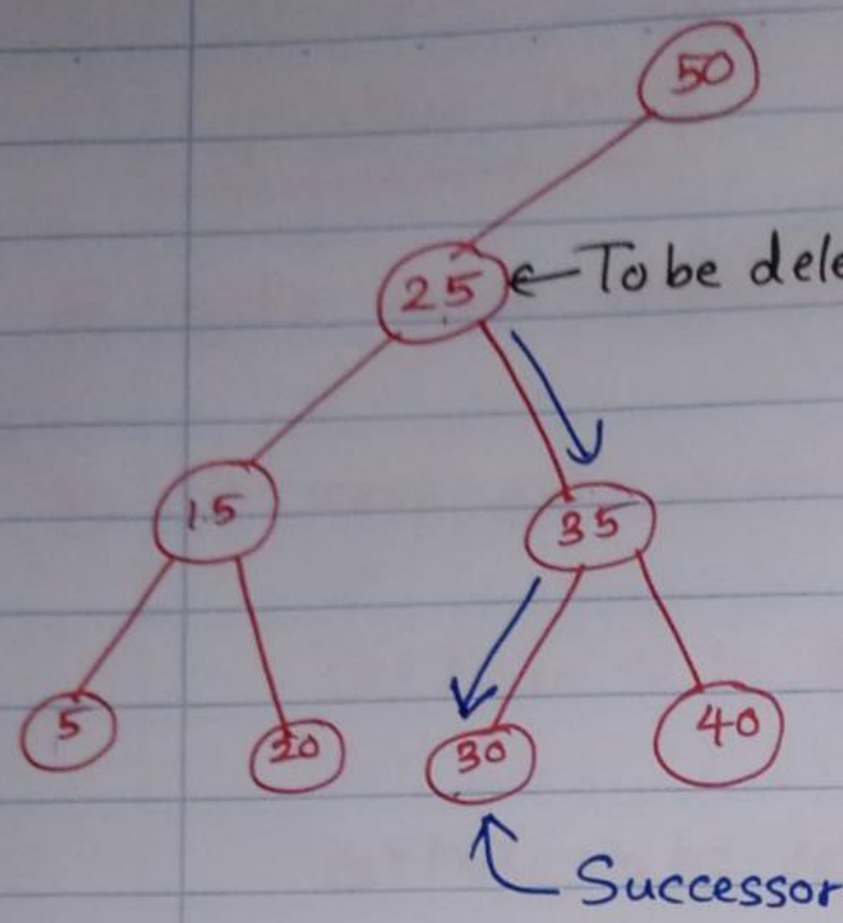
* successor 1 තියන්නේ delete කරන පුහාව ලොකු විශාල පුහාව. මේ example 1 ත් හරිය successor = 30 වෙනවා.

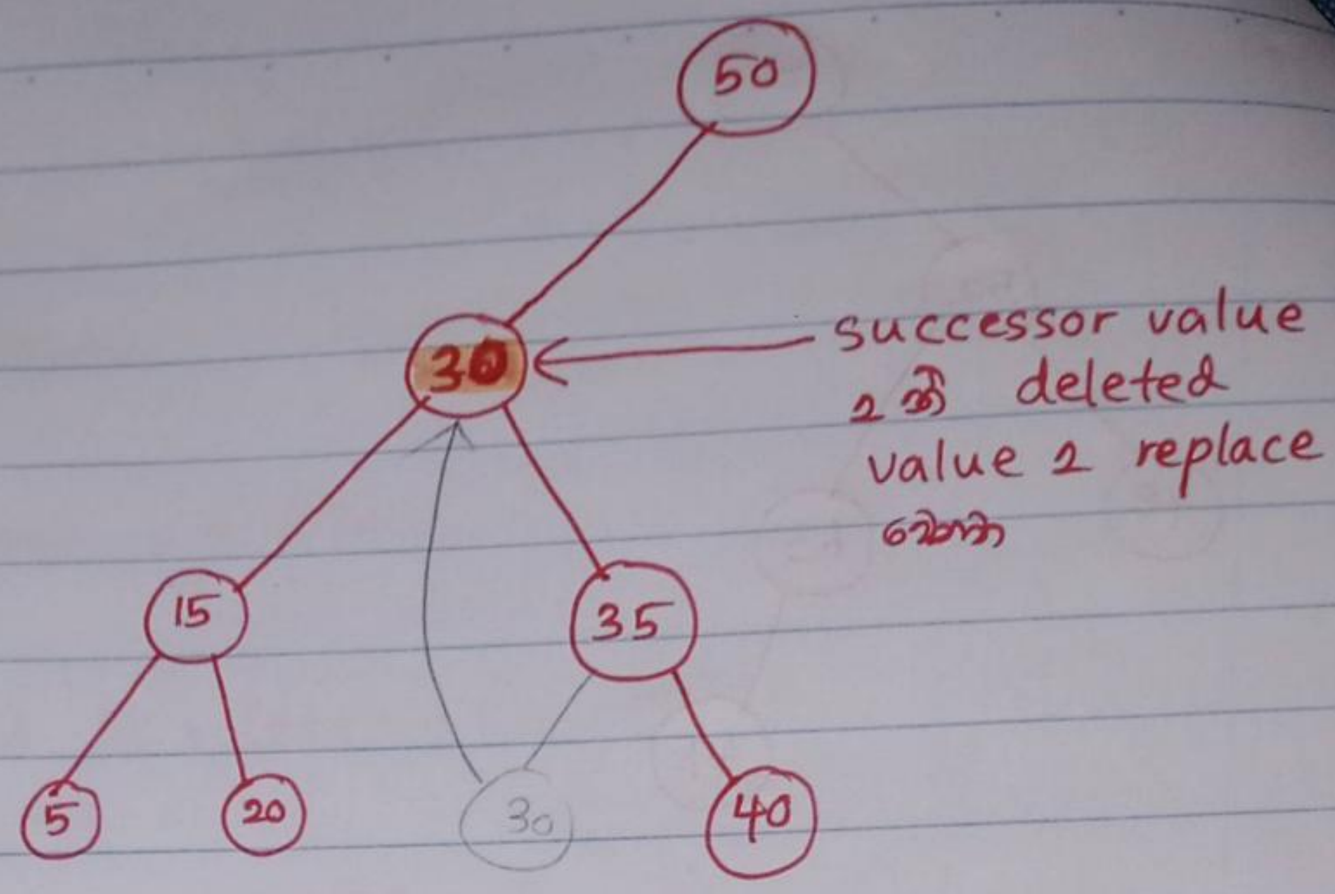
ඒ successor 1 තෝරගන්න විදිහක් තියනවා. මුලින්ම delete වෙන value 1ත් ඉඳලා එකගේ right child කෙනාව යන්න ඕන.
(∵ replace වෙන්න ඕන delete වෙන පුහාව එක විශාල පුහාවකින්)
ඊට පස්සේ leaf 1ත් නමුත් වෙනකම් පුහාවේ left child පුහේ එන්න ඕන.

එතකොට තමා successor value 1 නමුත් වෙන්නේ.

RATHNA

Scanned by TapScanner

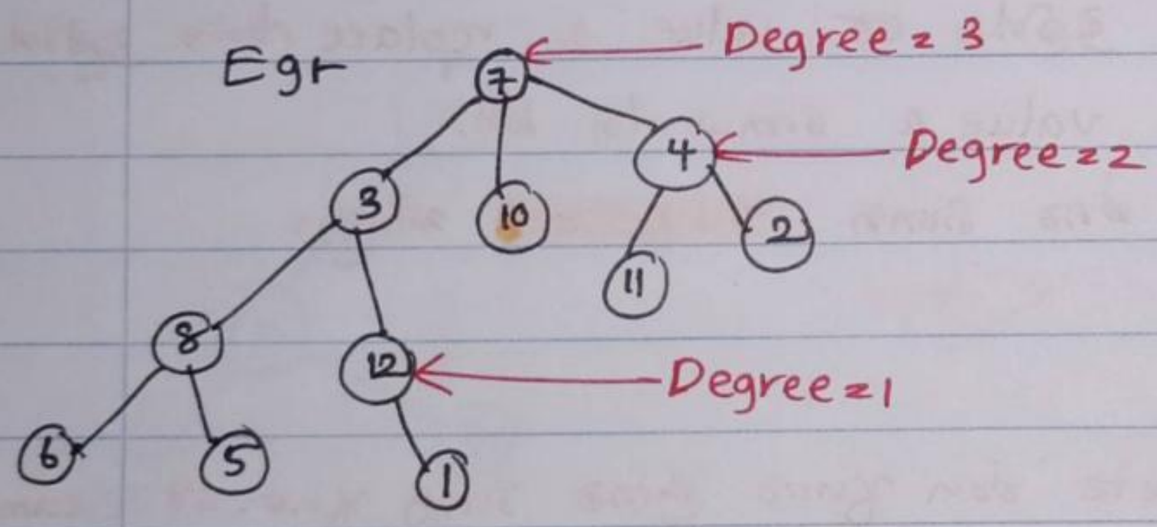




Tree Terminology

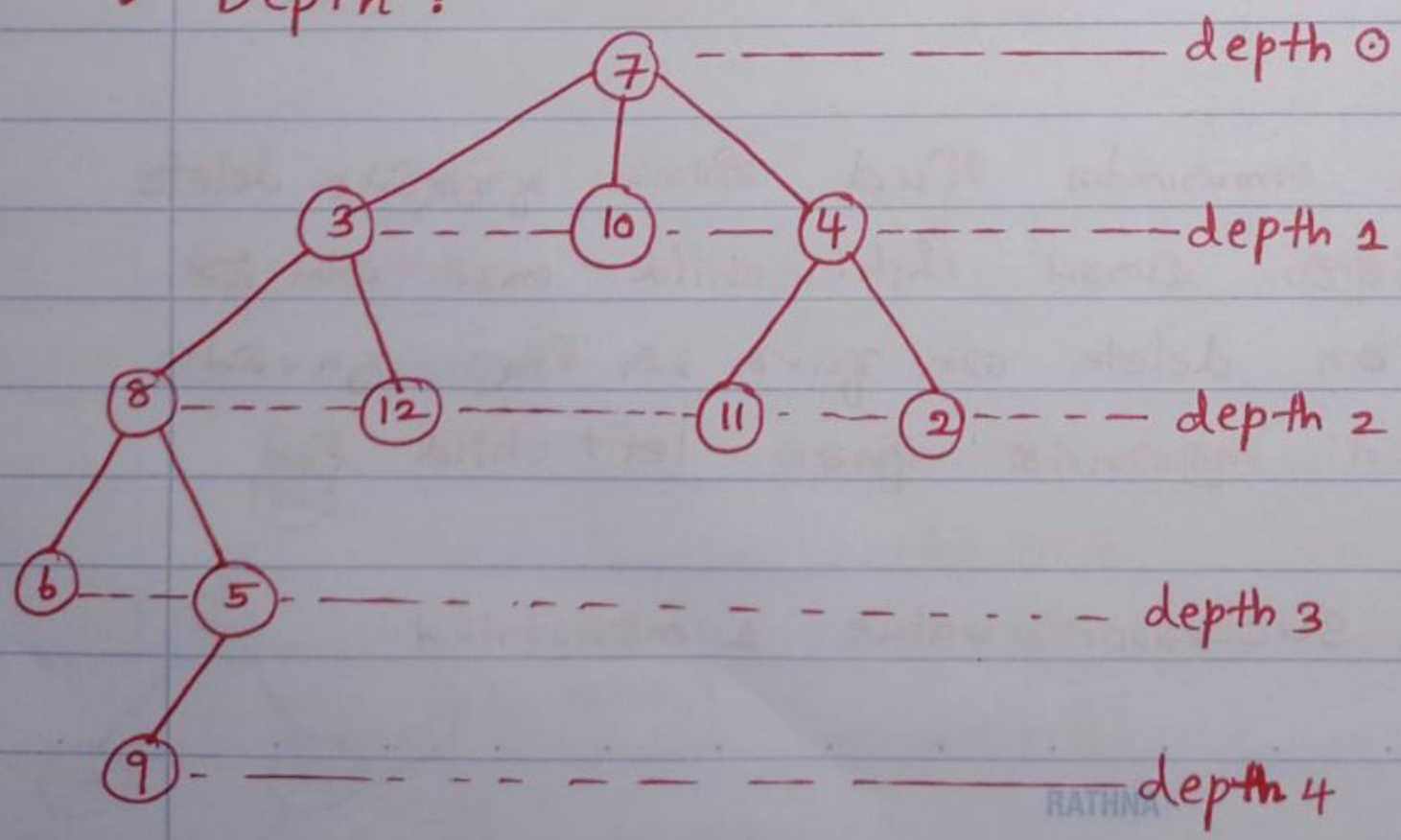
- Degree of a node :- The number of children it has

Egr



- Height :-
  Maximum depth നാണ് height കിട്ടുന്നത്.
  Height = 4

- Depth :

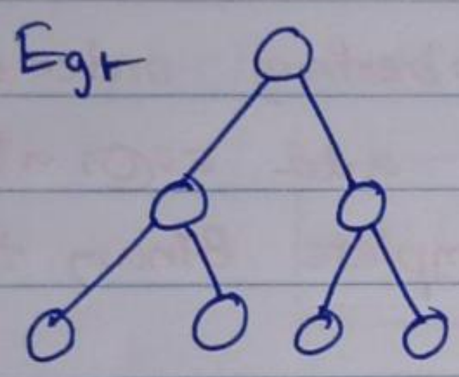


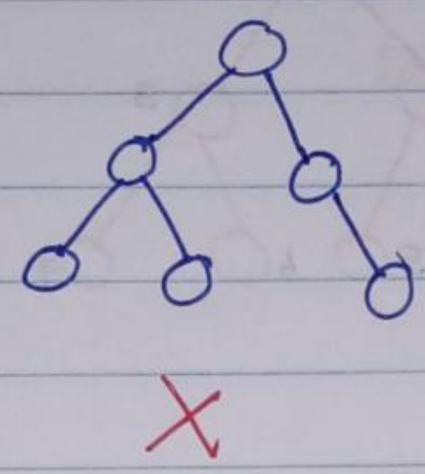
Scanned by TapScanner

Full Binary Tree

* Full binary tree ද්‍රැවක all the parents have two children. මේ නිසාත් fully completed වෙච binary tree ද්‍රැවක තමා full binary tree නිසාත්.

Eg:-

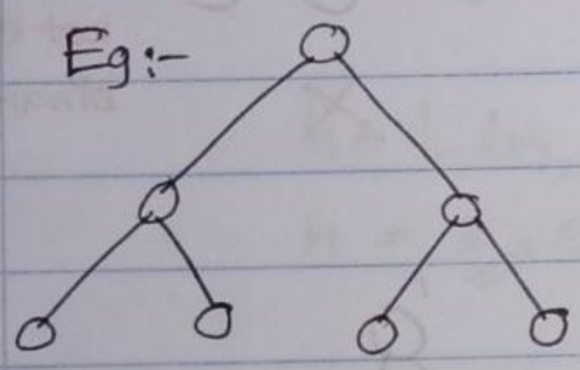




* මේකේ nodes වලට height එකක relationship ද්‍රැවක තියනවා.

$$\boxed{\text{No. of nodes } (n) = 2^{h+1} - 1}$$

Eg:-



Height = 2

$n = 2^{h+1} - 1$

$= 2^{2+1} - 1$

$= 2^3 - 1$

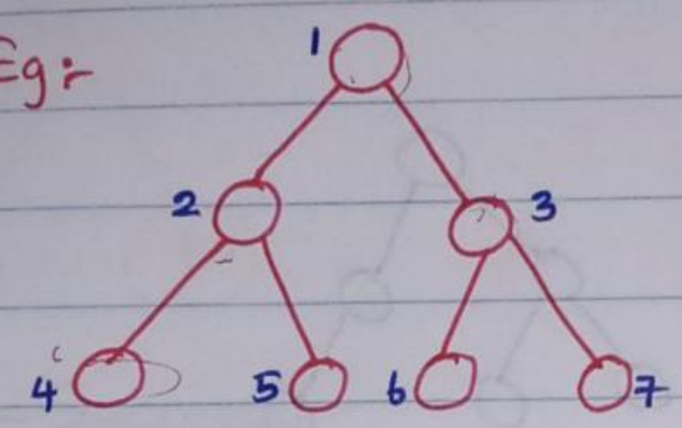
$= 7$ //

Scanned by TapScanner

Complete Binary Tree

* complete binary tree ආකාරේ each level පිළි පුරා ඉවර වීම left ේ සිට right. යටතට ගියොත් තමා එම complete binary tree ආකාරය.

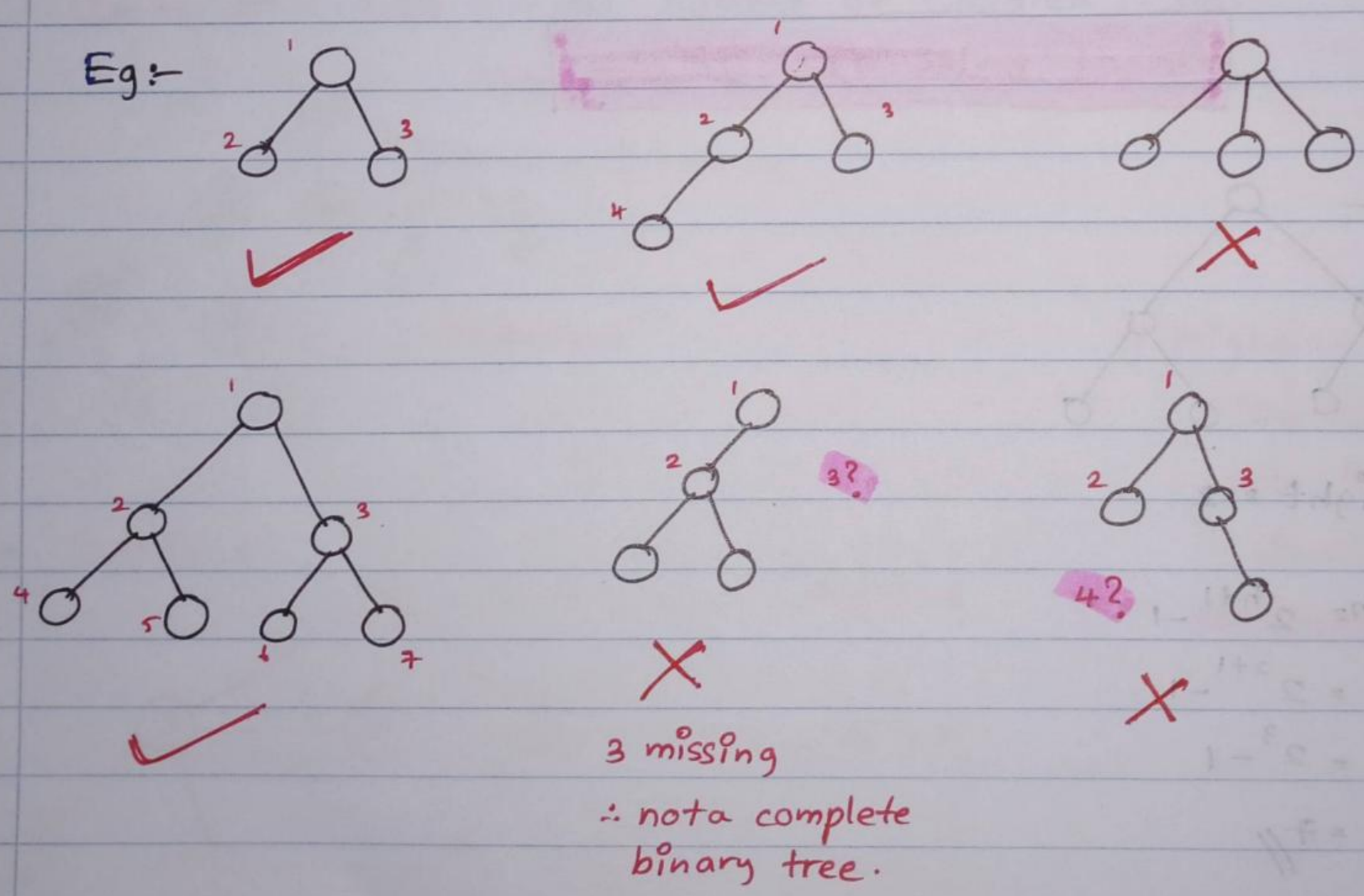
Eg:-



මේ numbering order ේ values add වෙලා නම් එය complete Binary tree ේ.

* Full binary tree is also a complete binary tree. But complete binary tree ේ සෑම කිසිවෙ full binary tree ේ වෙන්න ඕනෑ.

Eg:-



* This is not even binary tree.

∴ not a complete binary tree.

Scanned by TapScanner

Height of a Complete Binary Tree

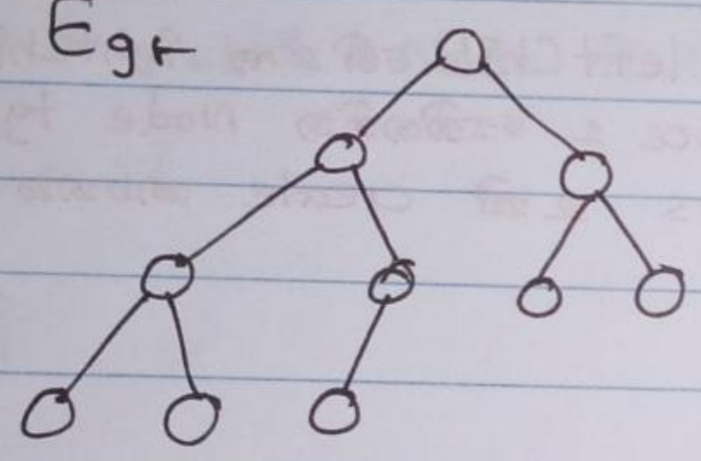
* Complete binary tree ඉන height එක යන්න සූත්‍රය $\lfloor \log_2(n) \rfloor$ බවින් දැනගමු.

n-elements

$$\boxed{h = \lfloor \log_2(n) \rfloor}$$

$\lfloor \rfloor$ - මෙතෙක් බිඳන්න ඇතැත්නම් පූර්ණ පරමාණුව ගන්න බිංදුව.

Egr



$n = 10$

$h = \lfloor \log_2 10 \rfloor$

$h = 3$

$\log_2 8 = 3$

$\log_2 16 = 4$

* 10 ගේ දශම අගේ බිංදු 3.5ට අඩු ගණන් සාමාන්‍ය අගයන්ගේ පූර්ණ පරමාණුව අරෙහිවල 3 කියන answer උ වෙන්නේ.

RATHNA

Scanned by TapScanner

Binary Search Tree - Implementation

Node Class

* Node contains the info an object. මේ node class එ { node එ කෙනි details සහ implementing තමා කරන්නේ.

```
class Node
{
    public int iData; // data item (මේකේ තමා අදාලේ node එ ඇතුලේ තියන value එ store කරන්නේ)
    public double dData; //other data
    public Node leftChild;
    public Node rightChild;
```

මෙතන leftChild මේ සහ rightChild කියන reference එ තෙනකින Node type එකේ pointers දෙකක් create කරනවා.

```
    public void displayNode(){
        S.o.p (iData);
        S.o.p (" , ");
        S.o.p (dData);
        ~~S.o.p~~
    }
}
```

Tree Class

* Node වලට class එකක් හදනවා වගේ tree එවන් class එකක් හදනවා.

```
class Tree
{
    private Node root; //first node of tree (මෙතනදි මුලින්ම root එට node object එකක් create කරනවා.)
```

Scanned by TapScanner

```
public Tree() {
        root = null;
}
```

//මෙතන constructor එදි root එකේ initial value එක විදිහට null දෙනවා. මොකද ආරම්භයේදි root එකේ value නැති නිසා. අපි values එකින් එක insert කරාම තමා root එකේ ඉඳන් tree එක හදන්නේ.

```
public void insert (int id , double dd) {
}

public boolean delete (int id) {
}
public Node find (int key) {
}

}
```

// මේ විදිහට මේ insert delete find method Tree class එක ඇතුලේ implement කරන්න ඕන.

Find Method

මෙතෙන්න තමා අපිට හොයන්න අවශ්‍ය number එක (key එක) දෙන්නේ.

```
public Node find (int key){
        Node current = root;
        while (current.iData != key)
        {
                if (key < current.iData)
                        current = current.leftChild;
                else
                        current = current.rightChild;
                if (current == null)
                        return null;
        }
        return current;
}
```

pointer name එක තමා current. එතකොට මුලින් initially point කරලා තියන්නේ root එට.

අපි check කරන current value එක key එට සමාන වෙනතෙක මේක repeat වෙනවා

Scanned by TapScanner

Insert Method

```
public void insert (int id, double dd) {
        Node newNode = new Node();
        newNode.iData = id;
        newnode.dData = dd;
        if (root == null) // no node in root
            root = newNode;
        else
        {
            Node current = root; //Start at root
            Node parent;
            while (true)
            {
                parent = current;
                    if (id < current.iData) //go left
                    {
                        current = current.leftChild;
                        if (current == null){
                            parent.leftChild = newNode;
                            return;
                        }
                    }
                else //go right
                {
                    current = current.rightChild;
                    if (current == null) {
                        parent.rightChild = newNode;
                        return;
                    }
                }
            }
        }
}
```

Scanned by TapScanner

Inorder Traversing Method

```
private void inOrder (Node localRoot)
{
    if (localRoot != null)
    {
        inOrder(localRoot.leftChild);
        localRoot.displayNode();
        inOrder (localRoot.rightChild);
    }
```

Preorder Traversing Method

```
private void preOrder (Node localRoot)
{
    if (localRoot != null)
    {
        localRoot.displayNode();
        preOrder (localRoot.leftChild);
        preorder (localRoot.rightChild);
    }
}
```

RATHNA

Scanned by TapScanner

-Recursion-

- Algorithms වල recursion කෙරෙන් apply වෙනවා.

- Recursion කියන්නේ අපි function 1 න් ඇතුලේ ආපහු ම function 1 ටම different parameter 1 න් එනිත call කරන එකට.

එනකොට function 1 use කරන කෙනා දෙකේම function 1 ගේ නම සමාන වෙන්නම ඕන. But parameters අවශ්‍ය 2දි වෙනස් වෙන්න ඕන. මොකද එකම parameter 1 pass කරලා function 1 call කරොත් එනකොට වෙන්නේ ම function එකම repeat වෙවි තියන එක. එතකොට function 1 stop වෙන්නේ නෑ. නමුත් recursion වලදි අනිවාර්යෙන් function 1 stop වෙන්න ඕන සහ output 1 න් එළියට දෙන්න ඕන.

∴ recursion වලදි හැමතිස්සෙම function name සමාන වෙන්න ඕන, parameters වෙනස් වෙන්න ඕන.

Eg:-
```
function (n)
{
    ________
    ________
    ________
    function (n-1);
    ________
    ________
}
```

Function Name should be same

Parameters should be different

* Factorial function is the very best example that we can demonstrate the recursion.

factorial

$5! = 5 \times 4 \times 3 \times 2 \times 1$

මෙතනදි recursive function 1 න් වෙන්නන්නේ නෑ. මොකද මෙතනදි 5! කියන එක වම් පැත්තේ තිබුණාම දකුණු පැත්තේ !(factorial) නැහැ. ඒ කියන්නේ මේක implement කරනකොට factorial කියන function 1 ඇතුලේ අපිට ආපහු factorial function 1ම වෙන්නන්න ඕ.

Scanned by TapScanner

Date

∴ recursive විදිහට අපිට මේක ලියන්න පුළුවන්.

$5! = 5 \times 4!$

මෙතනදි factorial කියන function 1 call වෙනවා different parameters එක්ක. එක්කම තමා recursion කියන්නේ.

General format:-

* $n! = n \times (n-1)!$

මෙතන ලිව්වට coding වලදි ගොඩක් use කරන්නේ පළල විදිහට.

* factorial (n) = n x factorial (n-1)

same function, with different parameters, calling itself

* left and right sides names should be exactly same.

* fact (n) = n x fact (n-1)

- factorial (n) = n x fact (n-1)

මෙතන තිබුණොත් recursion නෙවේ. recursion වලදි function name exactly same වෙන්න ඕන.

01 Recursive Relations

If we can write a relation to whatever the recursion problem to illustrate how the recursion works that equations call as "recursive relations".

Eg:- $n! = n \times (n-1)!$

factorial (n) = n x factorial (n-1)

fact (n) = n x fact (n-1)

RATHNA

Scanned by TapScanner

$5! = 5 \times 4!$

$4! = 4 \times 3!$

$3! = 3 \times 2!$

$2! = 2 \times 1!$

$1! = 1 \times 0!$

$0! = 1$ ← මෙතනදී recursion ය කි වෙනිනේ නෑ. මෙතනින් function ය නවතිනවා. ∴ මේකට කියනවා,

base/initial condition

කියලා.

* Recursion ය කරද අනිවාර්යෙන් base/initial condition ය කි තියෙන්නම ඕන. ඒක තමා recursion ය කෙ stop point ය වෙනිනේ. stop point ය කි නැතිනම් මේකෙන් output ය කි ගන්නනි බෑ. ∴ base/intial condition කියන ඒක අනිවාර්යයි recursion function ය කට.

* Recursion ය කදී අපි answer ය calculate කරනිනේ bottom ය කෙ ඉඳලා. මේකෙදී "bottom to top approach" තමා භාවිතා කරනිනේ.

$5! = 5 \times 4!$ (120, 24)

$4! = 4 \times 3!$ (24, 6)

$3! = 3 \times 2!$ (6, 2)

$2! = 2 \times 1!$ (2, 1)

$1! = 1 \times 0!$ (1, 1)

$0! = 1$ (1)

මේ විදිහට bottom to top approach ය භාවිතා කරලා තමයි answer ය ගන්නිනේ.

$5! = 120$

Scanned by TapScanner

02 Recursive Definition

Recursion 2 කට categories 2 කි තියෙනවා.

(01) Base Category (මෙකෙදි නොනන්න ඕන base value 1)

(02) Places where the recursion is applying (Recursive component category)

(මෙතනදි අපි recursion apply කරන function 2 ගේ පෙරළ නොනන්න ඕන) (right hand side of the recursion relation.)

- Recursive definition 2 දි කරන්නේ මේ categories 2 දෙකම කරන්න ඕන.

$$(n)! = \begin{cases} 1 & \text{if } n=0 \quad \text{(base)} \\ n*(n-1)! & \text{if } n>0 \end{cases}$$

recursive component category.

factorial negative නොවේ කියලා කියන්නේ n>0 කියලා නිසා

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n=0 \\ n * \text{factorial}(n-1) & \text{if } n>0 \end{cases}$$

name should be same

RATHNA

Scanned by TapScanner

03 Recursive Method

මේකෙදි කරන්නේ definition 1 අපි if-else statement 1 කට map කරන එක.

```
int factorial (int n)
  {
    if (n==0) {
        return 1; // base
      }
    else {
        return (n* factorial(n-1)); // recursive component
      }
  }
```

* function name should be same and parameters should be change.

What is recursion?

A function that calls itself directly or indirectly to solve a smaller version of its task until a final call which does not require a self-call is a recursive function.

* Recursion වලට types 2 කි තියෙනවා.
  (1) Directly recursion
  (2) Indirectly recursion

(01) Direct Recursion

මේකෙදි function 1 අතුලේ directly ම ඒ function 1 ට self call කරනවා smaller parameter 1 කි සමඟ.

Scanned by TapScanner

Eg:-

```
int factorial (int n)
{
    if (n==0)
        return 1; // base
    else
        return (n* factorial(n-1)); // recursive component
}
```

(02) Indirect Recursion

මේකෙදී අර function 1 ඇතුලේ වෙන function 1 කට call කරලා ඒ new function 1 ඇතුලේ first function 1ට call කරනවා different parameters 1 එක්ක.

Eg:-

```
f(x)                    g(x)
{                       {
  ------                  ------
  ------                  ------
  g(x);                   ------
  ------                  f(x-1);
  ------                  ------
}                       }
```

1st we call different function.

Inside this function indirectly we call the first function with different parameters.

RATHNA

Scanned by TapScanner

04 Reccurance Equation

- Mathematical function that defines the running time of recursive functions.

reccurance equation ↓ ↓ ↓ total running time ↓ recursive function ↓ ↓.

* දී recursion function ↓ ක්‍රම මේ T(n) function ↓ හොයාගන්න පුළුවන් recursive function ↓ ක් වෙනවා.

$T(n)$

Total running time

input size

Eg:- $T(N) = T(N-1) + b$

$T(N) = T(N/2) + c$

- T(n) කියන්නේ total running time ↓ කියන දී recursive function ↓ ගණිතමය කියන method body ↓ ඇතුලේ කියන statements වලට සම්පූර්ණයෙන් run වෙන්න ගතවෙන කාලය හොයන්න කියන.

Eg:-
```
int factorial(int n)  → T(n)
{
    if (n==0)  ← C1
        return 1;  ← C2
    else   keyword ↓ ක් නිසා process ↓ ක් වෙන්නේ නෑ. ∴ ඒක omit කරනවා.
        return (n * factorial(n-1));
}
```

C_3 $T(n-1)$

* මේ වගේ T(n) calculate කරද්දී අපිට simple if conditions වලට සහ simple mathematical calculation වලට executing time ↓ (c) constant ↓ ක් කියලා හරි යනවා.

Scanned by TapScanner

* function ඇතුළේ factorial කියන statement එක දෙගව බලා තමා running time යොදන්නේ.
* මෙතන factorial (n-1) නෙවෙයි running time එක constant එකක් විදිහට ගන්න පුළුවන්. මොකද මෙතන තමා recursion apply වෙන්නේ. මෙතන n ට අපි pass කරන අගය අනුව ඒ නෙවෙයි execute වෙන්න ඕනෙ වෙලාව වෙනස් වෙනවා n=5 කියන්නෙත් අපි දැක්කා. n=100 කියන්නෙත් වෙනවා.
∴ මෙතන runtime එක විදිහට ගන්නේ T(n-1).

* return කරන්න ඕන simple statement එකක් නිසා ඒකට constant එකක් ගන්න පුළුවන්.

Reccurence equation for factorial

* reccurence equation එක ලියන්නෙ අපි T(n) function එකේ recursion එක කියන්න තමයි. T(n) යි recursion function එක කියන්න නම් අපි අනිවාර්යෙන් else එක ඇතුළේ තියන factorial(n-1) statement එක execute කරන්නම ඕන.

∴ මෙතන factorial function එක ඇතුළේ else part එක අනිවාර්ය execute වෙන්න ඕන. ඒ කියන්නේ අපිට මේ reccurence equation එක ලියන්න පුළුවන් වෙන්නේ n>0 අවස්ථාවට විතරයි. මොකද n=0 කියන්නේ if part එක විතරයි execute වෙන්නේ. එතකොට කියන්නේ T(n) function එක recursion වෙන්නේ නෑ.

∴ $T(n) = C_1 + C_3 + T(n-1)$ (if එක ඇතුළ execute වෙන්නේ නැති නිසා C_2 ගන්න බැ.)

$T(n) = T(n-1) + c$ ← ($C_1 + C_3$ කියන්නෙ c constant එකක් ගන්න පුළුවන්)

Scanned by TapScanner

* Reccurrence equation 1 ගනිත්‍ය කෙසේද කියා විග්‍රහකරන්න සහ. වෙනදෙ අපි සොයන්න ඕන 'Bigo' value 1.
'Bigo' value සොයන්න methods 4 කි තියනවා.

* (01) repeated substitution / iteration method (-, /)
(02) reccurence tree method (/) } only for $T(n) = T(n-1) + c$
* (03) Master theorem (/)
(04) Substitution method (-, /) ← Both types

$T(n) = T(n-1) + c$ or $T(n) = T(n/2) + c$ both types ඉලව use කරන්න පුළුවන්

Repeated Substitution method (iteration method)

$T(n) = T(n-1) + c$
$T(n-1) = T(n-2) + c$
$T(n-2) = T(n-3) + c$
⋮
$T(3) = T(2) + c$
$T(2) = T(1) + c$
$T(1) = T(0) + c$
$T(0) = c1 + c2$

* මෙතන T(0) කියන්නේ n=0 වුණාම T value 1. එතකොට function 1 ගේ n==0 part 1 තමා execute වෙන්නේ (if part 1) එතකොට $T(0) = c1 + c2$ වලව.

$T(n) = c1 + c2 + c \cdot n$

$T(n) = c \cdot n + c4$ (c1 හා c2 වලට 1 constant 1 ක් හරින් ඉල්ලයි)

මේ ලැබෙන equation 1 කි 'BigO' value 1 සොයගන්න ඕන.

මෙතෙදි 'BigO' value 1 වෙන්නේ මේ function 1 ගේ කියන n ගෙ අගය.

$\therefore T(n) = O(n)$

T(n) function 1 ගේ 'BigO' value 1 වෙන්නේ n.

$T(n) = 3n^2 + 5n + 8n^5 - 10$
$= O(n^5)$

මේ වගේ function 1 ක "BigO' value 1 සොයන්න නිකන් හරින් ඕන අවම n value 1. function 1 T(n) කියන n ගෙ upper limit 1 තමා 'BigO' value 1ට හරින් ඕන

Scanned by TapScanner

Master Theorom (1)

මෙක apply වෙන්නේ recurence function වල divide කියන්නෙ විතරයි.

$$T(n) = aT(n/b) + f(n)$$

1st step - මෙකෙදි අපි a, b, f(n) values විතර හොයගන්න ඕන.

a- value infront of the T

b- value n is going to be devided by

f(n) - last part of the recurrence equation

Eg:- $T(n) = 2T(n/2) + cn$

$a = 2$ $f(n) = cn$

$b = 2$

2nd step - we have to do the comparison

$f(n)$ vs $n^{(\log_b a)}$

cn vs $n^{(\log_2 2)}$

cn vs n^{1} $(\because \log_2 2 = 1)$

cn vs n

මෙතන cn වල c කියන්නේ constant 2 නේ. මෙතනදි comparison වෙන්නේ n අගයයි අතර තියන c-constant 2 හොයලා නරින්න ඕනේ.

$\therefore n$ vs n

3rd step :- අපිට comparison 2 කරලා types 3 ක results ලැබෙන්න ඕනේ.

L.side < R.side ⟶ case 1

L.side = R.side ⟶ case 2

L.side > R.side ⟶ case 3

$\therefore n$ vs n කියන්නේ case 2 වල.

RATHNA

Scanned by TapScanner

~~answer = case 1 නම්,~~

~~result = case 2 නම්,~~

~~answer = $\theta(n^{\log_b a} \cdot \lg n)$~~

---

* result = case 1 නම්,

  answer $= \theta(n^{\log_b a})$

* result = case 2 නම්,

  answer $= \theta(n^{\log_b a} \cdot \lg n)$

  මෙතන lg n කියන්නේ $\log_{10}$ කියන එක. But අපි උත්තර බලද්දි lg n කියලා තියෙනවට $\log_2 n$ කියලා තමයි.

* result = case 3 නම්,

  answer $= \theta[f(n)]$

  මෙතන θ (Big θ) or O (Big O) දාන්න ඕනෙ. අපි ලබන්නේ.

∴ අපිට ලැබුණු result 1 case 2 නිසා,

answer 1 වෙන්නේ $\theta(n^{\log_b a} \cdot \lg n)$

~~answer = $n^{\log_b}$~~

~~answer = $(n^{\log_b a} \cdot \log_2 n)$~~

answer $= \Theta(n^{\log_b a} \cdot \log_2 n)$

$$T(n) = \Theta(n^{\log_b a} \cdot \log_2 n)$$
$$= \Theta(n^{\log_2 2} \cdot \log_2 n)$$
$$= \Theta(n^1 \cdot \log_2 n)$$
$$= \Theta(n \log_2 n)$$

Scanned by TapScanner

Date

Example ②

$T(n) = 9T(n/3) + n$

$a = 9$ $\quad f(n) = n$

$b = 3$

$\log_3 9 = 2$

$3^2 = 9$

$f(n)$ vs $n^{(\log_b a)}$

n vs $n^{(\log_3 9)}$

n vs n^2

$n < n^2$, $\therefore$ case 1

answer $= \Theta(n^{\log_b a})$

$= \Theta(n^{\log_3 9})$

$= \Theta(n^2)$

$f(n)$ vs $n^{(\log_b a)}$

cn^2 vs $n^{(\log_{\frac{3}{4}} 1)}$

cn^2 vs n^0

cn^2 vs 1

$O(1)$

a	b	f(n)
1	3/4	cn^2
		100

$P(n) = 10000 + 11\%$

Example ③

$T(n) = T(2n/3) + 1$

$a = 1$ $\quad f(n) = 1$

$b = 3/2$

$f(n)$ vs $n^{(\log_b a)}$

1 vs $n^{(\log_{3/2} 1)}$

1 vs

Scanned by TapScanner

- Intro. to Algorithms -

- Algorithm is any well defined computational procedure that takes some value or set of values as input and produce some value or set of values as output.

Input → Algorithm → Output

* Algorithm 1 කි කියන්නේ well defined computational procedure 1 කට. මේක හරියට method 1 කි වගේ. But method 1 කි වගේ change කරන්න ඒ. කීප දෙනෙක්ට inputs විකක් දීලා එකෙන් output 1 කි ගන්න කිව්වොත් එක එකිකෙනා use කරන method වෙනකි. එතකොට අවශ්‍ය විදිහට method change කරලා output 1 ගන්න පුළුවන්. But algorithm 1 කි එහෙම ඒ. එක හරියට constant 1 කි වගේ. free-defined කරලා කියන්නේ. කී දෙනෙක් use කරත් change වෙන්නේ ඒ. same input, same output, same procedure, same coding.

3, 1, 2, 7, 9, 8, 5, 4, 6 → Algorithm → 1, 2, 3, 4, 5, 6, 7, 8, 9

input → output

* මෙතනදී use කරලා කියන්නේ sorting algorithm 1 කි.

* Logic of this Algorithm:-

(01) Get the smallest value from the input

(02) Remove it and output

(03) Repeat above 1, 2 for remaining input until there is no item in the input.

Scanned by TapScanner

Properties of an Algorithm

- Be correct
- Be unambiguous (should be clear)
- Give the correct solution for all cases.
- Be simple
- It must terminate

Applications of an Algorithms

* we can use algorithms for many tasks.
  - Data retrieval
  - Networking routing
  - Sorting
  - Searching
  - Shortest paths in a graph

Pseudocode

* Algorithm should be clear and simple. ∴ Simple english is used to write algorithms.
* ඒ විදියට algorithms ලියන method 1 තමා "Pseudocode" කියලා හඳුන්වන්නේ.

- Method of writing down a algorithm.
- Easy to read and understand.
- Just like other programming language.
- More expressive method.
- Does not concern with the technique of software engineering.

RATHNA

Scanned by TapScanner

Pseudocode Conventions

- English
- Separate line for each instruction
- Looping constructs and conditional constructs.
- Indicate a comment line → // [◁ ← (මේකෙන් සමහර වෙලාවට use කරනවා)]
- = indicate a assignment (← මේකෙන් සමහර වෙලාවට use කරනවා)

  Eg:- int a ← 5 (මෙතනදි arrow head 1 තියෙනින ඕන variable name 1ට)

  int a=5

- **Indentation**

  * Indentation වලින් තමා code 1 කේ statements තියන විදිය (මොන block 1 කේද) කියලා අඳුනගන්නේ. මොනවගරි statements ටිකක් තවත් statement 1 ක් ඇතුලේ තමයි තියෙන්නේ ඒ statements normal statements ලියන vertical line 1ට දකුණු පැත්තට වෙන්න ලියනින ඕන (normally tab space 1 ක් කියලා) එතකොට තමා pseudocode 1 ක coding අඳුනගන්න පුළුවන් වෙන්නේ මේකට තමා indentation කියලා කියන්නේ.

Code

Eg:-
```
int a=5;
public void print()
{
    printf("Hello World!");
}
a++
```

pseudocode

```
int a=5
public void print()
    print Hello World
a++
```

(* { } , ; වගේ coding අයිකන් වෙනක pseudocode වලදි)

- **Array Elements**

  * Normally coding වලදි array elements වලට indexes දෙන්නේ 0 ඉඳන්. But pseudocode වලදි indexes දෙන්නේ 1 ඉඳන්.

  ∴ අනින pseudocode 1 ක් → code කරදදි array elements වල indexes වෙනස් වෙන්න ඕන.

Eg:-

	0	1	2	3
a =	5	7	6	10

coding

	1	2	3	4
a =	5	7	6	10

pseudocode

a[0]=5 ⟷ a[1]=5

Scanned by TapScanner

Analysis of Algorithms

* අපිට එකම task 1 කි කරන්න algorithms ගොඩක් තියෙනවා. Eg:- Sorting වලදී insertion sort, merge sort, quick sort වගේ ගොඩක් algorithms තියෙනවා.
∴ මේවා අතරින් best methods තෝරගන්න algorithms analyze කරනවා. Algorithms analyze කරන්න කලම ගොඩක් තියෙනවා.

- Memory (How much memory each algorithm have taken?)
- Logic Gates (How much logic gates it have been constructed?)
- Computational Time

  * මේ කියන්නේ Total running time 1.

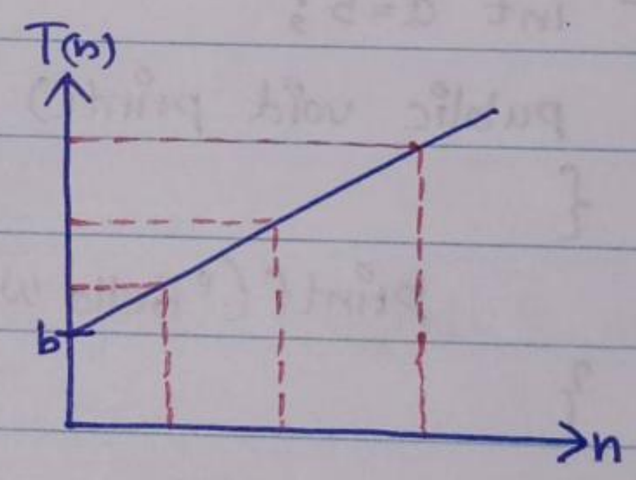
Total running time → $T(n)$ ← input size (array size)

  * Algorithm which is taken least time is the best.

* Why do we need an analysis?
  - To compare (and select the best one)
  - Predict the growth of run time.

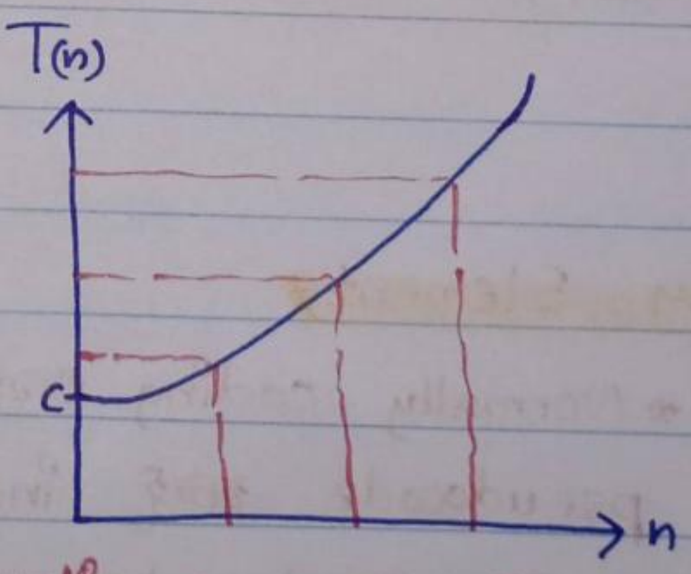
Eg:- ① $T(n) = an + b$

$y \quad mx + c$



② $T(n) = an^2 + bn + c$

$y \quad ax^2 + bx + c$



* T(n) එවම ගන්න equation 1 අනුව අපිට පුළුවන් මේ වගේ graph 1 කි අඳින්න. එතකොට ඕනම n අගයකට T(n) මේ graph 1 කි පොයාගන්න පුළුවන්. Algorithm 1 අනුව run කරන්න ඕන කෑ.

Scanned by TapScanner

Analysis Methods

* To do the analysis there are four methods.
  - Operation Count Methods
  - Step Count Method (RAM MODEL)
  - Exact Analysis
  - Asymptotic Notations
    - O big'o'
    - Ω Omega
    - Θ Theta

Operation Count

* Methods for time complexity analysis.
* Select one or more operations such as add, multiply and compare.
* Operations count considers the time spent on choosen operations but not all.

• මේකෙදි කරන්නේ algorithm 1 කෙ තියෙන හැම operation 1 කටම වෙනම time 1 calculate කරන එක. operations කිව්වම assigning, calculations print statements, conditions මේවා ඔක්කොම අයිති වෙනවා.

Eg:- int a = 5 assignment = 2ms
a = a + 5 * 2 assignment = 2ms addition = 1ms multiplication = 3ms
print a printing = 5ms

$T(n) = 2 + 3 + 5$

* මෙතන T(n) හොයද්දි ඔක්කොම operations වලට යන time 1 එකතු කරන්නේ නෑ.
  (1) Line 1 ක තියෙන්නේ 1 operation 1 නම් ඒකට යන time 1 T(n) වලට එකතු කරනවා වෙනකකි නොකර
  (2) Line 1 ක operations කීපයක් තියේනම් වැඩිම time 1 කි යන operation 1 කෙ time 1 තමා T(n) වලට එකතු කරන්නේ. මොකද එතනදි උපකල්පනය කරනවා maximum time operation 1 තමා T(n) වලට වලංගුවෙන් කරන්නේ කියලා.

Scanned by TapScanner

* නමුත් operation count method 1 practically use කරන්න විකක් අමාරුයි. මොකද operation එකින් එකට ගතවෙන කාලය අනි වෙන වෙනම හොයන්න ඕන. ඒක practical නෑ.

∴ Step count method 1 තමා වැඩිපුර T(n) හොයන්න use කරන්නේ.

Step Count (RAM model)

* මේකෙදි කරන්නේ algorithm 1 කේ තියන steps ~~[illegible]~~ වල එකතුව T(n) විදිහට ගන්න එක.

T(n) ⟶ sum of the steps

Eg:- int a = 5
a = a+5*2
print a

මේකෙදි මුලින්ම කරන්න ඕන මේකේ තියන steps ගණන හොයාගන්න එක. Step 1 කි තියෙන්නේ,

(1) All the mathematical operations
(2) All the conditional checkings
(3) All the assignments
(4) All the printings
(5) All the array accessing (A[i])
(6) All the increment and decrement operators

මේවා ඔක්කොම steps විදිහට ගණනය කරනවා.

int a=5 ⟶ step 1
a = a+5*2 ⟶ steps 3
print a ⟶ step 1

T(n) = sum of Steps
= 1+3+1 = 5

RATHNA

Scanned by TapScanner

Eg ② :- int a=0
for i=0 to 5
print i

* මේ වගේ එකක for loop 1 ඇතුලේ steps ගොඩක් තියෙනව. for i=0 to 5 කියන එක for (i=0; i<=5; i++) ලෙසයි implement වෙන්නේ එතකොට මේකේ assignment 1 කි, checking 1 කි සහ increment 1 කි තියෙනවා. ඒ විතරක් වගේම for loop 1 ඇතුලේ code 1 කි කීපවතාවක් run වෙනවා.

Assignment	checking	increments
i=0	0<=5 → T	i=1
i=1	1<=5 → T	i=2
i=2	2<=5 → T	i=3
i=3	3<=5 → T	i=4
i=4	4<=5 → T	i=5
i=5	5<=5 → T	i=6
i=6	6<=5 → F	

* ඒ අනුව increment 6 කරන් repeat වෙනවා. checking 7 කරන් repeat වෙනවා. assignment 7 කරන් repeat වෙනවා. for 1 ඇතුලේ print statement 1 6 කරන් execute වෙනවා.

∴ int a=0 → step 1
for i=0 to 5 → assigmnt=7 checking=7 inc=6
print i → 1*6

∴ T(n) = 1+ 7+7+6+6
= 27

* මේ for loop 1 කෙනි steps ගණන ගන්න හොඳි equation 2 කි තියෙනවා.

final value of for loop

assignments = checkings = final − first + 2

first value of for loop.

increments = no. of assignments − 1

inner statements = no. of assignments − 1

Eg:- final − first + 2 = 5 − 0 + 2 = 7

∴ assign = check = 7

increments = 7 − 1 = 6

print statement = 6

Eg③:- int j = 1 ⟶ 1 step

for i = 1 to n ⟶ inc = n , checkings = n+1 , assignments = n+1

print i ⟶ 1 * n steps

A[i] = A[i] + i ⟶ මෙතන line 1 කෙනි steps 4 කි තියෙනවා.

① ② ③ ④

n වතාවක් iterate වෙන නිසා,

4 * n steps

* final − first + 2 = n − 1 + 2

= n + 1

∴ T(n) = 1 + n + (n+1) + (n+1) + n + 4n

= 8n + 3

RATHNA

Scanned by TapScanner

Eg ④ :- int j=0 ⟶ step 1
while (j<=5) ⟶ step 7
print j ⟶ step 6*6
j=j+1 ⟶ step 2*6
① ②

*while loop ↳ for loop ↳ use කරපු equation 1 ම හදන්න පු.

$T(n) = 1+7+6+12$
$= 26$

Problems with ram model

- Differ number of steps with different architecture.
  Eg:- sum = sum + A[i] is a one step in the CISC processor.

- It is difficult to count the exact number of steps in the algorithm.

Insertion sort

* Insertion sort algorithm ↳ කරන්නේ අපි දෙන inputs විකක් [array ↳ ක්] acsending order විදියට output ↳ දෙනවා.

* මේකේ pseudocode ↳ අනුව illustrations ඇඳලා අපිට පුළුවන් මේ algorithm ↳ exactly analyze කරන්න.

Eg:- A

1	2	3	4	5
5	2	1	0	8

මේ A කියන array ↳ අපි insertion sort algorithm ↳ කෙනේ pseudocode ↳ ට pass කරලා ඒකට අදාලව තමා illustrations අඳිනවා. මෙතන array indexes 1→ඉඳලා පටන්ගෙන තියෙනේ pseudocode වලදී array indexes 1→n වෙනකම් ගන්න නිසා.

Scanned by TapScanner

* illustration 1 කි කියනේනේ අපිට pseudocode 1 කේ කියන හොඳ පද වෙනුවට අපේ inputs අනුව values ආදේශ කරලා අඳින කටයුත්තම.

* Insertion sort 1 කේ කියනේනේ for loop 1 කි කිහා කිකේ හැම iteration 1 කටම illustration අඳිනීන ඕන.
ඒ වගේම හැම iteration 1 කි ඉවරවුණාම හැම illustration 1 කටම තියේනේ array 1 update වෙලාද කියලා බලලා ඒ updated array 1 ඇඳගන්න ඕන. ඊළඟ illustration 1ට use කරනීනේ ඒ updated array 1.

Pseudocode for insertion sort

1	2	3	4	5
5	2	1	0	8

insertion-sort (A) ← මේ header 1 . මේකට A array 1 pass කරනවා කියලා වෙනත කියනීනේ.

for j=2 to 5

1 for j=2 to A.length ← 1st iteration 1 j=2 කිහා මේ illustration 1 අඳිනීනේ j=2 වලට.

key = A[2] = 2

2 key = A[j]

3 //insert A[j] into the sorted sequence A [1...j-1]

i = 2-1 = 1

4 i = j-1

While 1>0→T and 5>2 (මෙතන conditions 2ම true කිහා while loop 1 ඇතුලට කිහන.)

5 While i>0 and A[i] > key → while 0>0 →F

A [2] = 5

6 A [i+1] = A[i]

i = 0

7 i = i-1

8 A [i+1] = key

A [1] = 2

while loop 1කේ දෙවෙනි iteration 1 ඉවර වුණාම ආපහු line 5 ඇවිල්ලා බලනවා updated values තිකිත while condition true වෙනවද කියලා. මෙතනදී තිකෙම නැති කිහා line 8 execute වෙනවා.

* මේක තමා 1st illustration 1 . (j=2 iteration 1ට)
* ඊළඟට තියෙනේ array 1 update වෙලාද බලලා updated array 1 ඇඳගන්න ඕන.

Scanned by TapScanner

* updated array 1 අඳිනින නම් array indexes, values වෙනස්වෙලාද බලන්න ඕන. ඒකට අපිලා illustration 1 මුල ඉඳලා check කරගෙන එන්න ඕන. illustration 1 කේ array indexes වලට values assign වෙලා තියෙනවනම් ඒක තමා updated array 1 අඳිනින ඕන වෙන්නේ.

	1	2	3	4	5
A	2	5	1	0	8

according to line 8, A[1]=2

according to line 6, A[2]=5

මේවගේ වෙනස්කම් වෙන්නෑ. ∴ මේ විදිහටම array 1ට දානවා

Logic of the insertion sort

* insertion sort 1 ආරම්භකිරීමේ සලකන්නේ 2nd index 1 කි. (for j=2...)

1st iteration :-

* මේකෙදි සලකන්නේ j=2 අවස්ථාව. මේ කියන්නේ අපි array 1 කේ A[1] සහ A[2] values 2 විතරයි සලකන්නේ.

* key value 1 = 2 කරලා ඊට පස්සේ A[1] සහ A[2] 2 කේ තියන values 2 acending order 1ට දානවා.

* මේ විදිහට ඊළඟට j=3 iteration 1දී සලකන්නේ 1st values 3 විතරයි. මේ 3 acending order 1ට දානවා. ඊළඟට iteration 1 දී j=4 වල value 1 මේවගේ එකක compare කරලා මේ values 4 acending order 1ට දානවා.

* මේ විදිහට තමා logic 1 යන්නේ.

1st iteration

1	2	3	4	5
5	2	1	0	8

→

1	2	3	4	5
2	5	1	0	8

2nd iteration

1	2	3	4	5
2	5	1	0	8

→

1	2	3	4	5
1	2	5	0	8

Scanned by TapScanner

Date

* Exact analysis කරන්න එක විදියකින් තමා මේ විදිහට illustrations ඇඳලා සම්පූර්ණ algorithm process එකම analyze කරන එක.

* නමුත් එක practically කරන්න අමාරු නිසා exact analysis කරන්න mathematical way 2 කින් තියනවා.

Exact Analysis of Insertion sort

Insertion-sort steps	Cost	Times (No. of times relevent code line executing)
1	c_1	n
2	c_2	$n-1$
3	0	$n-1$
4	c_4	$n-1$
5	c_5	$\sum_{j=2}^{n} t_j$
6	c_6	$\sum_{j=2}^{n} (t_j-1)$
7	c_7	$\sum_{j=2}^{n} (t_j-1)$
8	c_8	$n-1$

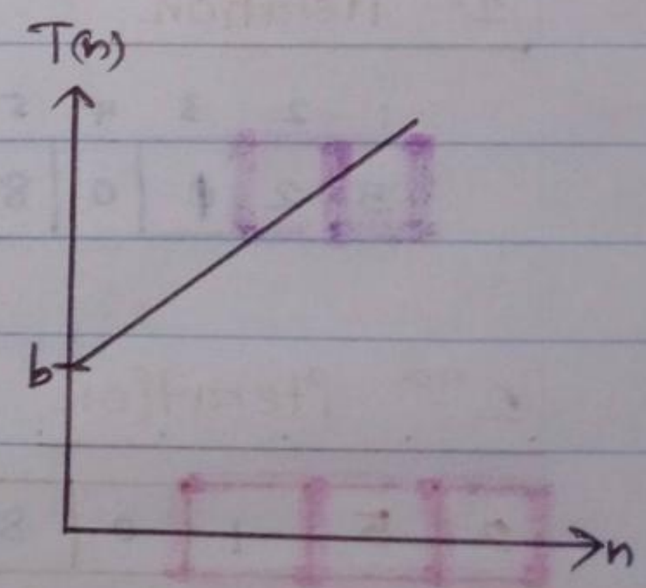
Running time :-

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j-1) + c_7 \sum_{j=2}^{n} (t_j-1) + c_8(n-1)$$

Best case and Worst case

* Best case කියන්නේ Total run time 1 අඩුවෙන්ම යන අවස්ථාවට. Insertion-sort 1 කදී T(n) අඩුම වෙන්නේ inputs ටිකක් ascending order 1ටම ඉන්නකොට.

∴ Best Case (Arrays in sorted order)



$$\boxed{T(n) \rightarrow an+b}$$

$y = mx + c$

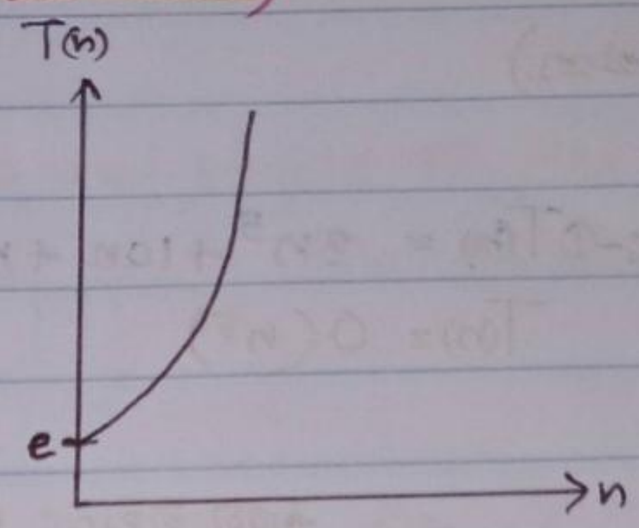
$T(n) = O(n) \leftarrow$ Big O value 1 වගේගන n

* Worst case කියන්නේ T(n) වැඩියෙන්ම යන අවස්ථාවට. Insertion-sort උදා T(n) වැඩිතරම යන්නේ inputs decending order ට දුන්නම.

∴ Worst Case (Array is in reverse sorted order)
descending order

$$\boxed{T(n) \longrightarrow cn^2 + dn + e}$$

$$y = ax^2 + bx + c$$



$T(n) = O(n^2)$ ← Big 'O' value උ වෙන්නේ n^2

*** Normally insertion-sort use කරන්නේ inputs ගණන (n) අඩු arrays වලට. නොඳද n වැඩිවුණොත් T(n) ගොඩක් වැඩිවෙනවා. එතකොට මේ algorithm උ use කරන එක efficient නැහැ.

Asymptotic Notations

- RAM model has some problems
- Exact analysis is very complicated.

Therefore we move to asymptotic notation

* මේ asymptotic notations use කරන්නේ inputs ගොඩක් තියන අවස්ථාවලදී විතරයි. (Sufficiently large size of n)

* There are three notations

O - Notation

θ - Notation

Ω - Notation

RATHNA

Scanned by TapScanner

Date / /

Big O - Notation

* We use Big-O notation to give an upper bound on a function.
(Big o value 1 ක් කරන්නේ function 1 ගේ upper bound 1 හොයන එක.)

Eg:- ① $T(n) = 3n^5 + 10n + n^2$

$T(n) = O(n^5)$

* මේ විදිය equation 1 ක තමි n ගේ වැඩිම බලය තියන එක තමා Big "O" value 1 වෙන්නේ.

② $T(n) = n \log n + n^2 + 5n + n!$

$T(n) = O(\quad)$

* මේ වගේ n ගේ බල වලට අමතරව තවත් දෙ තියනවනම් Big 'O' value 1 ගන්න upper bound 1 බලන්න ඕන.
* upper bounds තියන විදිවෙලක් තියනවා.

$(2^n, 3^n, 4^n, 5^n \ldots$ set 1 ම අධිකයි$)$

$$1 < \log n < n < n\log n < n^2 < 2^n < n! < n^n$$

$(n^2, n^3, n^4, n^5 \ldots$ set 1 ම අධිකයි$)$

$\therefore T(n) = O(n!)$ //

Scanned by TapScanner

③ Assignment (s=1)
Addition (s+1)
Multiplication (s*2)
Comparison (s<10)

*කෙලින්ම T(n) function 1 නෑනි වෙනුවට දුන්නත් Big-O value 1 නොයන්න පුළුවන් වෙන්න ඕන.

*මේ වගේ simple calculations වලට සහ simple if-else conditions වලට ගතවෙන කාලය [T(n)] නියතයක් කියලා ගන්න පුළුවන්.

∴ මෙතන, T(n) = C වෙනවා.

∴ Big-O value 1 ක් නියතයක් වෙනවා.

Big-O value නියතයක් කියලා යන්නේ O(1) ඒලා.

∴ T(n) = O(1)

④ for i=1 to n ← outer for loop 1 n වතාවක් iterate වෙනවා.
for j=1 to i ← inner for loop 1 $\underbrace{n \times n}_{n^2}$ වතාවක් iterate වෙනවා.
print j

∴ T(n) = O(n^2)

*මෙතන for loops වල n,i වෙනුවට 5,10 වගේ numbers තිබුණනම් මෙතන T(n) = C (නියතයක්.)
එනෙමක් නම් T(n) = O(1) වෙනවා.

Scanned by TapScanner

Ω - Notation

- Provides the lower bound of the function. Ω (ඔමෙගා) notation 1 කි කියන්නේ function 1 ගේ lower bound 1.

Eg:- $T(n) = 3n^5 + 10n + n^2$

$T(n) = O(n^5)$

$T(n) = \Omega(n^5)$

* මෙතනදී Ω value 1 n කියලා නිතෙන්න පුළුවන්. But එතෙම වෙන්නේ නෑ. මේවගේ T(n) function වල big-O value = Ω value
* ∴ ඕනම T(n) function එකක big-O value එකම Ω value 1 විදියට ගන්න ඕන අනිවා.

Θ - Notation

* මේකෙන් ගන්නේ function 1 ගේ average value 1 වගේ දෙයක්.
* In here also,

Θ value = big-O value

$\therefore T(n) = 3n^5 + 10 + n^2$

$T(n) = O(n^5)$

$= \Omega(n^5)$

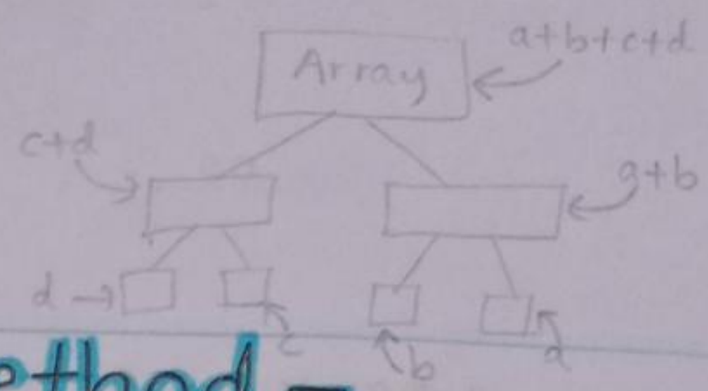
$= \text{Theta}(n^5)$

$\boxed{\text{big-O value} = \Omega \text{ value} = \Theta \text{ value}}$

RATHNA

Scanned by TapScanner

Date: 01



- Divider & Conquer Method -

Divide and Conquer

* Divide and conquer කියන්නේ method 1 කි. මේ method 1 use කරලා තමා සමහර algorithms implement කරන්නේ.
* Divide and conquer method has three steps.
  (1) Divide - Break the problem into sub problem recursively
  1st step 1 විදියට කරන්නේ divide කරන එක. මෙතනදී array 1 කි නම් තවදුරටත් divide කරන්න බැරි වෙනකම්ම sub parts වලට divide කරන්න ඕන. ඒ කියන්නේ index 1 කි වෙනකම්ම array 1 කි නම් divide කරනවා.
  (2) Conquer - Solve each such problems
  'ර'ට තේරෙන්නේ ඒ divide කරපු sub parts එක එක වෙනවෙනම විසඳනවා.
  (3) Combine - All the solutions of sub problems are combined to get the solutions of the original problem.
  මේ step 1 දී කරන්නේ original problem 1 කේ answer 1 ගන්න sub problems වල answers ඔක්කොම combine කරන එක.

Applications

* Divide and conquer method 1 use කරලා algorithms 2 කි develop කරලා.
  (01) Quick Sort
  - More work on divide phase
  - Less work for others.
  (02) Merge Sort
  - Vice versa of Quick sort.

* මේ algorithms 2 දී divide & conquer method 1 use කරන විදිය වෙනස්. Divide & conquer method 1 use කරනවා කියන්නේ අනිවාර්යෙන් divide, conquer, combine කියන parts 3 ම apply කරන්න ඕන.

Scanned by TapScanner

Date: 02

* එනකොට මේ 2 කියන difference 1 තමා,

• Quick sort 1දී divide part 1 difficult. But combining part 1 simple

• Merge sort 1දී divide part 1 simple. But combining part 1 too difficult.

QS	MS
↑ Div	Div ↓
Con	Con
↓ Com	Com ↑

Quick Sort

```
QUICKSORT (A,p,r)
if p<r
    q= PARTITION (A,p,r)
    QUICKSORT (A,p,q-1)
    QUICKSORT (A,q+1,r)
```

```
PARTITION (A,p,r)
x = A(r)
i = p-1
for j=p to r-1
    if A[j] ≤ x
        i=i+1
        exchange A[i] with A[j]
exchange A[i+1] with A[r]
return i+1
```

* Quicksort 1දී quicksort algorithm 1ට අමතරව partition algorithm 1ක් තියෙනවා. මොකද quicksort algorithm 1 ඇතුලෙදී ~~අපි~~ අපි partition algorithm 1ට call කරනවා. partition 1ක් return කරන value 1 variable 1 කට ගන්නවා.

* මේ වගේම quicksort කියන්නේ recursive algorithm 1 කි. මොකද quicksort 1 ඇතුලේ අනෙකුත් quicksort algorithm 1 ම different parameters වලින් call කරන නිසා.

Scanned by TapScanner

Date: 03

* Quicksort 1ක්‍රි dividing part 1 difficult. ∴ නමයි partition() කියලා වෙනම algorithm 1 ක් තමයි dividing part 1 කරන්නේ.

Illustration

	1	2	3	4	5	6	7	8
A	2	8	7	1	3	5	6	4

*මේ Array 1ට තමා illustration 1 කරන්නේ.

Quicksort(A, 1, 8)

QUICKSORT (A, p, r)
- A → Array name
- p → 1st index (not value)
- r → Last index (not value)

if 1<8 → True

1 if p<r

q = partition (A, 1, 8)

2 q = PARTITION (A, p, r)
(q = 4)

Quicksort (A, 1, 3)

3 QUICKSORT (A, p, q-1)

Quicksort (A, 5, 8)

4 QUICKSORT (A, q+1, r)

partition (A, 1, 8)

PARTITION (A, p, r)

x = A[8] = 4

1 x = A[r] — pivot element

i = 1-1 = 0

2 i = p-1

for j = 1 to 7

3 for j = p to r-1

if A[2] ≤ 4 → 8 ≤ 4 → False

if A[1] ≤ 4 → 2 ≤ 4 → True

4 if A[j] ≤ x

i = 0+1 = 1

5 i = i+1

exchange A[1] with A[1]

6 exchange A[i] with A[j]

exchange A[4] with A[8] (8 ⇄ 4)

7 exchange A[i+1] with A[r]

return 4

8 return i+1

*මෙතන අන්තිමටම for loop 1ගේ වෙනම iteration ඔක්කොම සවස execute වෙන වෙලේ දෙන

*මේ q කියන නමකි කියනවා q is called as "partition index".

මේ කියන්නේ array 1 partition වෙන index 1. (not value)

මේ example එකේ විදිහට නම් index 4.

1	2	3	4	5	6	7	8
2	1	3	4	7	5	6	8

partition algorithm 1 මුලදී සහ execute වුණාට පස්සේ ලැබෙන final array 1 තමා මේක.

***මේවගේ එක iteration 1 කිහිප සැරෙක් exchange 1 කි කියනවනම් ඊළඟ iteration 1ට use කරන්න ඕන ඒ exchange වුණු new array 1.
∴ exchange 1 කි වුණාට පස්සේ new array 1 [illegible]

Scanned by TapScanner

*** Special:-

අපිට quiz 1 කදී illustration අඳින්න පුළුවන් වන්න ඕනෙ. කලින් eg: 1 කේ අපි එකකබකි illustration 1 අඳපු නිසා අපිට quicksort 1 කේ recursive steps වලට illustration අඳින්න බැ. (Line 3, Line 4) ∴ මෙතනදී next line විතර diagramatically explain කරන්න වෙනවා. Logic 1 diagramatically apply කරලා තේරුන්න ඕන.

1

2

3 QuickSort (A, p, q-1) [A, 1, 3]

1	2	3
2	1	3

pivot = 3

<3	3	>3

මෙතනදී මේ array 1 කේ >3 values තුනි වුණාම diagram 1 කේ මේ විදිහට අඳින්න ඕන. මොකද logic 1 කේ තියෙන්නේ මේ විදිහට නිසා

4 QuickSort (A, q+1, r) [A, 5, 8]

5	6	7	8
7	5	6	8

pivot = 8

<8	8	>8

Logic

* Partition function 1 implement කරනකොට අපි element 1 කි තේරගන්නවා pivot element කියලා. එතකේ notation 1 → x.
* මේ pivot element 1ට අපිට හැම array element 1කම ගන්න බැ. එතකොට 1st array element (A[1]) or last array element 1 වෙන්න ඕන.

*** pivot element 1 ගන්න විදිහ අනුව partition algorithms 2 කි තියෙනවා. 1st element 1 ගන්න එකේ implementation වල තමයි වෙනස්කමක් තියෙනවා (tutes). Last element 1 ගන්න එක තමා මෙතනදී (lec) implement කරන්නේ.

∴ Quicksort has 2 different illustrations

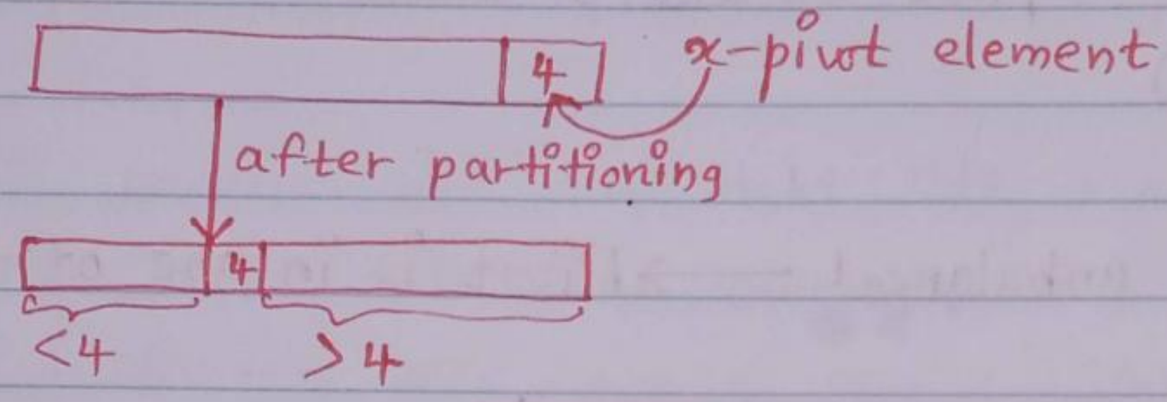
Scanned by TapScanner

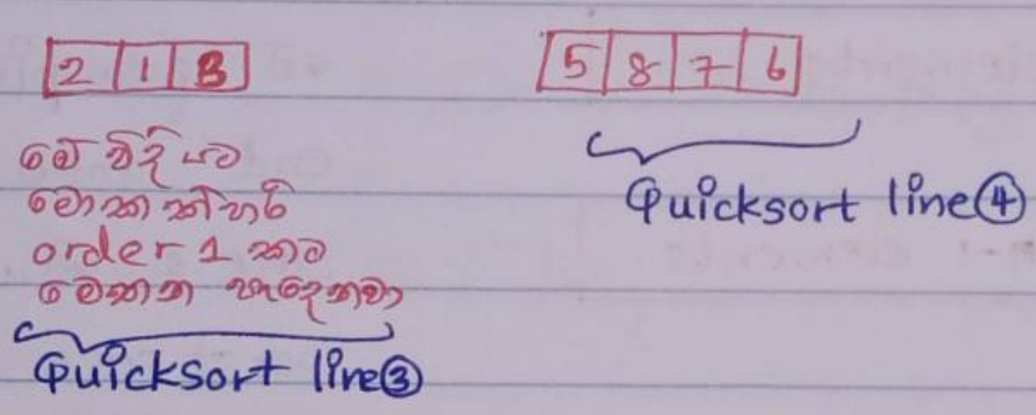
Date: 05

* එතකොට මේ partition algorithm 1 implement කරලා ඉවර වෙද්දි අපි මේ ගන්න pivot element 1 අපේ array 1 කේ මැද නොවෙන්න index 1 කට එන්න. exctly middle නෙවෙයි. Somewhere in the middle. It may be exactly middle or not.

* එතකදී pivot 1 ට leftside 1 ට ඇවිල්ලා තියෙන්නේ pivot element 1 ට වඩා අඩු අගයනේ. Right side 1 ට එන්නේ pivot 1 ට වඩා වැඩි අගයනේ.

Eg:-





* ඊට පස්සේ මේ pivot 1 නි වෙන වෙනම sub array parts වලටත් quicksort 1 ආපහු apply වෙනවා. ∴ තමයි quicksort algorithm 1 ඇතුලේ recursion 1 කි තියෙන්නේ. (Line 3 and 4)

* මේ විදිහට ආපහු ආපහු quicksort 1 apply වෙනවා. මේ විදිහ විසඳෙනකම් (most of the time තනි element 1 කට වගේ යනකම්)

Scanned by TapScanner

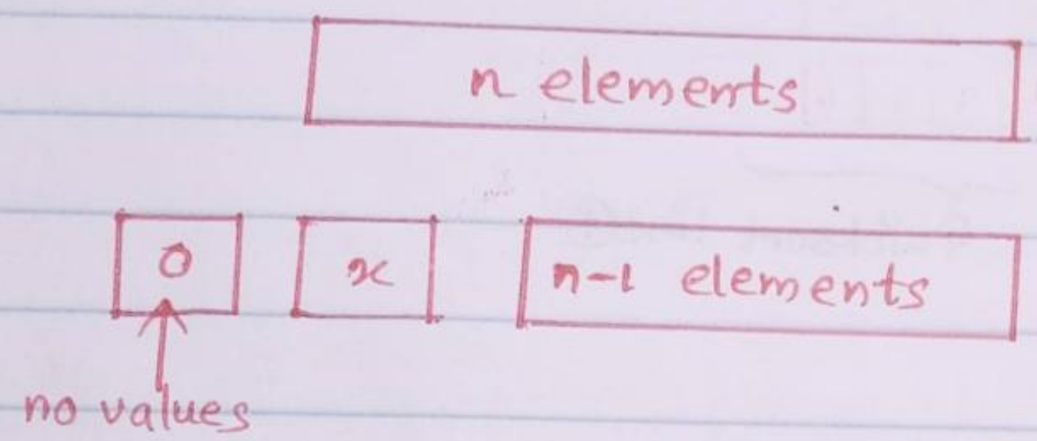
Analysis of Quick Sort

Worst Case

$$\boxed{T(n) = O(n^2)}$$

* Quick sort ද්‍රැස්ගැනීම worst case , best case බලන්නේ pivot element ද්‍රැස්ගැනීම තුල අනුව.
* Worst case ද්‍රැස්ගැනීම pivot element ද්‍රැස්ගැනීමේ array ද්‍රැස්ගැනීමේ මුල හෝ අග (one of the end)

** Partition is unbalanced ⟶ Pivot is in one of the end



මේ විදිහට pivot ද්‍රැස්ගැනීම one of the end අග්‍රයේ වෙන්න worst case ද්‍රැස්ගැනීම. මෙතනදී 'b' ලබාගන්න worst case ද්‍රැස්ගැනීම ගන්නවෙන්න Total run time ද්‍රැස්ගැනීම බිඳිනවා. 'b' ගැසීමේ දී T(n) අනුව O (big 'o') value ද්‍රැස්ගැනීම බිඳිනවා.

**

Quicksort (A, p, r) ⟶ T(n)

1 if p<r ⟶ C_1
2 q = partition (A, p, r) ⟶ $C_2 \cdot n$
3 Quicksort (A, p, q-1) ⟶ T(0)
4 Quicksort (A, q+1, r) ⟶ T(n-1)

*මෙතනදී Quicksort ද්‍රැස්ගැනීමේ run time ද්‍රැස්ගැනීම T(n) කියලා ගණිනවා. (n elements නිසා)

① if කියනකොට මෙතනදී simple condition ද්‍රැස්ගැනීම. ∴ ඒකට Time ද්‍රැස්ගැනීම constant(C_1) කියලා ගණිනවා.

② මෙතනදී partition කියලා වෙන algorithm ද්‍රැස්ගැනීම execute වෙන නිසා ඒක complex. ∴ මෙතනදී constant ගන්න බෑ. මේ partition algorithm ද්‍රැස්ගැනීම execute වෙන්න ගතවෙන කාලය array ද්‍රැස්ගැනීමේ size අනුව වෙනස් වෙනවා. ඒ නිසා array size ද්‍රැස්ගැනීම විශාලයි නම් for loops iterations වැඩි, if checkings වැඩිවෙන නිසා run time ද්‍රැස්ගැනීම වැඩියි. Size ද්‍රැස්ගැනීම අඩුයි නම් run time ද්‍රැස්ගැනීම අඩුයි.

parameters ඔක්කොම [illegible]

Scanned by TapScanner

$\therefore$ Partition time $\propto n$ අනුලෝමව සමානුපාතිකයි. මේ proportional mark ද අධික කරනිත පුද්ගලයි n නියතයකින් දක්ක හැක.

Partition time $= C_2 \cdot n$

③ Line ③ ද දී execute වෙන්නේ partition වලින් පස්සේ වැටෙන sub arrays වලින් left side ද. worst case ද දී වෙනදී left side ද හිස් නම් නිසා සහනදී Time = 0 වෙනවා. $\therefore T(0)$

④ Line ④ කෙන්දී partition වලින් පස්සේ right side ද තමා execute වෙන්නේ. සහනෙට වෙනදී right side ද elements $(n-1)$ නියතවා, Quicksort ද දී සහනෙන්ම n elements වලට ගතවෙන කාලය $T(n)$ නම් $(n-1)$ elements වලට ගතවෙන කාලය $T(n-1)$ කියලා ගන්නවා.

** $T(n) = C_1 + C_2 \cdot n + T(0) + T(n-1)$

*වෙනදී $T(0)$ අධික කරන්න ප්‍රළාන. ම වගේම වෙන quicksort recursive algorithm ද කි නිසා එහි $T(n)$ function equation ද recurance ~~function~~ ද වෙනවා.

$\therefore$ වෙනදී මේක විසඳන්න recurance equation විසඳන methods වලින් යමක් use කරන්න වෙනවා.

වෙනදී recursive කොටස් නියන්නේ $T(n-1)$. substraction ද නිසා repeated substitution method ද use කරනවා.

$T(n) = C_2 n + T(n-1)$

$T(n) = T(n-1) + C_2 n$ (repeated substitution method)

$T(n-1) = T(n-2) + C_2(n-1)$

$T(n-2) = T(n-3) + C_2(n-2)$

$\vdots$

$T(2) = T(1) + C_2 \cdot 2$

$T(1) = T(0) + C_2 \cdot 1$

$T(0) = 0$

$T(n) = C_2 n + C_2(n-1) + C_2(n-2) + \cdots + C_2 \cdot 2 + C_2 \cdot 1$

$T(n) = C_2 \underbrace{[n + (n-1) + (n-2) + \cdots + 2 + 1]}$

වෙනත් නියමයන් සමාන්තර ශ්‍රේණියක් $\therefore$ මේ වගේ sum ද ගන්න සමාන්තර ශ්‍රේණිවල sum ද ගන්න equation ද use කරනවා.

මේ වගේම reccurance equation සහනදී only one constant ද නිසේනම් එහි කොපමණ නිසන පුළුවන් $\therefore C_1$ omit කරනවා.

Scanned by TapScanner

Date: 08

$$\underbrace{1+2+\cdots+(n-2)+(n-1)+n}_{\text{Arithmetic series}}$$

n-number of terms (ඊලඟ ගණන

a- first term

L- last term

$$S(n) = \frac{n}{2}(a+L)$$

$$= \frac{n}{2}(1+n)$$

$$= \frac{(1+n)n}{2} = \frac{n}{2} + \frac{n^2}{2}$$

** $T(n) = C_2\left[\underbrace{1+2+\cdots+(n-2)+(n-1)+n}\right]$

$T(n) = C_2\left(\frac{n}{2} + \frac{n^2}{2}\right)$ ← මෙතනදී big 'o' value

input parameter

1 නොවන ඕන input parameter 2ගේ. මෙතන input parameter 2 → n කියන අගේ check කරලා ලන්න ඕන T(n) equation n කියන places වල.

එතකොට upper limit 2 කලා big 'o' value 2 ඔන්න,

** Big 'o' value,

$$T(n) = O(n^2)$$

Same as insertion sort

Scanned by TapScanner

Date: 09

Best Case

$$\boxed{T(n) = O(n \log_2 n)}$$

** Best case 1ද pivot element 1 මැද තැන (excatly middle) තියෙන්න ඕන.

[n elements]

[$\frac{(n-1)}{2}$ elements] [x] [$\frac{(n-1)}{2}$ elements]

** Quicksort (A, p, r) $\longrightarrow T(n)$

1 if $p<r \longrightarrow C_1$

2 q = partition (A, p, r) $\longrightarrow C_2 \cdot n$

3 quicksort (A, p, q-1) $\longrightarrow T\left(\frac{n-1}{2}\right)$

4 quicksort (A, q+1, r) $\longrightarrow T\left(\frac{n-1}{2}\right)$

~~** $T(n) = C_1 + C_2 \cdot n + T\ n+1$~~

** $T(n) = C_1 + C_2 \cdot n + T\left(\frac{(n-1)}{2}\right) + T\left(\frac{(n-1)}{2}\right)$

$T(n) = C_2 n + 2T\left(\frac{(n-1)}{2}\right)$

$T(n) = C_2 n + 2T\left(\frac{n}{2} - \frac{1}{2}\right)$

$T(n) = C_2 n + 2T\left(\frac{n}{2}\right)$

$T(n) = 2T(n/2) + C_2 n$

Asymptotic notation use කරලා අපි මේ big 'O' value හොයනකොට elements ගොඩක් තියන අවස්ථා වලදී මේ තියන්නෙ මොන තේ n අගය හොයන විශාල අගයක්. ∴ $\frac{n}{2}$ තියන්නෙත් හොයන විශාල අගයක්. එතකොට $\frac{1}{2}$ ක් අඩුකරන තියන්නෙ හොයලා ඉවරන ඉන්න මුල්ම දෙ පරක්. ∴ මොකද $-\frac{1}{2}$ හොයලා ඉවරක

මොකද recursive ඇල්ගෝ තියන්නෙ division 2ක් නිසා master theorem use කරලා තමා big 'O' value 1 හොයන්නේ.

Scanned by TapScanner

$T(n) = 2T(\frac{n}{2}) + C_2 n \rightarrow$ master theorem

$\left[T(n) = a\,T(\frac{n}{b}) + f(n) \right]$

1st step $\rightarrow$ $a = 2$ $b = 2$ $f(n) = C_2 n$

$2^{nd} \longrightarrow$ compare $f(n)$ and $n^{(\log_b a)}$

$f(n)$ vs $n^{(\log_b a)}$

$C_2 n$ vs $n^{(\log_2 2)}$

$C_2 n$ vs n^1

$C_2 n$ vs n

$\therefore C_2$ is constant. $\therefore$ we can omit it.

$\therefore n$ vs n

[if L.side = r.side $\rightarrow$ case 2]

case 2 answer is, $\Theta(n^{\log_b a} \cdot \log_2 n)$

$\therefore T(n) = \Theta(n \cdot \log_2 n)$

** Big 'O' value,

$T(n) = O(n \log_2 n)$

Scanned by TapScanner

Date: 01

- Heap Sort Algorithm -

Heaps

* Heap is an array object that can be viewed as a complete binary tree. There are two kinds of heaps,
  (1) max heaps
  (2) min heaps

* In both case, values in the nodes satisfy Heap property which depend on the kind of heap.

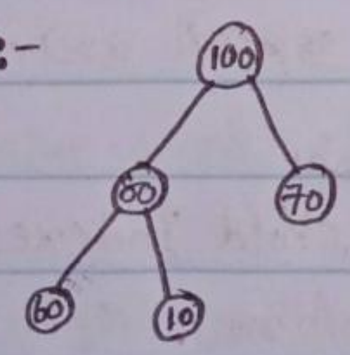
Heap 1 ක් වෙනින නම් අවශ්‍යතා 2 ක් සපුරනින ඕන.
(1) Heap 1 ක් complete binary tree 1 ක් වෙනින ඕන.
(2) Heap property 1 satisfy කරනින ඕන.

Heaps වර්ග 2 ක් තියනවා. මේවට අදාල heap properties 2 ක් තියනවා. ඒනකොට heap 1 ක් මේ property 1 ක් ඒකක CBT 1 ක් වෙනින ඕන.

- max-heap ⟶ max-heap property

The value of each node is greater than or equal to those of its children.
(max-heap 1 දී අනිවාර්යෙන් parent nodes values greater than or equal වෙනවා child nodes values වලට.)

Eg:-



CBT ✓

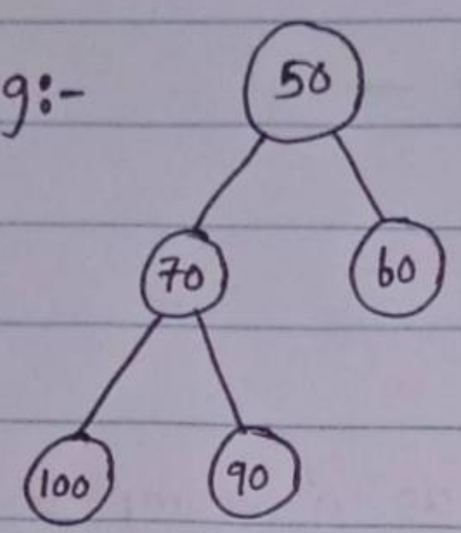
- min-heap ⟶ min-heap property

The value of each node is less than or equal to those of its children. (මෙකෙදී අනිවාර්යෙන් parent nodes values less than or equal වෙනවා child nodes values වලට)

Scanned by TapScanner

Date: 02

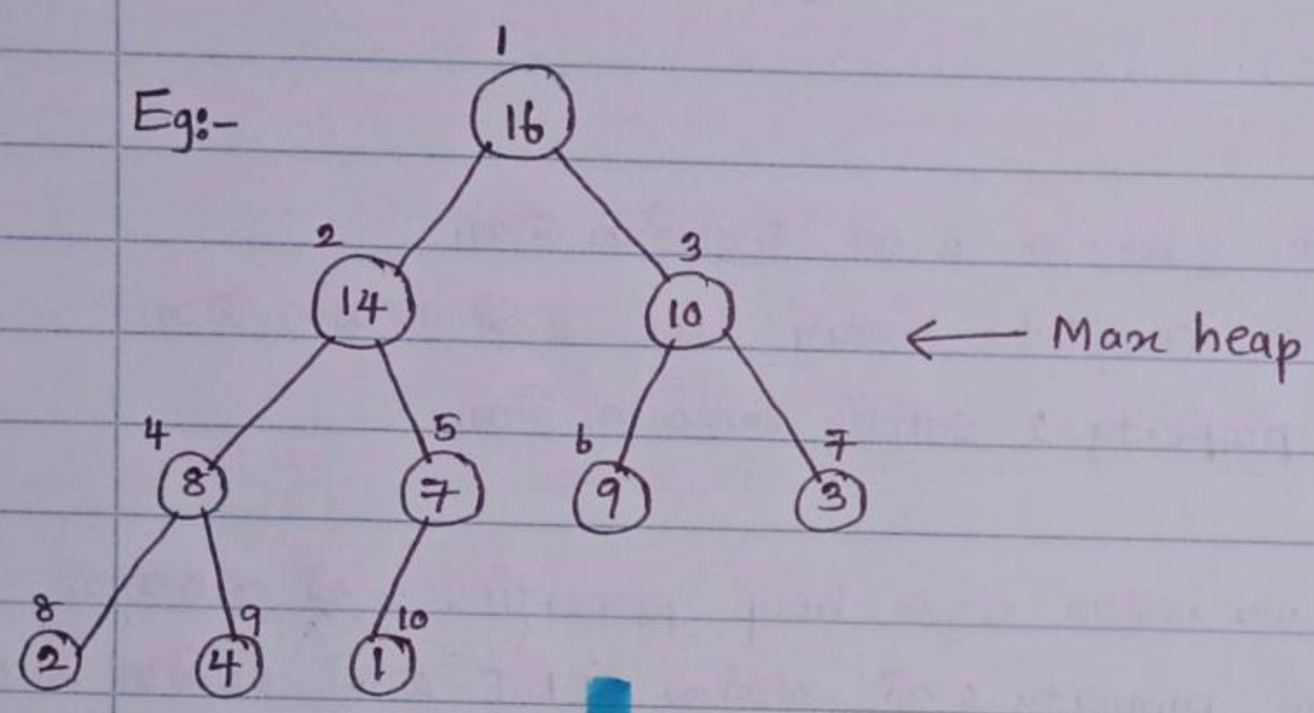
Eg:-



CBT ✓

* අපි heap 1 කි මේ වගේ diagram 1 ක දෙන්නවම computational task වල මෙක array 1 කි විදියට represent කරන්නේ.
* A heap can be represented in a one-dimensional array.

Eg:-



← Max heap

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

← Array

* Max heap 1 ඉ අපිට parents values/indexes, child values/indexes, left childs, right childs අඳුනගන්න පුළුවන් මුළුම array 1 ඉ ඕනෙම අඳුරගන්න බෑ.

∴ array 1 ඉ parents indexes, left child indexes, right child indexes හොයගන්න equation 3 කි use කරනවා.

(1) parent index (හොයන්න value 1 ක parent ගේ index 1 හොයගන්න පුළුවන්.)

$$\boxed{Parent[i] = \left\lfloor \frac{i}{2} \right\rfloor}$$

i = child index 1

Scanned by TapScanner

Date: 03

Eg:- අපිට 14 parent value 1 හොයාගන්න ඕන.

$$Parent[i] = \left\lfloor \frac{2}{2} \right\rfloor$$

14 රේ index 1 තමා මෙතෙන්ට දෙන්න ඕන. එතකොට එයාගේ parent ගේ index 1 තමා answer 1ට වෙන්නේ.

$= 1$

Parent value = A [1]

= 16

(2) Left child index (left child ගේ index 1 හොයන සූත්‍රය)

$$\boxed{Left\ child[i] = 2 * i}$$

Eg:- අපිට 14 රේ left child value 1 හොයාගන්න ඕන.

Left child[i] = 2 * 2

මෙතෙන්ට දෙන්නේ 14 index 1. left child index 1 තමා answer 1 විදිහට වෙන්නේ.

= 4

Left child value = A [4]

= 8

(3) Right child index (right child ගේ index 1 හොයන සූත්‍රය)

$$\boxed{Right\ child[i] = (2i + 1)}$$

Eg:- 14 value 1ගේ right side value 1 හොයාගන්න ඕන.

Right child [i] = (2*2) +1

14 index 1 මෙතෙන්ට දෙන්නේ. answer 1 විදිහට right child index 1 වෙනවා.

= 5

Right child value = A [5]

= 7

Scanned by TapScanner

Heap Algorithms

① Max_Heapify (HEAPIFY)

- To maintain max-heap property.

* Heapify algorithm ය recursive algorithm ය වේ.
* මේකෙන් කරන්නේ heap property ය maintain කරන එක.
* Heapify algorithm ය use කරන්නේ ය node එකක් විතරක් heap property ය violate කරලා තිබුණොත් විතරයි.
* Nodes කීයකින් property ය violate කරොත් වෙන algorithm ය කී කියනෙක එක maintain කරන්න. Heapify use කරන්නේ only one node violate වුණොත් විතරයි.

Array name

index of the violating node

```
MAX_HEAPIFY (A, i)
l = Left(i);
r = Right(i);
if l ≤ A.heap_size and A[l] > A[i]
        largest = l;
else largest = i;
if r ≤ A.heap_size and A[r] > A[largest]
        largest = r;
if largest ≠ i
        exchange A[i] with A[largest]
        MAX_HEAPIFY (A, largest)
```

මේ විදිහට (i) parameter ය කී pass කරලා තියෙනවා එතනදි හැමතිස්සෙම ගන්නේ index ය. (value ය නෙවේ)

මෙතනදි ගන්නේ heap ය කෝ size ය

recursion happens in this line

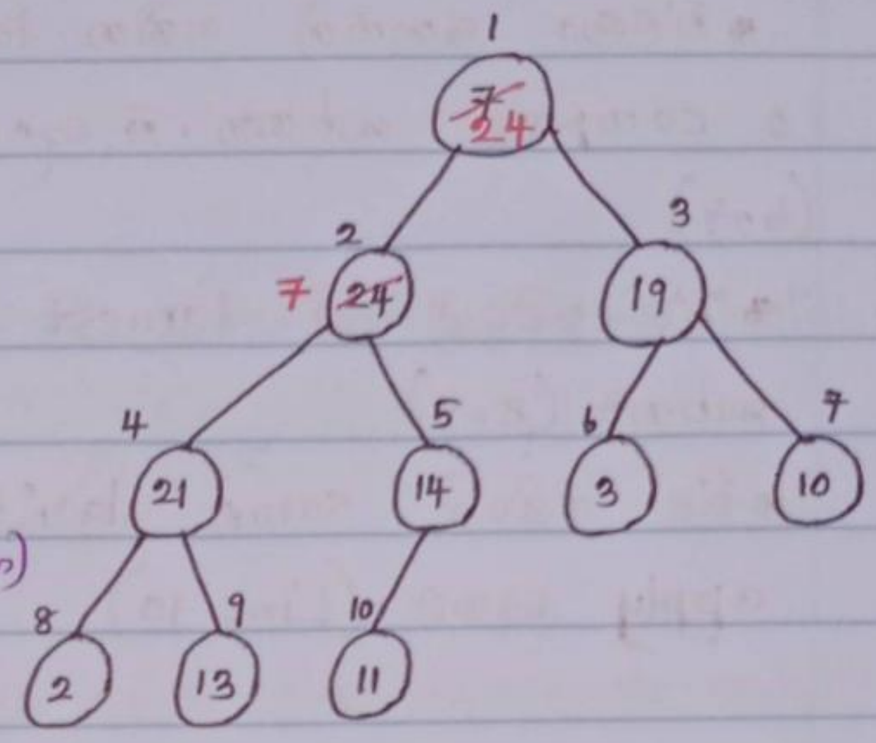
Scanned by TapScanner

Example

	1	2	3	4	5	6	7	8	9	10
A	7	24	19	21	14	3	10	2	13	11

* මේ Array 1ට තමා illustration අඳිනකේ.
* මුලින්ම මෙතන heap 1 කි විදිහට ඇඳගෙන ඉන්න ඕන.

මෙතන violate value 1 වෙනකේ 7. එකේ index 1 වෙනකේ 1



Max_heapify (A, 1)

1. l = 2
2. r = 3
3. if 2 ≤ 10 && 24 > 7 → T (if 1 ඇතුලට යනවා)
4. largest = 2
5. × (else 1 execute වෙනවා)
6. if 3 ≤ 10 && 19 > 24 → F (∴ if 1 ඇතුලට යන්නෑ)
7. ×
8. if 2 ≠ 1 → T (∴ if 1 ඇතුලට යනවා)
9. exchange 7 with 24
10. Max_heapify (A, 2)

* මේකෙදි මුලින්ම heapify 1 execute වුණාම 1st iteration 1දි 7 ↔ 24 exchange වෙනවා. ඊට පස්සේ ආපහු heapify algorithm 1 call කරනවා (recursive)
* මේ විදිහට recursive වලට ආයෙ ආයේ illustration අඳිනකේ නෑ. මුළු හරි අතිදට පස්සේ ඊළඟ විත පෙනින්නේ diagramatically විදිහට.
* Diagram 1 හරියට අඳින්න නම් logic එක දැනගෙන ඉන්න ඕන.

Scanned by TapScanner

Date: 06

Logic

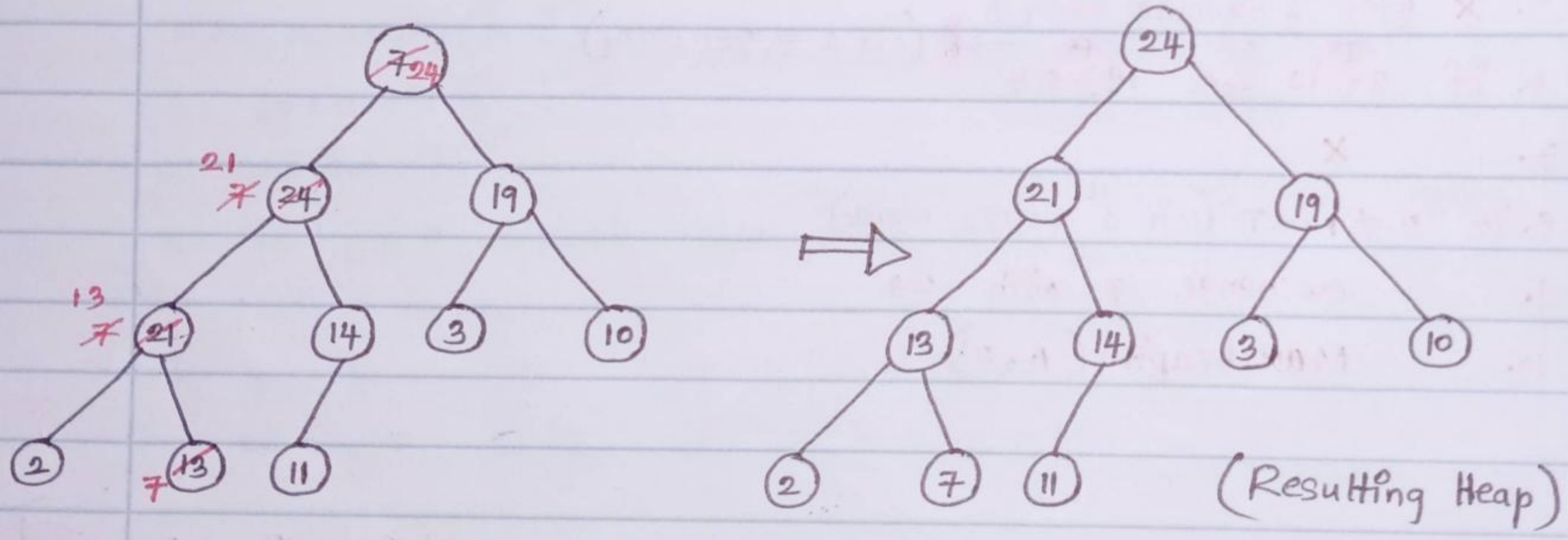
* මුලින්ම violating index ද algorithm ට දුන්නම එක check කරනක් වෙ index ට left & right child ඉන්නවද කියලා. එහෙම තියෙනවනම් ඊට පස්සේ ඒ values දෙක variable 2 කට assign කරගන්නවා. (1 to 2)

* ඊට පස්සේ violating value ට යි එයාගේ left child value ට යි compare කරනවා. කරලා ඒ දෙකෙන් largest value ට ගන්නවා. (3 to 5)

* ඊළඟට එතනින් ගන්න largest value ට යි එතන right child value ට compare කරනවා. ඒ දෙකෙන් ආපහු largest value ට select කරගන්නවා (6,7)

* ඊට පස්සේ ඒ largest value ට යි violation value ට යි exchange කරනවා (8,9)

* ඊට පස්සේ same algorithm ට ම ආපහු දැන් violation node ට ම apply වෙනවා. (Line 10)



* මේ විදිහට තමා diagramatically හෙමිහිට බලා, මුලි illustration ට යි පස්සේ.

Scanned by TapScanner

Date: 07

Analysis of Heapify Algorithm

* Heapify algorithm ඒකේ (total run time ඒ) $T(n)$, ගණන 2 කට වැඩි නිසා Normally Big 'O' value ඒකේ Big 'O' value ඒ

$$\boxed{T(n) = O(\log_2 n)}$$

කියලා තමා ගන්නේ.

* But heap ඒකේ කියන්නේ CBT ඒකේ නිසා, අපිට එකේ height ඒ ගන්න පුළුවන්,

$$\boxed{h = \lfloor \log_2 n \rfloor}$$

*මෙතනදී මේකේ flow value ගන්න එක තොරලක්ක හැරලා ($\lfloor\rfloor$) අපිට පුළුවන් Big 'O' value එකේ equation ඒකේ $\log_2 n$ වෙනුවට h ආදේශ කරන්න.

* ∴ එතකොට,

$$\boxed{T(n) = O(h)}$$

මෙහෙමත් Big 'O' value ඒ ලියන්න පුළුවන්.

Scanned by TapScanner

② BUILD_HEAP

* Heapify algorithm ↓ use කරන්න පුළුවන් one violation node ↓ කි තියෙනකොට විතරයි.
* violation nodes (හොඳක් කීපයක්) තියෙනකොට එක නිසා order ↓ ට ගන්න තමා build_heap algorithm ↓ use කරන්නේ.

Array name

BUILD_MAX_HEAP(A)

1. A.heap size = A.length
2. for i = ⌊A.length/2⌋ downto 1
3. MAX_HEAPIFY(A, i)

Array size = A.length , A.heap-size

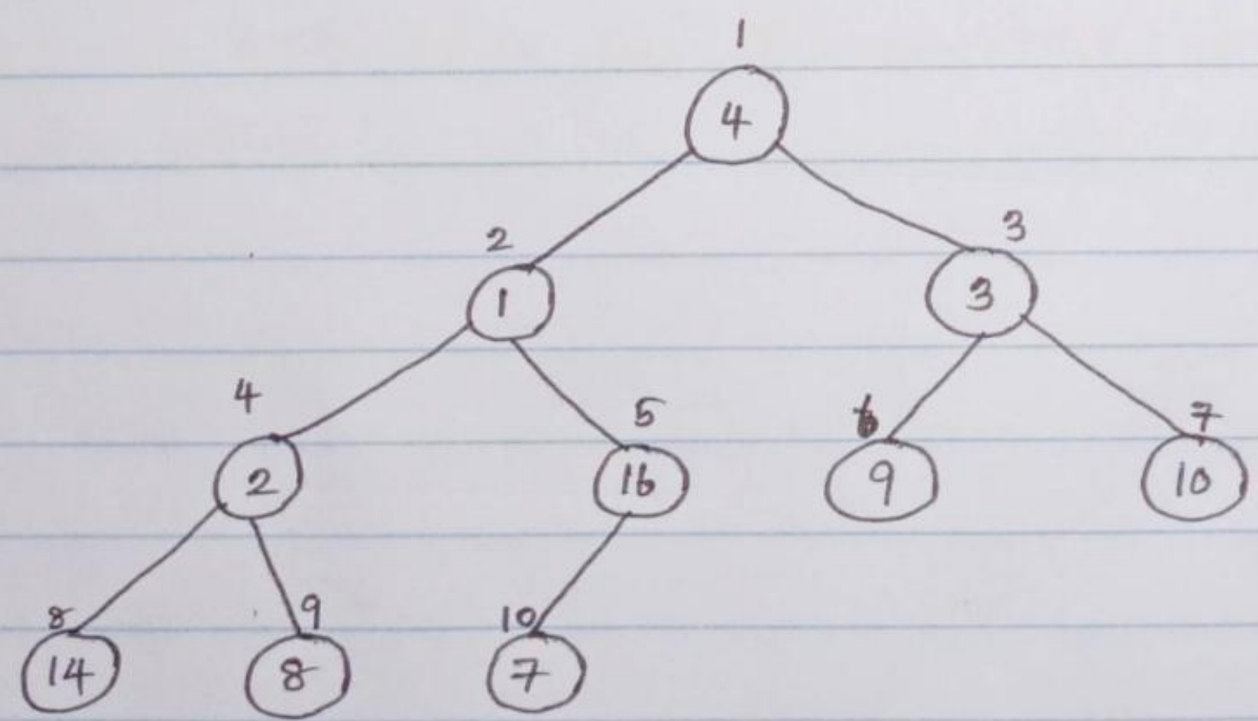
වෙන්න ගන්නේ flow value ↓

Build_heap ↓ දිගේ ආපහු heapify algorithm ↓ call කරනවා. මොකද වෙන්නේ violation nodes කීපයක් තිබුණොත් අපිට එක කරකට එක node ↓ නෙමෙයි තමා වෙන්නේ කරන්න වෙන්නේ. ∴ එතකදී heapify algorithm ↓ use කරනවා. ඔක්කොම nodes check වෙන්න ඕන නිසා තමා for loop ↓ කි use කරලා තියෙන්නේ.

Example:-

	1	2	3	4	5	6	7	8	9	10
A	4	1	3	2	16	9	10	14	8	7

* වෙන්න මුලින්ම heap ↓ කට convert කරගන්න ඕන.



Scanned by TapScanner

Date: 09

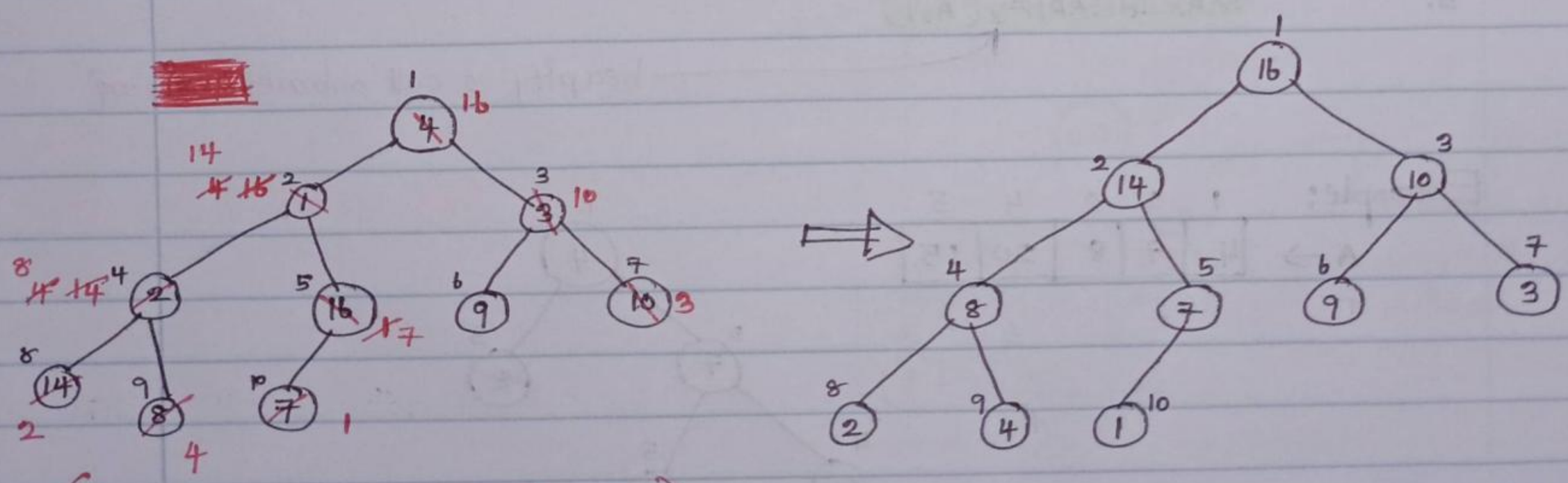
Build_max_heap(A)

1. A.heap size = 10
2. for i = ⌊5⌋ down to 1 — ⌊5⌋ = 5 වගෙයි. decremental for loop (i=5, 4, 3, 2, 1)
3. MAX-HEAPIFY(A, 5) — මෙතන 5 දෙනවිට මේක initial iteration 1 කියන. iteration 1 ක 10 ගේ value 1 වෙනවා වෙනවා. මේ 5 කියන්නේ මෙතනදී අපි 5th index node 10 heapify algorithm 1 apply කරන්න ඕන.

Max_heapify (A, 5)

1. l = 10
2. r = 11 ← 11th node 1 ක් නැතිවුණොත් මෙතනදී 11 කියලා දෙනවා. නොකද node 1 ක් තිබුණොත් අනිවා එකේ index = 11 නිසා
3. if 10 ≤ 10 && 7 > 16 (T && F → F)
4. X
5. else largest = 5
6. if 11 ≤ 10 && .... → F (if 1 ඇතුලට යන්න බැ)
7. X
8. if 5 ≠ 5 → F (if 1 ඇතුලට යන්න බැ)
9. X
10. X

*මෙතනදී illustration අතරේ for loop 1 ගේ (i=5) 1st iteration 1ට විතරයි. But 'b'ලොට i=4, i=3 ... කට illustrations කියනවා. But අපි සලකන්නේ heapify 1ට illustration අඳින්න කියන ඉදිකිරීම බැ. ∴ මෙතනදී Diagram use කරනවා වෙනවා.



(for i=4, i=3, i=2, i=1 iterations)

Scanned by TapScanner

Analysis of build-heap

* Build-heap ද්‍රැ කෙරෙන complexity analysis ද්‍රැ කිරීමේදී.

* For build-heap,

$$\boxed{T(n) = O(n)}$$ (Big 'O' value ද්‍රැ → n)

මේ කියන්නේ Run time ද්‍රැ array elements ගණන සමානුපාතික වෙනවා
($n\uparrow = T(n)\uparrow$, $n\downarrow = T(n)\downarrow$)

③ HEAPSORT Algorithm

* heapsort ද්‍රැ කරන්නෙන් array inputs විදිහට ascending order ද්‍රැ ගන්න එක.

* මේ heapsort ද්‍රැ implementing වලදී build-heap ද්‍රැ යි, heapify ද්‍රැ යි දෙකම භාවිතා වෙනවා.

Array name

```
HEAPSORT(A)
1.  BUILD-MAX-HEAP(A)
2.  for i = A.length down to 2
3.          exchange A[1] with A[i]
4.          A.heap-size = A.heap-size - 1;
5.          MAX-HEAPIFY(A,1)
```

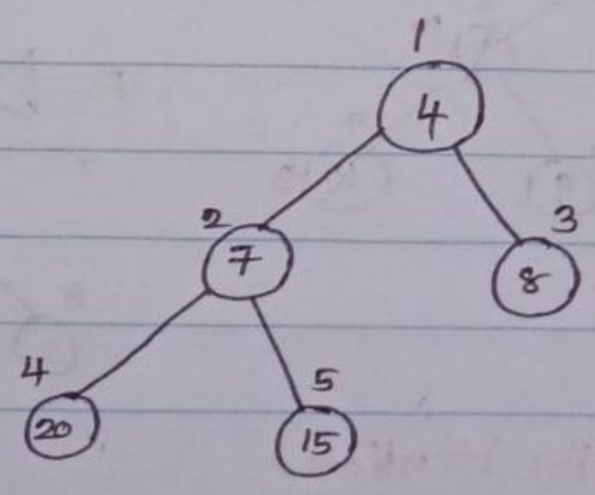
Build-heap ද්‍රැ call කරනවා මෙතනදී

heapify ද්‍රැ call කරනවා මෙතනදී

Example:

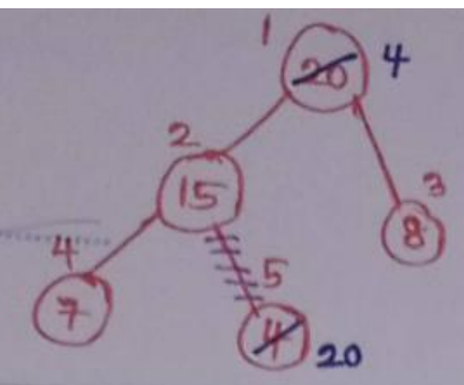
A →

1	2	3	4	5
4	7	8	20	15



Scanned by TapScanner

Date: 11



HEAPSORT(A)

1. BUILD-MAX-HEAP(A)
2. for i = 5 down to 2
3. exchange 20 with 4
4. A.heap-size = 5-1 = 4
5. MAX-HEAPIFY(A,1)

වෙනුවට වෙනිවේ last node 1 අයිති කරන්න ඕන heap 1ටයි.

BUILD-HEAP(A)

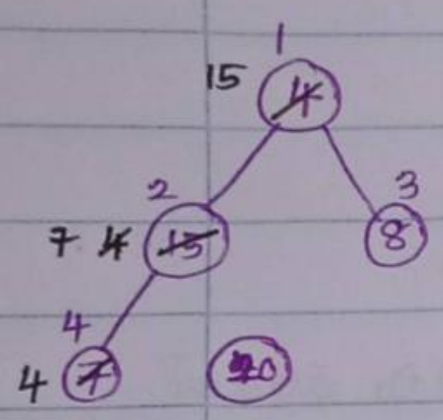
1. A.heap size = 5
2. for i = $\lfloor 5/2 \rfloor$ downto 1
3. MAX-HEAPIFY(A,2) ← මුල 1st iteration 10.

$\lfloor \frac{5}{2} \rfloor = 2$

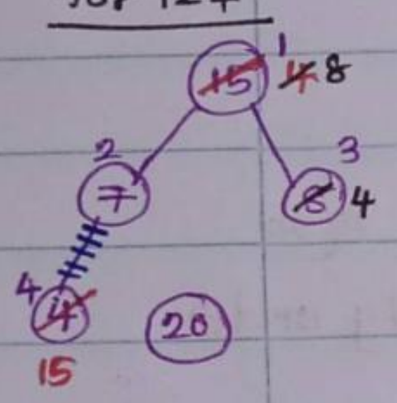
MAX_HEAPIFY(A,2)

1. l = 4
2. r = 5
3. if $4 \overset{T}{\leq} 5 \overset{\&\&}{\&\&} 20 \overset{T}{>} 7 \rightarrow T$
4. largest = 4
5. ×
6. if $5 \overset{T}{\leq} 5 \overset{\&\&}{\&\&} 15 \overset{F}{>} 20 \rightarrow F$
7. ×
8. if $4 \neq 2 \rightarrow T$
9. exchange 7 with 20
10. MAX_HEAPIFY(A,4)

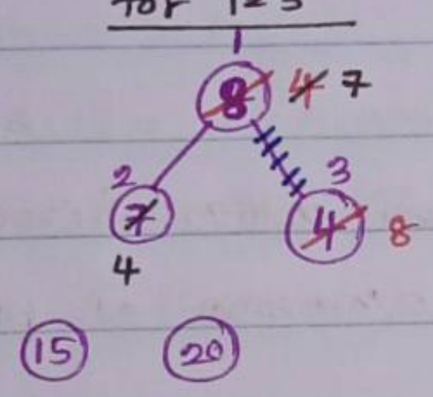
for i=5



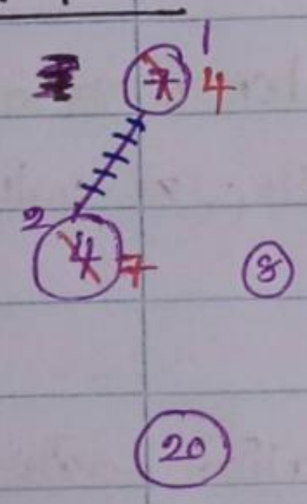
for i=4



for i=3

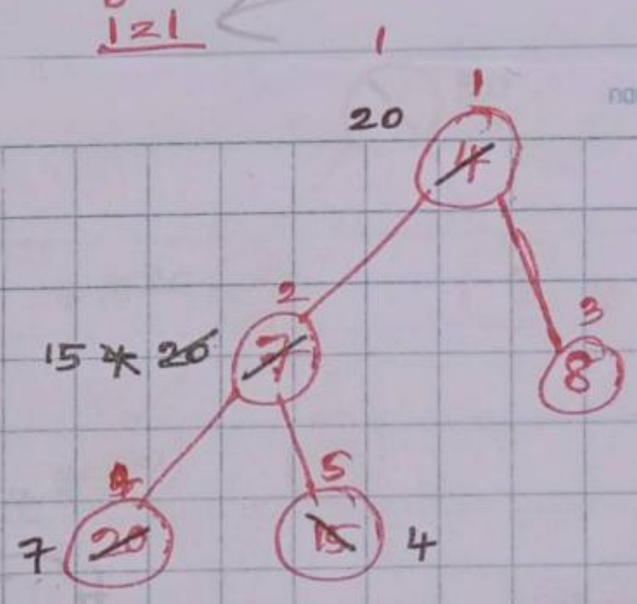


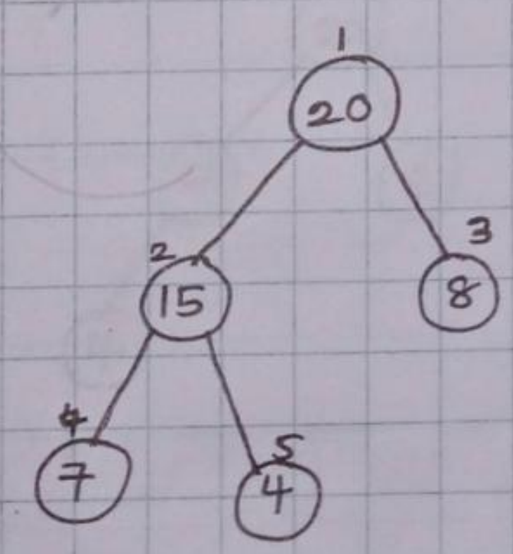
for i=2



exam 1ට මේ ඔක්කෝම අදිලා එන්ඩ ඕන. i=5 න කෙන්නලා likevise ascending order 10 ගැලපෙන්න නිලව කෙරෙනවා අපි

i=1





මේක තමා අපි heapsort 1ගේ execution වෙලා ගත්තේ.

Likevise,

4 7 8 15 20

Scanned by TapScanner

Heapsort running time complexity

* Running Time is $O(n \log_2 n)$

$$\boxed{T(n) = O(n \log_2 n)}$$

Priority Queues

* Heap data structure ද්වාරා usages ගොඩක් තියෙනවා.
* Heap ද්වාරා එක application එක තමා "priority queue"
* මේ priority queue ද්වාරා කියන්නේ normal queue ද්වාරා වඩා වෙනස් queue ද්වාරා. Normal ද්වාරා වගේ FIFO method ද්වාරා නෙවෙයි priority queue ද්වාරා වැඩ කරන්නේ.

* මේකෙදී වෙන්නේ අපි priority queue ද්වාරා තියෙන elements වලට priority value ද්වාරා (int no ද්වාරා) දෙනවා. එතකොට ඒ priority order ද්වාරා තමා මේකෙන් output ද්වාරා ලැබෙන්නේ.

* Arrival order ද්වාරා ගැන නෙවෙයි මේකෙදී කලකන්නේ. priority order ද්වාරා ගැන. ∴ front, rear variable දෙක අමතරව මේකෙදී තව priority කියලත් value ද්වාරා තියෙනවා.

* මේ priority queues use කරලා තමා heaps implementation වෙන්නේ.

* There are two kinds of priority queues:
  - max-priority queues
  - min-priority queues

* අපි lec ද්වාරා කරන්නේ max-heaps නිසා මෙතනදී කතා කරන්නෙක් max-priority queues ගැන.

Scanned by TapScanner

* max-priority queues වලදී වෙන්නේ maximum numbers වලට priority දෙන එක.

* අපිට max-priority queue එකේ කරන insert and extract (remove and returns) operation කරන්න පුළුවන්.

HEAP_EXTRACT_MAX

• මේ algorithm එකේ heap එකේ තියෙන maximum element එක remove කරලා return කරනවා.

heap name — heap size

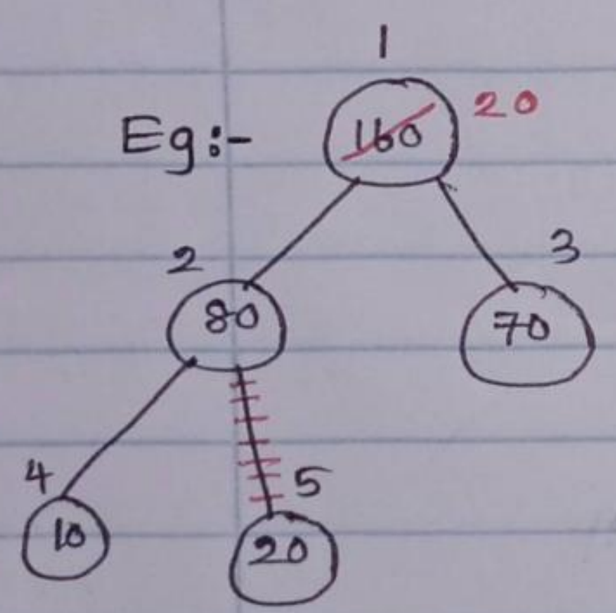
HEAP_EXTRACT_MAX (A[1...n])

```
if A.heap_size ≥ 1
    max = A[1]
    A[1] = A[A.heap_size]
    A.heap_size = A.heap_size - 1
    MAX_HEAPIFY(A, 1)
    return max
```

Logic

max heap එකේ maximum value එක තියෙන්නේ 1st index එකේ තියන එකේ තියෙන value එක max කියලා variable එකට දාගන්නවා. ඊට පස්සේ last node එකේ value එක 1st node එකට assign කරලා last node එක අයින් කරනවා. ඊට පස්සේ අපිට heapify algorithm එක call කරලා දැනි 1st node එකේ තියෙන violation value එක හරි place එකට දාගන්නවා. Finally max variable එකට තිබුන maximum value එක return කරනවා

Eg:-

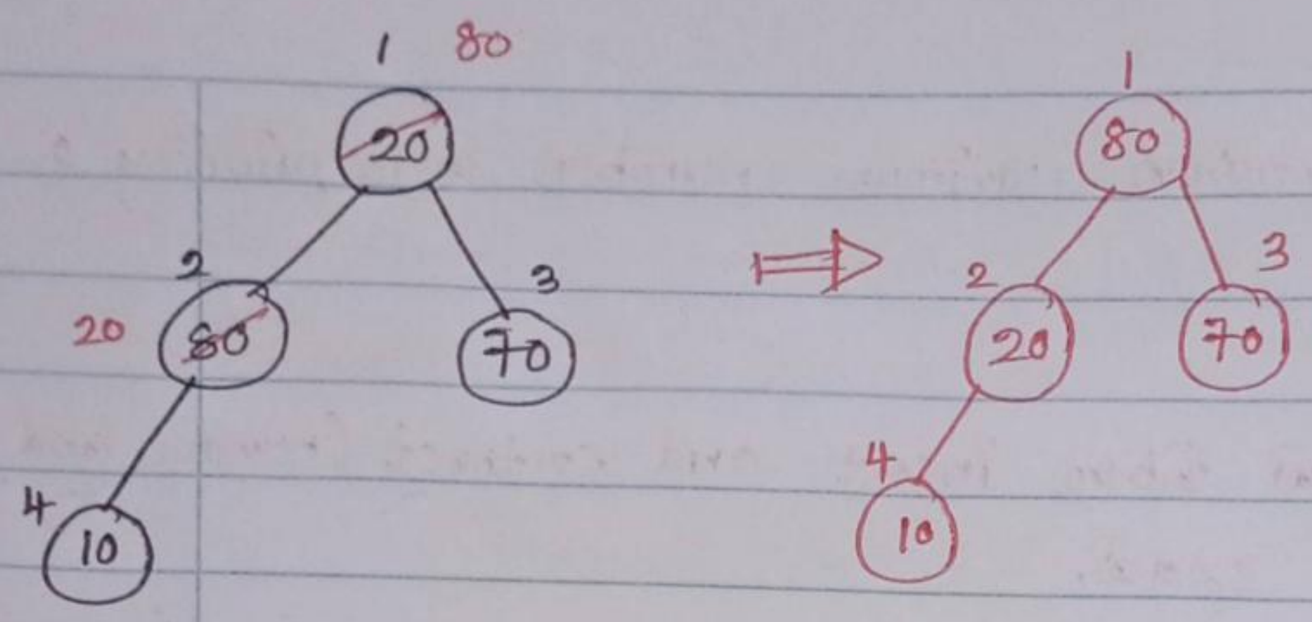


HEAP_EXTRACT_MAX (A[1..5])

```
if 5 ≥ 1 → T (∴ if එක ඇතුලට යනවා)
    max = 160
    A[1] = A[5] = 20
    A.heap_size = 4
    MAX_HEAPIFY(A, 1)
    return 160
```

**heapify එකේ illustration එක දැකිලා තියා. (But වෙනදේ දිලාව diagram එකේ හැටි ගන්න ඕන.)

Scanned by TapScanner



* Heap_extract_max algorithm 1 වගේ total run time, T(n)

$$\boxed{T(n) = O(\log_2 n)}$$

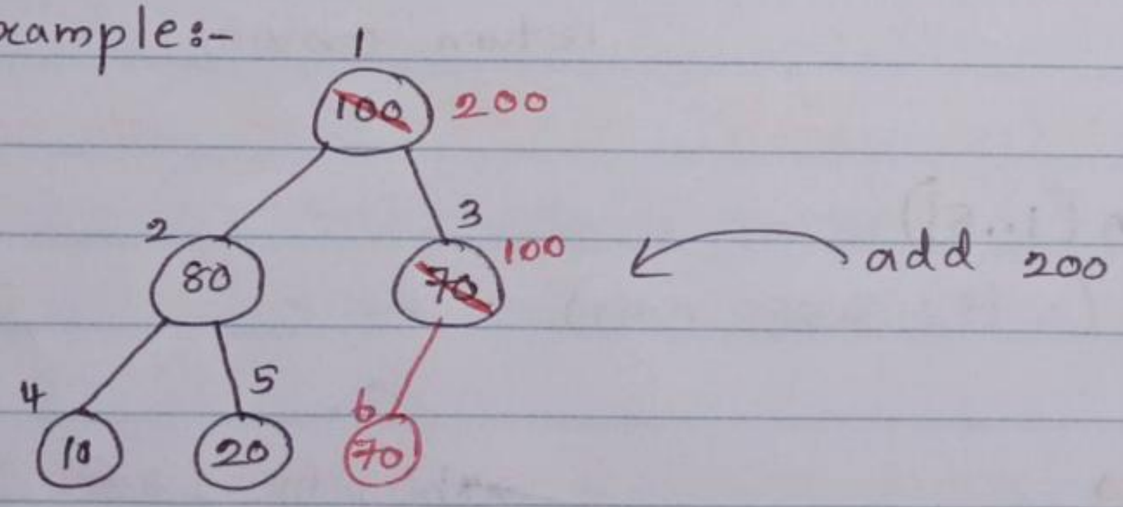
HEAP_INSERT

* මේකෙන් new element 1 කෑ add කරන්න පුළුවන් heap 1ට.

අපිට insert කරන්න ඕන value 1

```
HEAP_INSERT (A,key)
1. A.heap_size = A.heap_size+1
2. i= A.heap_size        //assume A[i] = -∞
3. while i>1 and A[PARENT(i)] < key
4.          A[i] = A[PARENT(i)]
5.          i = PARENT(i)
6. A[i] = key
```

Example:-



Scanned by TapScanner

heap_insert (A, 200)

1. A.heap_size = 6
2. i = 6
3. while 6 > 1 && A[3] < 200 → T (while inner statement executed)
   3>1 && 100<200 70 < 200
4. A[6] = 70 → T (for 2nd iteration)
5. i = 3 → A[3] = 100 → X
   i = 1
6. A[1] = 200

* Time complexity of heap_insert,

$$\boxed{T(n) = O(\log_2 n)}$$

**** heap_insert සහ heap_extract_max algorithm දෙ input 1 අනිවාර්යෙන් max heap 1 කි වෙන්න ඕන.
(without any violations)

Scanned by TapScanner

Date: 01

- String Matching Algorithms -

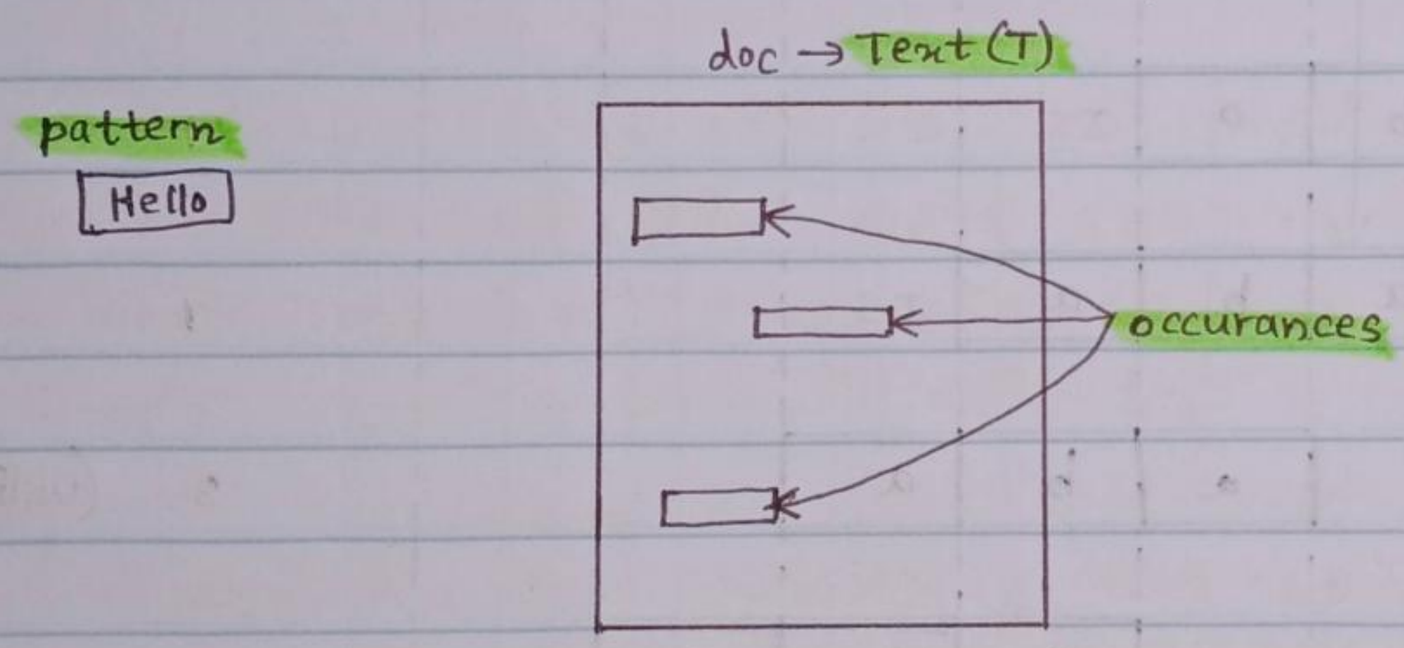
String Matching

* String matching කියලා කෙනෙක්ගේ given string 1 ක checking or matching කරන එක document 1 ක වගේ දෙයක්. (eg:-search function)

* කෙනෙක්ට string කියනකොට given numbers, given character, given character set වගේ වගේ ඕනම එකක් කෙනෙක් පුළුවන්.

- String Matching is finding occurance(s) of a given pattern string within another string or body of text.

Eg:- අපි word doc 1 ක ctrl+F ගහලා word 1 ක search කරනකොම ඒ word 1 තියෙන ඔක්කොම occurances අපිට එකතු කරගන්න පුළුවන්.



- There are many different algorithms for efficient searching.

Applications

* Text processing
* Text-editing (eg:-Find and change in word)
* Computer security (virus detection, password checking)
* DNA sequence analysis
* Data communications (header analysis)

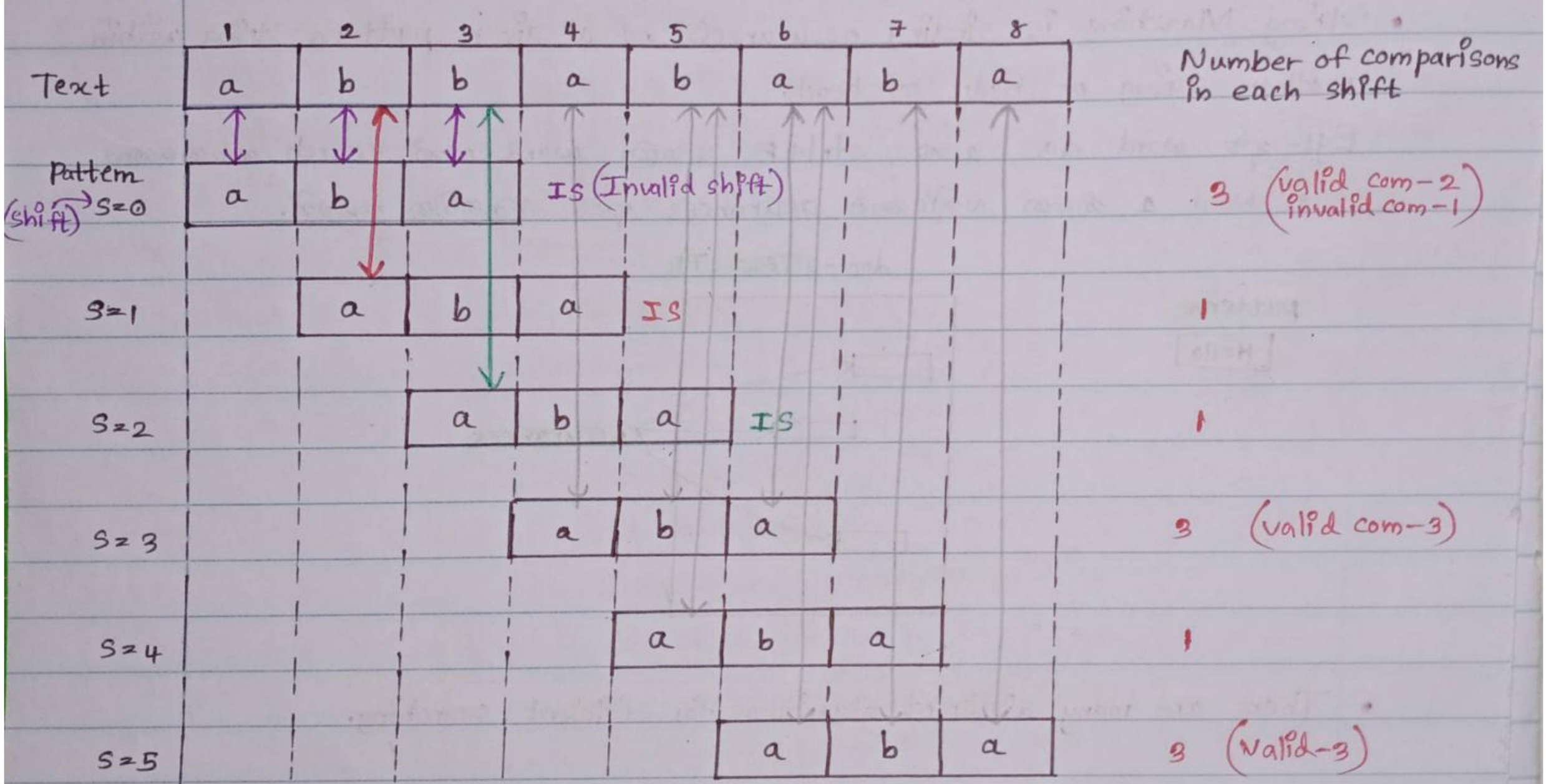
Scanned by TapScanner

Date: 02

Naive String Matching Algorithm

මේකෙදි කරන්නේ අපිට තියන text 1 ගේ L.H.S ඉඳලා R.H.S වෙනකම් අපි දෙන pattern 1 තියෙනවද කියලා check කරන එක. අවස්ථාගතේ L.H.S 1 ගේ ඉඳලා shift=0 වලදි. shift=0 කියන්නේ pattern 1 තියන placement 1 ට. shift=0 වලදි text 1 ගේ 1st character තියන line 1 ගේමයි pattern 1 ගේ 1st character 1 නි තියන්නේ.

මෙහෙම කියලා එක තරකම එක character 1 ගැණේ check කරන්න ඕන invalid (comparison) matching 1 නි අහුවෙනකම්.



Number of comparisons in each shift

* මේකේ total shifts ගණන්නෙ 0→5 shifts 6 නි තියෙන.

* මේ shift ගණන ගොඩනගන equation 1 නි තියෙන.

$$\boxed{\text{Total number of shifts} = n-m+1}$$

Scanned by TapScanner

Date: 03

n - Number of characters inside the text (T)
m - Number of characters inside the Pattern (P)

Eg:- previous example,

Total no. of Shifts = $n-m+1$
$= 8-3+1$
$= 6$ $(0 \rightarrow 5)$ *(not $1 \rightarrow 6$)

Best Case

* Best case කියන්නේ T(n) අවම case 1ට.
* එතකොට string matching වලදී run time 1ට වලින්නේ comparisons ගණන. comparisons ගණන ↑ නම් T(n) ↑. comparisons ↓ නම් T(n) ↓.

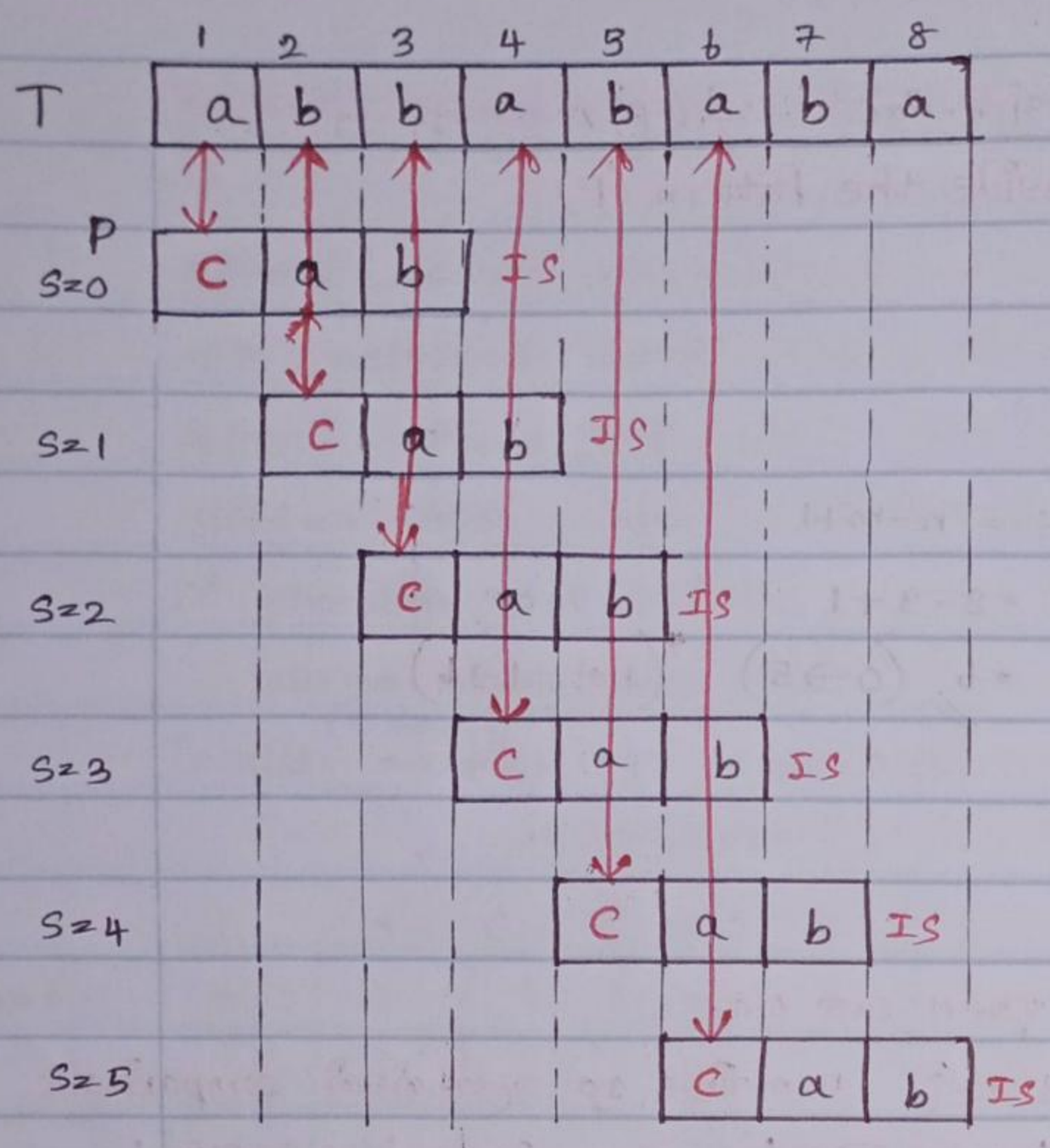
∴ Best case 1දී අපි ගන්න ඕන අඩුම comparisons ගණන්කි වෙන අවස්ථාව. Shift 1 කදී වෙන්න පුළුවන් අඩුම comparisons ගණන 1 යි. (1 invalid comparison) එහෙම වෙන්න නම් හැම shift 1 කෙදිම 1st character comparison 1 invalid වෙන්න ඕන.

එතකොට
ඒ කියන්නේ first character 1 text 1 කේ නැති invalid charater 1 කි වෙන්න ඕන. මොකද එතකොට තමා හැම shift 1 දිම 1 comparison 1 කි විතරක් වෙන්නේ.

∴ String Matching 1 කේ best case 1 දී pattern 1 කේ 1st character 1 text 1 කේ not existing character 1 කි වෙන්න ඕන.

Scanned by TapScanner

Date: 04



No. of comparisons in each shift

1

1

1

1

1

1

Time Complexity of Best Case

* Total no. of comparisons = 6 = total no. of Shifts

* Total no. of shifts = n−m+1

$$\boxed{\text{Time complexity} = O(n-m+1)}$$

Scanned by TapScanner

Date: 05

Worst Case

* worst case ඒදී comparisons ගණන maximum වෙනවා ඕන. එතකොට තමා T(n) ↑ වෙන්නේ.

* මේ කියන්නේ maximum comparison වෙනවා කියලා text එකේ තියෙන characters ඔක්කොම අමාරු වෙනවා ඕන සහ pattern එකෙන් text එකේ තියෙන characters ම නිකන්න ඕන. pattern ඒ විදි 2 කට තියෙන්න පුළුවන් එනකොට,

① T | a | a | a | a | a | a | a | a |

P | a | a | a |

* මෙතනදී comparison 3 කි (maximum) එනවා (3-valid)

② T | a | a | a | a | a | a | a | a |

P | a | a | b |

* මෙතනදීත් comparison 3 කි (maximum) එනවා. (2-valid, 1-invalid)

Eg:- T | a | a | a | a | a | a | a | a |

	No. of comparisons in each shifts
S=0 P: a a b	3
S=1 a a b	3
S=2 a a b	3
S=3 a a b	3
S=4 a a b	3
S=5 a a b	3

Time complexity of worst case

Total no. of comparisons = 18
= 3 × 6

3 → no. of characters inside the pattern (m)
6 → number of shifts → (n-m+1)

Scanned by TapScanner

Date: 06

∴ Total no. of comparisons = $m(n-m+1)$

$$\boxed{\text{Time Complexity} = O(m(n-m+1))}$$

The Rabin-Karp Algorithm

* Rabin-Karp algorithm ↳ characters, character set, numbers ඕනම ඒකක use කරන්න පුළුවන්.
* But මෙතනදී ගන්නේ numbers විතරයි.

Example:-

T = 2013004003005 02

P = 300

q (modulo value) = 10 ← මේ q value ↳ ගණන්කම් වැඩිදියි අපිම දෙනවා.

① මුලින්ම p % q ගේ අගය සොයගන්න ඕන.

② 'b' ලබාදෙන්නේ pattern ↳ තියන characters ගණන වෙලා ඒ ගණනට අදාල all posibilities විතර ලියගන්න ඕන. (Eg:- මෙතන p ගේ numbers 3 කි තියා නිසා t ගේ all posibilities of 3 numbers විතර ලියගන්න ඕන)

③ 'b' ඒ හැම value ↳ කම % q සොයන්න ඕන.

④ p % q ගේ answer ↳ ට සමාන answers තියන අවස්ථා විතර වෙන්කරලා අරගෙන ඒවා අතරින් valid hits සොයගන්න ඕන. (valid answers)

$$p \% q = 300 \% 10 = 0$$

Scanned by TapScanner

B

Date: 07

201 %10 = 1

013 %10 = 3

130 %10 = 0 → spurious hit

300 %10 = 0 → valid hit

004 %10 = 4

040 %10 = 0 → spurious hit

400 %10 = 0 → spurious hit

003 %10 = 3

030 %10 = 0 → spurious hit

300 %10 = 0 → valid hit

005 %10 = 5

050 %10 = 0 → spurious hit

502 %10 = 2

spurious hits;

කියන්නේ p%q ගේ answers සමහර වැරදුණු නිසා answers නොවෙන values වලට.

**මෙහිදී valid hits වලට වඩා spurious hits ගණන වැඩියි.

මේ spurious hits ගණන අඩු කරන්න අවශ්‍ය නම් අපිට q ගේ අගය තව කරන්න පුළුවන්.

(Eg:- q=10 → q=100 වගේ කරනවා නම් spurious values අඩු වෙනවා දැනට වඩා)

* මෙහෙදී time complexity 1 ට බලපාන්නේ hits ගණන (Number of hits)

* No. of hits ↓ නම් T(n) ↓, No. of hits ↑ නම් T(n) ↑

Best Case

* Best case 1 දී T(n) ↓ ට නිසා, [T(n) minimum]

Number of spurious hits & valid hits = 0

Worst Case

* Worst case 1 දී T(n) ↑ ට නිසා, [T(n) maximum]

(1) All the hits are valid hits

(2) All the hits are spurious hits

(3) All the hits are the combination of valid & spurious hits

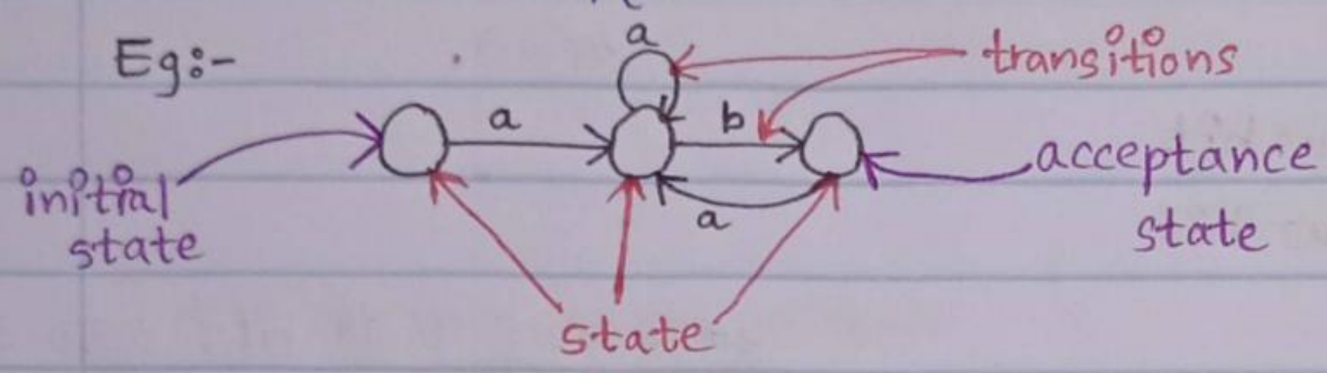
මෙහිදී Text 1 ගේ all අවස්ථා වික 0 වෙලා මේ අවස්ථා 3 යි 1 යි වෙනවා විතරයි.

Scanned by TapScanner

Finite Automata

* මේකෙදි කරන්නේ දීලා තියෙන pattern එකකට අනුවව State chart machine 1 කි අඳින එක.

Eg:-



State Chart Machines

Eg:- P = aba input alphabet = {a,b}

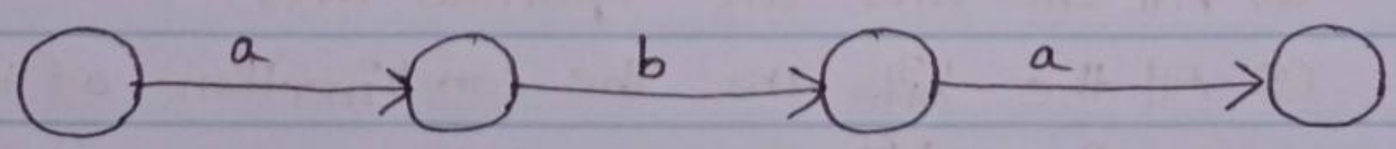
* State chart අඳින්න දෙනකොට අපිට දෙන්නේ pattern 1 විතරයි. Text 1 කි දෙන්නේ නෑ. ඊට අමතරව input alphabet එක දෙනවා. එනකොට අපි කරන්න ඕන මේ pattern 1 සහ input alphabet 1 use කරලා අනුව pattern 1 ලැබෙන්න පුළුවන් Text ඔක්කොටම state chart machine 1 ක හෙන්නන්න එක.

① State chart machine 1 අඳින්න නම් මුලින්ම States ගණන හොයාගෙන ඉන්න ඕන. ඒකට equation 1 කි තියෙනවා.

Number of States = No. of characters inside the P + 1

No. of states = 3+1 = 4

② ඊට පස්සේ මේ states වලට horizontal transitions වලට කිව්වා (arrows) ඒවාට input විදිහට pattern 1 ගේ characters ටික දෙනවා pattern 1 ගේ තියන විදිහටම.



* මෙතනදි Text 1 කි ~~aba~~ aba කියලා perfect match 1 තමා මේ විදිහට හෙන්නන්නේ.

Scanned by TapScanner

Date: 09

input alphabet = {a, b}

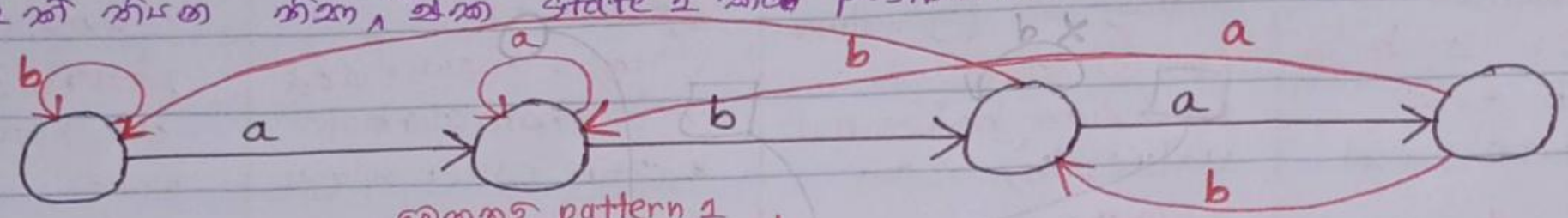
*input alphabet 1 කේ තියන characters ගණනට අදාල possibilities බලන්න පින. egr characters 3ක් තියෙනවනම්
T =
T =
T = 3ක් බලන්න පින

③ මේ perfect match 1 ට අමතරව මේ states මිකොන් pattern 1 ලැබෙන්න පුළුවන් ආකාර තොකක් තියෙනවා.

∴ 'b'ලගට කරන්නේ initial state 1 ක් තමන්ගෙන state 1 කින් 1 අරගෙන මේ pattern 1 ගන්න පුළුවන් විදිහට transitions (arrows) දාන එක.

(To find other posibilities)

*මේ විදිහට other posibilities ගන්න නම් input alphabet 1 අවශ්‍ය වෙනවා. අපිට initial 1 කේ ඉඳලා හැම state 1 කම බලනකොට මේ input alphabet 1 through නම් transitions අඳින්න වෙන්නේ. input alphabet 1 කේ එකතුකරන 1 character 1 කේ විතරයි insert කරන්න පුළුවන්. ∴ අපිට මේ දීලා තියන example 1 කේ නම් in. al. 1 කේ values 2 කේ තියන නිසා හැම state 1 කටම posibilities 2 කේ බලන්න වෙනවා.



මොනවද pattern 1 ලැබෙන නිසා මේක correct

① T = a ✓
② T = b ✗

① T = abaaba ✓
② T = ab ✓

① T = aba ✓
② T = abb ✗

① T = abaa
② T = abab

① T=a වුණාම, pattern 1 කේ තියනවනම් aba මොකද T කියන්නේ a ගෙනි. pattern 1 කියන්නේ a ගෙන් පටන් ගන්න මේ අකුරුන් මේ valid Text 1 කේ.

② මේක්ක විදිහට 1st character 1 b. But අපිට ඕන aba නිසා 1st character 1 a වෙන්න ඕන. ∴ මේක invalid character 1 ක්.

*invalid character 1 ක් තියේනම් options 2 ක් තියනවා.

② nd state 1 ක් තියෙන්නේ 2nd character 1.

*මොකද T= කියලා කියන්න ඕන initial 1 කේ ඉඳලම. ∴ මොකද T=aa/ T=ab කියලා කියා ලැබෙන්නේ

T=aa
T=ab
මේ a, b 2 ඉන්නේ input alphabet 1 කේ

① මේක invalid character 1 ක්. ∴ ඉන්නවලිලා use කරපු options 2 කේ 1 ක් මොකද්දක් ගන්න වෙනවා.

Normally මේ වගෙනේ priority 1 දෙන්නේ "Stay in same state" 1 ට.

*මේකෙදි තියන්නේ 3rd character 1 ගැන.

② මොකක්ද invalid නිසා options 2ම 1ම use කරන්න ඕනෙ.

1st → abbbba ✗

මෙතන output 1ම ලැබෙන්නේ 1st option 1 දී.

∴ මොකක්ද use කරන්න ඕනේ 2nd option 1. මොකද 1st option 1 කේ pattern 1 ගන්න බැරි නිසා.

∴ මොකක්ද 2nd 1 correct දී නිදලා බලන්න දෙයක් නෑ. මොකද 1st option 1 incorrect නිසාත් 2nd opt 1 ඉන්න ඕනි.

*මොකක්ද Text 1 ගේ already pattern 1 තියෙන්නේ අපි කොච්චර occurances තියනවා. මොකද state chart 1 කින් one occurance 1 කට වඩා තියන නිසා.

*මේ last state 1 දී අපි කෙලින්ම 2nd character 1 ගේ ඉඳලා cover කරලා posibilities බලනවා.

මෙන මුලින් වලකන්නේ නෑ.

(i) T = abaa

baa, aa දෙකේම pattern 1 ලැබෙන්නෑ. But a ගෙන් pattern 1 ගන්න පුළුවන්. ∴ p = aba

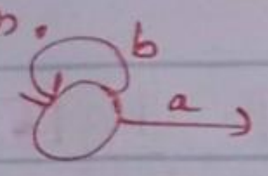
* මේ තියන්නේ 1st character දැනටම මොකක් තියනවා. ∴ අපි කරන්න ඕන 2nd character

① Stay in the same state
② Go back state

But මෙතනදී මේක initial state 1 නිසා option ② හරියෙන්නෑ.

∴ option ① තමයි.

මෙතන invalid character 1 b නිසා b ගෙන initial state 1කේ ඉන්නවා.



* මේක option correct නම් මේක use කරා incorrect වුණොත් 2nd option 1 ගන්නවා,

1st option 1 correct ද කියලා බලන්න stay 1 දීලා වගන stay 1 නිසා ලැබෙන iteration නිසාන් බිලකගන්නක text 1ට. b'ට තුන්වෙනි horizontal arrow බල දීපු values මිනි (ඒ state 1ට තුන්වෙනි නිසා horizontal values මිනි) b'ළඟට මියනවා.

තනිදී ඒ ලියපු whole text 1ගේ නෙනෙකුට pattern 1 නිසා තනි ඒක correct. නැත්නම් 2nd opt. 1ට යන්න වෙනවා

* But මෙතනදී අපි බලන්න ඕන back state 1කට යනවනම් මොන back state 1 මදි transition 1 දාන්න ඕන කියලා

මොකද මෙතන 3rd state 1ට ඊළඟට තුන්වෙනි state 1ක් නියනක.

* ඒක බලන්න කරන්නේ ඉන්න Text 1ට ලැබුණු answer 1ගේ 2nd characterගේ ඉඳලා cover කරලා check කරනවා අපිට මේ pattern 1 start කරන්න ඉඳන්නේ කියලා

(i) T = a bb

මෙතනට නියෙදිදී ගන්න බෑ. මොකද pattern = aba මෙතන තියන්නේ b ගෙන් (bb)

∴ මේක ✗

(ii) T = a b b

මේක නරියන්නෙත් නෑ. මොකද තියන්නේ b ගෙන්

මේ නියන්නේ මොකක 1 නිසාමයි pattern ගන්න බෑ.

∴ අපිට මුලම state 1ට (initial state) යන්න වෙනවා.

∴ මේ මගේ අවස්ථාවකදී කරන්නේ අපිට initial state 1ට transition 1ක් දාලා යමු.

1 නි. ∴ 2nd state 1ට තමා transition 1 දාන්නේ.

(iii) T = a bab ✗
T = ab ab ✓

මෙතනදී ab characters දෙකම නියනවා. pattern 1 තුදෙන්න ඕන 3rd character 1 විතරයි.

∴ 3rd state 1ට transition 1 දානවා.

** T = aba b

අපිට මෙතෙන්ට check කරන්නේ. මොකද උඩ ගන්න ab ගමන් b නි නියන නිසා