

Recursion & stack

There are 2 conditions for recursion:-

① Simple case

② Recursive case

Function stack memory
عن طريق "Stack" يتم تخزين
Function calls في الذاكرة

ex Sum of Natural Numbers:

```
→ int sum(int n) {
```

```
    if (n-1 == 0)
```

```
        return 1; or return n;
```

```
    else
```

```
        return n + sum(n-1);
```

```
}
```

Factorial

```
int fact(int n) {
```

```
    if (n-1 == 0 || n == 0)
```

```
        return n;
```

```
    else if (n == 0)
```

```
        return 1;
```

```
    else
```

```
        return n * fact(n-1);
```

* The power of number

```
int power(int num, int power) {
```

```
    if (p == 0)
```

```
        return 1;
```

```
    else if (p == 1)
```

```
        return n;
```

```
    else
```

```
        return n * power(n, p-1);
```

```
}
```


16.09.2021

Algorithm Analysis

Thursday

* أنا كمبرمج بغير أحد مسألة بأكثر من طريقة Algorithm ولكن لازم أختار أحسن واحدة من حيث تكون أسرع وقت وأقل مساحة

Def:- Algorithm: step by step procedure to solve a certain task.

Analysis

* The problem can be solved by more than algorithm. So the programmer should choose the best one.

less time less space



Resources (CPU) Memory (RAM)

* كل برنامج يفكره أو يستعمله خطه الكمبيوتر حال RAM

Analyze

* How to Analyze the problem:-

1) Step by step Analysis / line by line Analysis

```
void Function (int n) {  
    int i, j;  
    for (i=0; i<n; i++) {  
        for (j=0; j<n; j++) {  
            printf ("i.d j.d", i, j);  
        }  
    }  
}
```

ح يوصل لكونه خطه
البرنامج لأنه الشرط خطه

C "Constant"

$n+1$ ← مرة بفرقة

$n(n+1)$ ← كل

$n \cdot n$ ←

$T(n) = C + (n+1) + (n^2+n)$
+ n^2
← بفرقة

3 GHz means $\rightarrow 3 \times 10^9$ instructions in one second
3 جيجا هيرتز

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

نحتاج إلى أنه وكأنه هو ال max للوقت الذي سيأخذه البرنامج

*Q Assume the processor speed is 1 GHz
 $n = 2000000 = 2 \times 10^6$, find the time it will take.

Sol Time = $\frac{n}{\text{speed}} = \frac{2 \times 10^6}{1 \times 10^6} = 2 \text{ seconds}$

المرجع = المسافة
الزمن

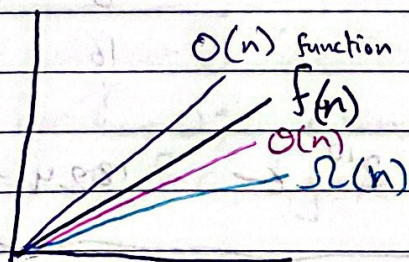
كل كل سرعة الزمن هو طول وبعدين مجموع كل

* Time Functions:-

① Big O $O(n)$: upper bound. "maximum of time that the process will take"

② Omega $\Omega(n)$: lower bound "minimum ..."

③ Theta $\Theta(n)$: upper and lower bound.



Sol

suppose

$$\frac{n}{2} = 0.5n$$

$$\frac{i}{n}$$

$$n-1$$

$$n-2$$

$$n-3$$

$$n-4$$

$$n/2 - 1$$

$$O(n)$$

* q for (i=n; i>0; i/=2)

Sol

suppose

$$n = 1024$$

$$\frac{i}{n}$$

$$1024$$

$$512$$

$$256$$

$$128$$

$$64$$

$$32$$

$$16$$

$$8$$

$$4$$

$$2$$

$$1$$

11 times

by

$$O(\log_2(n))$$

* q for (i=n; i>0; i/=3)

Sol

$$\log_3(n)$$

ما بظنه كأنه بفرضه قيمة n كثير كبيرة فهلوا مش راجع عن مليون واحد

Q# for (i=0; i<n; ++i) {

for (j=n; j>0; j/=10) {

$$n+1$$

$$\log_{10}(n)$$

sol

$$O(n \log_{10}(n))$$

* q for (i=0; i<n; ++i) {
 for (j=i; j<n; ++j) {
 ...
 }
 }

$O(n)$

Sol $\frac{i}{1} \frac{j}{0} \dots \frac{j}{n}$

$\left. \begin{array}{c} 1 \\ \vdots \\ n \end{array} \right\} n-1$

$$\Rightarrow 1+2+3+4+\dots+n-3+n-2+n-1+n$$

$$\begin{array}{r} 2 \quad n-2 \\ 3 \quad n-3 \\ 4 \quad n-4 \\ \vdots \\ n-1 \quad 1 \end{array}$$

$$= \sum_{i=1}^n i$$

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

$$= O(n^2) \text{ (Ignoring constant)} \underline{\underline{Q.E.D.}}$$

* q for (i=0; i<n; ++i) {
 for (j=0; j<n; ++j) {
 for (k=n; k>0; k-=4) {
 ...
 }
 }
 }

Solution $O(n^2 \log(n))$

Ex: for (i=0 ; i < n ; ++i) {

→ will stop → $i^2 \geq n$

$$\Rightarrow O(\sqrt{n})$$

Ex: int p=0

for (i=0 ; p < n ; ++i) {

p += i ;

}

sd $\Rightarrow p = 1 + 2 + 3 + 4 + 5 + \dots + n = \sum_{i=0}^n i = \frac{n(n+1)}{2} \approx n^2$

i	p
0	0
1	1
2	3
3	6
4	10
:	:
n	

Ex for (i=0 ; i < n ; ++i) {
for (j=0 ; j < i ; ++j) {
:
}
}

$$\Rightarrow 1^2 + 2^2 + 3^2 + \dots + n^2 = \sum_{i=0}^n i^2 = \frac{n(n+1)(n+2)}{6}$$

$$= \frac{1}{6} (n^3 + 3n^2 + 2n)$$

$$= \frac{1}{6} 2n^3 + 3n^2 + n$$

i	j
0	1
1	2
2	4
3	9
:	:
n	n^2

حل هين لاننا ما تعامل مباشرة مع n

باللوب التلاتي = $O(n^3)$


```

Ex  For(i=0; i<n; ++i) {
      For(j=0; j<n; ++j) {
      }
    }

```

→ $O(n^2)$

i \ j	0	1	2	...	n-1	n
0						
1						
2						
...						
n-1						
n						

* Recurrences :-

- Finding time used with recursive functions:-

```

* int Factorial(int n) {
    if (n==0)
        return 1;
    else
        return n * Factorial(n-1);
}

```

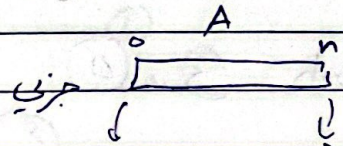
→ First, we find a Recursive Relation:-

$$T(n) = \begin{cases} d & ; n=0 \\ c + T(n-1) & ; n > 0 \end{cases}$$

$$\begin{aligned}
 &\rightarrow \left. \begin{aligned}
 T(n) &= T(n-1) + c \\
 T(n-1) &= T(n-2) + c \\
 T(n-2) &= T(n-3) + c \\
 &\vdots \\
 T(1) &= T(0) + c \\
 T(0) &= d
 \end{aligned} \right\} \begin{aligned}
 &\text{sum} \rightarrow T(n) = c + c + c + \dots + d \\
 &= c \cdot n + d \\
 &= O(n)
 \end{aligned}
 \end{aligned}$$

طالب
فقطاً حل السؤال السابق رج يكون أسرع "أقل وقت" من طريقة Recursion
لأنه كد مرة يستدعي ال Function

Ex merge sort



→ void mergeSort (int A[], int L, int H) {

if (L < H) {
mid = (L+H)/2;

mergeSort (A, L, mid);
mergeSort (A, mid+1, H);

Merge (A, L, H);
}

يحل القسم الثاني على تين كد رابط ينتج ، بعين بصيرت فيهم

$$\text{Time } T(n) = \begin{cases} d ; n=1 \\ 2T(\frac{n}{2}) + n ; n > 1 \end{cases}$$

Merge Func. ك
بستدعيه كل مرة

استدعي mergeSort
لو استدعيته 5 جزئ بدو
وهيك
كد مرة بقسم على تين
عشان المعادلة بقسم
على تين من طرف وكذا
المعادلة تقسم على 2 وأما $\frac{n}{2}$

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

$$T(\frac{n}{2}) = 2T(\frac{n}{2^2}) + \frac{n}{2} \quad \text{②}$$

$$T(\frac{n}{2^2}) = 2T(\frac{n}{2^3}) + \frac{n}{2^2} \quad \text{③}$$

sub ② in ①

$$T(n) = 2 \left[2T\left(\frac{n}{2^2}\right) + \frac{n}{2} \right] + n$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + n + n \quad \text{--- (4)}$$

Sub (3) in (4)

$$T(n) = 2^2 \left[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} \right] + n + n$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + n + n + n$$

* after k^{th} step

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

→ will stop $\frac{n}{2^k} = 1$

$$2^k = n$$

$$k = \log_2(n)$$

$$\rightarrow T(n) = n T(1) + \log(n) \cdot n$$

$$= d \cdot n + n \cdot \log(n)$$

(د، ثابت) (ن، متغیر)

$$O(n \log(n))$$

Ex: (مثال قبل)

$$T(n) = \begin{cases} d & ; n=1 \\ 2T\left(\frac{n}{2}\right) + 10 & ; n>1 \end{cases}$$

sol $T(n) = 2T\left(\frac{n}{2}\right) + 10 \dots \textcircled{1}$

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2^2}\right) + 10$ constant $\dots \textcircled{2}$

$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + 10 \dots \textcircled{3}$

$\textcircled{2} \rightarrow \textcircled{1}$

$T(n) = 2 \left[2T\left(\frac{n}{2^2}\right) + 10 \right] + 10$

$T(n) = 2^2 T\left(\frac{n}{2^2}\right) + 2 \times 10 + 10 \dots \textcircled{4}$

$\textcircled{3} \rightarrow \textcircled{4}$

$T(n) = 2^2 \left[2T\left(\frac{n}{2^3}\right) + 10 \right] + 2(10) + 10$

$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 2^3(10) + 2(10) + 10$

$+ 10(2^2 + 2 + 1)$

$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 10 \sum_{i=0}^3 2^i$

→ after K steps:-

$T(n) = 2^K T\left(\frac{n}{2^K}\right) + 10 \sum_{i=0}^K 2^i$

$\frac{n}{2^K} = 1 \rightarrow n = 2^K$

$K = \log_c n$ geometric series

Note Geometric $\rightarrow \frac{r^k - 1}{r - 1} = \frac{2^k - 1}{2 - 1} = 2^k - 1$

$$\rightarrow T(n) = n T(1) + 2^k - 1 = d \cdot n + n - 1$$

Final $\Rightarrow \underline{O(n)}$
ans

Ex: $T(n) = \begin{cases} d & ; n=0 \\ 2T(n-1) + c & ; n>0 \end{cases}$

Sol $T(n) = 2T(n-1) + c \quad \dots (1)$
 $T(n-1) = 2T(n-2) + c \quad \dots (2)$
 $T(n-2) = 2T(n-3) + c \quad \dots (3)$

(2) in (1)

$$T(n) = 2[2T(n-2) + c] + c$$

$$T(n) = 2^2 T(n-2) + 2c + c \quad \dots (4)$$

3 in 4

$$T(n) = 2^2 [2T(n-3) + c] + 2c + c$$

$$= 2^3 T(n-3) + 2^2 c + 2c + c$$

$$= 2^3 T(n-3) + c \sum_{i=0}^2 2^i$$

after k steps:

$$T(n) = 2^k T(n-k) + c \sum_{i=0}^{k-1} 2^i$$

$\rightarrow n-k=0 \rightarrow n=k$

$$= 2^n T(0) + c (2^k - 1)$$

$$2^n d + c (2^n - 1) = O(2^n)$$

24.09.2021

Linked lists

Thursday

* Array : The simplest data structure

مثال →

① Fixed size, ~~cannot change its size~~ ~~لا يمكن تغيير حجمه~~

→ it leads for wasting space.

(certain)

② insert new element ~~in~~ in a position. → takes more time

③ Delete an element. "we need to make shifts."

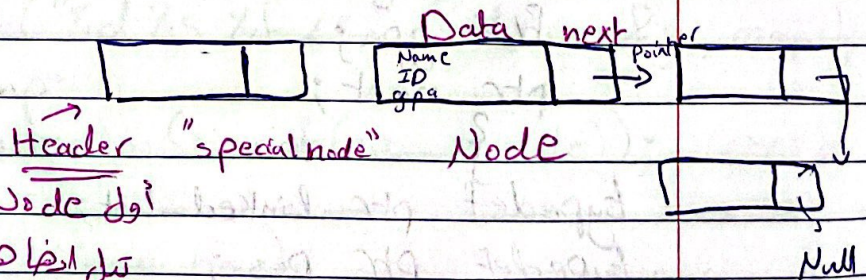
④ The need of sequential memory cells.

Defo- List : it is a collection of elements that are organized sequentially.

- The array follows the strategy of allocating the memory for all its elements in one block.

- The Linked List uses the strategy of allocating memory for each element separately and only when necessary.

- The linked list consists of a series of structures called nodes.



Node أول ويكون فارغة من البيانات عنان
Node آخر يكون فارغة من البيانات

أخرا

* Types of Linked List:-

- Single L.L (Normal)
- Doubly L.L
- Circular L.L
- Doubly circular L.L.

* Operations on Linked list:

- | | | |
|-----------|---------------|-------------|
| 1) create | 2) insert | 3) delete |
| 4) search | 5) is Empty | 6) get Size |
| 7) sort | 8) print List | |

* The creation of L.L :-

main

```
struct Node;
```

```
typedef struct Node * ptr;
```

```
struct Node {  
    char Name [25];  
    int ID;  
    char major [20];  
    float gpa;  
    ptr next;  
}
```

```
typedef ptr Linked_List;  
typedef ptr pos;
```

position

pointer to node

typedef → for
creation of new
datatype

}
نوع البيانات
الخاص

بترجع pointer

```
Linked_List L;
L = (Linked_List) malloc (sizeof (struct Node));
L->next = NULL;
```

هنا انشأت
السير وحج
باليموري
مساحة تفين

ptr next 2 byte
gpa 8 byte
major 20 byte
ID 4 byte
name 25 byte

دعاً أول مرة أدخل لل Null
ال arrays التي عرفتم قبل

2) Insert in L.L :-

```
void insert (Linked_List L, pos p, ptr temp) {
```

```
if (temp != NULL && L != NULL) {
    temp->next = p->next;
    p->next = temp;
    printf("new node is inserted\n");
} else {
    printf("Error in insertion\n");
}
```

هنا الخطوة لأنه لو أنق
ال temp على NULL
run-time error
Detection لل bug
قبل ما نبدأ

main

```
int id;
float g;
char nm[25], major[20];
```

while (ID >= 0)

```
printf("Enter the student's Information");
```

أدخلك
أدخلك

```
scanf("%d %s %f %s", &id, nm, &g, major);
```

```
ptr temp;
```

```
temp = (ptr) malloc (sizeof (struct Node));
```

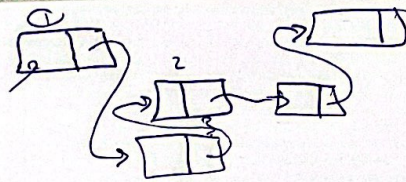
(ptr) هو

```
temp->ID = id
```

data التي نريد
casting
نوع ال data

```
strcpy (temp->Name, nm);
```

```
temp->gpa = g;
```

pointers

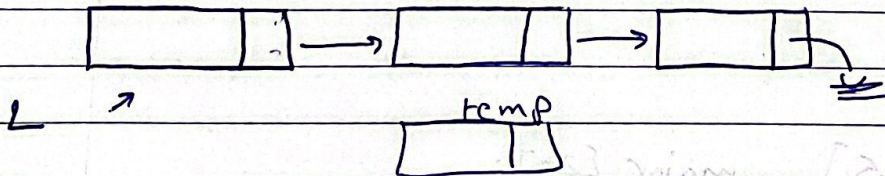
insert(L, L, temp);

هناك دليلا يبين ان كل node جديدة في Node ال header "عالبداية"
 لكن بوضو بنوع اضيفها وين بي انا ، بقدر ادمق اسم ال (L) وال (temp) تالست
 ال Node الى انا موصية اضيف عليها "قبلا او وراها" Insert(L, P2, new)

لو بي اضيف nodes الى آخر ال list
 Function insert_last
 → Void insert_last(Linked_List L, pos P2, ptr temp) {

pos P2 = L → next;
 while (P2 → next != NULL)
 P2 = P2 → next;
 P2 → next = temp;
 temp → next = NULL;

if statement لا يذاك في ال temp و L Null ≠



L: The name of the Linked List
 P: the position where the new node to be inserted
 temp: new node

Q * How to print the L.L?

```
void printList (Linked_List L) {
```

```
    ptr p;
```

```
    p = L → next;
```

```
    while (p != NULL) {
```

```
        printf ("ID: %d \t Name: %s \n",
```

```
            p->id, p->name);
```

```
        p = p->next;
```

```
    }
```

```
}
```

Function for search:-

```
Ptr search (Linked_List L, int id) {
```

```
    ptr p;
```

```
    p = L → next;
```

```
    is found = 0
```

```
    while (found == 0) (p != NULL) {
```

```
        if (p->id == id) {
```

```
            return p;
```

```
        }
```

```
        p = p->next;
```

```
        return NULL NULL;
```


* Delete:- I can delete nodes by:

- ith node
- specific property
- all nodes.

رقم "نادر" ما يستعمل
هناك بيانات أخرى

Function
for
deleting
by ith

void delete_ith (Linked-List, int i) {

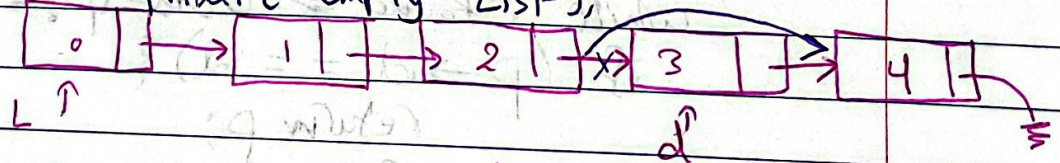
int a = 0;
ptr p = L; d;

if (!isEmpty(L)) {
while (p->next != NULL && a < i-1) {
p = p->next; // الرقم "a" node
a++; // إلى ربي أحذفه.
}
d = p->next;

هناك ptr p التي
على ربي التي بدي أحذفها

p->next = d->next;
free(d);

} else printf("Empty List-");
}



لوبي أحذف أوله إلى بجالي 2 تأخر على 4 بعين حذف في خشان ما دقيج
باجي معلومات ال L.L

Ex. delete the node Name is Ahmad "Function"

Pointers

```

void delete (Linked-List L, char Name[]) {
    ptr p = L, d;
    if (!is Empty (L)) {
        while (strcmp (p->next->Name, Name) != 0)
            p = p->next;
        d = p->next;
        p->next = d->next;
        Free (d);
    }
    else
        printf ("Empty List");
}

```

لو في مكان حرج غير انه احمد && خذ الياك

delete all Nodes; لازم واحد واحد نخلصه

لانه لو حذفنا الاول بس الباقيين في ذاكرة memory حتما حرجنا لـ data wasted

→ Void delete (Linked-List L) {

ptr p = L->next;

while (p != NULL) {

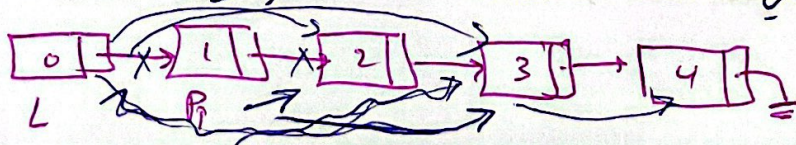
L->next = p->next

Free (p)

p = L->next;

}

Free (L);



الوقت الذي
يأخذه

عشيرة ما احذف واحد
اشترط ان يكون في

لا يغير تأثير على ولا اشترط ان يكون اول ووحيد

لو نبي اعد ال Function السابق d recursion

```
→ void delete(Linked_List L) {  
    if (L → next == NULL) {  
        Free (L);  
    }  
    else {  
        ptr p = L → next;  
        L → next = p → next;  
        Free (p);  
        delete (L);  
    }  
}
```

لو نبي اصيب Time "بفعل" $O(n)$ بـ "ن" .

$$\rightarrow T(n) = \begin{cases} d, & n=0 \\ T(n-1) + c, & n>0 \end{cases}$$

لأنه كلمة حذف node

نولاً حقا لو التبين $O(n)$ لكن ال recursion ياخذ وقتاً أكثر
لأنه يستدعي Function كلمة .

Function getSize:

```
int getSize(Linked_List L) {  
    int counter = 0;  
    ptr p = L;  
    while (p != NULL) {  
        ++ counter;  
    }  
}
```

$O(n)$

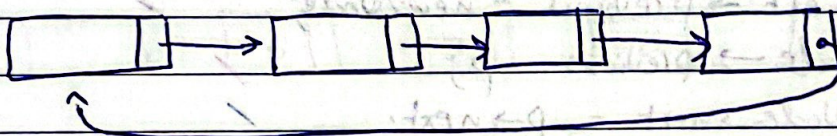

```

    p = p -> next;
}
return counter;
}

```

* **Time Complexity** = constant $\leftarrow$ يعني global variable
 قبل ال main ويعرفه $\text{int size} = 0$ وكل ما عمل insert
 node ++size وكل ما حذف node --size
 وانا في function ال get Size وانا في function ال get Size
 بس return size وسكون من ال size كل ال مكتوب جواه

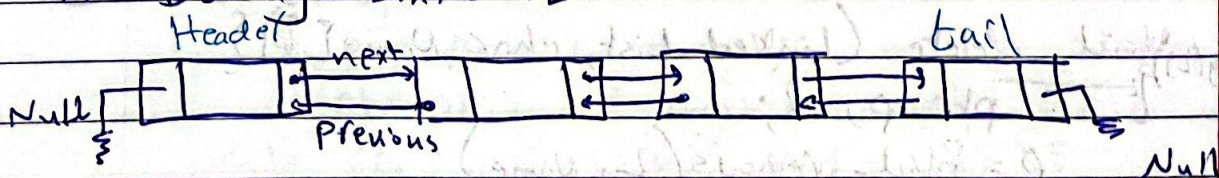
* Circular Linked List:



الفرق هو انه آخر node بدل ما تاخر على NULL بتروح بتاخر
 علوق node $\leftarrow$ يعني $\text{p} \rightarrow \text{next} = \text{L}$ هو ان ال p

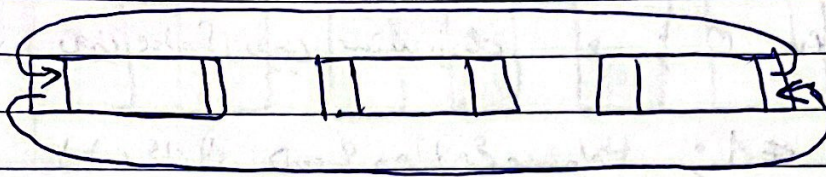
بالنسبة ال Functions تاسمها بتكون نفس قبل لكن آخر
 بدل ما اكتب $\text{L} \rightarrow \text{next} = \text{NULL}$ بكتب $\text{L} \rightarrow \text{next} = \text{L}$ وبس

* Doubly Linked List:



بدل ما اكتب next واحد next واحد previous واحد

4 Circular doubly LL:-

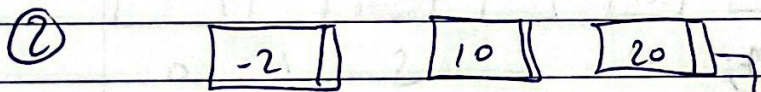
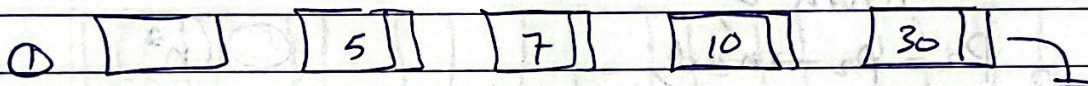


مرتبه ترتيب و تاييد

Q Assume you have 2 sorted Linked Lists, write a function that determine if these two linked lists are disjoint or not.

→ not common elements.

فكره الكل : انما نحل pointer نكافئ عنصري من اخر رقم وبقارن



بما انهم كل ال -2 اصغر من غيره ✓ ولا يتساوون بها
 يرجع الى القيمة نفس الشيء هل يتساوي القيمة - لا وهي اصغر
 من ال 10 وال 20 من ال 10 - لا وهي اصغر
 فيبقى ال 20 بال L1 تاخذها هل القيمة 10 = 20 - نعم
 فبرجع false

```
Function: int Disjoint (Linked-List L1, Linked-List L2) {
    ptr p1 = L1 → next;
    ptr p2 = L2 → next;
    while (p1 != NULL && p2 != NULL) {
        if (p1 → data < p2 → data)
            p1 = p1 → next;
        else if (p2 → data < p1 → data)
            p2 = p2 → next;
        else
            return 0;
    }
    return 1;
}
```


$P_2 = P_2 \rightarrow \text{next};$

else

$\{ \text{return } 0; \}$ → disjoint (فصلی) False (نہیں)

$\{ \text{return } 1; \}$ → Loop (حل) ملاقات مسالوہ
disjoint (فصلی) True (ہاں)

$O(n)$
↑
مقام

nested loop $\leftarrow O(n^2)$ ← sorted (منظم) * $O(n^2)$

→ while ($P_1 \neq \text{NULL}$) {

$P_2 = L_2 \rightarrow \text{next}$

while ($P_2 \neq \text{NULL}$) {

if ($P_1 \rightarrow \text{data} == P_2 \rightarrow \text{data}$)

$\text{return } 0;$

$\{ P_2 = P_2 \rightarrow \text{next};$

$O(n^2)$

$\{ P_1 = P_1 \rightarrow \text{next};$

$\text{return } 1;$

* Radix (Sort:-) ترتیب دے گا

- used for integers and strings.

Examples Assume 876 253 426 913 81 51 7
64 27

steps = # digits in max num.

"اکیس منزلیں ارقام ہیں من 3 منزلیں ← 3 steps"

Step # 1: حسب منزلة الأعداد

(n)

		51		913			426	27		
		81		253	64		876	7		
0	1	2	3	4	5	6	7	8	9	

Step # 2: حسب المنزلة التالية من اليمين

(n)

			27			253				
7	913	426			51	64	876	81		
0	1	2	3	4	5	6	7	8	9	

Step # 3: حسب المنزلة الثالثة من اليمين

(n)

81	64	51	27							
				253		426			876	913
0	1	2	3	4	5	6	7	8	9	

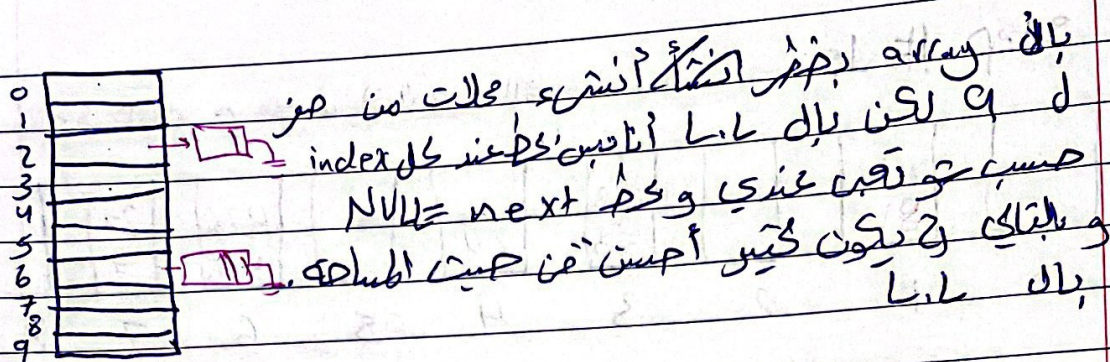
→ 7, 27, 51, 64, 81, 253, 426, 876, 913

$$T(n) = n + n + n = O(n)$$

n x n 2-D array
 step 1
 step 2
 step 3

Assume $n = 10^4$
 ⇒ 2 Arrays (at least) $n \times n = 2 \times 10^4 \times 4 \text{ bytes} = 8 \times 10^4 \text{ bytes}$

→ 2 Linked-List $2 \times n = 2 \times 10^4 \times 4 \text{ bytes} = 8 \times 10^4 \text{ bytes}$



سؤال: لو بي أعمل Function جيبي خانة "منزلة" مملووية من رقم

```

→ int getDigit (int n, int i) {
    if (i == 0)
        return
    while (i > 1) {
        n = n / 10;
        --i;
    }
    return n % 10;
}

```

لو عندي اسماء بي أرتبها حسب Radix ، بعد نفس الآن هناك ، بسوف أحوال اسم وطول Steps حساب ، وكل ما عندي اسم ملا أرتج من هين للسؤال ، بجيب ملا عندي الاسم محل الفاضي

اما NULL أو space Nuzmai Sara Heba Khalid

شرح كود الورقة بال L.L.A Cursor Imp

X- ملا ما بيكون في Pointers ← مرموز Dynamic Memory Allocation بروج

Array : fixed size

هلا هو مرموز حاصر 100 ، لكن نبيد أعمل أكثر من L.L.A زي B و C

والع عنبر ال A

05.10.2021

Stack

Tuesday

def stack: list of elements with restriction on the insertion and deletion, whereas they are performed only on one position which called the top of stack

in stack : insert : push

delete : pop

في صف واحد " أنا باخذ الأول أو الأخير وحده واحد "

* Applications on stack:

[1] Functions call: Functions A, B, C مثلاً لو عندي
في A نأخذ B نستدعي C والـ A نستدعي B الذي هو أصلاً استدعي

[2] Redo / Undo

[3] expressions evaluations

[4] browsers → google → Facebook → link game - - -

* Implementation For stack: ← 1- Linked List / صف عاقل / أمزاني
أي قبل "لكن" أمزاني صف 2- array / مكاننا صفني

الـ stack على كبري ممكن نستخدم array لكننا

```
#define MAXSIZE 100;
```

```
struct stack {
```

```
int Elements[MAXSIZE];
```

```
int Top;
```

```
};
```



```
typedef struct stack STACK;
```

```
STACK s;
```

```
s.Top = -1;
```

هيك انشاءات stack

← empty stack

Function

push

```
void push (STACK s, int e) {
```

```
if (s.Top < MAXSIZE-1) {
```

```
++s.Top;
```

```
s.Elements[s.Top] = e;
```

```
}
```

```
else {
```

```
printf("stack overflow \n");
```

```
}
```

```
}
```

بدي اضافة انا في حال اضيفنا على
عشان بدي من -1
s.Elements[s.Top] = e;

Function

pop

int

```
pop (STACK s) {
```

```
if (s.Top > -1) {
```

```
int e;
```

```
e = s.Element[s.Top];
```

```
--s.Top;
```

```
return e;
```

```
}
```

```
else {
```

```
printf("stack is empty \n");
```

```
return -1;
```

```
}
```

```
}
```

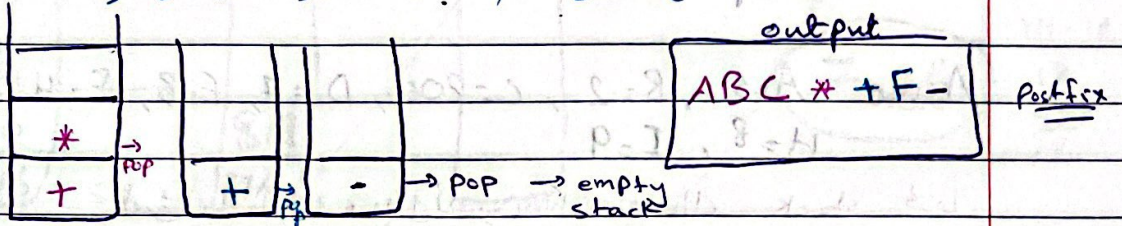
لو عايز نخرج القيمة ناعنا e = -1 كين بدي جوا اذا هي
غير فيها -1 وبينا نخرج ولا هي فاضيق؟ ← ما استخرج ال Function جوا
"كين لو -1 > s.Top ← استخرج pop ← عر هيك ما تستخرج"

العملية الحسابية تكون جوا

① Conversion from Infix expression to Postfix

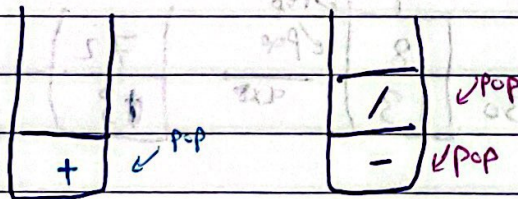
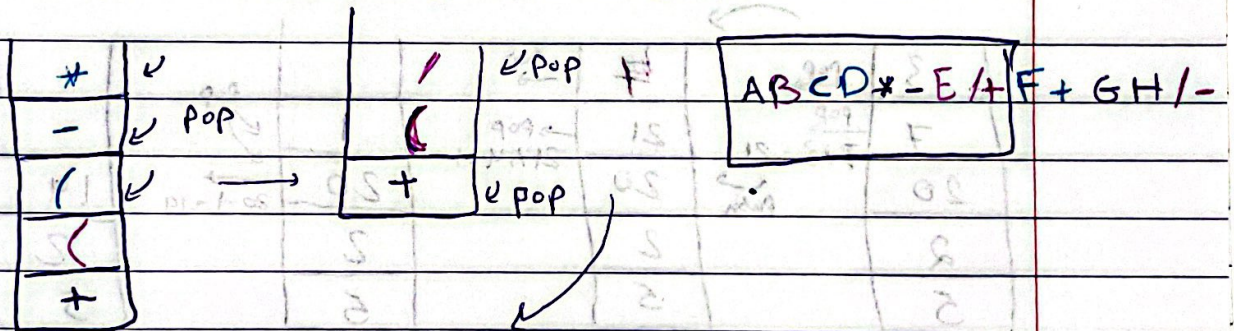
أولاً في جدول معين نضع تعبيرنا

الفكرة انه حين العملية على stack ، ونضيف فوقها الى اليمين اولوية
فقط ، لكن مثلاً المجموع والجمع لهم نفس الأولوية ونضعها في مكان آخر



كلنا نعلم pop نزلنا على output

Ex $A + (B - C * D) / E + F - G / H$



Remarks: الأقواس يجب كتابتها في ما قبلها ، ونحن كأنه من موجود الى قبلها
فبعد pop الى حولها ، وما أقبل نهاية القوس () ، نزلنا على output
أي أوله) ، الأقواس لا نضيفها على الناتج ، ولوبيد أضفنا عملية جمع
وكان موجود أصلاً على output ، ونضيف الجبرده

نفس أولية * قيمة

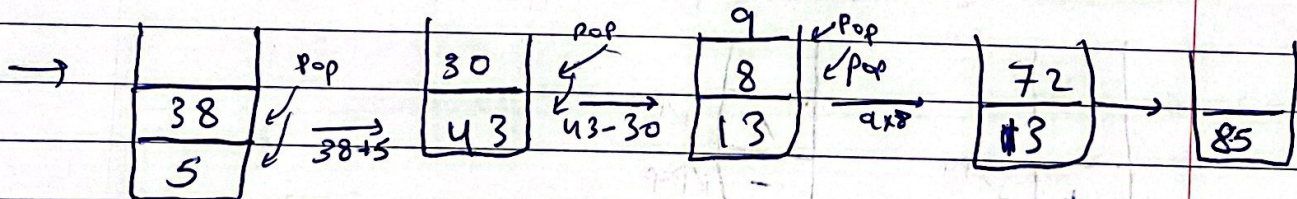
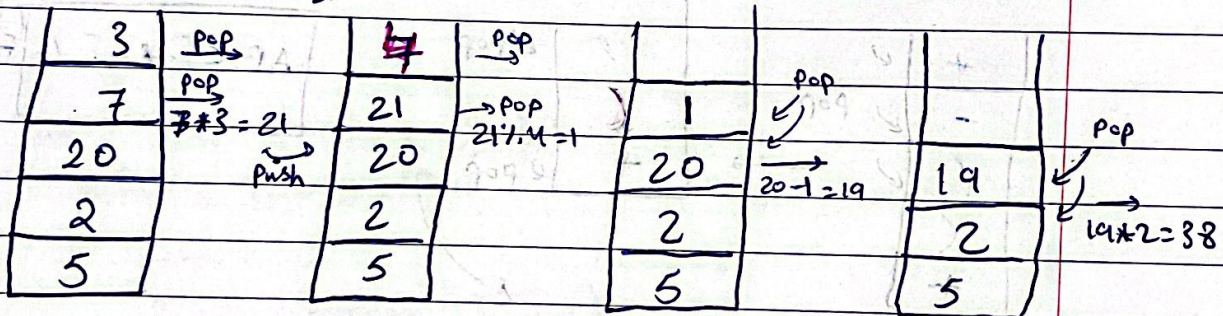
Ex Convert to postfix: $A + B * (C - D * E \% F) - G + H * I$

Sol ABCDEF * % - + G - HI * +

2 Postfix expression evaluation

Assume $A=5, B=2, C=20, D=7, E=3, F=4, G=30, H=8, I=9$

حسب ناتج السؤال السابق، وبذلك أضفنا الـ variables على stack كما
الآن عملية حسابية، وخرج بعض الخرج 2 elements وبذلك علينا
هذه العملية، وبذلك push على stack كان من الناتج.



عشان أعرف إذا كان معي - لأزني آخر خطوة يكون فيها جواب واحد
ومفاتيح وخطأ أو أي العمليات لأزني على الأقل أدخل قيمة متغيرين.

Evaluate $ABCD * - E / + F + GH / -$

$A=2, B=3, C=4, D=5, E=6, F=7, G=8, H=9$

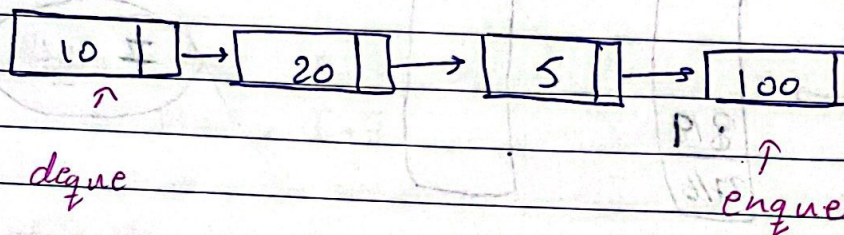
0.7.10.2021

Queue

Thursday

def

Queue: is a list, whereas the insertion is performed always at the end of queue, while deletion is performed at the beginning



* Queue is FIFO "First in First out"

First In last out — stack use

- Applications :

- 1) Printers
- 2) calling centers
- 3) Registration system

* Implementation:-

1- Linked List

2- Array

← how?

```
#define SIZE 5;
struct Queue {
    int Data[SIZE];
```

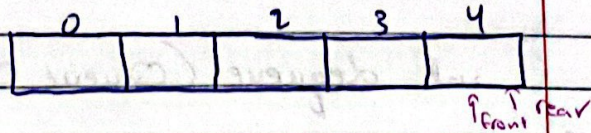
```
    int front;
    int rear;
```

```
} Q; ① ps → ps (upar) * Q ke queue var slo
```

```
Q.front = size - 1;
```

```
Q.rear = size - 1;
```


ما يقبل عناصر على كيو ملين ولا يقبل من هو ذا



Function
الأقفافه

void enqueue (Queue Q, int x) {

int temp = Q.rear;

if (Q.rear == Size-1)

Q.rear = 0;

else

++ Q.rear;

هو ديس (بفوق) على 0 1 2 3 4
إذا كل موقع على 0
(عشان أحرك Pointer وأعطى موقع صح)

if (Q.rear == Q.front) {

printf("Queue is full\n");

Q.rear = temp;

else {

Q.Data[Q.rear] = x;

}

}

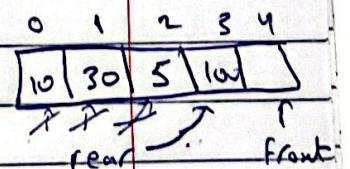
عشان لا يكون إذا
مستعمله ولا يقبل إضيف

O(1)

constant

Assume 10, 30, 5, 100, 70, 60

Q.front	Q.rear	x	temp
4	4	10	4
	0	30	0
	1	5	1
	2	100	2
	3	70	3
	4		
	3		



الفاضي يحبس needle و يمكن يكون أي مكان

```
Func int dequeue (Queue Q) {
dequeue
```

if (!is Empty (Q)) {

if (Q.front == size - 1)

-Q, front = 0;

else

++ Q. front;

```
return Q.Data[Q.Front];
```

3

else {

```
printf("Empty Queue\n");
```

return -1;

3

3

لازم "الأفضل" قبل ما استنسخ هاد ال Function أفصح من الأساس، بال main
إذا isEmpty وعضان قيمة ال (1-) تابع ال Func ورج قترينك

```
int isEmpty(QueueQ){
```

return Q.front == Q.rear]

فولياً إنما ما يحذف لكن يخفى فوق القيمة الموجود أو هالها hide هو هوفها
 يروح القيمة فليس لا شيء الي كتبه 6 لكن لو في 1.1.1 عنجز فهو حذف قيمة
 كلها من 1.1.1

① (1) constant

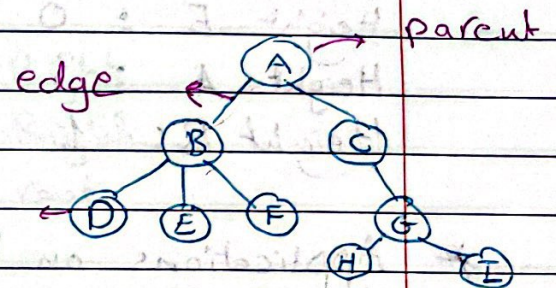
12/10/2021

Trees

Tuesday

* Linear data structure:
 "كل شيء يسبقه والي بعده"
 "كل شيء قبل آخره هو هيك"

Def Tree:- is a non-linear data structure
 or is a finite set of nodes, together with
 a finite set of edges, that define the
 parent-child relationship.



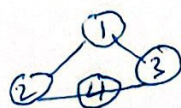
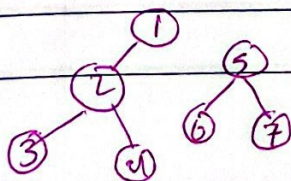
Def Path from node to node m : set of edges/
 nodes from node 1 to node m , with length of $m-1$

Ex Path from A to H: A, C, G, H

4 nodes

* Tree properties:-

- 1- It has a designated node, called root, that has no parent A
- 2- Every node, except root, has exactly one parent.
- 3- Every node has zero or more children.
- 4- There is a unique directed path from the root to each node.



① → tree ✓

X not a tree because there are 2 roots!



Def * Depth of node v :- the length of the path from the root to node v

ex Depth of H : 3

Depth of B : 1

Depth of A : 0

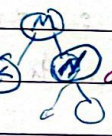
Def: * Height of node v : The longest path from node v to the leaf node.

Height E : 0

Height A : 3

Height B : 1

* Applications on tree:-

1- Dictionary  Searching

2- Directory

3- Decision Tree

4- File directories

* Implementation of trees:-

Struct Node {

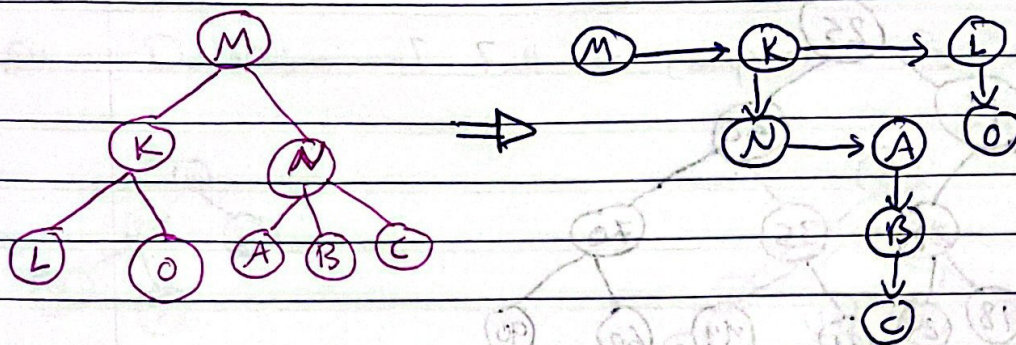
ptr next;

ptr siblings;

}

(Note) $\rightarrow$ Siblings: nodes with same parent

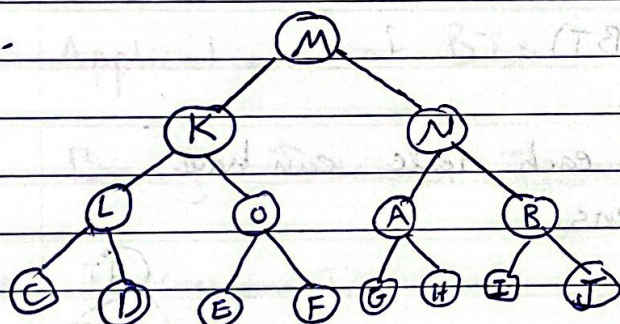
ptr next
ptr sibling ↓



Defn Tree Traversal:- Visiting all nodes. By:-

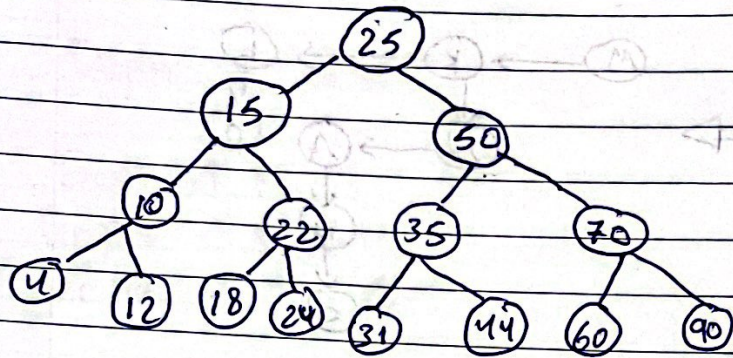
- 1- preorder: root - Left - Right
- 2- Inorder: Left - root - Right
- 3- postorder: Left - Right - root

Ex:-



1. pre-order: M, K, L, E, D, O, E, F, N, A, G, H, B, I, J
2. Inorder :- E, L, D, K, E, O, F, M, G, A, H, N, I, B, J
3. Post-order : E, D, L, E, F, O, K, G, H, A, I, J, B, N, M

Ex: Write the pre, In and post order traversal for the following tree:-

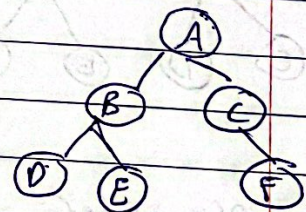


- 1- Pre-o: 25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 60, 90
- 2- In-o: 4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 60, 70, 90
- 3- Post-o: 4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 60, 90, 70, 50, 25

الترتيب حسب 3 2 1 وبقاع رسمي :

* Binary Tree:- (BT)

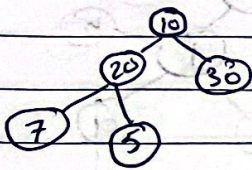
def is a tree in which each node can have maximum 2 childrens.



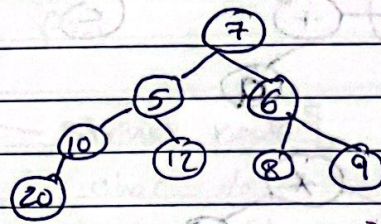
* Types of Binary Tree:

- [1] Full BT : every node has ^{zero} 0 or 2 children
- [2] Complete BT: every level, except the last, must be completely filled. The last level is filled as far left as possible.

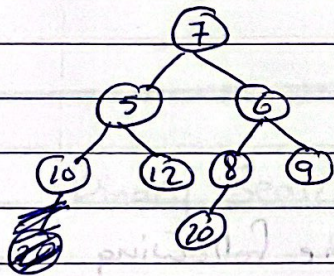
Ex Determine if Full or complete BT



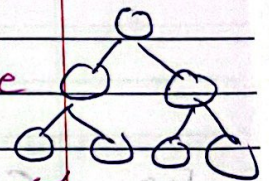
(Full BT) and (complete)



(Not Full BT)
complete BT



Not complete nor full
Just BT (filling must be from left to right)
Full + complete



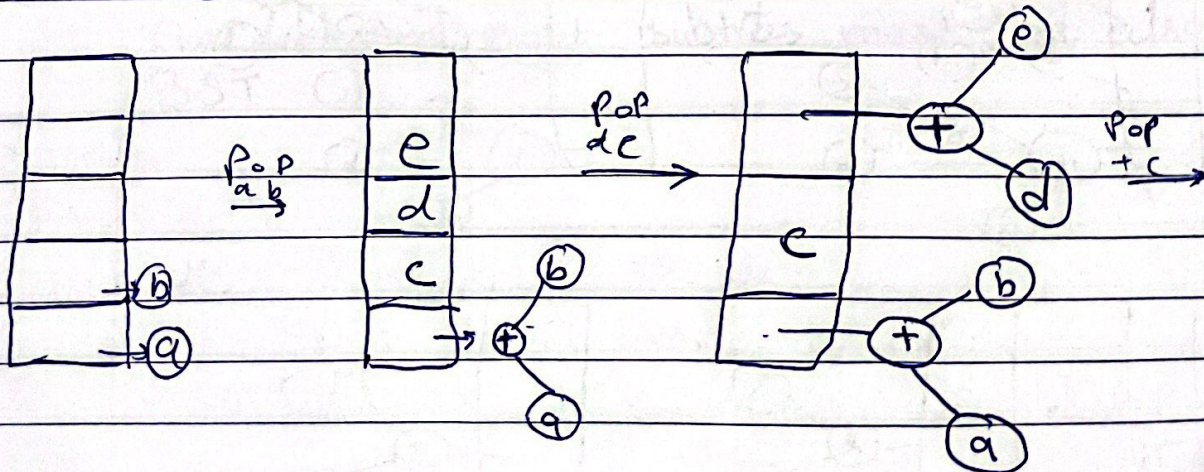
* Applications of BT:-

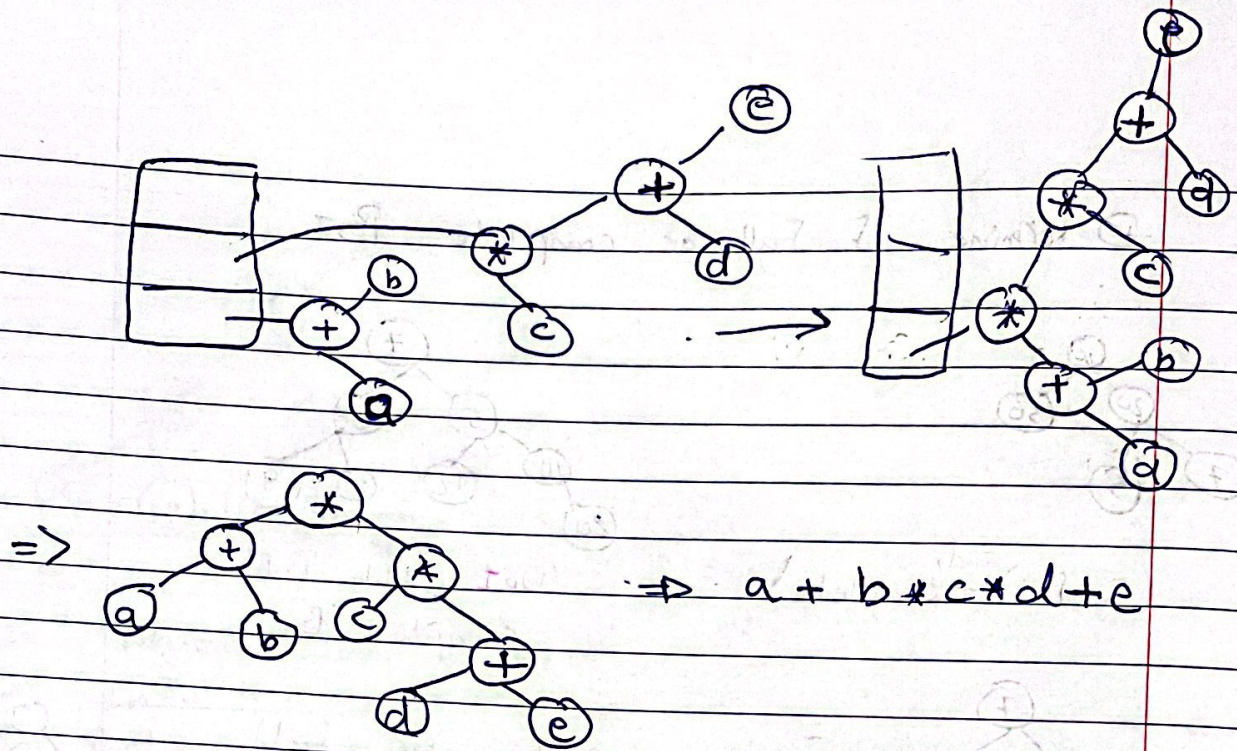
1- Binary Decision Tree

2- Expression Tree.

Ex: $ab + cde + **$

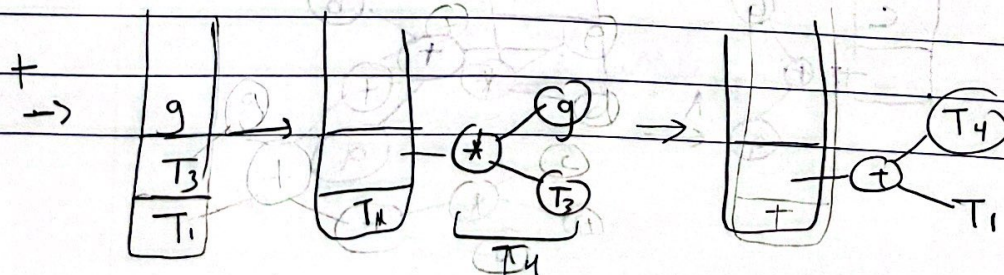
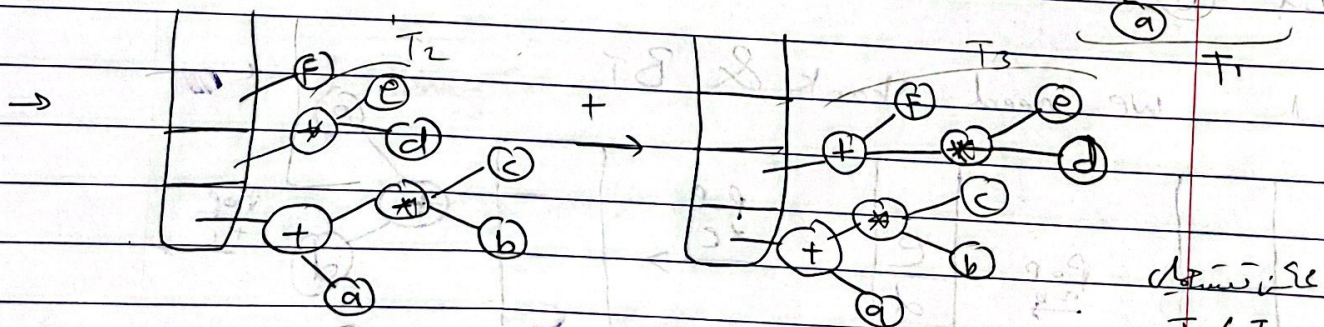
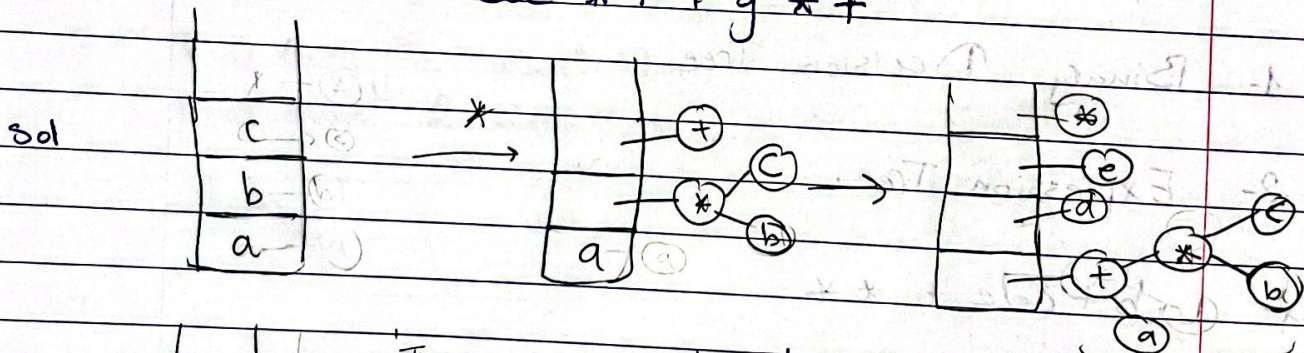
Sol we need stack & BT

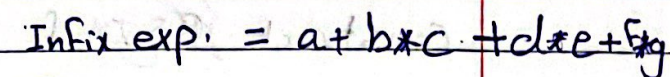




Ex: Construct the exp. tree for the following postfix exp.

$abc * + de * f + g * +$





post

```
char data;
```

ptr left;

ptr right;

3.

def BST: is a binary Tree, that satisfies the following properties:

- node's Key → فليكن رقم واحد
ال struct الذي ما ينكر (ID) مثلا ، ويصير أبيض وأحمر

- 3- The left and right subtrees must be also BST

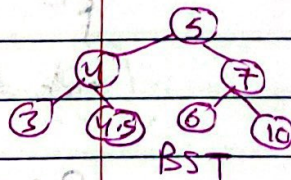
node {

```
char [ ] name;
```

```
int ID;
```

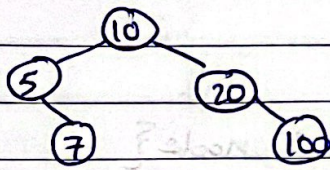
final gpa:

ptr left;
3 ptr right;



Q-1) Construct BST of 10, 20, 5, 7, 5, 100

Sol



منویہ اکثر و استعمال کرتی

* Operations on BST:-

- creation
- insertion
- deletion
- search
- Traversions.

* creation:

```
struct Node;  
typedef struct Node * BST;  
struct Node {
```

یہیں طالب/طالبہ
لکھیں گے

```
    BST left;  
    BST Right;  
}
```

```
int main() {
```

```
    BST T;  
    T = (BST) malloc(sizeof(struct Node));  
    T->left = NULL;  
    T->Right = NULL;  
}
```


node ptr, i.e. node

Search

BST Search (BST T, int key) {

BST current = T;

int Found = 0;

while (current != NULL && !Found) {

if (current->ID == key)

Found = 1;

else if (current->ID > key) {

current = current->left;

else

current = current->right;

}

return current;

}

$O(\log(n))$

Q. Write search in Recursive form.

BST Search (BST T, int key) {

if (T == NULL) {

if (T->id == key)

return 1;

else if (T->id > key)

return search (T->left, key);

else

return search (T->right, key);

}

return NULL;

}

$$T(n) = \begin{cases} d & n=0 \\ T(\frac{n}{2}) + c & n>0 \end{cases}$$

$O(\log(n)) \rightarrow$ if balanced BST
if the tree ideal

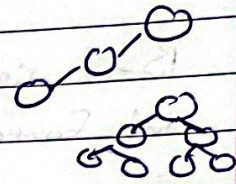
Syntax error

Time comp. For search Functions:

Worst case $O(n)$

Best Case $O(1)$ →

Balanced $O(\log n)$



فيما فيسكن دمايع كل النودز

void inorderTraversal (BST T) {

if (T != NULL) {

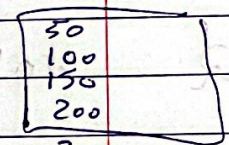
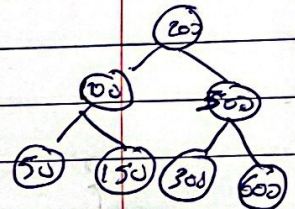
inorderTraversal (T->Left);

- Printf("%d \n", T->Data);

- inorderTraversal (T->right);

}

}



300
500
600

→ PostOrder: يبدا بين السمين = فقط
ولا بين pre: تفرز ان Left مع ال Right

* Insertion:-

- The idea is to do iterative traversal of the given tree.
- If we find a node whose left child is empty, we make a new key as left child of the node.

9

Function:- $\text{BST insert (BST T, int x)}$

```
T = (BST) malloc(sizeof(tree-Node));
```

$T \rightarrow \text{Data} = x$: البعدين السفلي طاريون في أكثر

```
else {
```

if $(x < T) \rightarrow \text{Data}$

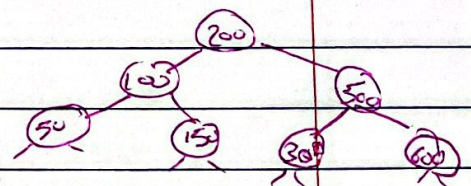
$\boxed{T \rightarrow \text{left} = \text{insert}(T \rightarrow \text{left}, x)}$

else if ($x > T \rightarrow \text{Data}$)

$T \rightarrow \text{right} = \text{insert}(T \rightarrow \text{right}, x);$

```
printf("The values should be unique");
```

3



120, 170, 650, 570, 900

عمل ال Functionality من void عنوان يرد الى subtrees مع بعض
رجوع طلائع ال 120 بالآخر يرجع ال subtree تابعها ورجوعها
تأيت ال 150 ورجوعها 200 ورجوعها 200

* Deletion

- Three cases :-

- 1- Node to be deleted is leaf, simply Remove it.
- 2- Node to be deleted has one child, copy the child to the node and delete the child.
- 3- Node to be deleted has 2 children, either copy the minimum from right "leaf" or copy the maximum left from left

Function: Find minimum

```
BST Find_Min (BST T) {  
    if (T == NULL)  
        return T;  
    else if (T->left == NULL)  
        return T;  
    else  
        return (Find_Min (T->left));  
}
```

```
Function BST delete (BST T, int x) {  
    delete if (T == NULL)  
        return T;  
    else if (x < T->Data)  
        T->left = delete (T->left, x);  
    else if (x > T->Data)  
        T->right = delete (T->right, x);  
    else {  
        →
```


Case $\neq \text{NULL}$

```
if (T → left && T → right) {
    BST temp;
    temp = Find-Min (T → right);
    T → Data = temp → Data;
    T → right = delete (T → right, T → left);
    T → left = delete (T → left, T → Data);
}
else {
    BST temp, c = T;
    if (T → left == NULL)
        temp = T → right;
    if (T → right == NULL)
        temp = T → left;
    Free(c);
    return temp;
}
```

حذف از درخت و جابجایی داده به درخت کوچکتر
و درخت بزرگتر را به جای آن قرار دادن

در این بخش 13 سوال است
* AVL Trees:-

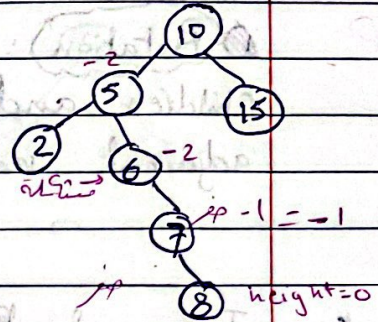
Def: AVL: is a binary search tree with height balance property.

- For any node v , the height of the subtrees of v is differ by at most 1.
- Any subtree of an AVL tree is also an AVL.

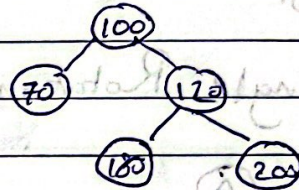
+3 خالص اربعة وعاليين 3

Ex

Not AVL →



Not AVL becoz it's not
Binary tree → $180 > 120$



✓ AVL فيستعمل 110 و 180

* Why AVL tree?

- insertion or deletion in BST can cause large imbalance.

Problems of BST

- The time complexity of search, insertion or delete of any node could be $O(n)$ in worst case.

لكن بال AVL كل ما فينا اوزان متغيرة، كل اوزان و اوزان
balance

- In AVL the balance property will ensure that the height of the tree will remain $O(\log(n))$

* Balancing is done by Rotation operation.

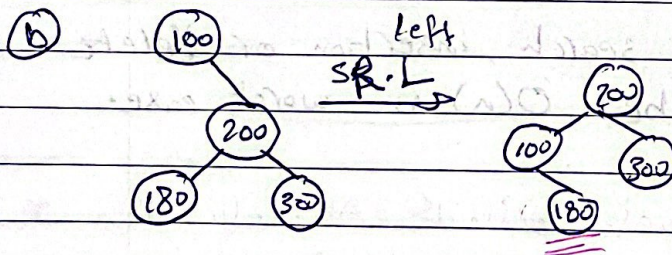
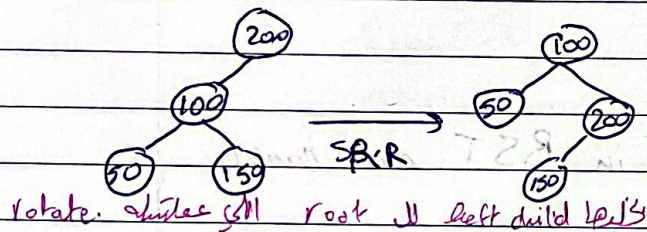
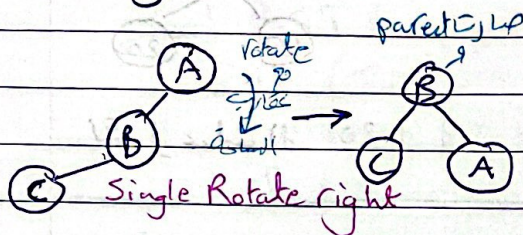
def Rotation: is a process of switching children and parents among two or three adjacent nodes.

* Types of Rotation:-

1-1 Single Rotation

2- double Rotation.

1) a-

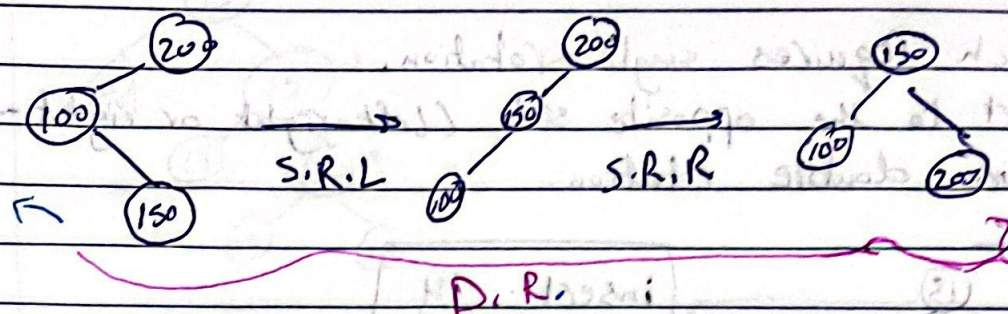


2- Double Rotation: - (first left then right or first right then left)

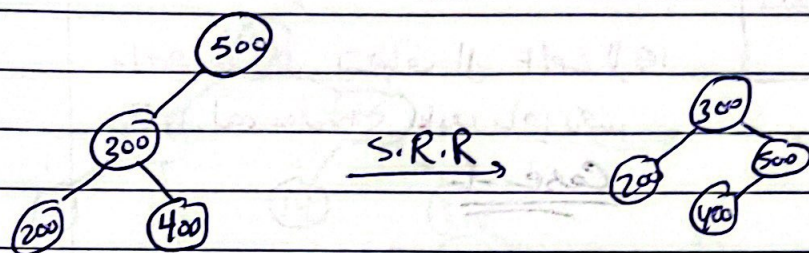
a- Double right-left

b- D. Left-Right

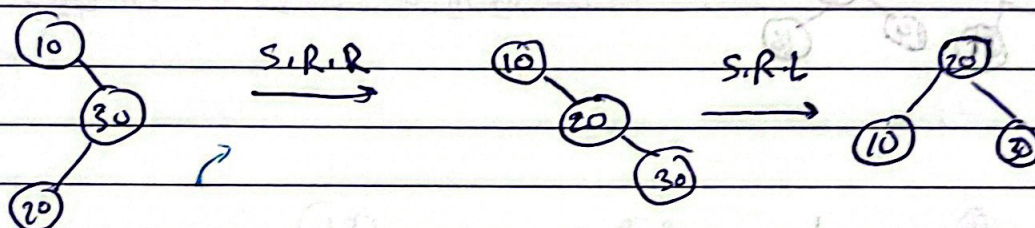
Ex



Ex



Ex



* Operations on AVL:-

1. insertion
2. deletion
3. Search
4. Rotation
5. get Balance
6. Traversal

* Insertion: same algorithm used in BST, in addition to balancing the tree.

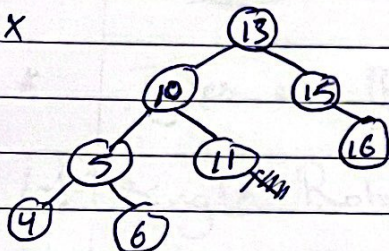
There are 3-cases in insertion

1. insert that does not cause imbalance.
2. insert to the same side (left-left or right-right)

which requires single rotation.

3- insert to the opposite side (left-right or right-left) require double rotation.

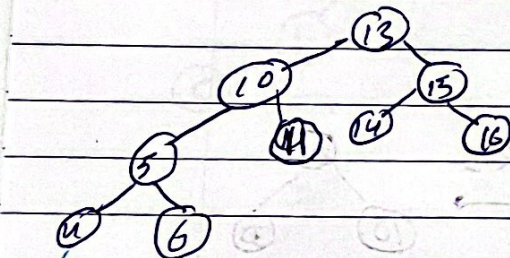
Ex



insert 14

ما فیہا مسئلہ کی پہچان ال Left ال 15
وہ پہلے balanced کای ما فیہا دیکھیں

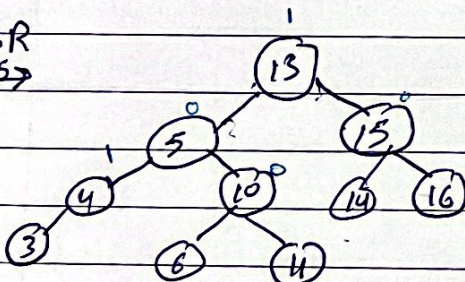
Case-1-



insert 3

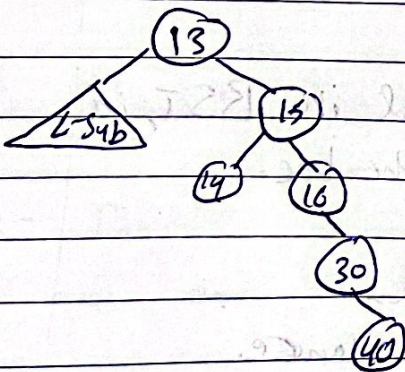
یہی ال Left ال 4 واپس مسئلہ 919

S.R.R
105

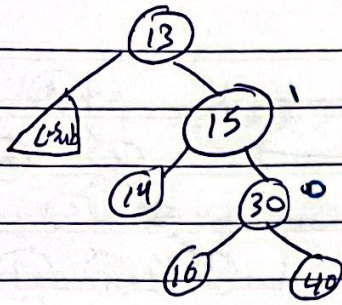


insert, 30 and 40

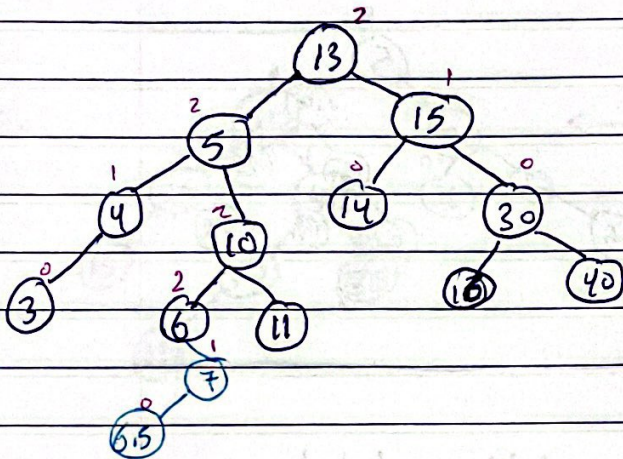
میں آئیے
دیکھیں



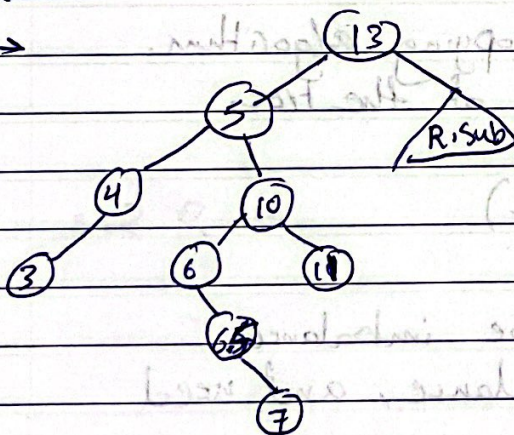
S.R.L
30-16



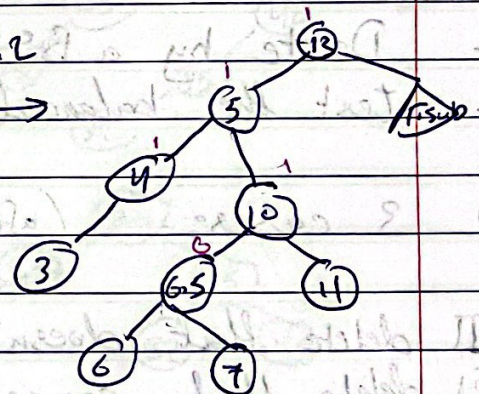
insert 7 $\hat{c} \rightarrow 6.5$



S.R.R
→

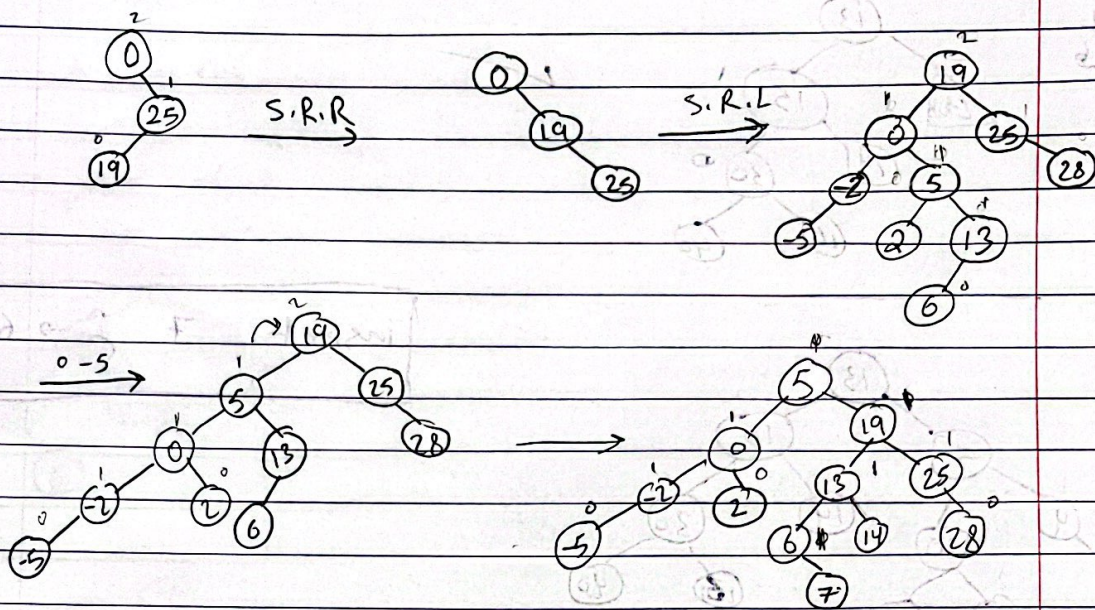


S.R.L



Ex: insert the following key values into an initially empty AVL tree.

0, 25, 19, 5, -2, 28, 13, -5, 2, 6, 14, 7



* Deletion:-

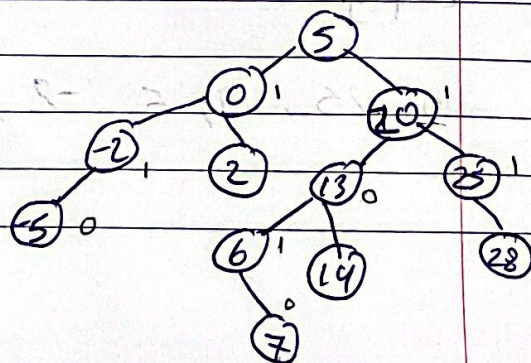
- Delete by a BST by copying algorithm.
- test the balanced factor of the tree.

→ 3 cases:- (after test)

- 1] delete that doesn't cause imbalance.
- 2] delete that cause imbalance, and need single rotation.
- 3] delete that cause imbalance, and need double rotation.

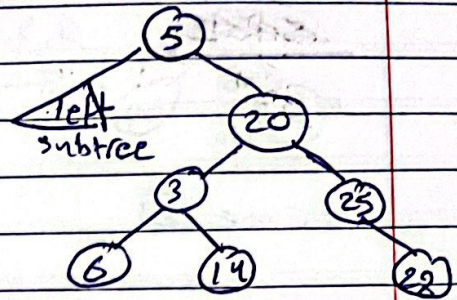
Ex

→ This AVL Tree is balanced

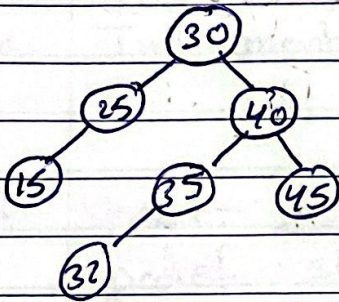


* delete 7 →

still balanced and ^{has} does not been affected. case 1

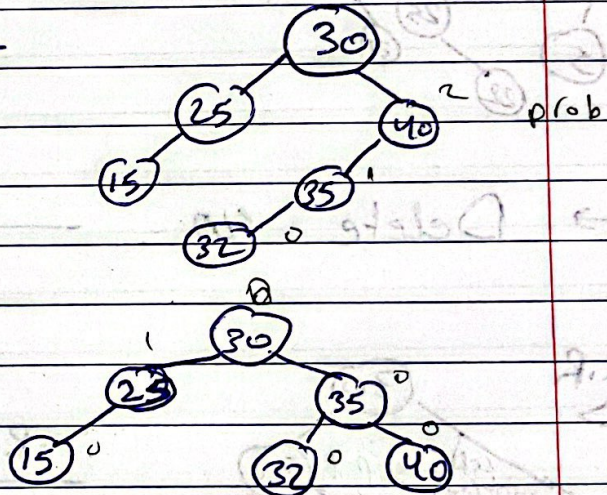


Ex



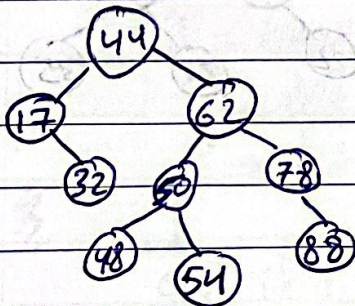
This is Balanced AVL Tree

→ delete 45 →



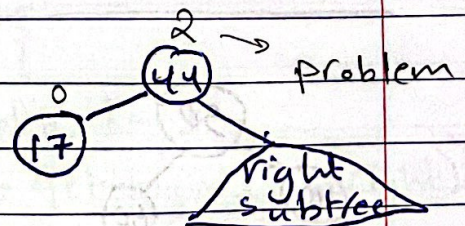
→ S.R.R →

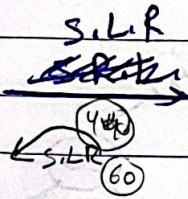
case 2



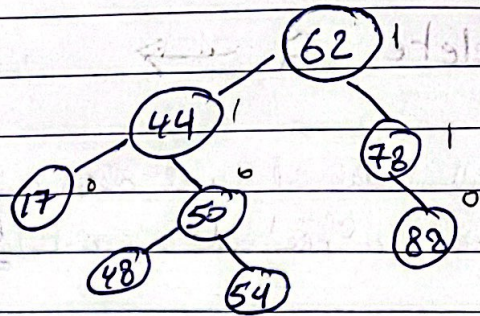
This is balanced AVL Tree

Delete 32





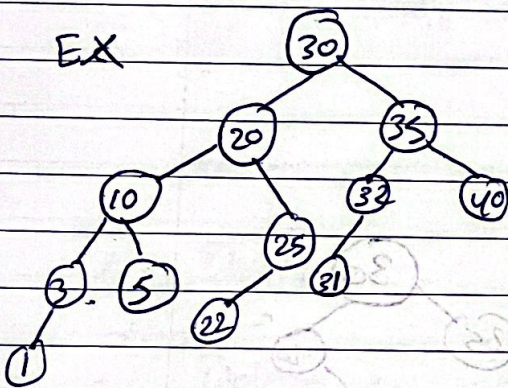
case 2



دائماً طارحس ما في balanced وفكري جيبي نود من الجهة الأكبر منها فونز
زي لليزان جيبي من الجهة الأقل لالأخف مسكان يتوارثوا

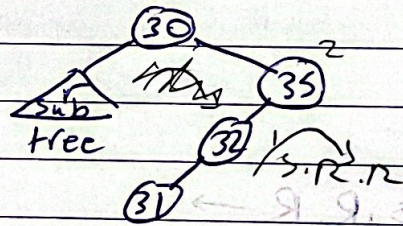
Note

EX

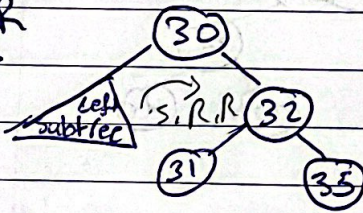


→ This is AVL T ✓

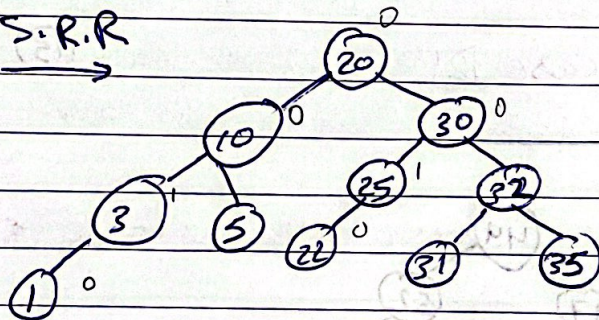
→ Delete 40



S.R.R

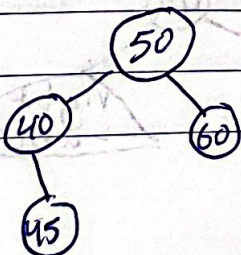


S.R.R

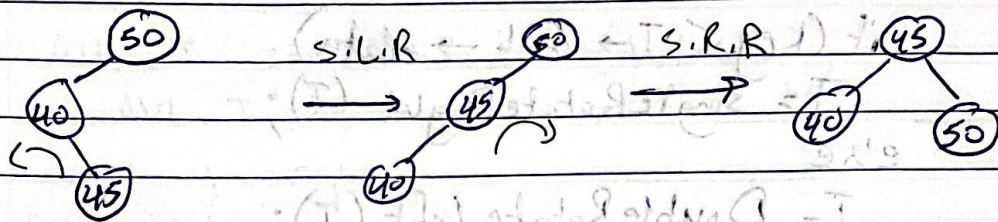


case 3

Ex



→ Delete 60



Case 3

* Implementation of AVL Tree:-

```

→ struct AVLnode;
typedef struct AVLnode * AVL-T;
struct AVLnode {
    double data;
    AVL-T right;
    AVL-T left;
    int Height;
}

```

* Function insert:-

→ AVL-T insert (AVL-T T, double key){

if (T == NULL){

T = (AVL-T) malloc (sizeof (struct AVLnode));

T → left = T → right = NULL;

T → data = key;

}

else if (Key < T → data) {

T → left = insert (T → left, key);

if ((getHeight (T → left) - getHeight (T → right)) >= 2){

balance • abs
قسمه مطابقة
مساواة على الطرفين

موجود بالترتيب
نكتبه بـ بجانب

X

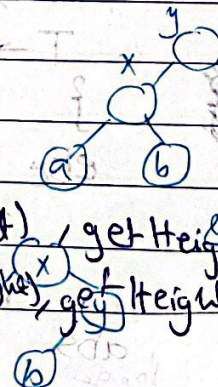

```

    if (key < T->left->data)
        T = singleRotateRight(T);
    else
        T = DoubleRotateLeft(T);
    else if (key > T->data) {
        T->right = insert(T->right, key);
        if (abs(getHeight(T->right) - getHeight(T->left)) >= 2) {
            if (key > T->right->data)
                T = singleRotateLeft(T);
            else
                T = DoubleRotateRight(T);
        }
        T->Height = 1 + Max(getHeight(T->left), getHeight(T->right));
        return T;
    }
}

```

getheight Function: `int getheight(AVL-T T) {`
`return T->Height;`
`}`

Function: `AVL-T singleRotateRight(AVL-T y) {`
`AVL-T x;`
`x = y->left;`
`y->left = x->right;`
`x->right = y;`
`x->Height = 1 + Max(getheight(x->right), getheight(x->left));`
`y->Height = 1 + Max(getheight(y->right), getheight(y->left));`
`return x;`
`}`



Function: AVL-T singleRotateLeft (AVL-T y) {

S.R.L AVL-T x;

x = y → right;

y → right = x → left;

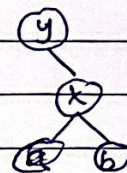
x → left = y;

x → Height = 1 + Max (getHeight(x → right), getHeight(x → left));

y → Height = 1 + Max (getHeight(y → right), getHeight(y → left));

return x;

}



Function: AVL-T doubleRotateLeft (AVL-T k₃) {

Double K₃ → Left = singleRotateLeft (K₃ → Left);

Rotate return (singleRotateRight (K₃));

Left then ?

right

* def splay Tree : is BST that:-

- Are not perfectly balanced all the time
- It assumes that recently accessed nodes are most likely accessed again
- Allow search and insertion operations to run faster.
- After node v is accessed, perform splaying operation.

def - splay : bring the node up to become the tree root.

- Active nodes (the recently accessed) will be moved towards the root, while in active nodes.

نیل مرے T یون $O(n)$ ← لیناویکی اہل خانہ مرے عملیہ (کتی مثلاً)
 علی ہذا الجہر بتکیر $O(1)$

will be moved far from the root.

* Splaying: - double / Right / Left ← rotates

rotate

1- Zig to left-right

2- Zag to right-left

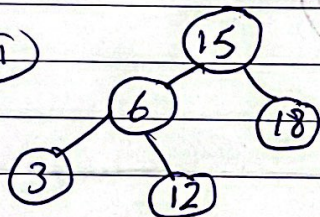
3- Zig-Zag

4- Zag-Zig

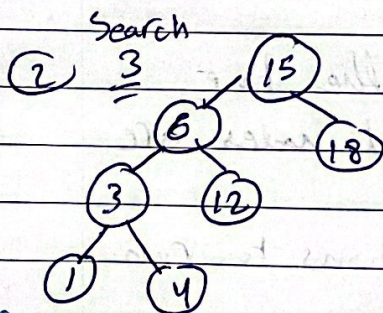
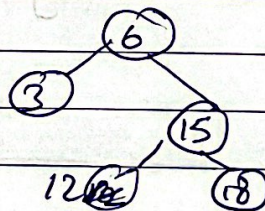
5- Zig-Zig

6- Zag-Zag

Exps: ①



Zig
Search 6

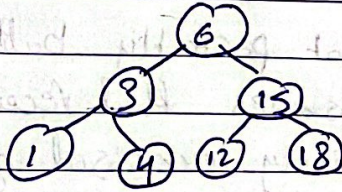


Search

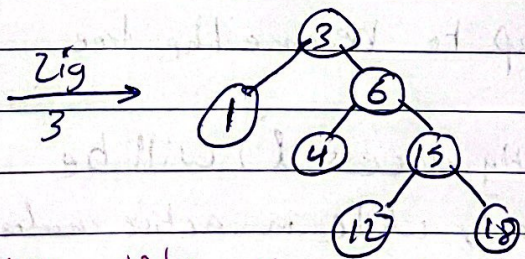
②

3

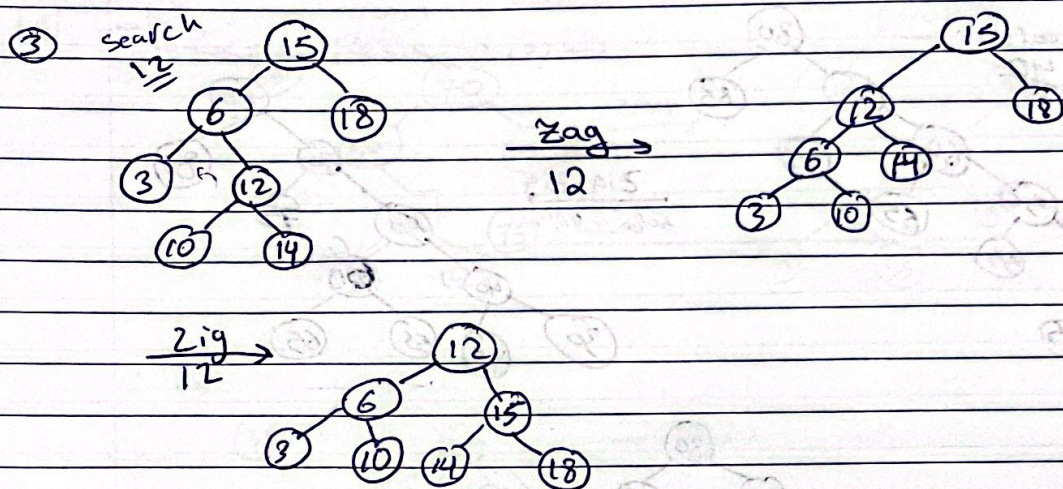
Zig Zag



3 child
 6 parent
 15 Grandparent
 اول مرے بچہ ال
 مع ال parent
 مع ال parent

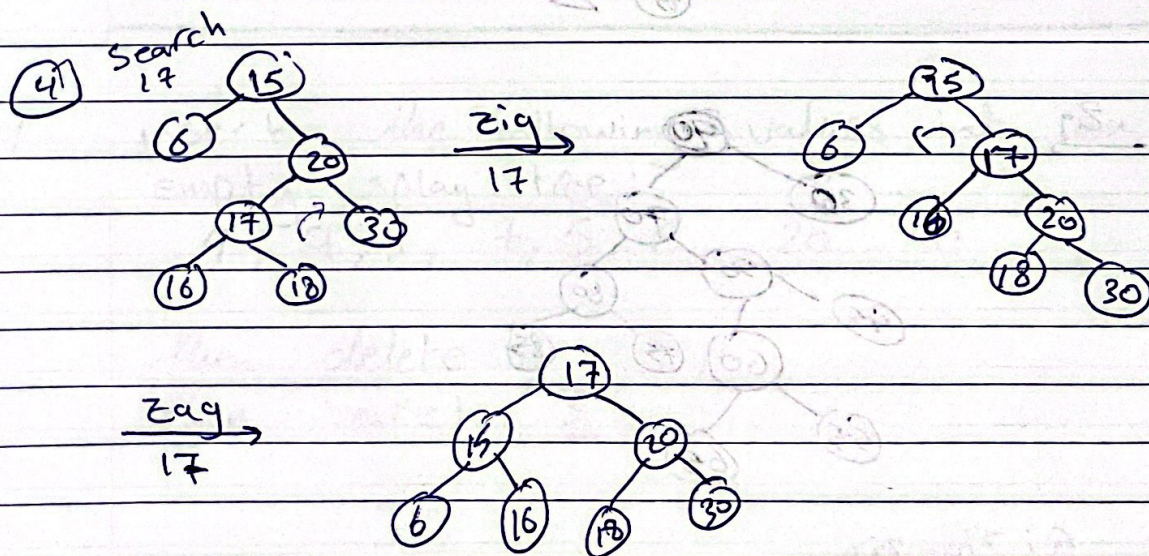


لا چڑی انہ ہوں عثمان اول ال 3 بروج مرتبہ بنفسیہ الانجاء left left
 و بوجہا



هونا عثمان اولاد ال 12 رحت left رحت right عثمان هيك
هونا اول اسب بقول ال child مع ال Parent موبعين ال Parent مع
ال grand parent

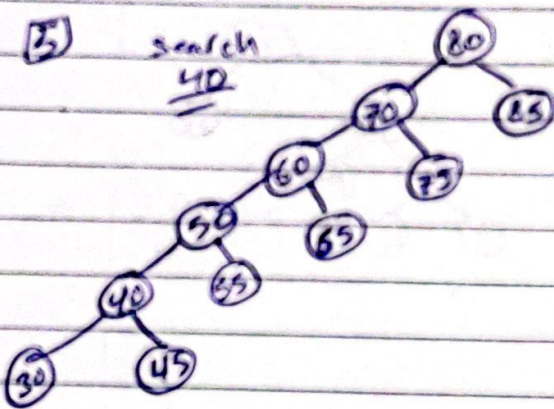
ال عثمان على splay ما بقى الكتب في ال Implementation
في سبب: ال BST وبقاك اعد ال search في ال عثمان
عثمان دسر ال عثمان



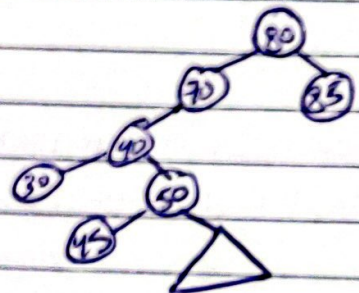
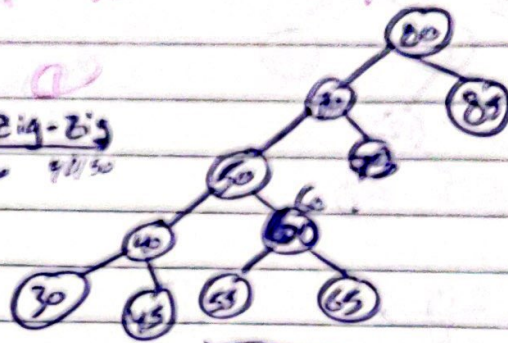
5)

Search
40

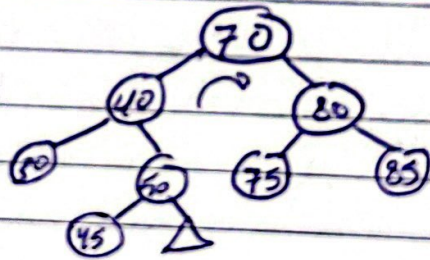
فاصله بین الیاف 2 و 3



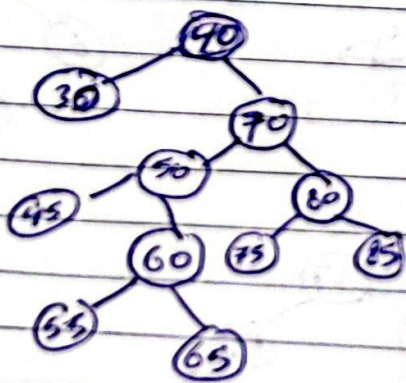
Zig-Zig
50/60 75/80



Zig
2

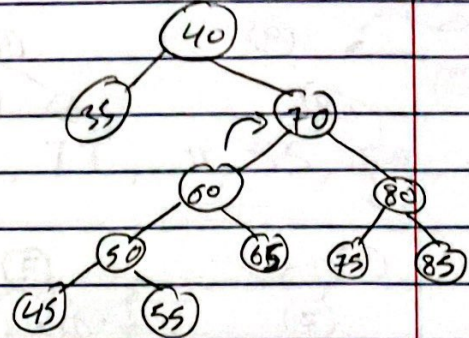
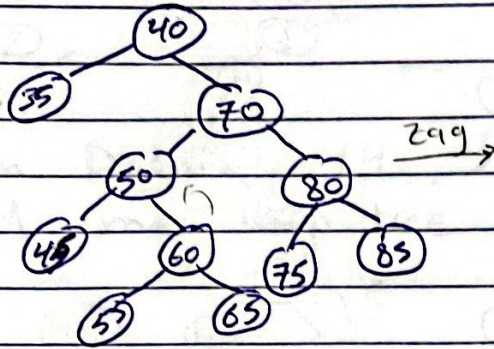


Zig

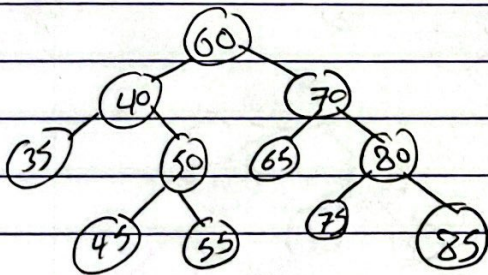
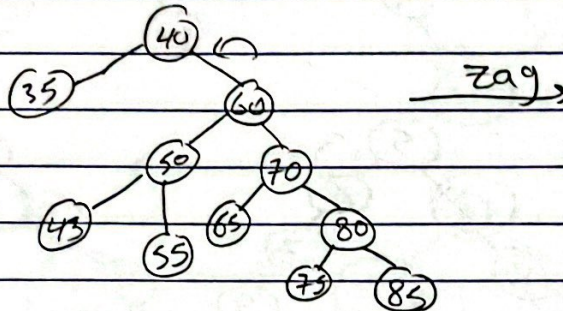


- ① Zig-Zig
- ② Zig-Zig

6 search
60



Zig



Ex. insert the following values into an empty splay tree:

4, 9, 3, 7, 5, 6, 20, 40

then delete 9

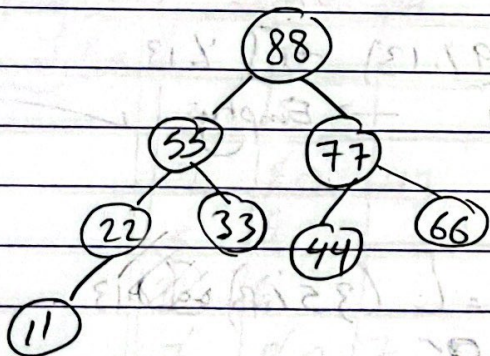
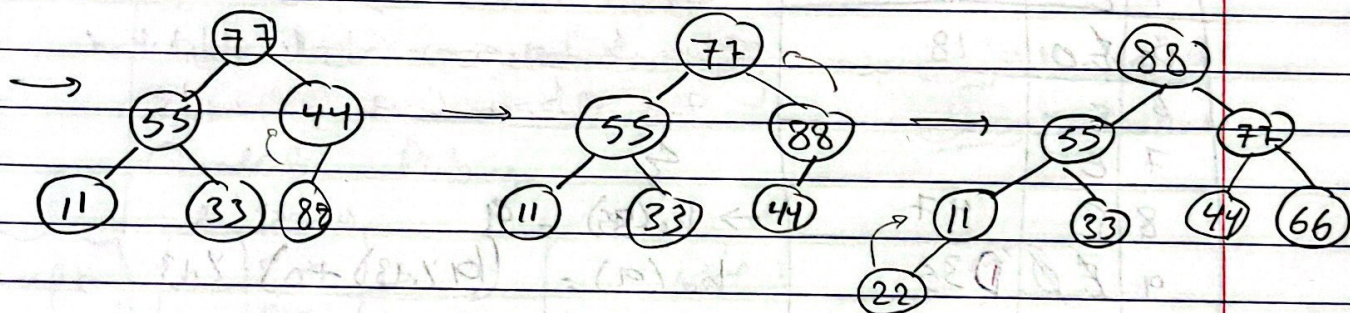
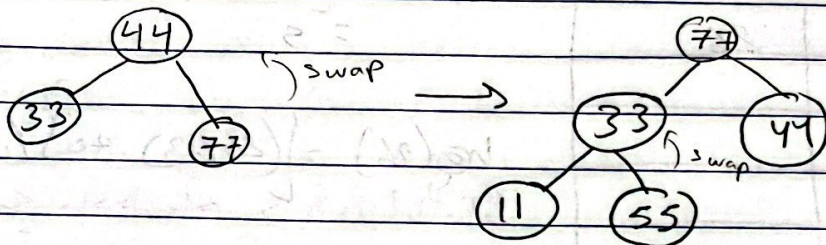
then search 3

16/11/2021

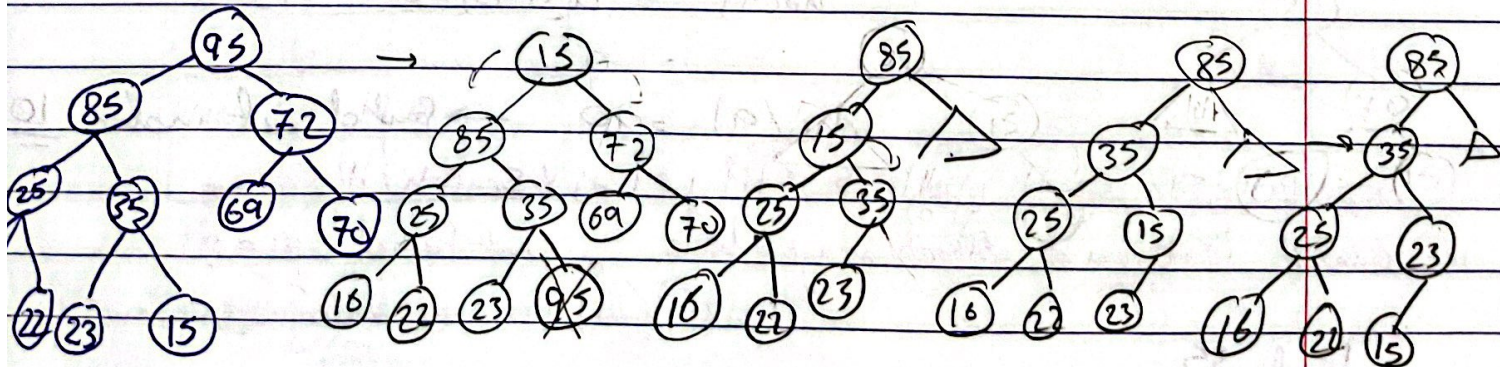
Heap

Tueschay

Q. Given Data: 44, 33, 77, 11, 55, 88, 66, 22
→ Build max heap tree.



Q Delete root 95



Hashing

* Linear Probing (Open Addressing)

* 18 Example
 index Status

0	E	0	26
1	E		
2	E		
3	E		
4	E		
5	E	0	18
6	E		
7	E		
8	E		47
9	E	0	35
10	E	0	9
11	E	0	48
12	E		64

$$h_0(18) = [(18 \times 13) + 0] \% 13 = 5$$

$$h_0(26) = [(26 \times 13) + 0] \% 13$$

ليه ماحط واحد؟ لأننا أنا دائماً
 بيبأ من صف لكن لو صار دقارب
 بروج - 1 --- وهذا كد ما يلاقى
 حل فإني بوجين رجلا فيه

$$\rightarrow h_0(35) = 9$$

$$h_0(9) = [(9 \times 13) + 0] \% 13$$

= 9 collision

$$h_1(9) = [(9 \times 13) + 1] \% 13 = 10 \rightarrow \text{Empty}$$

E: Empty
 O: occupied

$$* \text{Delete}(35) = [(35 \times 13) + 0] \% 13 = 9$$

$$\text{Delete}(40) \rightarrow h_0(40) = 1 \text{ empty}$$

Find 9

$$h_0(9) = [(9 \times 13) + 0] \% 13$$

$$= 9$$

$$h_1(9) = 10 \rightarrow \text{Found at index } 10$$

عملية ال Search لازم أدخل أنا كد ما القى الرقم، لكن لو ما القيته
 linear Probing disadvantages

و ممكن يمشي $O(n)$

find 35

$$h_0 = [(35 \times 13) + 0] \% 13$$

$$= 9 \rightarrow \text{not found} \rightarrow \text{"already deleted"}$$

* لو كان الـ index رقم زوجي الـ 35 عزي 22 مثلاً
وتمت عمل Find 35 ورجع الـ index مسجود فأنظر
أدور أدور لـ ما ليس عزي شيء Empty فهو not found
ويزيد بـ index الـ search

* إذا كانت موجودة بـ index مسجلها delete ورجعنا
لـ search عليها، عندها خدش بتوي هي من قبل كامل
فمنه خالين بـ index الـ موجودة للشيء وبالتالي خدش
not found

إذا كان الـ index بـ Sorting عزي عن الـ hash table
لـ index بـ وقتك

Quadratic Probing

Example Page 22 in slides.

index	Status	
0	EO	13
1	EO	14
2	E	
3	E	
4	EO	47
5	E	
6	E	
7	EO	7
8	EO	21
9	EO	815
10	EO	13
11	E	
12	EO	34

مبني :

$$h_0(34) = 8 \quad \text{collision}$$

$$h_1(34) = (34 \cdot 13 + 1^2) \cdot 13 = 9 \quad \text{collision}$$

$$h_2(34) = (34 \cdot 13 + 4) \cdot 13 = (8 + 4) \cdot 13$$

$$= 12$$

$$h_0(47) = 8 \quad \text{collision}$$

$$h_1(47) = 9 \quad h_2 \text{ coll}$$

$$h_3(47) = (8 + 9^2) \cdot 13 = 17 \cdot 13 = 4$$

الـ index الـ توزيع الـ "توزيع"
والـ index الـ توزيع الـ "توزيع"
الـ index الـ توزيع الـ "توزيع"

الـ index الـ توزيع الـ "توزيع"

int HashTable[13]; parallel array
int status [13]

struct
{ int element;
char status; }

الـ index الـ توزيع الـ "توزيع"
الـ index الـ توزيع الـ "توزيع"
الـ index الـ توزيع الـ "توزيع"

المسألة ٢٦
Hashing double
table

* Double hashing :-

Example

P 25

@

Status		
0	EO	26
1	E	
2	E	
3	EO	70
4	E	
5	EO	18
6	EO	9
7	EO	96
8	EO	47
9	EO	35
10	EO	36
11	E	
12	E	

$$h_0(a) = 9 \div 12 = 9 \text{ collision}$$

$$h_1(a) = \left[\left(\frac{9 \div 13 + 1}{h_0} \right) + \left(9 \div 13 \right) \right] \div 13$$

$$= 19 \div 13 = 6$$

$$h_0(96) = 5 \text{ coll}$$

$$h_1(96) = \left[\left(\frac{96 \div 13 + 1}{h_0} \right) + \left(96 \div 13 \right) \right] \div 13$$

$$= \left[\frac{5 + 1}{1} \right] \div 13$$

$$= 7 \div 13 = 7$$

$$h_0(70) = 5 \text{ coll}$$

$$h_1(70) = \left[\left(\frac{5 + 1 \times (1 + 70 \div 12)}{h_0} \right) \right] \div 13$$

$$= \left[\frac{5 + 11}{1} \right] \div 13 = 16 \div 13 = 3$$