Recursion & stack

There are 2 conditions for recursion:

① simple case ② recursive case

* طالما يتم استدعاء Function يتم تخزينها بالطوابق داخل memory
عن طريق عملية تسمى stack " خلاص أول كل نزوح
عالي بوجه بتعبيه و بنظل كدا لحد ما كل Function ال

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 || 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 بقدر أحل مسألة بأكتر من طريقة Algorithm ولكن لازم أختار أحسن واحدة منهم بحيث 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 ↓ Resources (CPU)

less space ↓ Memory (RAM)

* كل برنامج بفتحه أو بستقبله بخزنه الكمبيوتر على RAM

Analise

* How to ~~Analysize~~ 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 ("%d %d", i,j); }
    }
}
```

لح يوصل لهون حيث بكل البرنامج لأنك الشرط خطأ

C "constant"

n+1 ← مرة بنفذ

n(n+1) ← كل

n.n ←

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

بسيط

3 GHz means → 3×10^9 instruction in one second
جيجا هيرتز ثانية

Time
↓
→ $T(n) = O(n^2)$

نبأخذ أعلى أس وكأنه هو الـ max للوقت الكي عكي ياخذه البرنامج

*Q Assume the processor speed is 1 GHz, $n = 2000\,000 = 2 \times 10^6$, find the time it will take.

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

الزمن = المسافة / السرعة

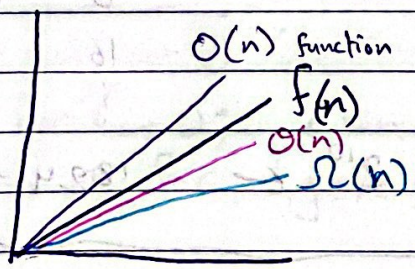
كل كل سطر لوين لح يوصل ونعدين نجمعهم كلهم.

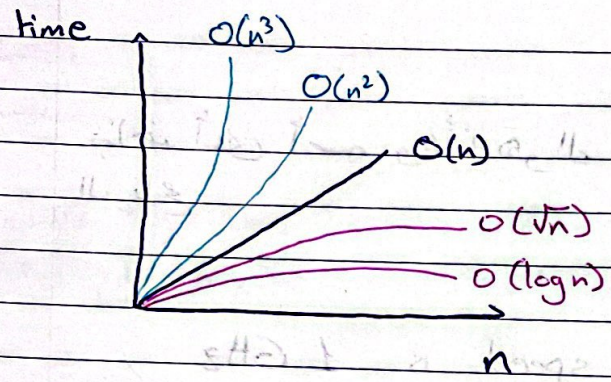
* Time Functions :-

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.





* لاحظي لما n تكون صغيرة بزقش بالوقت بس لما تكبر n بزقش الـ logn عن n^3

* Time (Big O) is neglected when n is small

question:- for (i=1 ; i<n ; i*=2){

}

What is the max time it will take?

لكل أول اشي بزقش قيمة ل n وكاول أوجد علاقة

لو كانت n=1024 ←

i
1
2
4
8
16
⋮
1024

10 times

$1024 \rightarrow 2^{10} \rightarrow 10 = \log_2 1024 \Rightarrow O(\log_2 n)$

* بنفس الطريقة

for (i=n ; i>n\2 ; --i)

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

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

i	
1024	
512	
256	11 times
⋮	
4	
2	
1	

بدنا $\underline{=} O(\log_2(n))$

---

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

Sol $\log_3(n)$

ملاحظة: كأنه بنفرض قيمة n كتير كبيرة فهلبوتا مش رح تفرق عن مليون و 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

i	j
0	0 … n (n)
1	1 … n (n-1)
2	n-2
3	n-3
4	n-4
	2
n-1	1

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

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

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

$= O(n^2)$ Ignoring constant

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

Solution $O(n^2 \log_4(n))$

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

→ will stop → $i^2 \geq n$

i^k

⇒ $O(\sqrt{n})$

Ex: int p=0

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

p += i ;

}

sol

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

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*i ; ++j) {

:

}

}

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

⇒ $1^2+2^2+3^4+\dots+n^2 = \sum_{i=0}^{n} i^2 = \frac{n(n+1)(n+2)}{6}$

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

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

حل هيك لأنو ما بقامل مباشرة مع n بالوب التالي.

$= O(n^3)$

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

i	j
0	n
1	n
2	n
3	n
:	n
n	n

$n + n + n + n + \dots + n$

→ $O(n^2)$

---

* 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:-

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

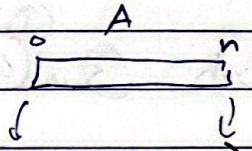
→
$$\left.\begin{aligned} T(n) &= T(n-1) + c \\ T(n-1) &= T(n-2) + c \\ T(n-2) &= T(n-3) + c \\ &\vdots \\ T(1) &= T(0) + c \\ T(0) &= d \end{aligned}\right\} \xrightarrow{\text{sum}} T(n) = c + c + c + \dots + d$$

$$= c \cdot n + d$$

$$= O(n)$$

فعلياً حل السؤال السابق رح يكون أسرع "أقل وقت" من طريقة recursion بالوب
لأنه كل مرة بستدعي الـ Function

Ex merge sort



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

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

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

  Merge (A, L, H);
}
```

بفصل نقسم المصفوفة على نصين كل ما يظل ينقص ، بعدين بجيب ترتب فيهم .

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

Merge Func. بستدعيها كل مرة

كل مرة بقسم على نصين عشان المعادلة تقسم على نصين من شرط مكتوب المعادلة تقسم على نصف واحد $\frac{n}{2}$

mergeSort استدعيت مرتين واستدعيت 5 جزء وهيك

تكملة ⟹ JP

$$T(n) = 2T(\tfrac{n}{2}) + n \quad \dots (1)$$

$$T(\tfrac{n}{2}) = 2T(\tfrac{n}{2^2}) + \tfrac{n}{2} \quad \dots (2)$$

$$T(\tfrac{n}{2^2}) = 2T(\tfrac{n}{2^3}) + \tfrac{n}{2^2} \quad \dots (3)$$

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 $\frac{n}{2^k} = 1$

$$2^k = n$$
$$k = \log_2(n)$$

→ $$T(n) = n\,T(1) + \log(n) \cdot n$$
$$= d \cdot n + n \cdot \log(n)$$

قيمة معدومة مقارنة مع الأخرى — أكبر

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

Ex: نفس قبل لكن

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

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

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

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

(2) → (1)

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

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

(3) → (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 ans $\Rightarrow \underline{O(n)}$

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

Sol

$T(n) = 2\,T(n-1) + c \quad \ldots (1)$

$T(n-1) = 2\,T(n-2) + c \quad \ldots (2)$

$T(n-2) = 2\,T(n-3) + c \quad \ldots (3)$

(2) in (1)

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

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

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.

سلبياتها → ① fixed size, سلبية، عشان ممكن نحجزها أكبر من اللازم

→ it leads for wasting space.

② insert new element in a (certain) position. → takes more time

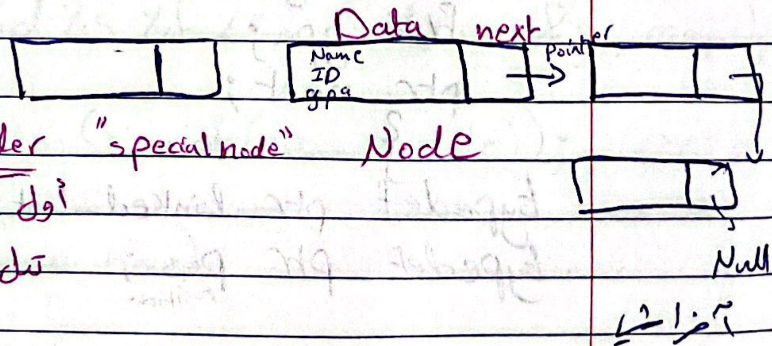
③ Delete an element . "we need to make shifts".

④ The need of sequential memory cells.

Def.- 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 seperately and only when necessary.

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



أول Node وبتكون فارغة من البيانات عشان تدل انها هون البداية

L. L

* Types of Linked List:-

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

* Operations on Linked list: → ماذا هو حجم عدد Size

1) create 2) insert 3) delete
4) search 5) is Empty 6) getSize
7) sort 8) printList

* The creation of L.L :-

struct Node;
typedef struct Node *ptr; pointer to node

typedef → for creation of new datatype

struct Node {
char Name [25];
int ID;
char major [20]
float gpa;
ptr next;
}
typedef ptr Linked_List;
typedef ptr pos; position

} يبدي اسماء الهم

main

```
Linked_List L;                                   pointer برجّع
L = (Linked_List) malloc (sizeof (struct Node));  هون انشأت الهيدر وحجز بالميموري
L -> next = NULL;                                 دائماً أول مرة بأخلّ الـ Null
```

ptr next 2byte - gpa 8 byte - major 20 byte - ID 4 byte - name 25byte مساحة تكون حسب حجم الـ arrays اللي عرفتهم قبل

---

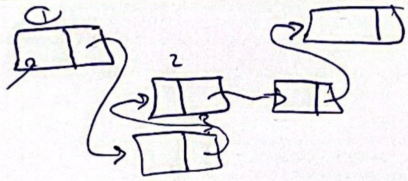
[2] Insert in LL :- (علاوة على الكود اللي قبل — كله تكملة)

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

    if (temp != NULL && L != NULL) {       هاي الخطوة لازم لو أشّر
        temp -> next = p -> next;          الـ pointer على NULL بعمل
        p -> next = temp;                  run-time error بتصير
        printf("new node is inserted\n");  بعمل Detection للبوينتر
    }                                      قبل ما أبدأ
    else
        printf("Error in insertion\n");
}
```

بالـ main

```
int id;
float g;
char nm [25] , major [20];
while (ID>=0) {
printf ("Enter the student's Information");
بضل يطلب مني أدخل بيانات
ptr temp; scanf("%d %s %f %s", &id, nm, &g, major);
ptr temp;
temp = (ptr) malloc(sizeof(struct Node));   (ptr) هو casting
temp -> ID = id                             لأنه الـ ptr رح يرجّع نفس نوع الـ data (pointer)
strcpy (temp -> Name, nm);                  فلازم أوضح وأحدد شو نوع الـ data
temp -> gpa = g; }
```



pointers

insert (L, L, temp);

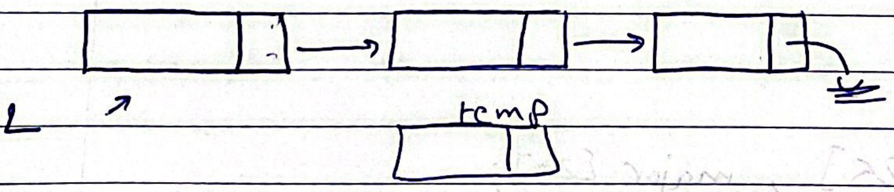
هيك بجيب بضيف كل node جديدة بعد ال Node ال header "عالبداية"
لكن برضو بتقدر أضيفها وين بدي أنا ، بقدر أعطيه اسم ال (L) و ال (Temp) تاعت
ال Node الي أنا معنية أضيف عليها "قبلها أو وراها" Insert(L, P2, new)

لو بدي أضيف nodes على آخر شي بال L L شو اسوي؟

Function → Void insert_last (Linked_List L, ptr temp){

```
pos p2 = L -> Next;
while (p2 -> next != NULL)
        p2 = p2 -> next;
        p2 -> next = temp;
        temp -> next = NULL;
}
```

لازم تعملي if statement تتأكدي فيها اذا 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=p->next;
    }
}
```

---

1. Function for search:-

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

    ptr p;
    p= L->next;
    int found=0;
    while (found==0 && (p != NULL)){
        if (p->id == id) {
            return p;
        }
        p = p -> next; }
    return NULL;
```

* Delete :- I can delete nodes by:

- ith node
- specific property
- all nodes.

وقد "نادراً ما يستخدموها" هاي بستخدموها أكثر

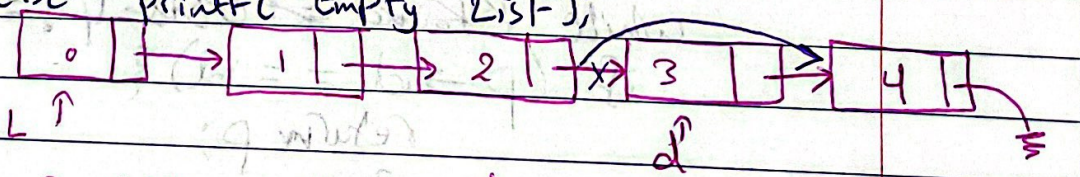
Function for deleting by ith

```
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;
            ++a;
        }
        d = p→next;
        p→next = d→next;
        free (d);
    } else printf(" Empty List");
}
```

نتأكد انه لسا مش فاضية

الرمز "ال node" الي بدي أحذفه.

خلي ptr الـ next يأخذ على بعد الي بدي أحذفها



لو بدي أحذف 3: أول اشي بخلي 2 تأخذ على 4 بعدين بحذف 3 عشان ما يضيع باقي معلومات الـ LL

Ex. delete the node Name is Ahmad "Function"

(Pointers)

```
void delete (Linked_List L, char Name[]) {
    ptr p = L, d;
    if (!isEmpty (L)) {
    while (strcmp (P->next->Name, Name) != 0)
            p = p->next;
    }
    d = p->next;
    p->next = d->next;
      free (d);
    }
    else
        printf ("Empty List");
}
```

مش متساويين

while عشان && لازم Ahmad لو في احتمال ترجع غير انو موجود

---

delete all Nodes; "لازم واحد واحد تحذفهم"

لأنه لو حذفت الأول بس الباقيين رح يظلهم بال memory محجوزة بس ما نستخدمهم data wasted

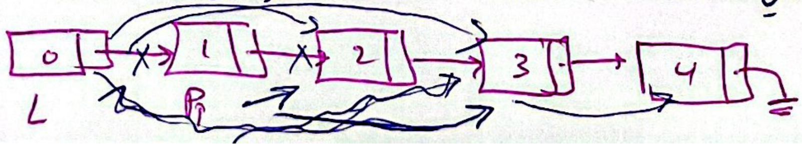
الوقت اللي بياخذه n

```
→ Void delete (Linked_List L) {
    ptr p = L->next;
    while (P != NULL) {
    L->next = p->next
    free (p)
    p = L->next;
}
free (L);
```

أشر عليه عشان لما احذف واحد

أمشي عالي بعدها احذف 2

لما يصير أشر على ولا اشي احذف (L) أول موجود



لو بدي أعمل الـ Function السابق لـ 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 بدي احسب Time بطلع O(n) برضه.

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

لأنو كلمة خذف node

نفلياً حتى لو التنين O(n) لكن الـ recursion بياخد وقت أكتر لأنه بستدعي الـ Function كل مرة.

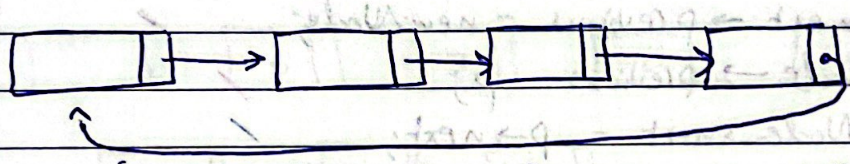
Function getSize:

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

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

★ لو بدي أخلي Time constante ← بعرف global variable
قبل ال main وبعرفه int size=0 وكل ما أعمل insert
node بعمل ++size وكل ما احذف node بعمل --size
وبالآخر ال function getSize ولازم نكتب بال function
بس return size فبكون مش محتاج كل الي مكتوب جوا .

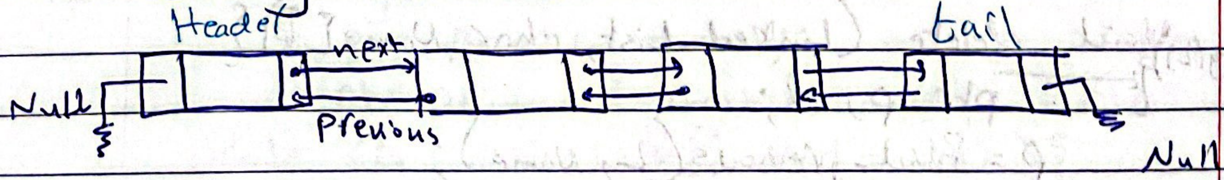
* Circular Linked_List:



الفرق هو انه آخر node بدل ما تأشر على NULL بترجع تأشر
على أول node ← يعني لما p→next=L معناه (انه خلص)

بالنسبة لل Functions تبعتها بتكون نفس قبل لكن آخر وحدة
بدل ما أكتب L→next=NULL بكتب L→Next=L وبس .

* Doubly Linked_List:



بدل ما أنشئ بوينتر واحد لل next بعمل اثنين واحد لل previous

... استخدام الها هو بال Search بيشتغل موضوع size وممكن أتسمحلها 2 ".
"جري"

ال Functions

① ← أول اشي بعدل عال creation بال struct ← بضيف ptr previous

```
← struct Node{
      ;
    ptr next;
    ptr previous;
}
```

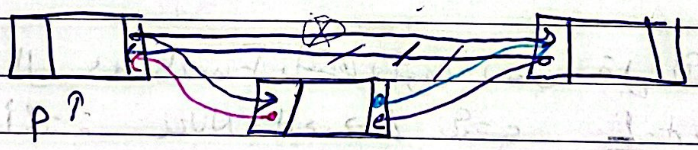
وكمان بضيف بعد

```
L = malloc ---
L->next = NULL;
L->previous = NULL;
```

② تعديل ال insert

```
void insert (Linked_List L, ptr p, ptr newNode){
    p->next->previous = newNode;
    newNode->previous = p;
    newNode->next = p->next;
    p->next = newNode;
```

أسهم مهم



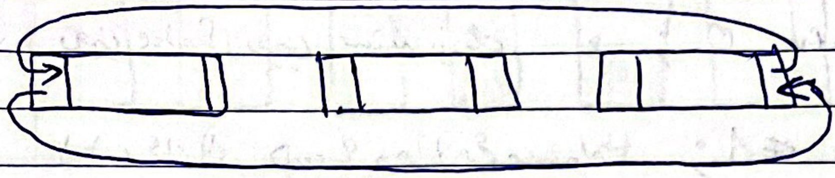
p ↑

New Node

③ تعديل ال delete

```
if(!isEmpty(L)){ void delete (Linked_List, char Name[]){
    ptr p, d;
    p = find_previous(L, Name);
    d = p->next;
    p->next = d->next;
    p->next->previous = d->previous;
} free(d)
}
```

4 Circular doubly L.L :-

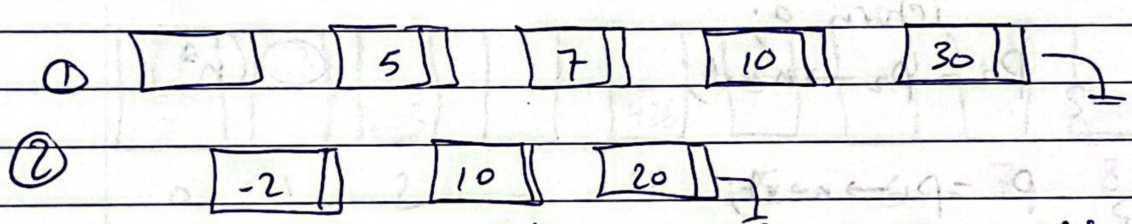


مرتبة ترتيب تصاعدي أو تنازلي

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.

فكرة الحل : أنا بعمل pointer لكل ذاك عشان من أصغر رقم ونقارن



بيبدأ أقارن هل ال 2- أصغر من خمسة ✓ ولا بتساويها بعدها

بروح عالخمسة نفس الشيء هل بتساوي العشرة ← لا وهي أصغر

بتحرك عند السبعة ← هل السبعة ~~أصغر من~~ تساوي 10 ← لا وهي أصغر

فبتحرك بال L.L تاعتها ، هل العشرة ب ① = 10 ب ② ← نعم

فبرجع false

```
Function: int Disjoint (Linked-List L1, Linked-List L2) {
    ptr p1 = L1 -> next;    // عشان ما نعمل اشي بال header
    ptr p2 = L2 -> next;    // لازم أعمل ptr يأشر عالي بعده
  while (p1 != NULL && p2 != NULL) {
      if (p1 -> data < p2 -> data)
            p1 = p1 -> next;
      else if (p2 -> data < p1 -> data)
```

$P_2 = P_2 \rightarrow next;$

else

return 0; → disjoint مش False يعني رجعت

}

return 1; لو لفت كل ال Loop وما لاقت مساواة معناها رح تطلع disjoint

}

$O(n)$ ← مهم

* لو مش sorted → $O(n^2)$ → nested loop

```
→ while (P1 != NULL){
      P2 = L2 -> next
   while (P2 != NULL){
   if (P1 -> data == P2 -> data)
       return 0;
     P2 = P2 -> next;
   }
     P1 = P1 -> next;
 }
   return 1;
```

$O(n^2)$

* Radix Sort:- ترتيب رقم ادري

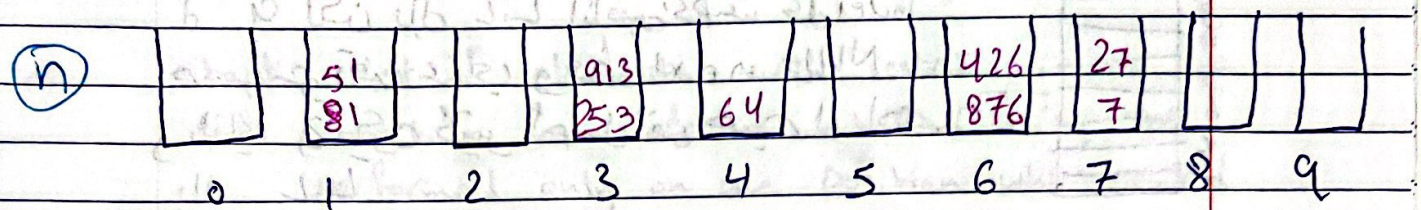
- used for integers and strings.

Example: Assume 876 253 426 913 81 51 7 64 27

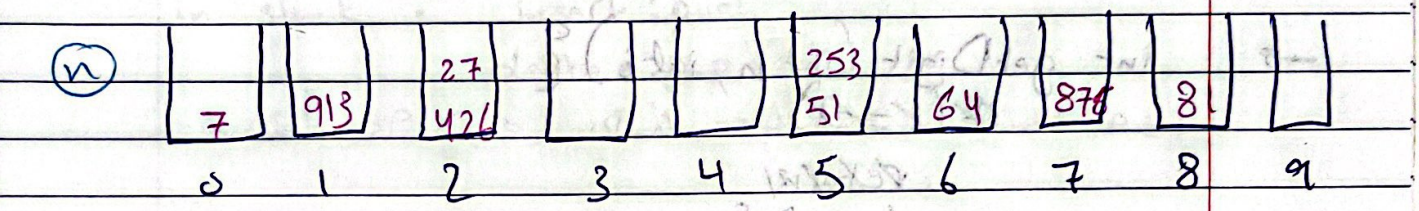
\#steps = \#digits in max num.

"أكبر رقم فيها من 3 منازل → 3 خطوات

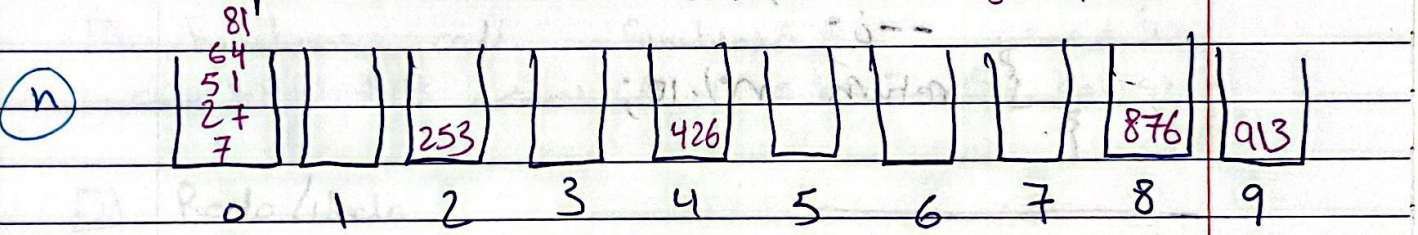
- Step # 1: نبدأ كل رقم حسب منزلة الآحاد له



- step # 2: حسب المنزلة الثانية من اليمين



- Step # 3: حسب المنزلة الثالثة من اليمين



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

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

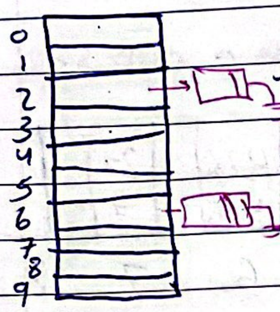
nxn لـ 2-D array بنشأ step 1 بـ -

— — - - — - step 2 بـ -

step 1 - array ممكن أقدر استغل step 3 بـ -

⇒ 2 Arrays (at least) $n = 10^4$ Assume $= 2 \times 10^8 \times 4$ bytes $= 14 \times 10^8$ bytes (2x)
nxn

→ 2 Linked-List $2xn = 2 * 10^4 * 4$ bytes $= 8 \times 10^4$ bytes



بالـ array بجهز أشكال أنشئ محلات من صفر
لـ 9 لكن بالـ L.L أنا بنشئ nodes عند كل index
حسب شو بحتاج عندي وبحط next = NULL
وبالتالي رح يكون كثير أحسن من حيث المساحة
بالـ L.L

سؤال: لو بدي أعمل Function بتجيبلي خانة "منزلة" مطلوبة من رقم

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

لو عندي اسماء بدي أرتبها حسب Radix ، بعمل نفس الاشي
هناك ، بشوف أطول اسم وبعمل steps حسبها ، وكل ما يخلص
اسم لما أروح من يمين للشمال ، بحسب لما يخلص الاسم محل الفاضي
اما NULL أو space

Khaled Heba_ Sara _ Mai

شرح كود الورقة Cursor Imp. of L.L

X بدون Dynamic Memory Allocation ← ممنوع لما ما يكون في pointers

حجم ثابت Array : fixed size

هلأ هو حاجز 100 ، لكن بقدر أعمل أكثر من L.L زي B و C
ولا يغير الـ A

05.10.2021 — Stack — Tuesday

Stack

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

زي صف صحون "أنا باخد الصحن الأعلى أو بجيب صحن على واحد"

* Applications on stack:

[1] Functions call: functions A, B, C — مثلاً لو عندي ع أطلع B بستدعي C وال A بستدعي B الي هو أصلاً مستدعي C

[2] Redo / Undo

[3] expressions evaluations

[4] browsers → google →(باضغط على) facebook → link game - ——

* Implementation for stack: 1- Linked-List / 2- array

(صيف عاول / آمراش — أي قبل "لكن أخذناه نفس مكان ما مفتي")

→ ال stack عادة بحور عشان هيك بستعملوا array أكثر من L.L

```c
#define MAXSIZE 100;
struct stack {
    int Elements[MAXSIZE];
    int Top;
};
```

```c
typedef struct stack STACK;
STACK S;                  // هيك أنا أنشأت stack
S.Top = -1;               // تدل على empty stack ←
```

Function push

(void ← عشان أخليه يرجع 1 اذا بنجح أضيف وصفر اذا ما بنجح)

```c
void push (STACK s, int e /* element */){
    if (s.Top < MAXSIZE-1){   // بدي أشوف اذا في مجال أضيف عليها
                              // -1 عشان بيبدأ من
        ++s.Top ;
        s.Elements [s.Top]= e;
    }
    else{
        printf ("stack overflow \n");
    }
}
```

Function pop

(int ← بدي أرجع قيمة)

```c
int pop (STACK s) {
    if (s.Top > -1) {         // يعني اذا في اشي مخزن فيها
        int e;
        e = s.Element [s.Top];
        --s.Top;
        return e;
    }
    else{
        printf ("stack is empty\n");
        return -1;
    }
}
```

لو هلا فرضنا القيمة تاعتنا e = -1 ، كيف بده يعرفنا اذا هي if رجعوها جوا الـ function ما استدعي ← s? فاضية ولا هي مخزنة ولا هي -1 وبدنا نخزن فيها

"حيث لو s.Top > -1 ← استدعي pop ← غير هيك ما تستدعي"

العملية الحسابية بتكون جوا

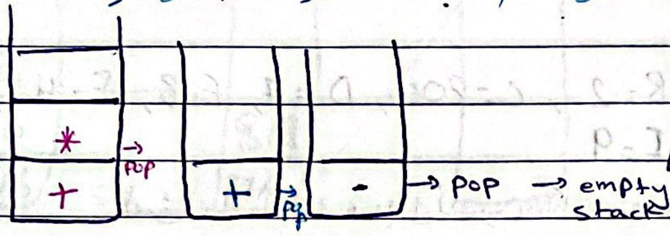
① Conversion from Infix expression to Postfix:

A + B * C - F

أول اشي بجيب تعبير: جمع بجيب وطرح

/ * - +

الفكرة انه بضيف العملية عال stack ، و بضيف فوقها الي الها أولوية فوقها ، لكن مثلاً الجمع والطرح الهم نفس الأولوية وبطلع الي تحت عشان أقدر أضيفها



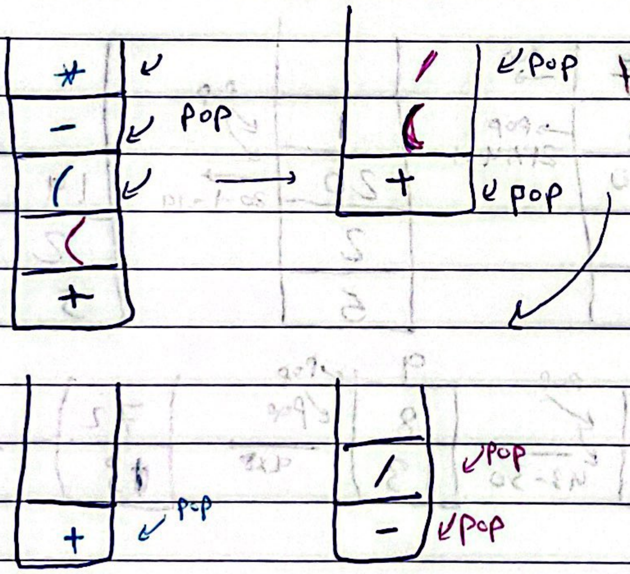
output

ABC * + F -

Postfix

كلشي بعمله pop بنزله عال output

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



ABCD * - E / + F + GH / -

Remarks: الأقواس تحجب كل شي ماقبلها ، يعني كأنه مش موجود الي قبلها فبعمل pop عالي جوها ، ولما أقابل نهاية القوس ( ) ، بعمل pop لحد ما أمحي أوله (، الأقواس ما بضيفها عالناتج ، ولو بدي أضيف عملية جمع وكان موجود أصلاً عملية جمع ← نمحي الموجودة (بنزلها عال output) ونضيف الجديد.

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

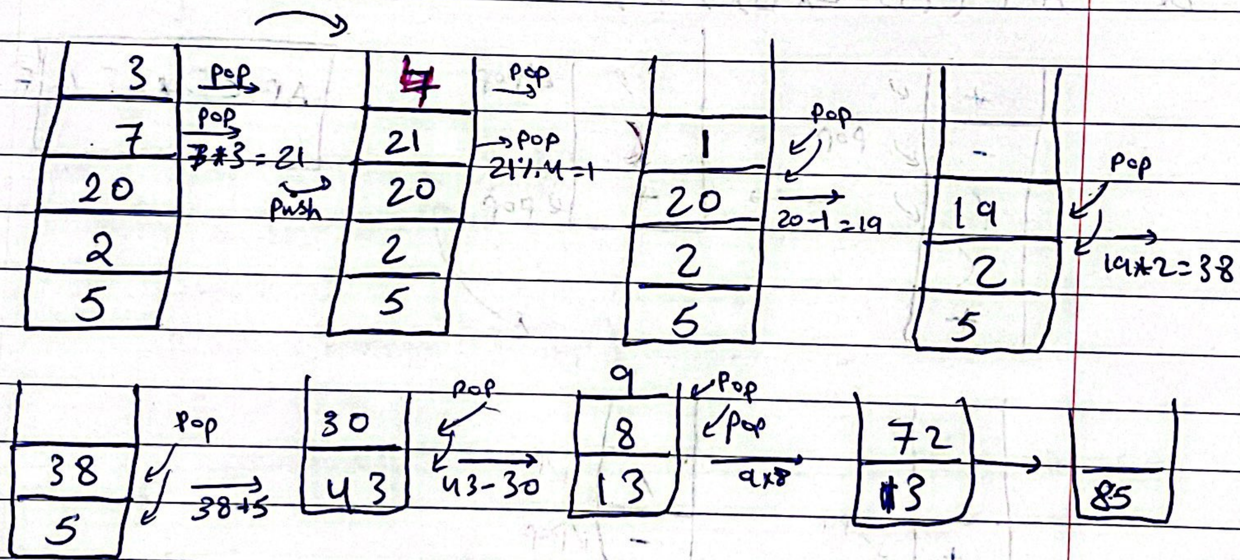
(نفس أولوية * ٪ قسمة)

Sol ABCDE*F%-*+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 ، بحط أضيف عال variables الـ stack لحد ما
الاقي عملية حسابية ، بروح بجيب اخر 2 elements وبعمل عليهم
هاي العملية ، وبعمل push عال stack كمان مرة للناتج.

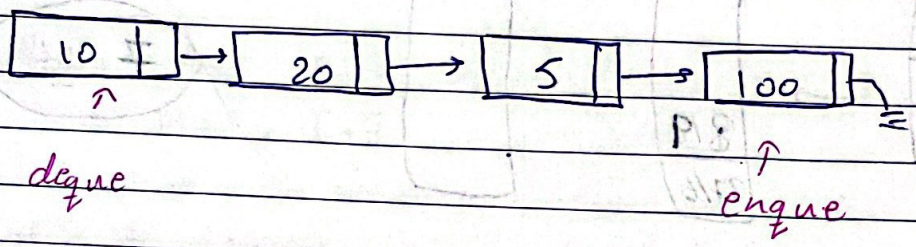


عشان أعرف اذا حلي صح ← لازم آخر خطوة يكون عندنا جواب واحد فقط ، ووطا أبدأ هاي العمليات لازم عالأقل أدخل قيمة متغيرين.

Ex Evaluate ABCD*-E/+F+GH/-
A=2 B=3 C=4 D=5 E=6 F=7 G=8 H=9

10.7.10.2021 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 ما بنستخدمش

- Applications :

1) printers 2) calling centers 3) Registration system

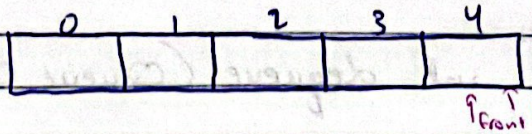
* Implementation:-

1- Linked List. 2- Array ← how?

```
#define SIZE 5;
Struct Queue {
    int Data[size];
    int front;
    int rear;
} Q;
Q.front = size -1;
Q.rear = size -1;
```

Scanned with CamScanner

ما بتبدأ أضيف على كومليان ولا أضيف مرتين ضايعين



Function الأقترانية

```
void enqueue (Queue Q, int x) {

    int temp = Q.rear;
    if (Q.rear == size-1)
        Q.rear = 0;
    else
        ++Q.rear;

    if (Q.rear == Q.front) {
        printf("Queue is full\n");
        Q.rear = temp;
    }
    else {
        Q.Data[Q.rear] = x;
    }
}
```

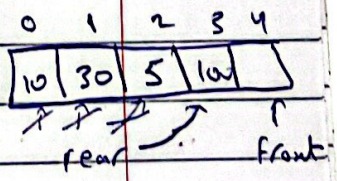
هو بس بخوف على 0 1 2 3 4
انا كل مرة على 0 --- 4
(عشان أحرك pointer وأعرف موقع كل)

عشان لما يشوف انا
مجيبة وما بنفع أضيف

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 يعتبر header وممكن يكون أي مكان

Func dequeue

```
int dequeue (Queue Q) {
    if (!isEmpty (Q)){
        if(Q.front ==sizE -1)
            Q.front=0;
        else
            ++ Q.front;
        return Q.Data [Q.front];
    }
    else{
        printf ("Empty Queue\n");
        return -1;
    }
}
```

لازم "الأفضل" قبل ما استدعي هاد ال Function أفحص من الأساس بال main
اذا isEmpty ، عشان قيمة ال (-1) كانت ال Func رح تخربيشك

```
int isEmpty(QueQ){
    return Q.front==Q.rear}
```

وفعلياً أنا ما بحذف لكن بخزن فوقا القيمة الموجودة أو بعملها hide ، هو وقعا
برجع القيمة تبعت الاشي الي كتبته ، لكن لو في سالب عنجد فهو كحذف قيمة
كلها من ال L.L

O(1) constant

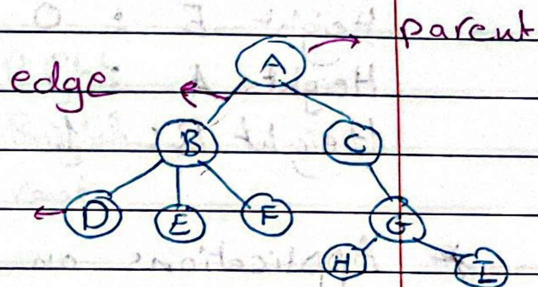
12/10/2021

Trees

Tuesday

* Linear data structure: يعني كل عنصر يؤشر عالذي بعده
"كلشي قبل أ قدام هو هيك".

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



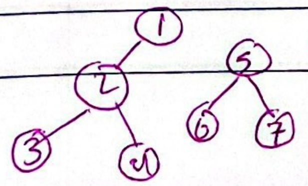
Def Path from node to node m: set of edges/nodes from node1 to node m, with length of m-1

Ex Path from A to H: A, C, G, H
4 عقد

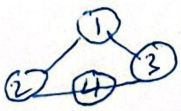
* Tree properties:-

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.

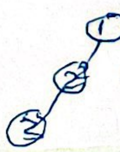
① → tree ✓



X not a tree because there are 2 roots.



→ X



✓

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

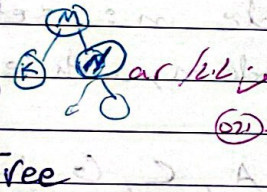
ex Depth of H : 3
Depth of B : 1
Depth of A : 0

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
2- Directory
3- Decision Tree
4- File directories

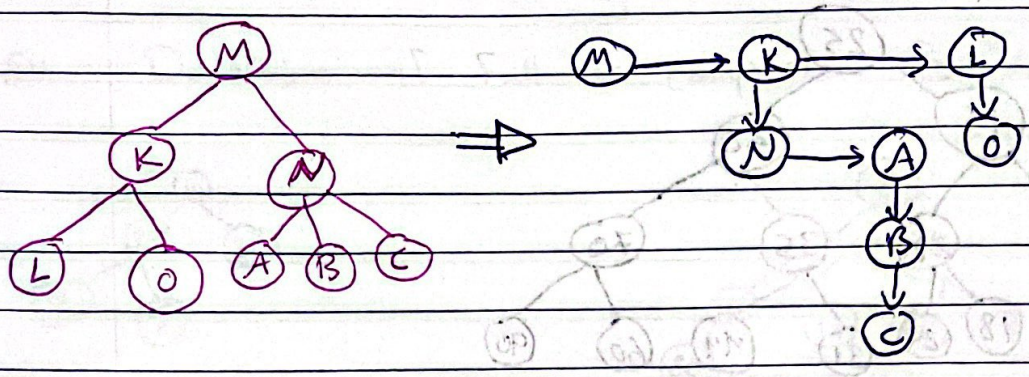


* Implementation of tree:-

```
Struct Node{
    ;
ptr next;
ptr siblings;
}
```

Note → siblings: nodes with same parent

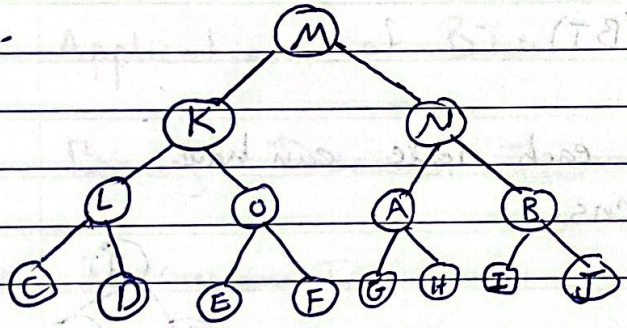
ptr next →
ptr sibling ↓



Def:* Tree Traversal:- visiting all nodes. by:-

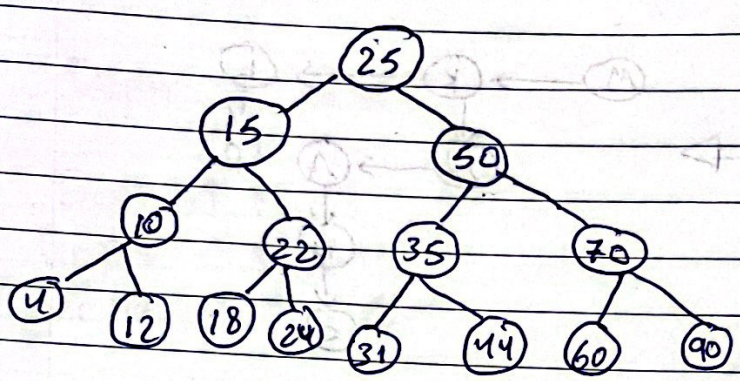
1- preorder: root - Left - Right
2- Inorder: Left - root - Right
3- postorder: Left - Right - root

Ex:-



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



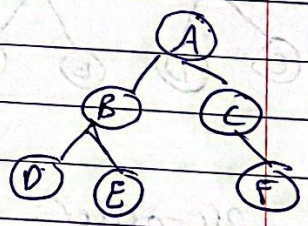
1- Pre-o: 25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 60, 90
2- In-o: 4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 60, 70, 90
3- Post-o: 4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 60, 90, 70, 50, 25

ملاحظة: ممكن يجيب 1 2 3 ويقلك ارسمي.

---

* Binary Tree:- (BT)

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

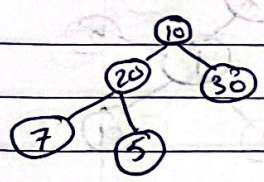


* Types of Binary Tree:

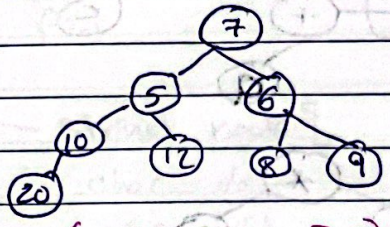
[1] Full BT: every node has zero or 2 children

[2] Complete BT: every level, except the last, must be completely filled. The last level is filled as far left as possible.

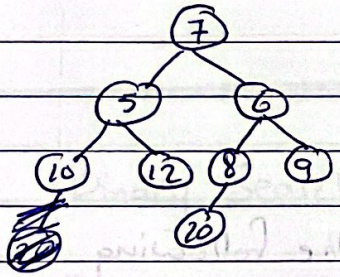
Ex Determine if Full or complete BT



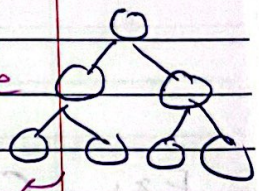
(Full BT) and + (complete)



(Not FULL BT)
complete BT



{ Not complete nor full
Just BT. (filling must be from left to right).



full + complete

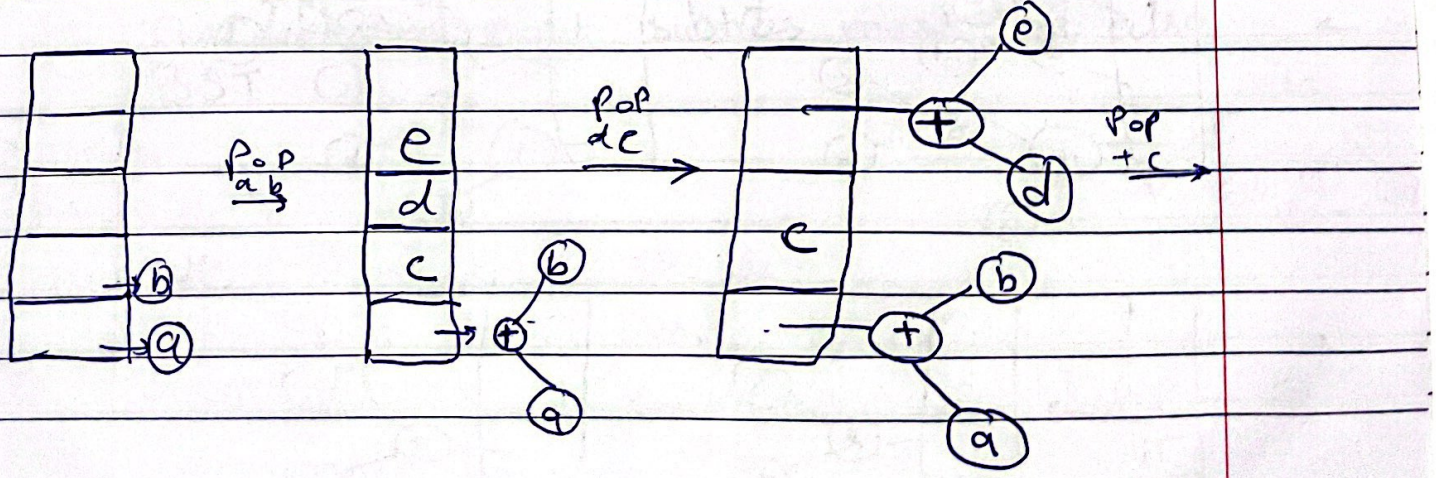
* Applications of BT :-

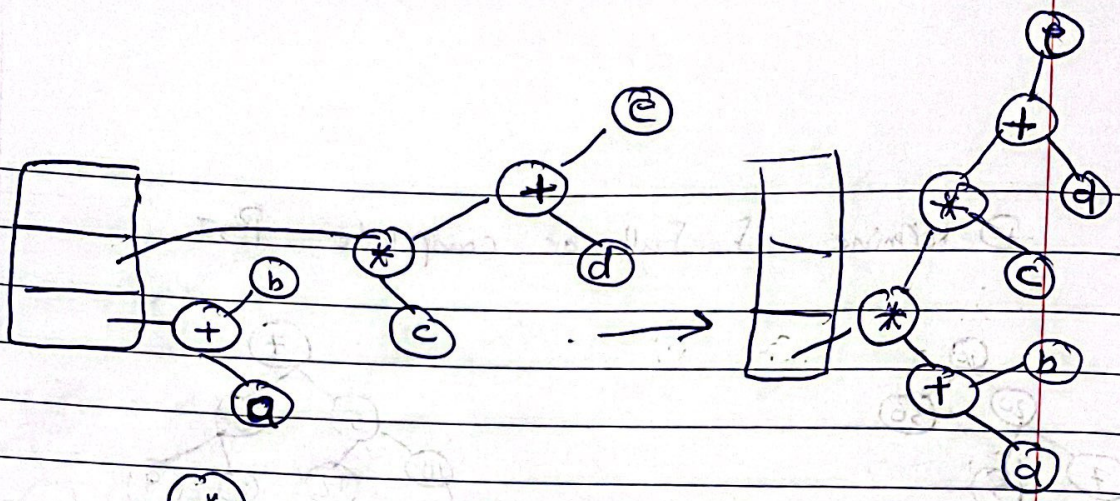
1- Binary Decision Tree

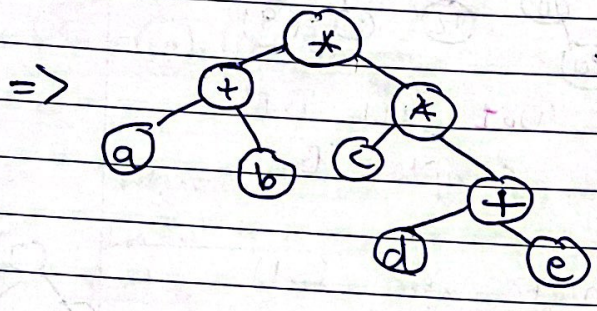
2- Expression Tree.

Ex: ab + cde + **

Sol we need stack & BT





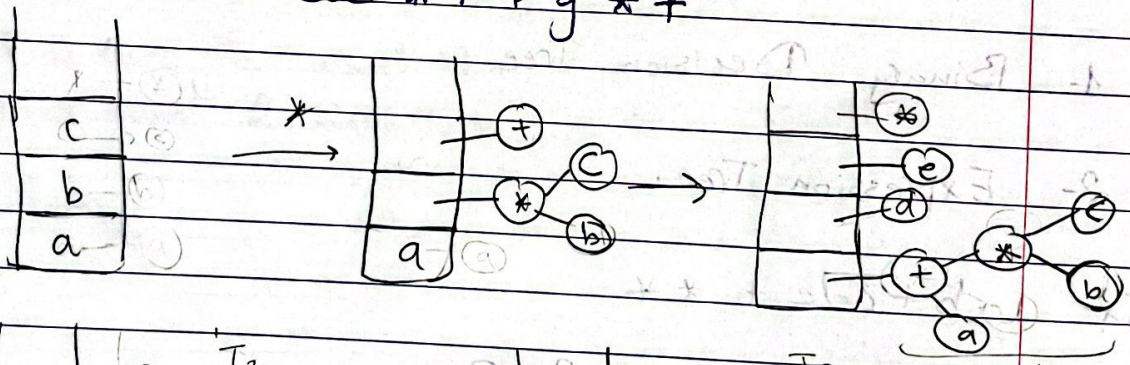


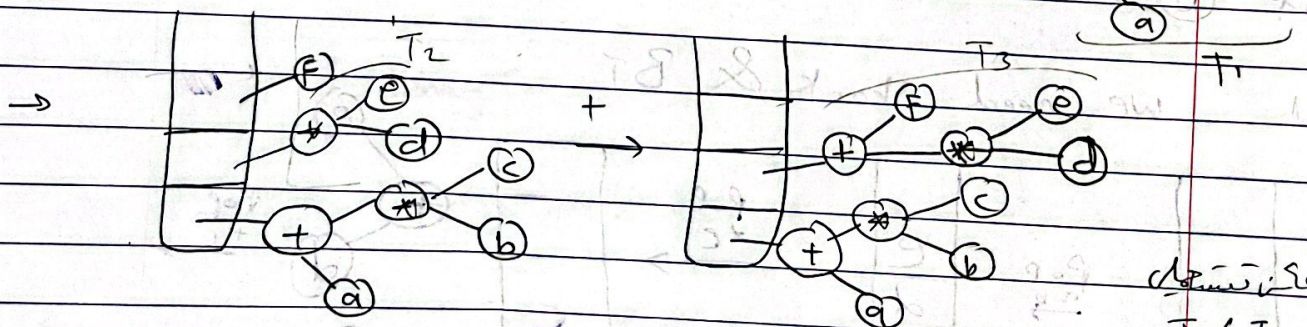
$\Rightarrow a + b * c * d + e$

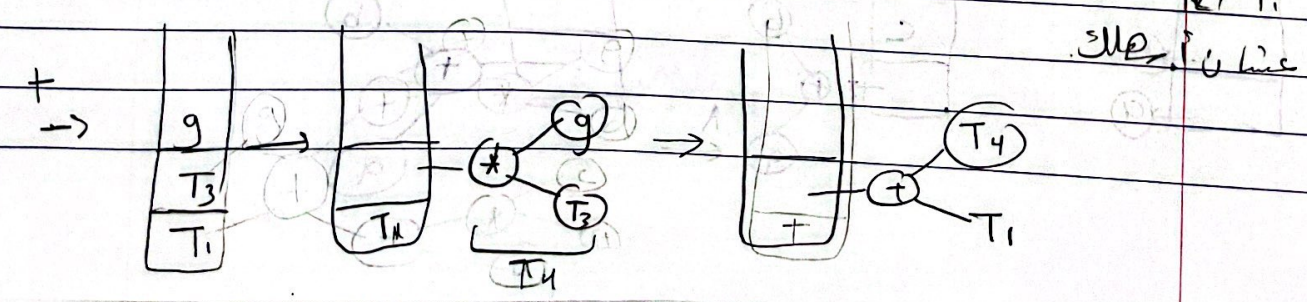
Ex: Construct the exp. tree for the following postfix exp.

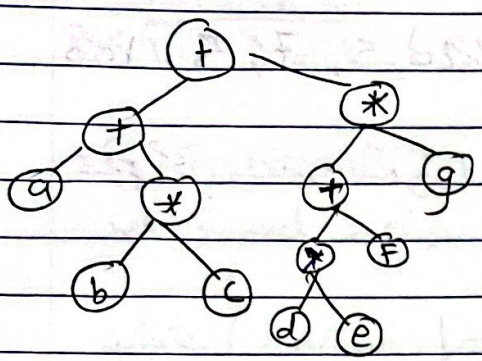
abc * + de * f + g * +

Sol









Infix exp. = a+b*c-+d*e+f*g

pre

post

```
struct node{
    char data;
    ptr left;
    ptr right;
};
```

---

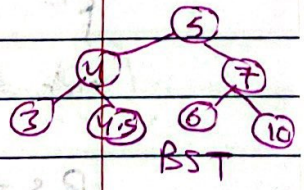
* Binary Search Tree (BST)

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

1- The left subtree of any node contains only the nodes with Key values less than the node's Key.

2- The right " " " " " " "
" " " " " greater " "
" "

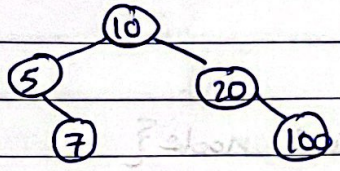
3- The left and right subtrees must be also BST




```
node {
char [ ] name;
int ID;
float gpa;
ptr left;
ptr right;
}
```

Construct BST of 10, 20, 5, 7, 5, 100

Sol



محتوى ذكروا استخدم كمرشد

---

* Operations on BST:-

- creation
- insertion
- deletion
- search
- Traversions.

---

* creation:

```
struct Node;
typedef struct Node * BST;
struct Node{
        ⋮
BST left;
BST Right;
}
int main(){

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

بيرد node ال كامل

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.

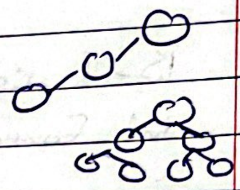
```
BST Search (BST T, int Key){
    if (T != NULL){
        if (T->id == Key)
            return 1;
        else if (T->id > Key)
            return search (T->left, Key);
        else
            return search (T->right, Key);
    }
    return NULL;
}
```

لو مش حاطينها بيطلع Syntax error

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

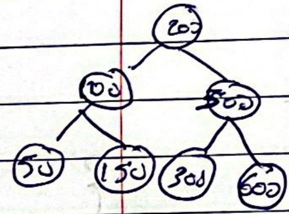
$O(\log(n))$ → if balanced BST
if the tree ideal

Time comp. for search functions:



Worst case O(n)
Best Case O(1) →
Balanced O(log(n))

بدنا فنكشن دطبع كل النودز



```
Void inOrderTraversal(BST T){

    if (T != NULL){
        inOrderTraversal(T->Left);
     -  Printf("%d \n", T->Data);
     -  inOrderTraversal(T->right);
    }
}
```

50
100
150
200
300
500
600

→ لو بدي PostOrder بنبدل بين السطرين = نقط
ولو بدي pre بنغير ال 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 (الي بدنا نضيفها) as Left child of the node.

- If we find a node whose right child " "
  " " " " Key as right child — — —

ولكن في أكثر من معلومة جوا ال node ع يا اما بيعمل كلهم أو بيبعت بوينتر بيؤشر عليها

Function:-

```
BST insert (BST T, int x) {

    if (T == NULL) {
        T = (BST) malloc(sizeof(tree-Node));
        T→left = T→right = NULL;
        T→Data = x;
    }
    else {
        if (x < T→Data)
            T→left = insert (T→Left, x);
        else if (x > T→Data)
            T→right = insert (T→right, x);
        else
            printf("The values should be unique");
    }
    return T;
}
```

هاي العبارة مسموحة عشان فراغ من اليمين الشمال ما بيكون في أكثر من يساوي وحدة.

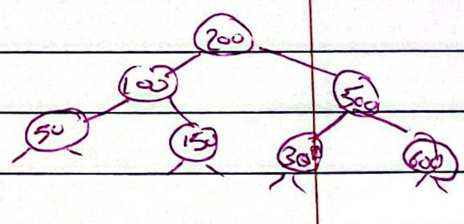
subtree

لما الها بوينترها عادي تكون أكثر لان ال BST ممنوع تتكرر ال Key فيها.

Time complexity:-

كل مرة بتقسم على 2 — balanced tree

- avg case $O(\log(n))$
- لما تكون كلها جهة وحدة — worst case $O(n)$
- Best case $O(1)$



Ex:- insert the following values:- 120, 170, 650, 570, 900

عمل ال function مش void عشان يرجع ال subtrees مع بعض
يعني لما أضيف ال 120 بالآخر برجع ال subtree تاعتها وبعدها
تاعت ال 150 وبعدها 100 وبعدها برجعهم مع ال 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 leaf from left

Function: find minimum

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

Function delete

```
BST delete (BST T, int x){
    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 {
```

→

نفس ≠ NULL

```
if ( T->left && T->right ){
    BST temp;
    temp = Find_Min (T->right);
    T->Data = temp -> Data;
    T->right = delete(T->right, T->left);   copy
    T->right = delete (T->right, 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;
}
}
```

لو دخلت قيمة مش موجودة عشان أخرها مش ح يصير error

ح يرجع بربط NULL بآخر إشي وخلص

رمز لأول حرف من 3 أسماء علماء أوجدوه

* AVL Trees:-

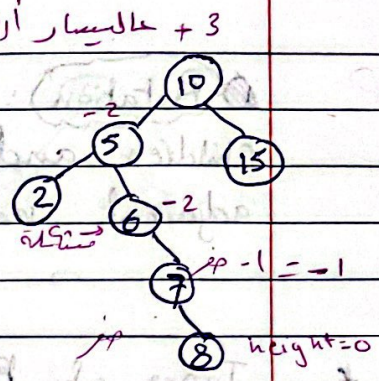
Def: AVL: is a binary search tree with height balance property.

- For any node v, the height of the subtrees of v is differ by at most 1.

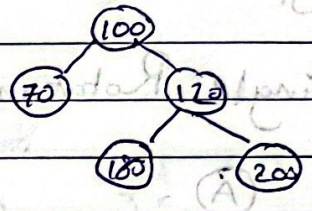
- Any subtree of an AVL tree is also an AVL.

Ex

Not AVL ⟶



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



لكن لو عدلت الـ 180 لـ 110 فبتصير AVL ✓

---

* Why AVL tree?

Problems of BST
- insertion or deletion in BST can cause large inbalance.
- The time complexity of search, insertion or delete of any node could be O(n) in worst case.

لكن بال AVL كل ما أضيف أو أحذف عنصر بقل أتفحص وأزبط اله balance تاعها.

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

---

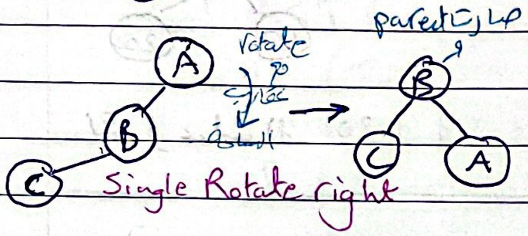
* Balancing is done by Rotation operation.

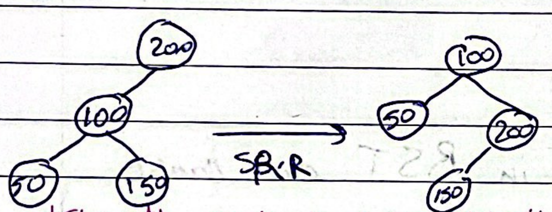
def **Rotation**: is a process of switching children and parents among two or three adjacent nodes.

* Types of Rotation:-

1- Single Rotation   2- double Rotation.

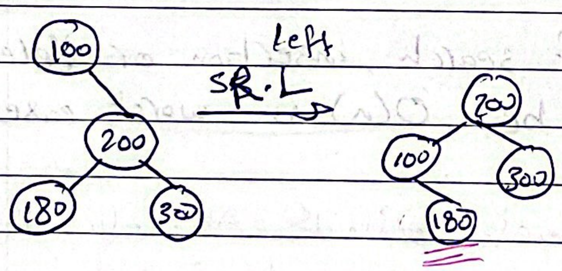
1) a-





rotate. عقدته اللي root ال left child كليها

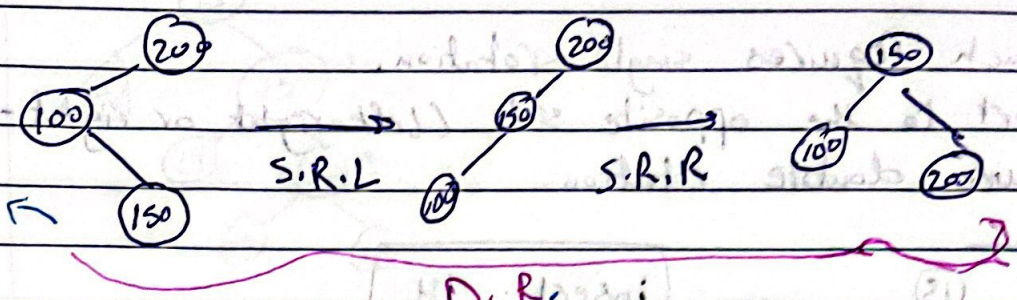
(b)



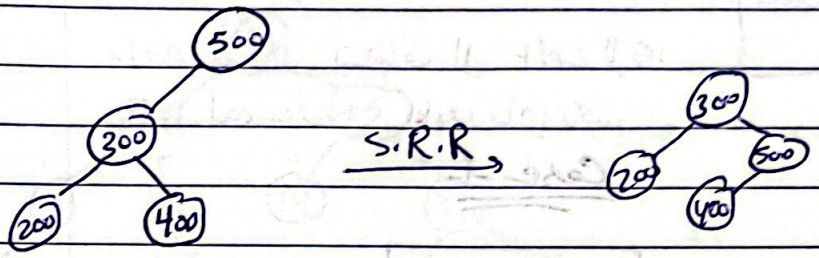
2- Double Rotation:- بعمل right بعدين left أو العكس

a- Double right - left
b- D. Left - Right

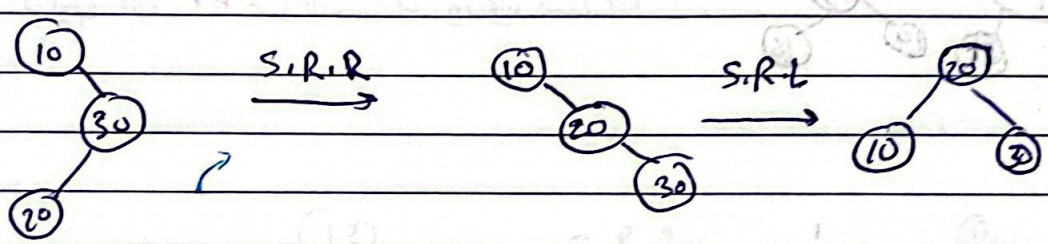
Ex



Ex



Ex



---

* ~~Oppst~~ Operations on AVL :-

1. insertion 2. deletion 3. search
4. Rotation 5. get Balance 6. Traversal (print)

---

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

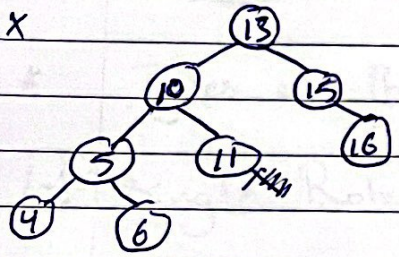
There are 3-cases in insertion

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.

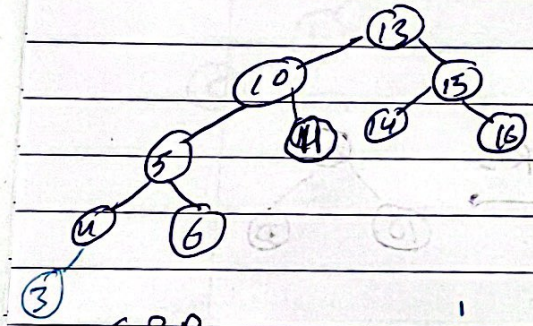
Ex



insert 14

ما فيها مشاكل بتخالف ال left لل 15 وبتكمل balanced عادي ما فيا تغيير.

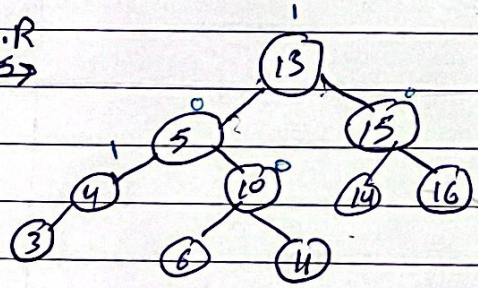
Case 1-



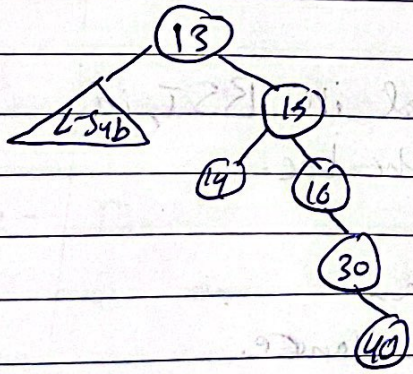
insert 3

بتكون ال left لل 4 وبصير متوازن

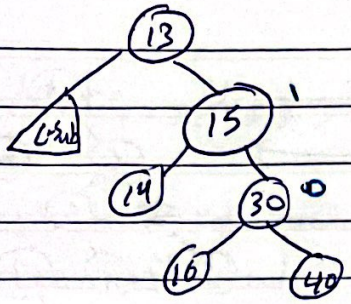
S.R.R



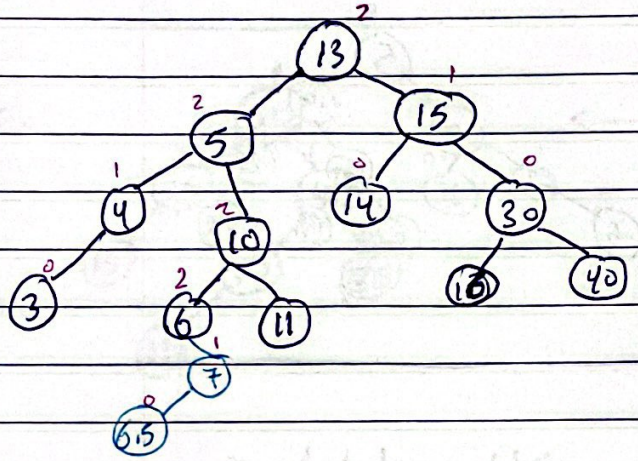
insert, 30 and 40 على الترتيب وحالتها.



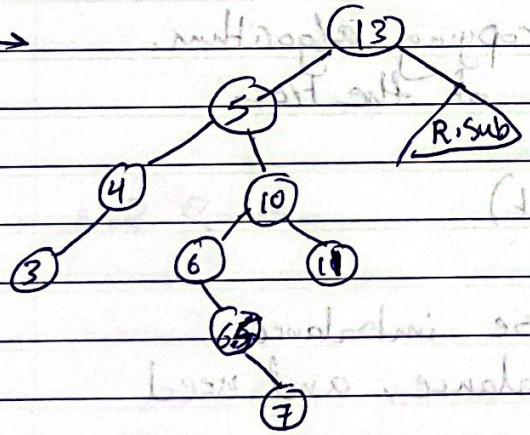
S.R.L
30-16



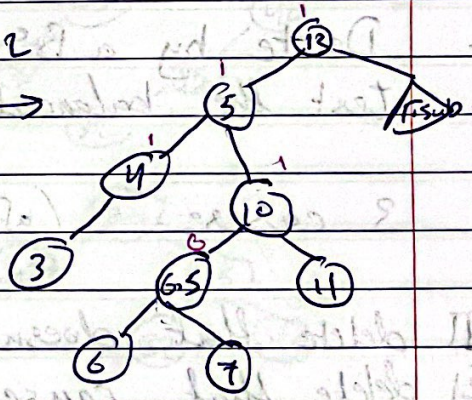
insert 7 $\hat{\imath}$ → 6.5



S.R.R →

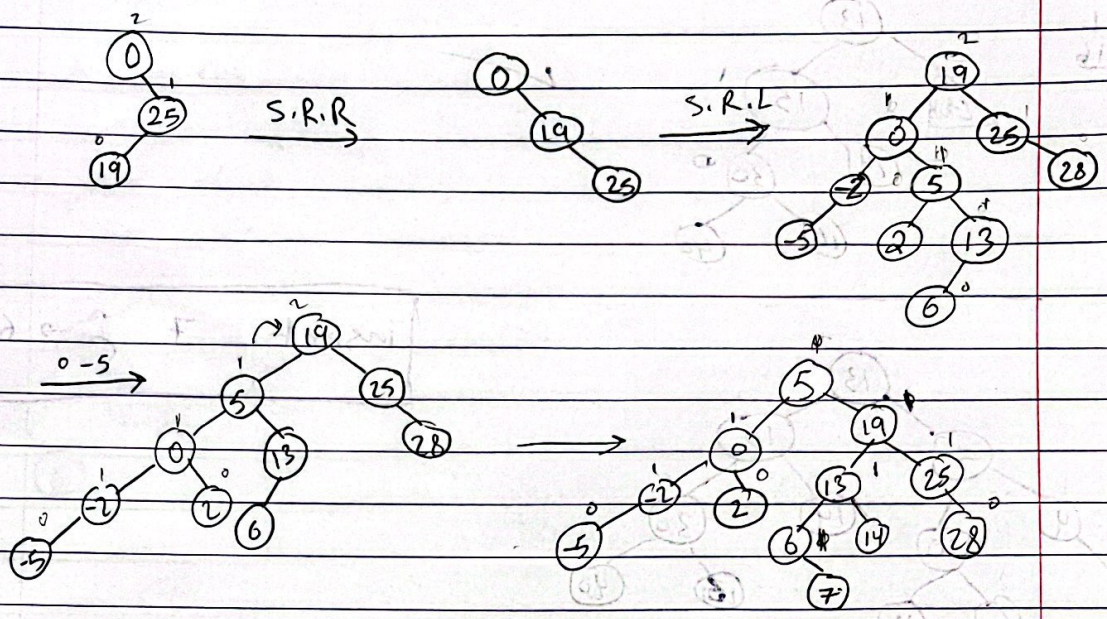


S.R.L →



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 :-

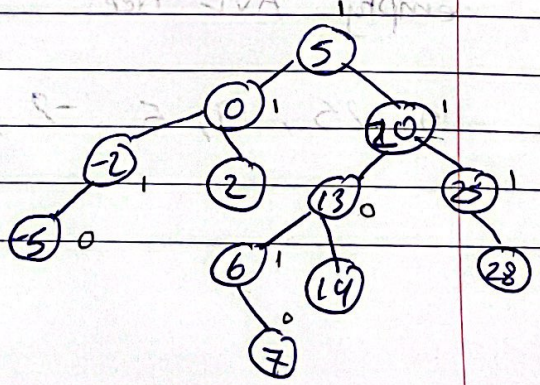
- Delete by a BST by copying algorithm.
- test the balanced factor of the tree.

→ 3 cases:- (after test)

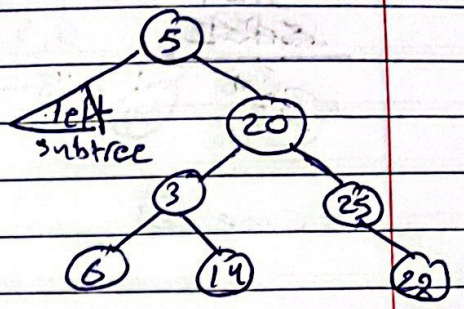
[1] delete that doesn't cause imbalance.
[2] delete that cause imbalance, and need single rotation.
[3] delete that cause imbalance, and need double rotation.

Ex

→ This AVL tree is balanced

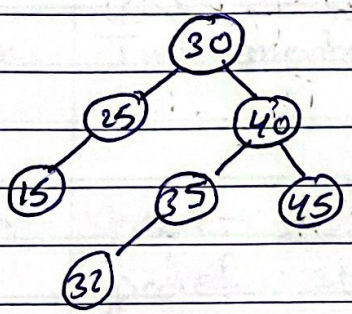


* delete 7 →



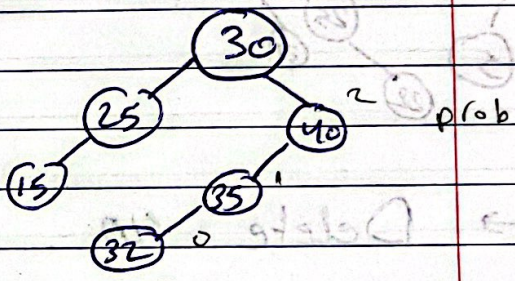
has
still balanced and ~~does~~ not
been affected. case [1]

Ex

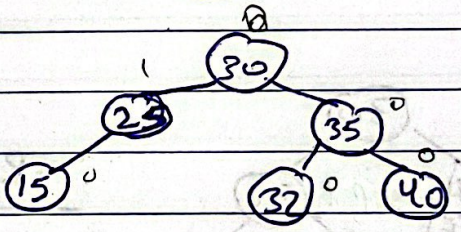


This is Balanced AVL Tree

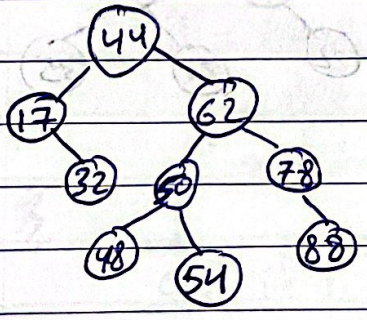
→ delete 45 →



→ S.R.R →

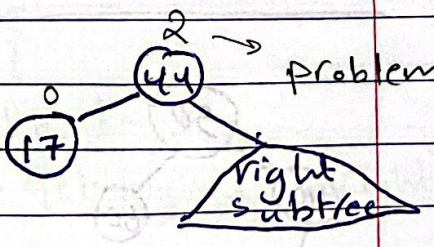


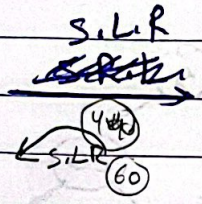
case [2]



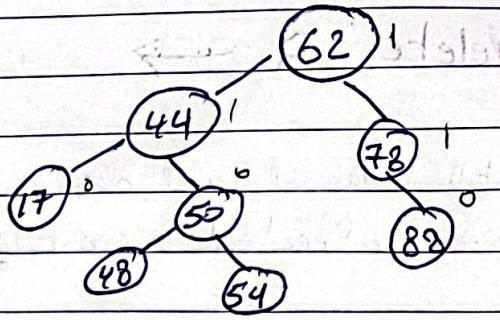
This is balanced AVL Tree

Delete 32



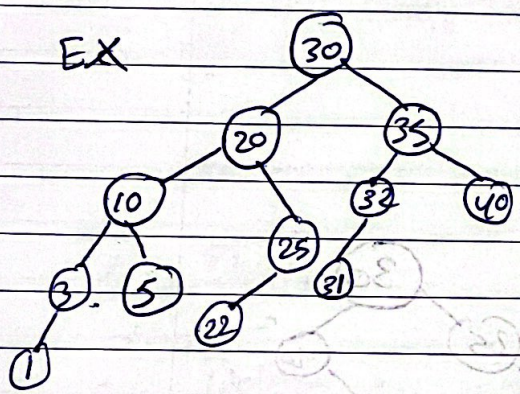


case [2]



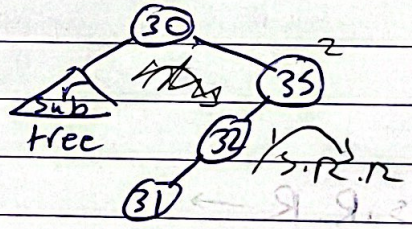
Note دائماً ما يصير ما في balanced ، فكري تجيبي نود من الجهة الأكثر فيها نودز زي الميزان تجيب من الجهة الأقل الأخف مسئان يتوازنوا

EX

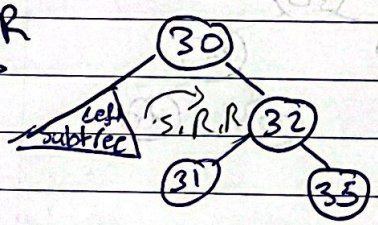


→ This is AVL T ✓

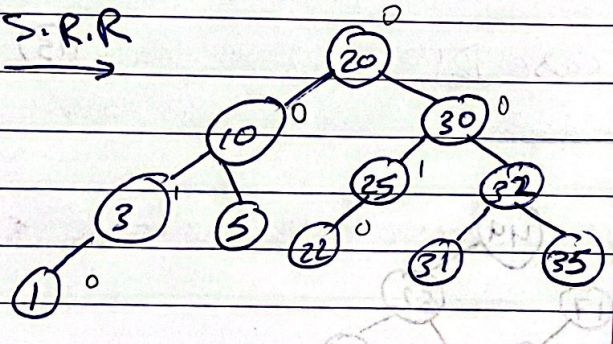
→ Delete 40 →



S.R.R ⇒

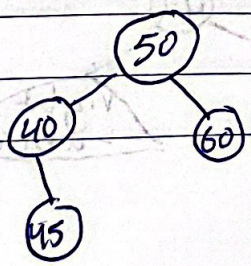


S.R.R →

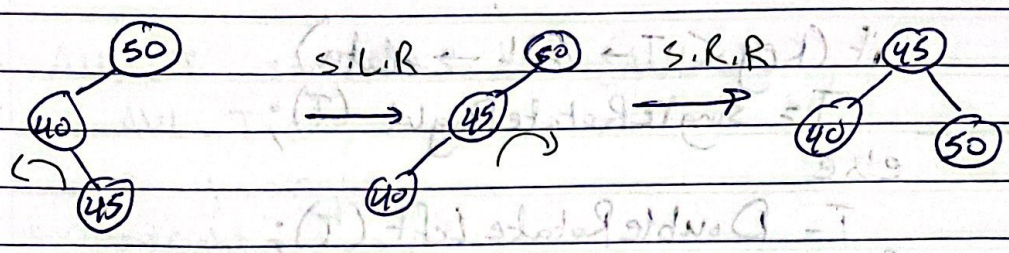


case [3]

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){
```

shift للترتيب

balance يعني

abs قيمة مطلقة عشان عكس الاطرح يطلع سالب

موجود بالزتاج في نبكتبه بعدين

```
        if (key < T→left→data)
            T= singleRotateRight (T);
        else
            T= DoubleRotate Left (T);
    }
}
else if ( key > T→data) {
    T→right= insert (T→right, key);
    if (abs(getHeight (T→right) - getHeight (T→left)) >=2){
        if (k > T→right→data)
            T= single RotateLeft(T);
        else
            T= DoubleRotateRight(T)
    }
}
T→Height = 1 + Max (getHeight (T→left), getHeight(
                                    T→right));
return T;
}
```

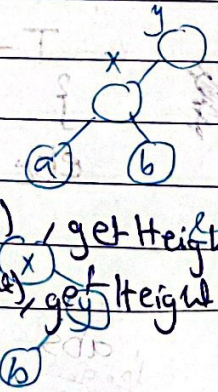
getheight Function:

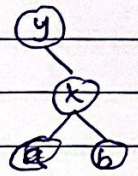
```
int getHeight (AVL_T T) {
    return T→Height;
}
```

Function: AVL_T singleRotateRight (AVL_T y){

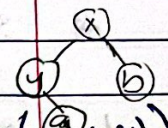
S.R.R

```
    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 K_3) {
Double Rotate Left then right
K_3 → Left = singleRotateLeft(K_3 → Left);
return (singleRotate Right(K_3));
}

---

* 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.

نقل مرة T بكون O(n) ← بعدين لو بدي اعمل عملية (find) مثلاً O(log n)
على هذا العنصر بتصير O(1)

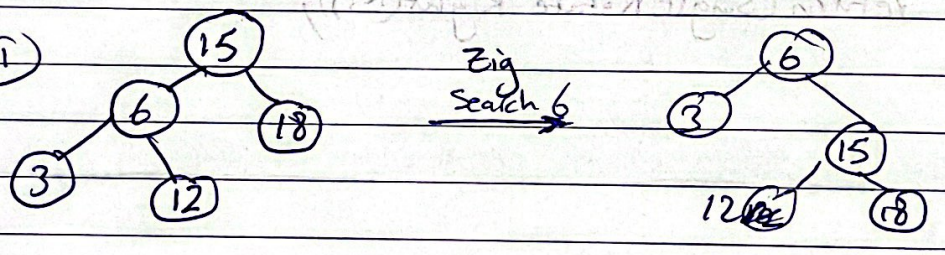
will be moved far from the root.

---

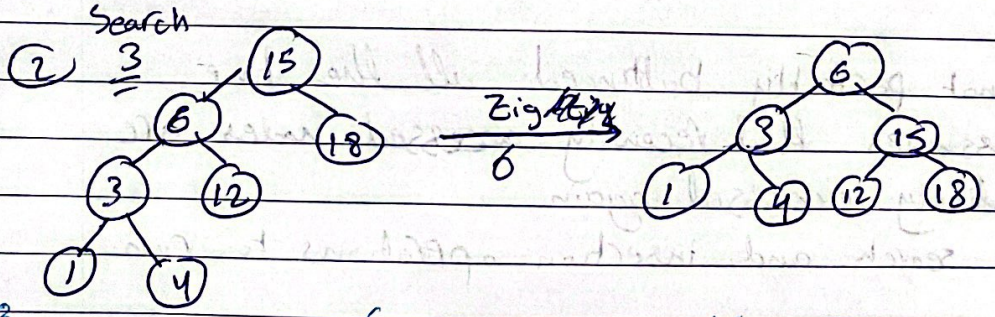
* Splaying:- هي عملية rotate ← double/Right/Left

1- Zig rotate to right
2- Zag rotate to left
3- Zig-Zag
4- Zag-Zig
5- Zig-Zig
6- Zag-Zag.

Exp:1

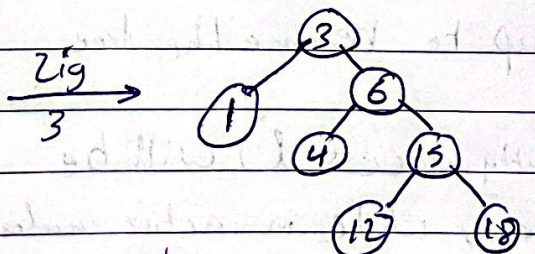


Search



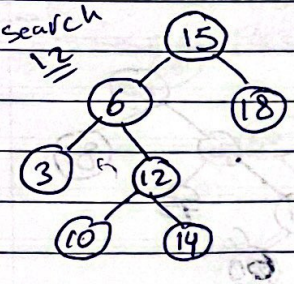
3 6 15

أول مرة بعمل ال Grandparent مع ال parent بعدين بعمل ال child مع ال parent

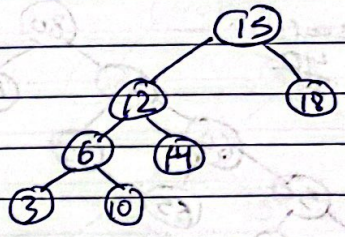


لاحظ انه هون عشان أوصل ال 3 بروح ريتنا بنفس الاتجاه left left وبوصلها.

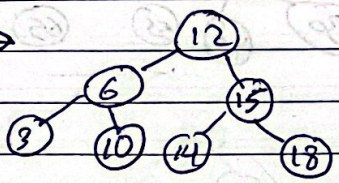
(3) search 12



zag 12



zig 12

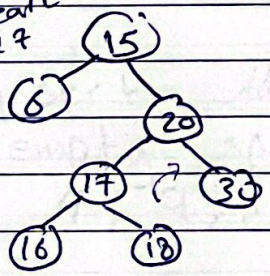


* هون عشان أوصل ال 12 رحت left بعدين right عشان هيك هون أول اشي بقل ال child مع ال parent وبعدين ال parent مع ال grandparent

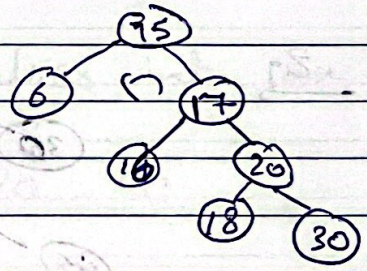
* الأسئلة على splay ما بتجي اكتب كود Implementation بس بيب: BST وبقلك اعملي search، مع دوري استعملها عشان تسرع الموضوع

---

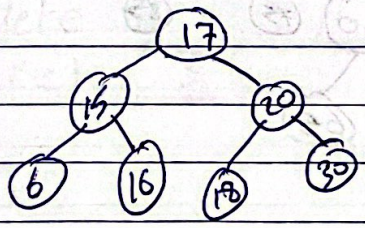
(41) search 17



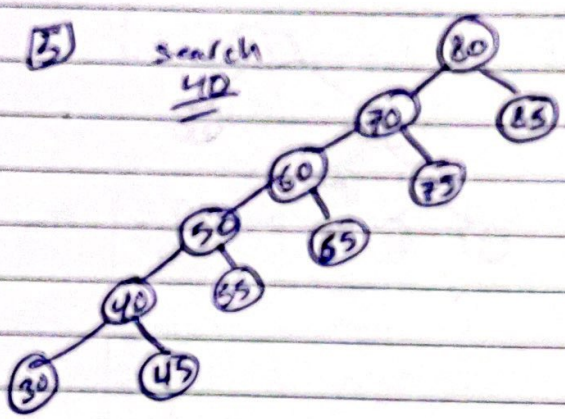
zig 17



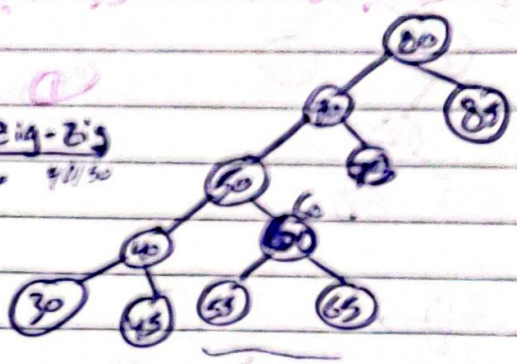
zag 17

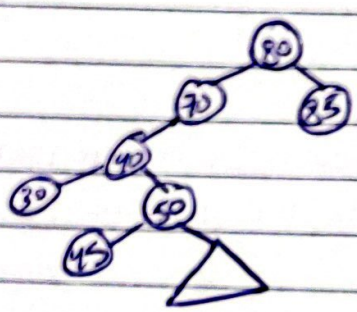


5) search 40

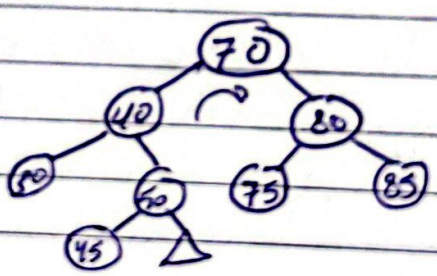


① Zig-Zig

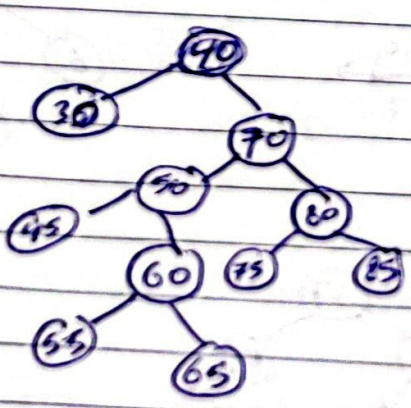




② Zig

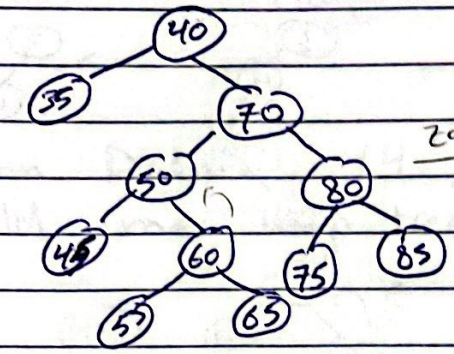


Zig

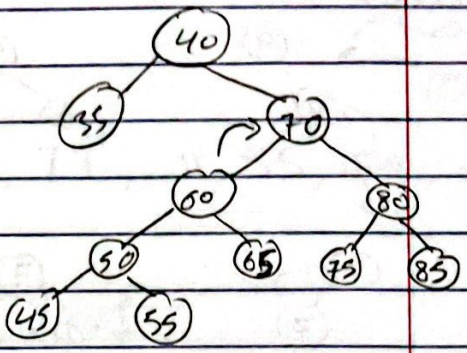


① Zig-Zig
② Zig-Zig

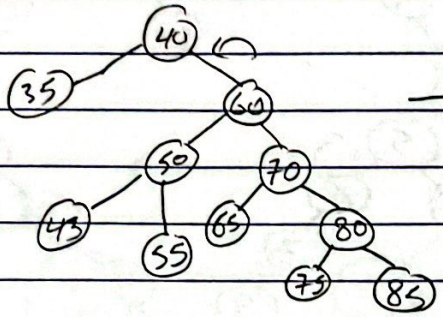
6 search 60



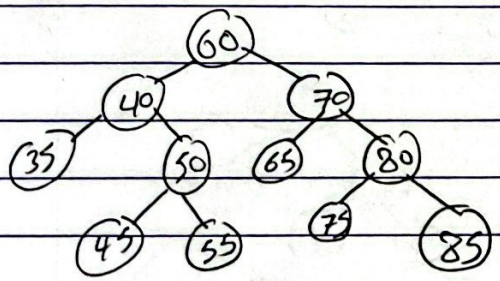
zag →



zig →



zag →



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

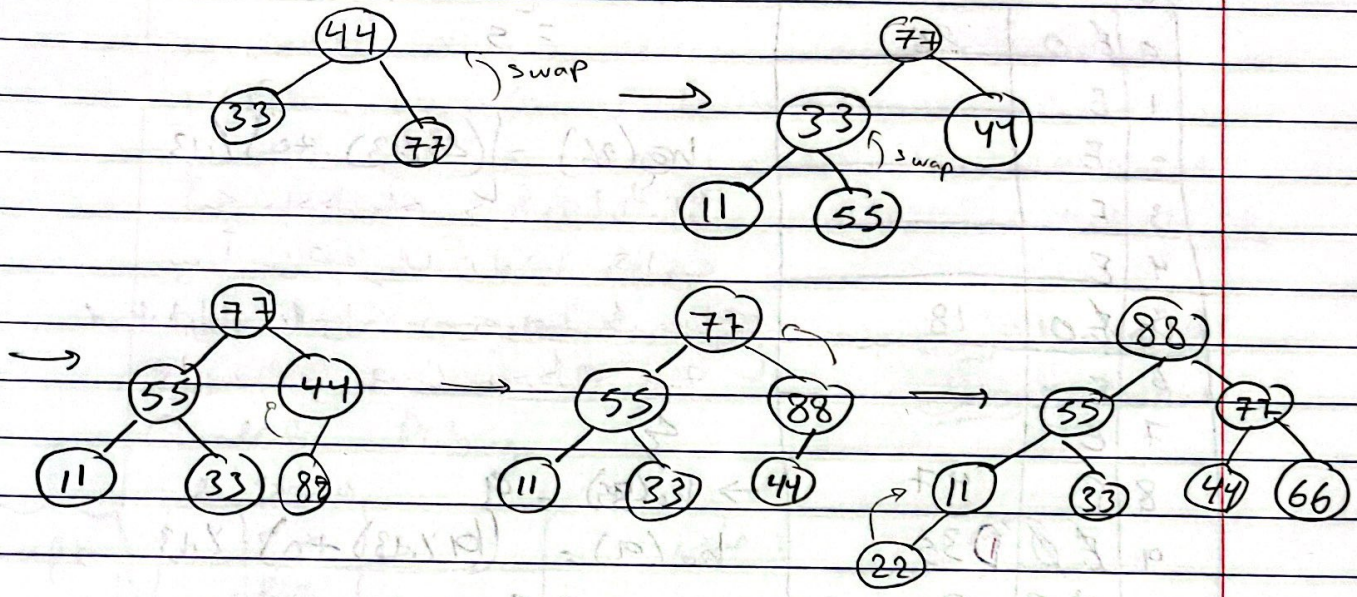
4, 9, 3, 7, 5, 6, 20, 40

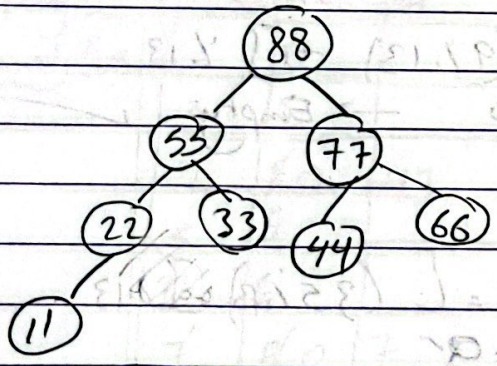
then delete 9

then search 3

16/11/2021 Heap Tuesday

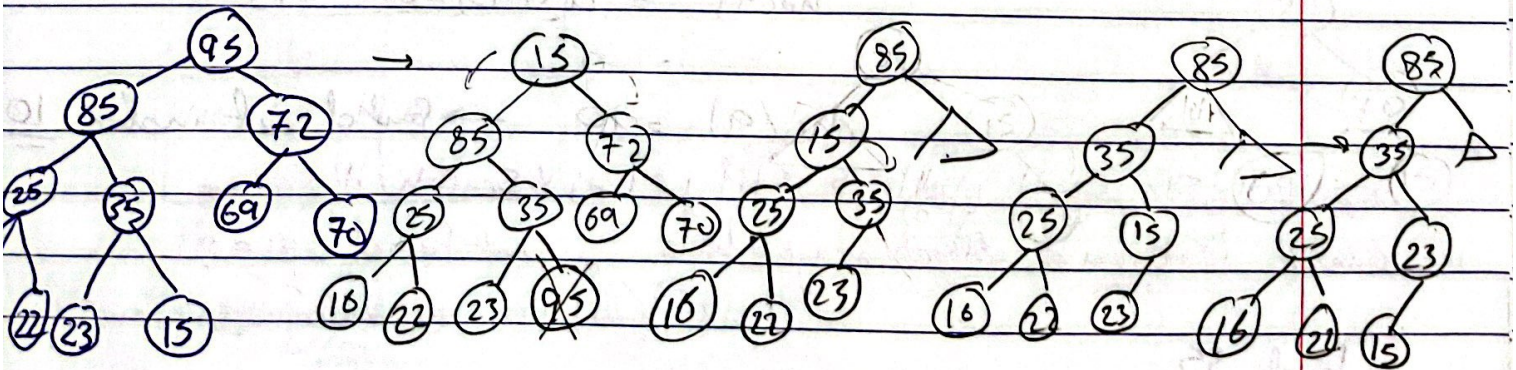
Q. Given Data: 44, 33, 77, 11, 55, 88, 66, 22
→ Build max heap tree.





1

Q Delete root 95



Hashing

Linear Probing (open Addressing)

Example

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

E : Empty
O : occupied

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

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

$\rightarrow h_0(35) = 9$
$h_0(9) = (9 \% 13) + 0) \% 13$
$= 9$ collision
$h_1(9) = [(9 \% 13) + 1] \% 13$
$= 10 \rightarrow$ Empty

* Delete (35) $= [(35 \% 13) + 0] \% 13$
$= 9$

Ddete (40) $\rightarrow h_0(40) = 1$ empty

Find 9
$h_0(9) = [(9 \% 13) + 0] \% 13$
$= 9$
$h_1(9) = 10 \rightarrow$ found at index 10

Find 35
$h_0 = [(35 \% 13) + 0] \% 13$
$= 9 \rightarrow$ not found $\rightarrow$ already deleted

13 ﮐ 815

* لو كان بال index 9 رقم بعطي نفس ال 35 زي 22 مثلاً وطلبت أعمل find 35 رح اقرا index 9 مشغول فأضطر أني أدور لحد ما يصير عندي ال Empty فبعطيني not found ويزيد بهيك ال search

* اذا كانت قيمة موجودة ب index وعملنالها delete ورجعنا طلبنا search عليها ، عندها خلص بتكون هي مش بشكل كامل ممنوعة لكن بعرف انها كانت موجودة مسبقاً وبالتالي خلص بعطي not found

اذا كان الترتيب بهمه Sorting ابعدي عن ال hashtable لأنه رح يضيع ويتوه

Quadratic Probing

Example Page 22 in slides.

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

صبحي :

$h_0(34) = 8$ collision

$h_1(34) = ((34 \% 13) + 1^2) \% 13$

$= 9$ collision

$h_2(34) = (34 \% 13 + 4) \% 13$ (2^2)

$= (8 + 4) \% 13$

$= 12$

$h_0(47) = 8$ collision

$h_1(47) = 9$ h_2 coll

$h_3(47) = (8 + 3^2) \% 13 = 17 \% 13 = 4$

طالما يصير coll بعمل قفزات "تربيعية" وبالتالي احتمال توزيع العناصر وعدم تكديسهم "أفضل"

العشوائية بال Quadratic أعلى من Linear وهاد بعتبر أفضل ، لكن مش مضمون كل تاني يجيب احتمال انك تلاقي Empty Cell

هاي ال E

int HashTable[13]; parallel array

int status [13]

or → struct أحكي
{int element; وأرتاح
char status;}

ال 26 التالية ، المفروض الـ Hash table ما نكرر العمليات

* Double hashing :-

Example p 25

@

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

$h_0(9) = 9 \% 13 = 9$ collision

$h_1(9) = [[9 \% 13 + 1]] + [9 \% 13]] \% 13$ (h_0)

$= 19 \% 13 = 6$

$h_0(96) = 5$ coll

$h_1(96) = [(96 \% 13) + 1 + (96 \% 12)]$

$= [5 + 1] \% 13$

$= 7 \% 13 = 7$

$h_0(70) = 5$ coll

$h_1(70) = [5 + 1 \times (1 + 70 \% 12)] \% 13$

$= [5 + 11] = 16 \% 13 = 3$