Data Structer

COMP 242

أ/ سوسن

chapter 1

Recursion

1] Hand shake problem

$h(2) = 1$, $h(3) = 2$, $h(4) = 3$ ...

$$h(n) = \begin{cases} 1 & , n = 2 \\ h(n-1) + (n-1) & , n > 2 \end{cases}$$

base

Recursive Part

```
Public Static int handShake(int n){
    if (n == 2)
        return 1;
    else
        return handShake(n-1) + (n-1);
}
```

(n-1) + hand(n-1)

2] Sum(int n)

```
if (n == 1)
    return 1;
else
    return n + Sum(n-1);
}
```

* كود رقم 100

$$n + Sum(n-1) = \sum_{i=1}^{100} i = 1 + 2 + 3 + \dots + 100$$

10

$$1 + Sum(n-1) = 1 + Sum(9)$$
$$1 + Sum(8)$$
$$1 + Sum(7)$$
$$\vdots$$

31 Factorial

```
Public Static long Fact(int n){
    if (n == 0)
        return 1;
    else
        return n * Fact(n-1);
}
```

$$F(n) = \begin{cases} 1, & n == 0 \\ n * Fact(n-1), & n > 0 \end{cases}$$

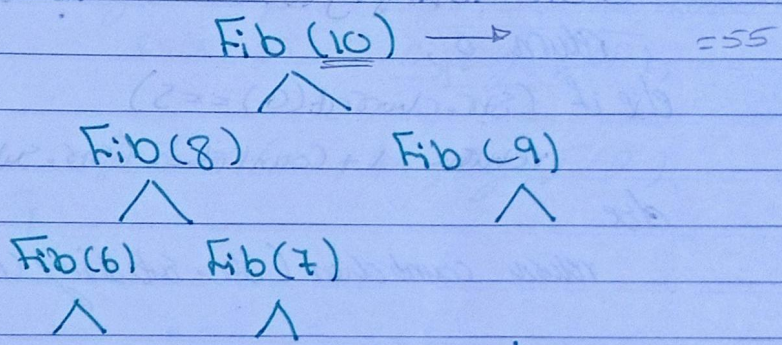
* يا إما بعطيني الكود وبده التركيز أو العكس.

[4] Fibonacci

0 1 1 2 3 5 8 13 21 ...

$$Fib(n) = \begin{cases} 0 & , n==0 \\ 1 & , n==1 \\ Fib(n-1) + Fib(n-2) & , n>1 \end{cases}$$

```
Public static int Fib(int n){
    if (n==0)
      return 0;
    if (n==1)
      return 1;
    else
        Fib(n-1) + Fib(n-2);
```



[5] Reverce

321 ← 123 : مثال لعكس الأرقام

```
Static int Sum = 0;

Public Static int Reverce (int n){
    if (n == 0)
        return Sum;
    else
        Sum = Sum * 10 + n % 10;
        Reverce n / 10;
}
```

[6] Count Char

لعد عدد الأحرف.

```
Public Static int countchar ( String Str , char s ){
    if ( Str.length() == 0 )
        ( Str.isEmpty() )
        return 0;
    else if (Str.charAt(0) == s)
        return 1 + Count Char ( Str.SubString(1) , s );
    else
        return countchar ( Str.subString(1) , s );
}
```

لبعد استبدالها بـ

[7] Count String

لعد كلمة بداخلة.

```
int Count String ( String Str , String s ){
    if (Str.length() < s.length())
        return 0;
    if (Str.subString(0, s.length(), equals(s))
        return 1 + countString (Str.substring(1) , s);
    else
        return countString (Str.subString(1) , s);
}
```

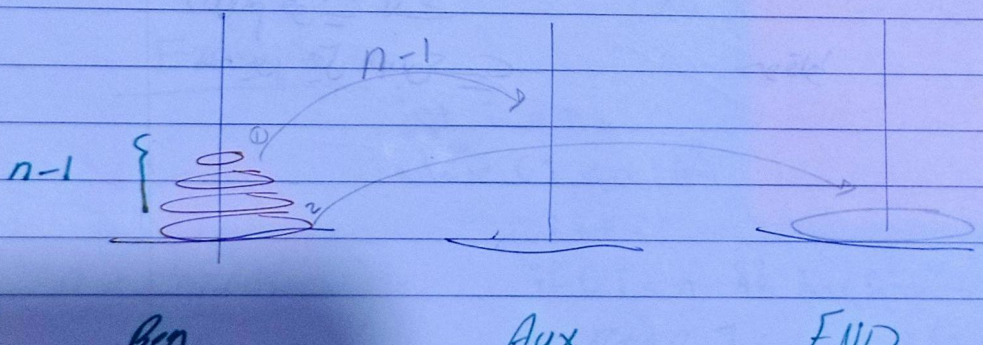
[5] Revers String.

```
public static String Reverce (String str ){
    if (str. is Empty ())
        return str ;
    else
        return Reverce (str. subString (1)) +
                str . charAt (0);
}
```

namolleh ← Hello man

[9] Hani Tower.

```
Hani Tower (int n , char Beg, char Aux, char END) {
    if (n>0) {
        Hani Tower (n-1, Beg, end, aux);
        System.out.printf ("\n Mov. disk %d from %c to %c "
            Disk , n, Beg, END);
        Hani Tower (n-1, Aux, Beg, END );
    }
```



Palindrome

```
Pablic Static boolean isPali (int [] arr , int i , int j){
    if (i > j)
        return True ;

    else if (arr [i] != arr [j] )
        return false ;

    else
        return isPali (arr, ++i, --j );
}
```

* Done

بس حلي اكتب سؤال.

لا حول ولا قوة
إلا بالله العلي
العظيم - ه

اللهم قوني بك
حين تقل حيلي

Analysis of Algorithm

chapter 2

(DS - AOA)

Problem Solving :-

Problem

A1 A2 A3 → * لازم يكونوا على نفس الـ Hardware & Software.

أهم اشي في الـ Algorithm

- Speed (قديش آخذ وقت).
- Space.

عشان أعرف أنو الصح والأفضل المفروض أجرب

تجارب 1:- Experimantel → implement

timing time();

Ploting بشوف إذا الرسمة بتطلع فنكشن معروفة

رياضيات 2:- Asymptotic → T(N)

All possibeltion. جميع الاحتمالات.

Example :-

n = element in Array

```
int i = 0;
while (i < n && Arr[i] != key) {
    i++;
if (i < n && Arr[i] == key)
    return 1;
else
    return 0;
}
```

Best case → 1 : في عندي

Average case → Array ال element

Worest case → N

$1 \leq \log n < n < n\log n < n^2 < 2^n < \infty$

$n! < \infty < n^n$

* Best → Worest

في عندي 3 تعريفات :

Big-O → upper bound

Big-Ω → lower bound

Big-Θ → Average bound.

$T(N) = O(g(n))$ if there exist constant C and n_0, such that $T(n) \leq cg(n)$ where $C > 0$, $n_0 \geq 0$

$T(n) = \frac{n(n-1)}{2} = \frac{n^2-n}{2} \leq \frac{1}{2}n^2 \leq n^2$

$\therefore O(n^2)$

* أنا بوقف هنا order

Big O : $f(n) = O(g(n))$ if $\exists$ constant c, n_0 such $\forall$ $c \geq 0$, $n_0 \geq 0$ where
$f(n) \leq c\,g(n)$.

Big Ω : $f(n) = \Omega(g, n)$, if $\exists$ constant c, n_0 such $\forall$ $c \geq 0$, $n_0 \geq 0$ where
$f(n) \geq c\,g(n)$

Big θ : $f(n) = \theta(g, n)$ if $\exists$ constant c_1, c_2, n_0 where $(c_1, c_2, n_0) \geq 0$ such that
$c_1 g(n) \leq f(n) \leq c_2 g(n)$.

Ex $f(n) = 2n + 3$

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

$f(n) \leq c\,g(n)$

$2n + 3 \leq 2n + 3n$ ✓

$2n + 3 \leq 5n$

* ثم أصلاً بنشيل ال constant

$\therefore f(n) = \Omega(n)$

$f(n) \geq c\,g(n)$

$2n + 3 \geq 1n$ ✓

$\therefore f(n) = \theta(n)$

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

$1n \leq 2n + 3 \leq 5n$

Ex $n^2 \log n$.

$f(n) = O(n^2 \log n)$ ← $f(n) \leq c\,g(n)$, $n^2 \log n + c \leq 10\, n^2 \log n$.

$f(n) = \Omega(n^2 \log n)$ ← $f(n) \geq c\,g(n)$, $n^2 \log n + n \geq n^2 \log n$.

$f(n) = \Theta(n^2 \log n)$.

Ex $f(n) = n!$ $\quad 1 \leq 1 \times 2 \times 3 \times (n-1) \times n < n^n$

$f(n) = O(n^n)$ (upper)

$f(n) = \Omega(1)$ (lower)

$f(n) = \Theta$ unknown.

$f(n) = n^2 + n$

$g(n) = n \log n$.

1. $f(n) + g(n) = \max(f(n), g(n))$ $\quad (n^2 + n)$

2. $f(n) * g(n) = O(f(n) * g(n))$ $\quad (O(n^3 \log n))$

$\log n! \rightarrow O(n \log n)$

$f(n) = n^2 \log(n)$ $\quad \log n^2 \log(n)$ $\quad \log n (\log n)^{10}$

$g(n) = n(\log n)^{10}$ $\quad \frac{2\log n}{n^2} + \frac{\log n \log n}{n} > \frac{\log n}{n} + \frac{10 \log n \log n}{10n}$

$f(n) = 3n^{\sqrt{n}}$ $\quad 3n^{\sqrt{n}}$ $\quad 2^{\sqrt{n}\log n}$

$g(n) = 2^{\sqrt{n}\log n}$ $\quad 3n^{\sqrt{n}} = (n^{\sqrt{n}})^{\log_2 3}$

$\log_a^b = \log_c^a$

$f(n) = 2^{\log n}$ $\quad \log 2^{\log n}$ $\quad \log n^{\sqrt{n}}$

$g(n) = n^{\sqrt{n}}$ $\quad \log(n)(\log_2 2) < \sqrt{n} \log n$

$f(n) = 2^n$ $\quad \log 2^n$ $\quad \log 2^{2n}$

$g(n) = 2^{2n}$ $\quad n \log_2 2$ $\quad \log_1 2^{2n}$

$\frac{n \log_1 2}{} < \frac{2n \log_2 2}{}$

$f(n) = n^{\log n}$ $\quad \log 2^n$ $\quad \log n^{\log n}$

$g(n) = 2^n$ $\quad n \log_2 2$ $\quad (\log n)(\log n)$

$\log n$ $\quad \log(\log(n))^2$

$\log n < 2\log(\log n)$

```
Sum (A, n) {
    S = 0;                          — (1)                  (1)
    for (i = 0 ; i < n ; i++)       — (3)     2n + 2
        S = S + A[i];               — (2)     2n
    return S;                       — (1)
}
```

(Above the for loop: 1 over $i=0$, $n+1$ over $i<n$, n over $i++$)

Big O (n) = $2n + 2 + 2n$ — Time Complexity

$= 4n + 4$

$= \boxed{n}$

S = 0 — (1)

i = — (1)

Array — $\boxed{n}$

Space Complixity.

* اكم variable عندي

وبعدهم بيزيدوا عندي في ال Ram.

```
Add (A, B, n) {
(n+1)     — for (i ≠ 0 ; i < n ; i++)          (1)
n(n+1)    — for (j = 0 ; j < n ; j++)          (1)
      n^2 — C[i][j] = A[i][j] + B[i][j];
}
```

(Under C[i][j], A[i][j], B[i][j]: n^2, n^2, n^2)

Time $O(n) = 2n^2 + 2n + 1$

$= n^2$

Space. $3n^2 + 2$

$\boxed{n^2}$

```
Multi (A, B, n) {
n+1         for (i = 0 ; i < n ; i++)
n(n+1)          for (j = 0 ; j < n ; j++)
n x n (n+1)         for (k = 0 ; k < n ; k++)
  n x n x n             C = C + A + B ;
}
```

Time = $O(n^3)$

Space = $O(n^2)$. ?

```
for (int i = 0; i < n; i++)        → O(n)
    for (int j = 0; j < n; j++) {  → O(n)
        Statment;
    }
```

* الـ أخذنا عدد الـ مرة
الـ قبل ><

stop. $i \geq n$

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

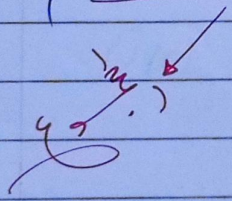
i	j	time
0	0	0
1	1	1
2	2	2
⋮	⋮	
k	⋮	k

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

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

* $O(n^2)$

```
p = 0;                          p > n  stopeal.
for (int i = 1; p <= n; i++) {
    p = p + i;
}
```



i	p
1	0+1 = 1
2	0+1+2 = 3
3	1+2+3 = 6
4	1+2+3+4 = 10
⋮	⋮
k	1+2+3+4+⋯k ⟹ $\frac{k(k+1)}{2}$

$$\frac{k(k+1)}{2} > n$$

$$\sqrt{k^2} > \sqrt{n}$$

* $= O(\sqrt{n})$

```
for(int i=0; i<=n; i+=2) {
    Statment;
}
```

i = 0, 2, 4, 6, ... k

Stop in $i > n$

$k > n$

* increment and decrement $\therefore O(n)$.

same time (O( ))

```
for( int i=1; i<=n; i*=2) {
    Statment;
}
```

$i = 2^0, 2^1, \dots 2^k$

Stop in $i > n$

$\log 2^k > \log n$

$\therefore O(\log n)$. ← $k > \log n$

```
for (int i=n; i>=1; i/=2) {
    Statment;
}
```

i

$n \rightarrow \frac{n}{2^0}$

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

$\vdots$

$\frac{n}{m} \rightarrow \frac{n}{2^k}$

Stop in $i < 1$

$\log \frac{n}{2^k} < 1 \log$

$\therefore O(\log n)$

$O(n)$ ← ضرب/قسمة

$O(\log n)$ ← ضرب/قسمة

*

```
P = 0;
for (i = 1; i < n; i = i * 2) {        → log(n)
    P ++;
}
                 هيساوي logn
for (j = 1; j < P; j *= 2) {
    Statment;                          → log(n)
}
```

∴ log (log n).

```
for (i ≠ 0; i < n; i++)                → n
    for (j = 1; j < n; j=*2)           → logn
    {
        Statment
    }
```

∴ O(n log n).

```
P ≠ 0;
for (int i = 1; P < n; i++) {
    P += i;   ⟹ P = P + i
    for (int j = n; j > 1; j /= 2) {
        Statment
    }
```

$\sqrt{n}$ · logn

∴ O($\sqrt{n}$ logn).

سواء for أو while
نفس النتيجة

Recursion.

```
Test (int n) {
    if (n>0) {
        print (n);
        Test (n-1); }
}
```

* Print Down.

* لما تكون مع for تقدر تتحكم بال max

1. $T(n) = \begin{cases} 1 & n \leq 0 \\ T(n-1)+1 & n > 0 \end{cases}$

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

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

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

* بعوضهم جوا بعض

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

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

$\vdots$

$T(n) = T(n-k) + k$

* Assume $n - k = 0$

$n = k$

$T(k) = T(0) + n$

$T(k) = \cancel{1} + n$

$\therefore O(n)$.

```
void Test (int n){
    if (n>0)
        for (int i = 1; i<n; i*= 2) ⇒ log n
            System.out.print(i);
        Test (n-1);
    }
```

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

$T(n-1) = T(n-2) + \log(n-1)$ ~ ①

$T(n-2) = T(n-3) + \log(n-2)$ ~ ②

$T(n) = (T(n-2) + \log(n-1)) + \log n$

$T(n) = T(n-2) + \log(n-1) + \log(n)$

$T(n) = T(n-3) + \log(n-2) + \log(n-1) + \log n$

⋮

$T(n) = T(n-k) + \log(n-k-1) + \log(n-k+2) \sim + \log(n)$

Asume $k-n=0$

$k=n$

$T(n) = T(n-n) + \log(n-n-1) + \log(n-1) + \log(n)$

$= T(0) + \log(1) + \log(2) + \sim + \log(n-1) + \log(n)$.

$T(n) = 1 + \log(1 \times 2 \times 3 \times 4 \times \sim (n-1) \times n$

$T(n) = \log n!$

$= O(n \log n)$.

$$T(n) = 4T(n/2) + n + 1$$

$$T(n/2) = 4T(n/2^2) + n/2 + 1 \quad \text{--- ①}$$
$$T(n/2^2) = 4T(n/2^3) + n/2^2 + 1 \quad \text{--- ②}$$

$$T(n) = 4T(n/2) + n + 1$$
$$= 4[4T(n/2^2) + n/2 + 1] + n + 1$$
$$= 4^2 T(n/2^2) + 2n + 4 + n + 1$$
$$= 4^3 T(n/2^3) + 4^2 n/2^2 + 4 + 2n + 4 + n + 1$$

$$= 4^3 T(n/2^3) + 7n + 5$$

$$T(n) = 4^k T(n/2^k) + cn$$ (c: constant)

$$\log n = 2^k \log$$

$$\log n = k$$

$$2^n(1) + cn$$

Big O (2^n)

✳

✳ هاد حل السؤال الأخير
في آخر صفحة
في الكتاب كومبليكسي

$T(n) = T(n+1) + 1 \Rightarrow O(n)$

$= T(n+1) + n \Rightarrow O(n^2)$

$= T(n) + \log n \Rightarrow O(n \log n)$

$= T(n) + n^2 \Rightarrow O(n^3)$

$= T(n) + n\log n \Rightarrow O(n^2 \log n)$.

constant

$T(n) = T(n-2) + 1 \quad O(n)$

$T(n-a) + b \quad O(nb)$

لـ مش شرط رقم ممكن فنكشن

algo

```
Test (int n) {
    if (n>0) {
        Print (n);
        Test (n-1);
        Test (n-1);
}
```

مهم جدا

$$T(n) = \begin{cases} 1 & n == 0 \\ 2T(n-1)+1 & n > 0 \end{cases}$$

$T(n) = 2T(n-1) + 1 \rightarrow T(n-1) = 2T(n-2) + 1$ ①

$T(n-2) = 2T(n-3) + 1$ ②

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

$= 2^2\, T(n-2) + 2 + 1$ — by ①

$= 2^2 (2T(n-3)+1) + 2 + 1$

$= 2^3\, T(n-3) + 2^2 + 2^1 + 2^0$ — by ②

Assume $n - k = 0 \rightarrow \boxed{n = k}$

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

$= 2^n + 2^{n-1} + 2^{n-2} + \cdots + 2^0$ $\quad n=k$ 대입 *

$\sum_{k=0}^{n} a^k = \frac{1-a^{n+1}}{1-a} \Rightarrow \sum_{i=0}^{n} 2^i = \frac{1-2^{n+1}}{1-2} = \boxed{2^{n+1}-1}$

* $\boxed{O(2^n)}$

$T(n) = 2T(n-1) + 1 \longrightarrow O(2^n)$

$T(n) = 3T(n-1) + 1 \longrightarrow O(\frac{3}{2}3^n)$

$T(n) = 3T(n-1) + n \longrightarrow O(n\,3^n)$

$T(n) = 3T(n-1) + \log n \longrightarrow O(3^n \log n)$

```
Public static int Test(int n){
    if (n == 0)
        return 0;
    else
        return 2T(n/2) + n;
}
```

$$T(n) = \begin{cases} 1, & n = 0 \\ 2T(n/2) + n, & n > 0 \end{cases}$$

$T(n/2) = 2T(n/2^2) + (n/2)$ ... ①

$T(n/2^2) = 2T(n/2^3) + (n/2^2)$ ... ②

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

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

$= 2^2 T(n/2^2) + 2n/2^1 + n/2^0$

$= 2^2(2T(n/2^3) + n/2^2) + 2n/2^1 + n/2^0$

$= 2^3 T(n/2^3) + 2^2 n/2^2 + 2n/2 + n/2^0$

$2^3 T(n/2^3) + n$

$T(n) = 2^k T(n/2^k) + kn$

$n + n\log n$

$\therefore O(n \log n)$

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

$\log n = 2^k \log 2$

$\log n = k$

* حل آخر للمثال

```
Test (int n){
    if (n==1)
        return 1;
    else{
        Print (n);
        return 4T(n/2) + n;
    }
}
```

*

Big O
2^n

$$T(n) = \begin{cases} 1 & , n==1 \\ 4T(n/2)+n+1 & , n>0 \end{cases}$$

Chapter 3 :-

Linked List

Node

data	next

↳ location in Memory

Node → Data بيشيل ال

header → ماسكة الداتا من أولها.

```
Class Node {
    Privete String name;
    Privete int id;
    Node next;
```

Node next → عشان أعرف هي وين رايح.

Operations:-

① insert ② delet ③ update ④ Traverse.

```
Public class Node {
    privet roll;
    privet int age;
    Node next;
                        → Node () {}
    Node (int roll, int age){
        this.roll = roll;
        this.age = age;
        Next = null; }
```

Node () {} ← constructer فاضي

* ونعمل setter , getter

ونعمل toString ← أو بدالها نعمل printinfo

method بدل فيها شو بدي اطبع

Driver

```
class single ll {

    Node header = new Node();                    File بدنا نقرأ
    File file = new File("data.txt");
    Scanner in = new Scanner(file);

    while (in.hasNextLine()) {
        String str = in.nextLine();
        بدي أقطع ← String [] data = str.split(":");

        header = insertLL(header, Integer.parseInt(data[0]),
                 Integer.parseInt(data[1]));

    } (while)
}
```

* insert في البداية.

```
public static Node insertLL(Node header, int roll, int age){
    *هون - أنا حطيت فوق الـ header    Node first = header;
    *نشأنا الـ Node.    Node newnode = new Node(roll, age);
        if (first == null) {
            return newnode;    → header = newnode;        إذا كانت فاضية.
        }
        else {
            newnode.next = first;
            first = newnode;    → header = newnode;        مش فاضية
        }
        return first;
}
```

Big O(1)
* constant

عشان أتأكد بعمل method تطبع شغلي.

```
printList(header);
```

```
printList(Node header){
    Node ptr = header;
    while(ptr != null){
        ptr.printInfo();
        ptr = ptr.next;
    }
}
```

O(n).

```
public static Node insertAtPosition(Node header, int roll,
                                    int age, int key){
    Node ptr = header;
    Node newnode = new Node(roll, age);
    if(ptr == null)
        ptr = newnode;
    else {
        while(ptr != null && ptr.getRoll() != key){
            ptr = ptr.next;
            if(ptr.getRoll() == key){
                newnode.next = ptr.next;
                ptr.next = newnode;}
            if(Ptr.next == null){
                newnode.next = ptr;
                Ptr = newnode;
            }
        }
    return ptr;
```

Array and linked list store collection of data.

Array ⇒ Store all element in one block of memory.

Linkedlist ⇒ allocate memery for each element separately and only when necessary.

Disadvantages of Array.

[1] Size is fixed. → حجمها ثابت مقدرش اعدل عليها واذا تغير حجمها ديناميكياً دح تحتاج وحدات من الذاكرة
ال Array اللي حجمها كبير في الها عيوب
(1) 90 أو 30 ٪ بس بستخدم من الاماكن والباقي بكون فاضي وحجز مساحة ع الفاضي.

[2] Inserting and deleting elements in to the middle of array is potentially expensive because existing elements need to be shifted over to make room.

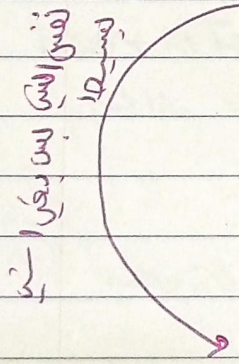
اضافة وحذف عناصر من منتصف الـ Array مكلف جداً لأن العناصر الموجودة بحاجة لـ ــــــ اعمل shift وهذا شي بطيء.

Array versur linked list:-

مناسبة Array ⇒ insert and delete at end.
Randomly accesing any element وصول عشوائي
searching the list for a particular value. البحث عن قيمة معينة

Linked list ⇒ Insert and Delete
تطبيقات بتطلب وصول متسلسل Application where sequential access is required.
الحالات اللي ما بقدر اتنبأ فيها number of element can't be predicted
بعدد العناصر مسبقاً befor hand.

```
public void insertLast (int data){
    Node ptr = header;
    Node newnode = new Node(data);

    if (ptr == null){
        ptr = newNode;
    }
    else { while (ptr.next != null){
            ptr = ptr.next; }
        ptr.next = newnode;
    }
}

Public void deleteFirst (){
    Node ptr = head;
    if (head == null){
        System.out.println("Empty!!"); }
    else if (head.next == null){
        head = null;
    }else{
        ptr = head.next;
        head.next = ptr;
    }}
```



delete last

```
else if ( head.next.next == null ){
    head.next = null ; }
else { while ( ptr.next.next != null ){
    ptr = ptr.next ;
    }
    ptr.next = null ; }
```

```
public void deletePossision (int key){
    Node ptr = head.next;
    Node prev = head;

    if (head == null){
        System.out.println("Linked List Empty !");
    }
    else{
        while ( ptr != null && ptr.getdata() != key){
            prev = ptr;
            ptr = ptr.next;
        }
        if (ptr == null){
            System.out.Print("Not found");
        }
        else{
            prev.next = ptr.next;
            ptr.next = null;
        }
    }
}
```

double LinkedList

Chapter 4

```
class Node {
    private key;
    Node next;
    Node prev;

    Node () {
    }
    Node (int key) {
        this.key = key;
    }
```

getter , setter

to string

```
class DLinkedList {
    DLinkedList () {
        Node header, tailer;
    }
```

insert At First.

```
public void insertAtFirstD (int key) {
    Node newnode = new Node ();
    if (header == null) {
        header = tailer = newnode;
    }
    else {
        newnode.next = header;
        header.prev = newnode;
        header = newnode;
    }
}
```

→ insertAtLastDl

```
public void insertAtlastD(int key) {        ← key
    Node newnode = new Node();
    if (tailer == null) {
        tailer = header = newnode;
    }
    else {
        tailer.next = newnode;
        newnode.prev = tailer;
        tailer = newnode;
    }
}
```

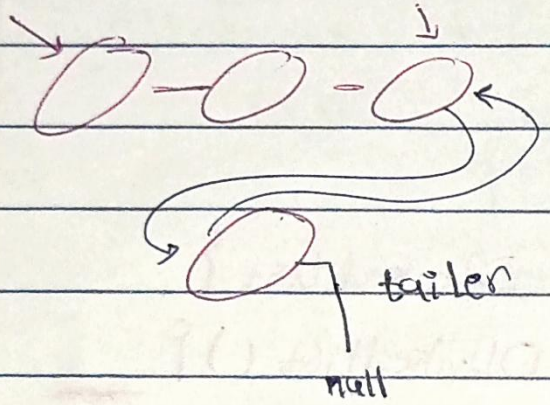
Sorgular:

tailer.next = newnode;

header.prev =

header

~~tailer~~



} ← Sorgular için

Delete from position :-

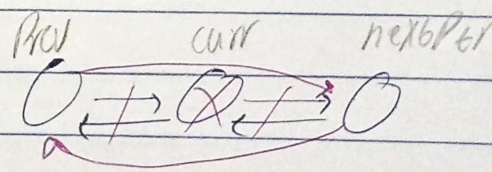
```
public void deleteDLElement (int key){
*  Node Ptr = header ;        Node current ;
        if(header == null){
            System.out.print("out of flow");
            return ;
        }

        else if ( header.getKey() == key && header.next == null()
            header = null ;
        }

        else {
            while( Ptr != null && Ptr.getkey() != key )
            ~~current = Ptr ;~~
            ~~if (Ptr.getkey() == key~~
            Ptr = Ptr.next ;
            if (ptr.getkey() == key ){
                Prev.next = Ptr.next ;
                Ptr.next.prev = Prev Ptr ;
                Ptr.next = null ;
                Ptr.Prev = null ;
            }
```

```
~~Ptr =~~
Prev = Ptr ;
Ptr = Ptr.next ;
```

أفضل اشي لازم يكون عندي 3 شغلات

- Prev
- current
- next Ptr.

```
Current = header
Prev = current
next ptr = current.next ;
```




```
Prev.next = next Ptr ;
next Ptr.prev = previous ;
current.next = null ;
  "    .prev = null
```

~~Delete at first~~

Delete from first :-

```
public void delebFromFirsb ( ) {
    Node ptr ;
    if (header == null) {
    —.print(" empty");
    }
    else {
        ptr = header.next;
        header.next = null;
        Ptr. prev = null;
        heder = ptr ;
    }
}
```

⇒ else if (header. next == null)
header = null;

System. gc (); | memory تفضيه تستخدم *

delete from last :-

```
public void deletefromlast () {

    if (tailer == null ){
        — print(" empty");
    }
    else if ( tailer .prev == null) {
        tailer = null; }
    else {
        ptr = tailer .prev;
        tailer. prev = null;
        ptr. net = null;
        tailer = ptr ;
} }
```

insert At position :-

```
public void insertAtposision (int data , int element){
    Node current = header;
    Node newnode = new Node(data);
    if (header == null ){
        print (" data not found ");
        header = tailer = new node;
    }
    else{ while (curre.next != null && current.getdata()
                                        != element)
        curr = curr.next;
        else if (curr.getdata() == element && curr.next != null){
        newnode.next = curr.next;
        newnode.prev = curr;
        curr.next.prev = newnode;
        curr.next = newnode;
    }
    else{
        curr.next = newnode;
        newnode.prev = curr;
    }
}
}
```

```
RadixSort (int [] a){
    int max = getMax(a);
    for (int i=max ; i>0 ; max /=10){
        countSort(a, p);
        p= p*10;
    }
```

phase.

element

$O(d * (n+k))$

- d ← number of digit
- k ← Range

* هل استقرار كل الخوارزميات؟

Normalization ← الى

```
public int getMax(int arr[]){
    int max = arr[0];
    for(int i=1; i<arr.length(); i++)
        if(arr[i]>max)
            max = arr[i];
    return max;
}
```

* Couting Sort :- no comparssion, Sort according to keys counting the dome having distinct key values.

Ex

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	1	1	0	2	5	4	0	2	8	7	7	9	2	0	1	9

Range → $0 \le a[i] \le 9$ (K)

Size = k+1

	0	1	2	3	4	5	6	7	8	9 (k)
count	3	3	4	0	1	1	0	2	1	2

	0	1	2	3	4	5	6	7	8	9
update/count	3	4	10	10	11	12	12	14	15	17

phase = طور.

```
CountSort (a, int(p)){
count   for (int i=0; i<n; i++){
            ++count[a[i]/p%10]);
        }
                                  Big O(n)
update  for (int i=1; i<k; i++){
            count[i] = count[i] + count[i-1];
        }
                                  Big O(k)
```

المشكلة في Range الـ كبيرة كتير

```
b عشان   for (int i=n-1; i>=0; i--){   Big O(n)
            b[--count[a[i]]] = a[i];   [a[i]/p%10]
        }
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
b	0	0	0	1	1	1	2	2	2	2	4	5	7	7	8	9	9

```
a عشان b   for (int i=0; i<n; i++){
               a[i] = b[i];
           }
```

* كل هذه مقدمة Radix ال

$O(n+k)$

Stack

O(1)	insert	← Push	عندي عمليتين مهمات هن ١)
O(1)	delete.	← PoP	٢)

stack over flow ⇒ لما تكون الـ stack ملانة.

stack under flow ⇒ لما تكون الـ stack فاضية. size (-1)

الاشي الي بأشر ← Top

دي عندي ~~both~~ ← هاد بجيب الي على الواجهة . Peek()

الحذف والاضافة من النهاية.

FILO :

* نفس فكرة الـ LinkedList بس تغير - مصطلحات

Application of stack :

دائما بجي سؤال عليهن .

Main uses stak are :

1- Evaluation of Arithmatic. expression

عشان * $E = A - ((B + C) * D)$

2- Execute recursive function.

→ depend on praiority :

في السلايدات موجود هاد الكلي :

1 ( ) الأولوية للأقواس

2 ^ القوة

3 * / الضرب & القسمة

4 + - الجمع & الطرح

infex → Postfix <x> <opp> <y> → <x> <y> <opp>

الأخير → الأوسط

* شيل الي فوق وحطه تحت والعملية
بالاول

infex → Prefix

الأول → الأوسط <x> <opp> <y> → <opp> <x> <y>

* شيل الي فوق وحطه تحت
والعملية بالاخر

* التركيز على الـ infex → Postfix

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

Pre → in

فابتدي من النهاية
بالاول و بدل

HG/FEDC*B-/A++- ← Reverse

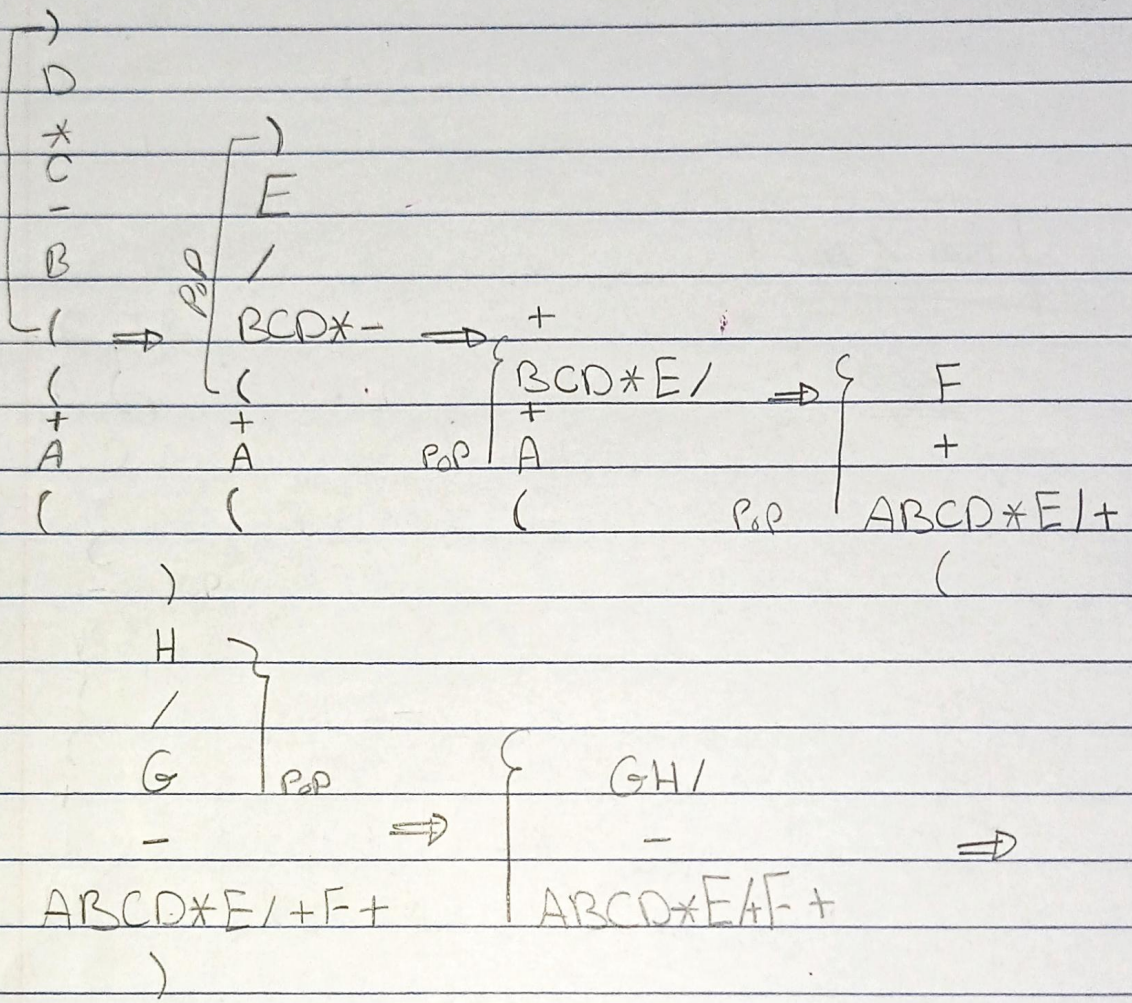
<y> <op> <x>

Stack	Expression
(	HG
(/	HGF
(/	HGFEDC (x = D, y = C)
(*	HGFED/C (x) (B) (y)
(- /	HGFE B* C/D (A)
(-	HGFE A/B* C/D
(+	HGF A/B*C/D-E
(+	HG A/B*C/D-E+F
(-	H A/B*C/D-E+F+G
	A/B * C/D - E + F + G - H → infix

Examples

1. $A+((B-C*D)/E)+F-G/H$ in → Post

<x><opp><y> → <x><y><opp>

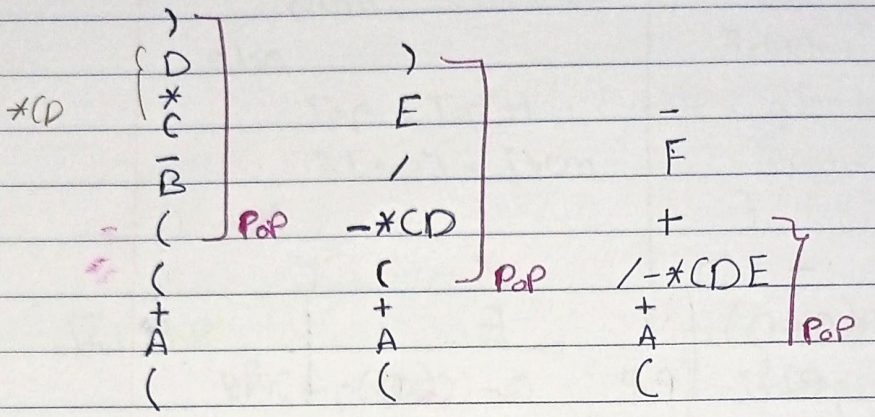


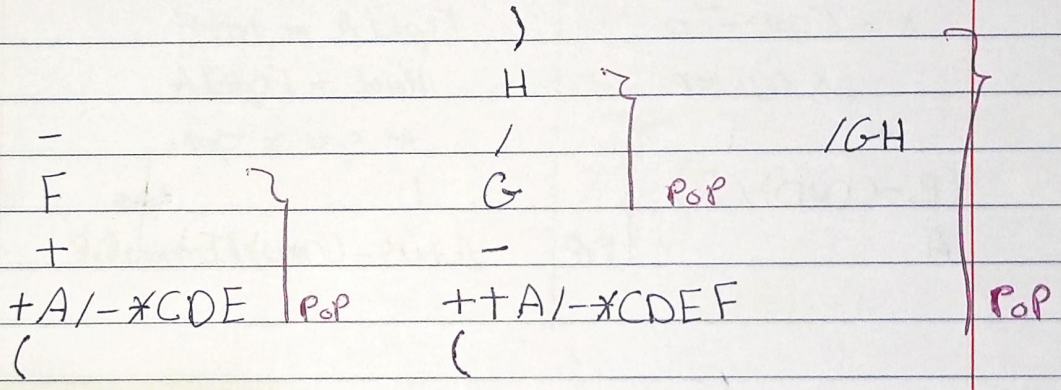
ABCD*E/+F+GH/-

Postfix

[2] A+((B-C*D)/E)+F-G/H in → Pre

<X><opp><Y> → <opp><X><Y>





| -++A/-*CDEF/GH | Prefix

[3] ABCD*-E/+F+GH/- Post → in

<x><y><opp> → <x><opp><y>

*		-		/	
D				E	
C	PoP	(C*D)			
B		B	PoP	B-(C*D)	PoP
A		A		A	

+		+	
(B-(C*D)/E)		F	
A	PoP	A+(B-(C*D)/E)	PoP

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

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

infix

Stack using Array :-

[1] Push.

```
if (top > size)
    print " over flow"
else
    Top = Top +1.
    A[top] = item.
end
```

```
() Push (int x) {
if (top >= (size -1)){
    Println(" over flow");
}else{
    a[++ top] = x;
    Println( x + " Pushed");
}
```

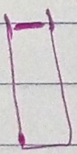
[2] PoP

```
if (top < 0)
    print " under flow"
else
    item = A[top]
    A[top] = Null
    top = top -1
end
return(item)
```

```
int PoP (int x) {
if (top < 0) {
    Println("under flow");
}else{
    a[--top] = x
    return x;
}
}
```

```
int Peek () {
    if (top < 0) {
        Print (" over under flow");
    }
    else {
        int x = a [top];
        return x;
    }
}
```

هاد الفنكشن
بيطبع آخر قيمة
موجودة في الStack
اللي هي الوجه oo



Stack using Linked List :-

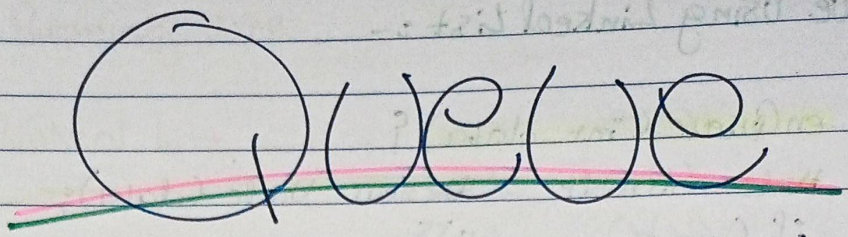
1.
```
boolean isEmpty() {
    if (header == null) {
        return true;
    } else {
        return false;
    }
}
```

2.
```
void push(int data) {
    Node temp = header;
    Node newnode = new Node(data);
    if (header == null) {
        header = newnode;
    } else {
        header = newnode;
        newnode.next = temp;
    }
}
```

3.
```
void pop() {
    if (header == null) {
        Print("Stack is empty");
    }
    else {
        header = header.next;
    }
```

```
[4] int Peek () {
        if (header == null) {
            Println("Stak empty!");
        }
        else {
            retern header.data;
        }
    }
```

بيعرض القيمة الي داخل
الهد :



مبدأ الـ Queue ← زي الطابور.

FiFo ← اللي بدخل أول يطلع أول

LiLo ← اللي بدخل آخر اشي يطلع آخر اشي.

* الإضافة بكل الأحوال ← من الآخر
enqueue = List.insertFirst()
dequeue = List.deleteLast() ← والحذف من الأول

* بتحتاج 2 pointer ← front : بتشير على الأول
rear : بتشير على الآخر.

Stacks ⇒ solve problem works on recursion.

Queue ⇒ solve problem have sequantial ~~prossing~~ processing.

Queue using Linked List :-

[1] void enQueue (int data) {

```
    Node newnode = new Node (data);
    if ( rear == null) {
        front = rear = newnode;
        rear.next = front;
    }
    else {
        rear.next = newnode;     ← يشاور على الـ next
        rear = newnode;          ← الرير الجديد يصير الـnewnode
        rear.next = front; }     ← الرير نكسته تاخذ الـnewnode ذيرجع الـ front.
    }
```

* إذا كانت الـ Queue فاضية حطي الـ rear و الـ front يساوي الـ newnode و الـ next للـ rear حطي = الـ front

[2] int deQueue () {

```
    Node temp;
    if (front == null) {     → rear = null;
        Print (" Empty! ");
        return 0;
    }
    else {
        temp = front;        أو ممكن Node temp = front;
        front = front.next;
        rear.next = front;
        temp.next = null;
    }
    return temp;
}
```

Running time

Linked List → - insert O(1)
- delete O(1)
- Find O(N)

Stack → Push O(1)
Pop O(1)

Queue → enQueue O(1)
deQueue O(1)

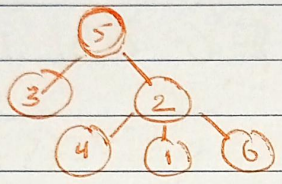
Stack	Queue
based in the Lifo or filo first in Last out or Last in first out.	based on the FIFO or LIFO, first in first out or last in Last out.
insertion and deleting in stacks take place only from one end of the list called the top	insertion ~~tack~~ takes place at the rear of the list and deletion from front.
insertion called Push Delet called Pop	insertion called enQueue delete called deQueue
we need one pointer which always point to the last element called top	we need ~~to~~ two pointer front → first element inserted in the list and is still present, rear → always point to the Last inserted element.
Solving problems work on recursion	solve problems having sequential processing.

Chapter 4

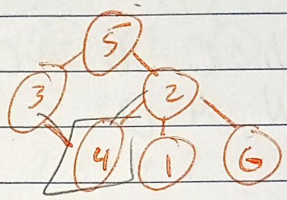
Tree

في شروط الـ Tree :

١. الـ root هو عبارة عن Node بس فيش إله أهل.
٢. كل Node إله أهل واحد.
٣. كل Node إلها صفر أو أكتر من الأولاد.
٤. عشان أوصل لـ Node كلهم يبقى إلها طريق واحد فقط.

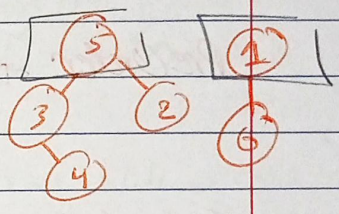


tree.



Not tree

في لـ (4) طريقين



Not tree

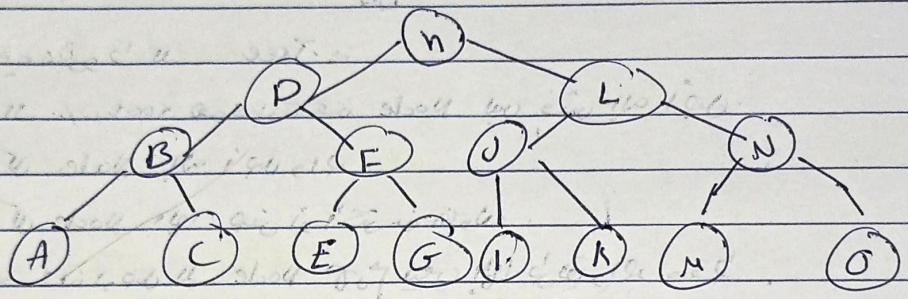
في 2 root

مصطلحات مهمة 👀

Ancestor
Proper ancestor
Decsendent
Proper descendent
Subtree
Leafe
internal
Siblings
Size
Level
Hight

Tree traversal

1. Pre Order ⇒ root → left → right
2. In Order ⇒ left → root → right
3. Post Order ⇒ left → right → root.



PreOrder ⇒ h D B A C F E G L J I K N M O

InOrder ⇒ A B C D E F G h I J K L M N O

Post Order ⇒ A C B E G F D I K J M O N L h

Exercise

* important

Construct the binary tree whose traversal are given below.

root

Pre: H D B A C F E G L J I K N M O — root lift right

Post: A C B E G F D I K J M O N L H — lift righ root.

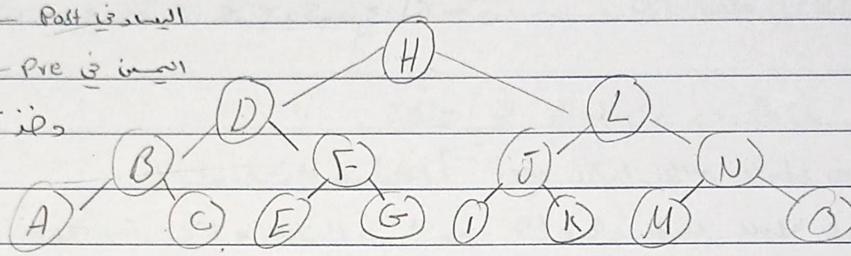
اليسار في Post ← يمين الـ tree

اليمين في Pre ← يسار الـ tree

وهنمشي كل حرف لحال.

مرة في الـ Post

ومرة في الـ Pre



مهم جدا *

Prefex: / − + 9 9 + * 2 3 4 2 — root → lift → right

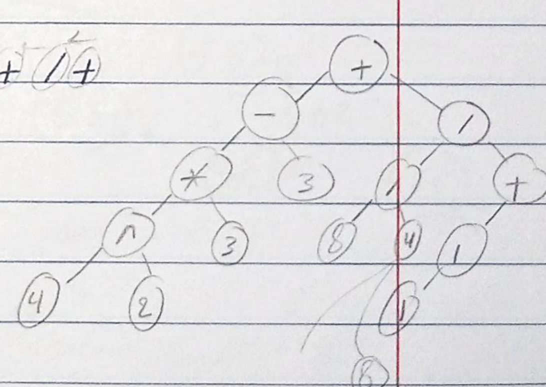
infex: 9 + 9 − 2 * 3 + 4 / 2 — lift → root → right

Postfix ⇒ ?

Pre ⇒ + − * ^ 4 2 3 3 / / 8 4 + 1 1

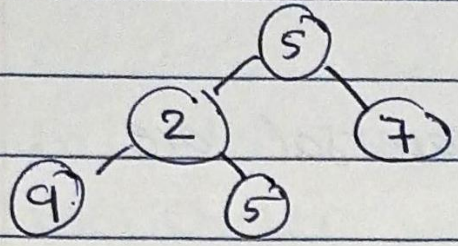
infix ⇒ ?

Post ⇒ 4 2 ^ 3 * 3 − 8 4 / 1 1 + / +



Binary tree

Tree شرطها كل الأقل Node 0
كل الأكثر Node 2



Full Binary tree ⇒

Complete Binary tree ⇒ لازم تكون معبية من اليسار لليمين *
وإذا كان في نقص لازم يكون في الأيمن.

Binary search tree

$O(\log N)$

* كل رقم الي على اليسار يكون أكبر من الـ root والي على اليمين أصغر.

* عشان أضيف Node جديدة بشوف إذا أكبر منه روح على اليمين وإذا كان أكبر من الـ Node اللي على يمين الـ root روح يمين وهكذا حتى ألاقي مكان فاضي.

* بالنسبة للحذف بدي أواجه 3 حالات.

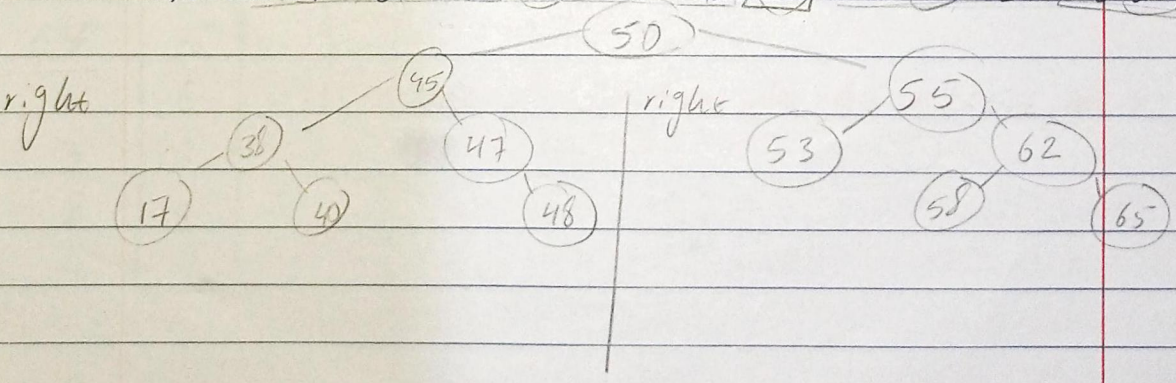
1. احذف Node مالها أولاد يعني leaf ← بحذفها وعادي.
2. احذف Node عندها ولد واحد ← بحذف الـ Node وبخلي ابنها يواجه أبوها.
3. احذف Node عندها ولدين ← بدور أواجه الـ Min أو الـ Max للـ Node اللي بدي احذفها وأحطه مكانه.

~~Min ← أكبر رقم في الـ left (الأخير)~~
~~Max ← أكبر رقم في الـ right (الأخير).~~

* يكون فيها left ← Min ⇒ أصغر رقم في الـ right
* يكون فيها right ← Max ⇒ أكبر رقم في الـ left

Ex

Post ⇒	17	40	38	48	47	45	53	58	65	62	55	50	root
in ⇒	17	38	40	45	47	48	50	53	55	~~57~~	58	62	65



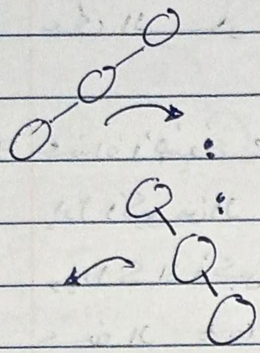
AVL tree

* هو Binary search tree

بس شرطها انها ← height balanced

(1-, 0, 1)

عشان اعدل ال tree ← balance



بعمل Rotation → : Right Rotation

→ : Left Rotation.

Single Rotation → Right
→ Left

Double Rotation → Left Right
→ Right Left.

* بتعمل insert او delete ركز ازاي ال Balanced

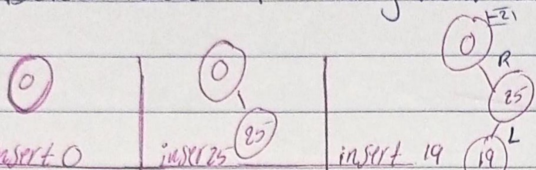
* هيجي مثال وكود

هيجي القسم التاني

Insert an integer number concecuating in an empty AVL tree. *

1) 0, 25, 19, 5, -2, 28, 13, -5, 2, 6, 14, 7

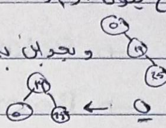
2) Delete node 19 using next suscceesors.

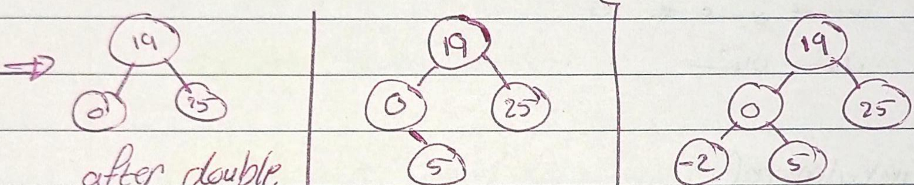


insert 0	insert 25	insert 19
Balanced	Balanced	imbalacede
no need to balance	no need to balance	need to balanced
		Problem : RL

Solve : RL rotation

* الحل رح يكون اوخذ الـ 19 واعملها راس ، وبجوانب بعمل لفة للـ 0

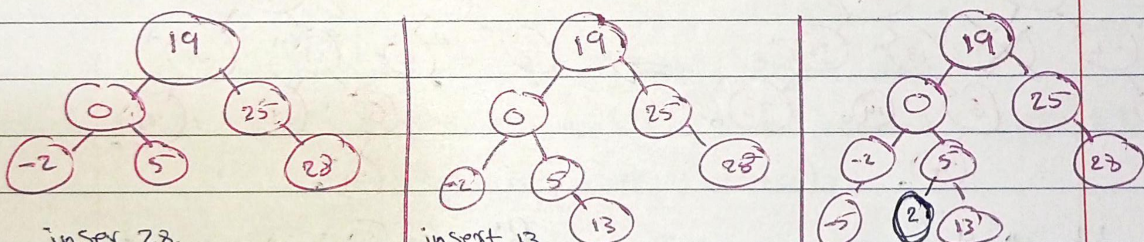




after double rotation R → L

insert 5 ✓

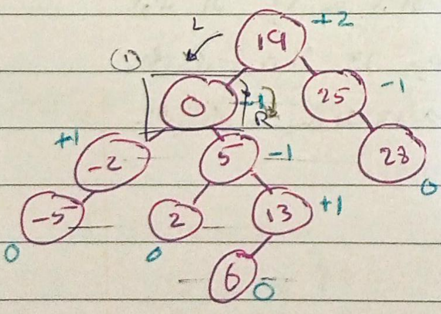
insert -2 ✓



inser 28 ✓

insert 13 ✓

insert -5, 2 ✓



insert 6

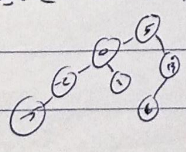
imbalanced

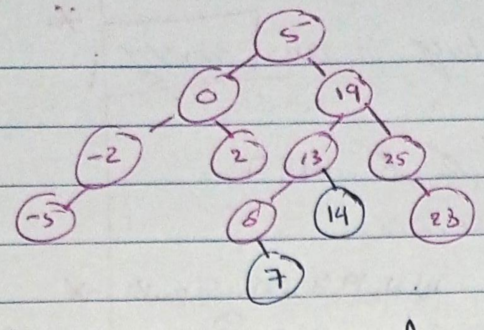
need to balanced

Problem ⇒ LR

Solve ⇒ LR Rotation

* الـ 5 بتصير مكان الـ 0 والـ 2 بتصير دك بين الـ 0





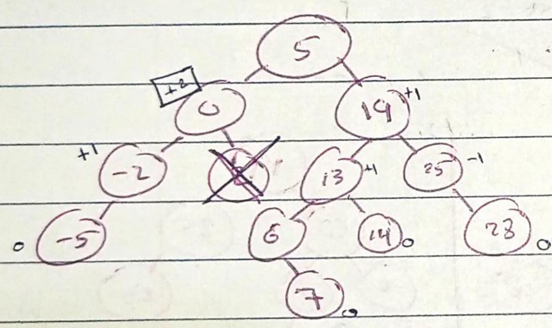
هسا بتصير الـ 5 مكان الـ 19
واللي كان على يمين الـ 5 بصير على
يسار الـ 19

insert 14 → Balanced

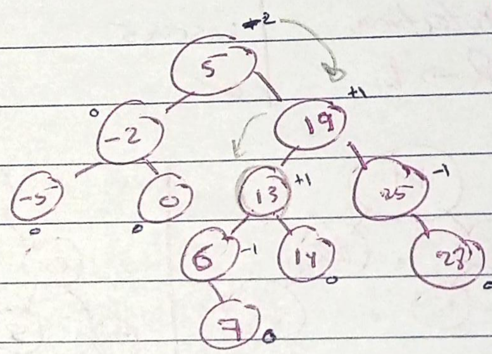
insert 7 → Balanced

* كيف أحدد المشكلة وين
اللي بحدد الـ Balanced ← + ⇐ شمال
− ⇐ يمين
0 ⇐ يمين يمين
شمال شمال.

Delete 2 ⇒ imbalanced

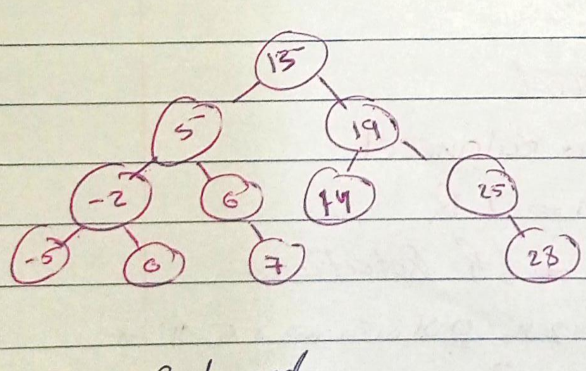


⇒



الحالة LL
ولا RR

يعني مشكلة في (5) ⇐ RL
الحل ← RL
بوخذ الـ 13 بحطها في الـ 5
واللي كان تحت الـ 13 بصير على يمين الـ 5
واللي كان يمين الـ 13 // تحت الـ 19



Balanced

Tree

```
Node tree{
    object data;
    Node left;
    Node right;

    Node (object data){
        this.data = data;
    }
```

```
class tree {
    Private Node root;
```

ممنوع في ال Main ← آكشف عن ال root
لأنها privet

```
    public void insert(Object data){
        root = insert (root , data)
    }
```

وبعمل 2 method عشان توصل لل root
دهي Privet ودهي Public عشان توصل للـ[?]
واستدعيها بال Main

```
    private ~~void~~ Node insert (Nood root , Object data){
        if (root == null){
            return newnode (data);
        }
```

→ Node newnode = new Node(data);

أكبر روح يمين

```
        else if (root.getData() < data){
            return root.setRight (insert (root.get~~data~~ , data);
```

Right

أصغر روح شمال.

```
        else if (root.get data() > data)
            return root.setlift (insert (root.get ~~data~~() , data);
```

Left.

* بعمل recursion كذا مرة
بدون بارامتر.

```
}
```

```
Public void InOrder(){
    InOrder(root);
}

Private void InOrder(Node root){
    if(root == null)
PreOrder  InOrder(root.getLeft());
    System.out.println(root.getData());
    InOrder(root.getRight());
}
```

PostOrder

```
public boolean delete(int data){
    if(root == null){
        return false;
    }
    else if(!
        return
    }
    else{
        return delete(root, data);
    }
```

B-Trees

DataBase ✱

[1] كل الـ leave في نفس الـ level

[2] الـ Node اللي بين الـ root والـ leafe إلها m أكبر عدد Childe $\frac{m}{2}$

[3] ؟ عدد الـ element اللي جوا أكبر شي m-1 وأقل شي m/2

[4] كل root إله Child لازم يكون كذا.

✱ أهم شيء الترتيب :-

Example

* Creat a B-tree of order 5

AGFBKDHMJES1RXCLNTUP

* لما يصير أكبر من الـ size بدل Split وعملية الانقسام OP اللي بالنص

* لما يدي أضيف K بدل Split ويطلع F وف ← وأكبر ولد اللي أصغر على اليسار.

وهكذا الترتيب.

ترتيب ~~AGFT~~ ① ~~ABFG~~

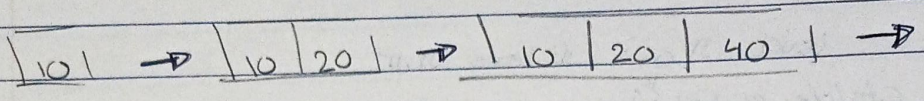
~~F~~

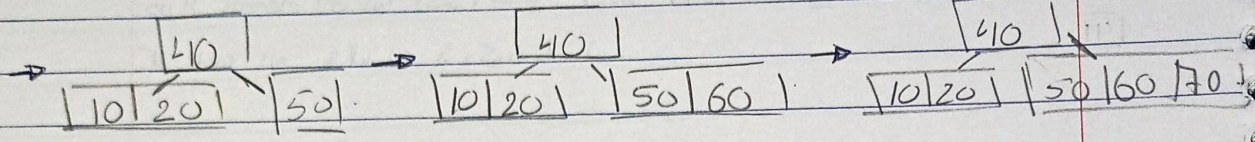
~~F~~ → ~~ABGK~~

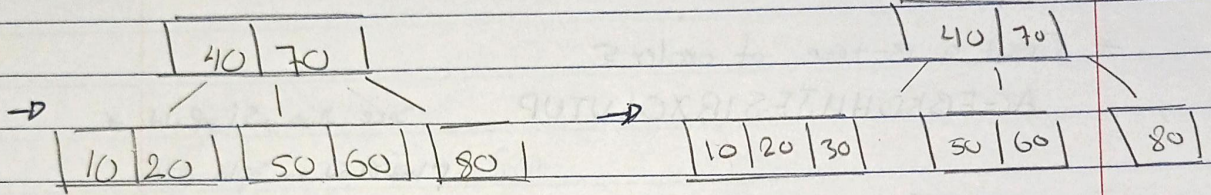
~~ABGK~~

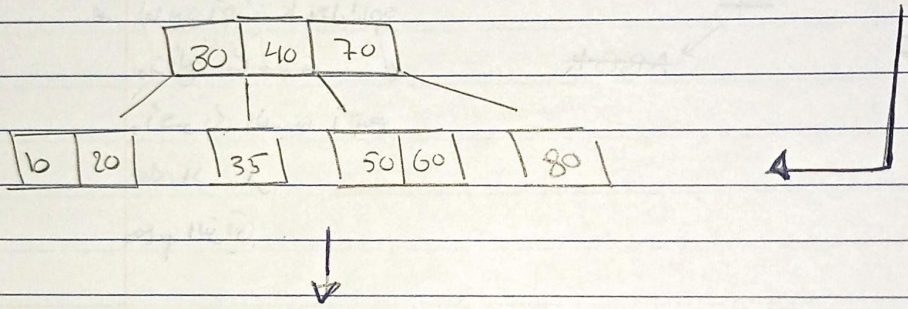
4

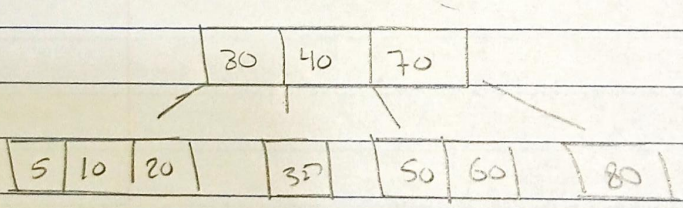
10 20 40 50 60 70 80 30 35 5 15 60

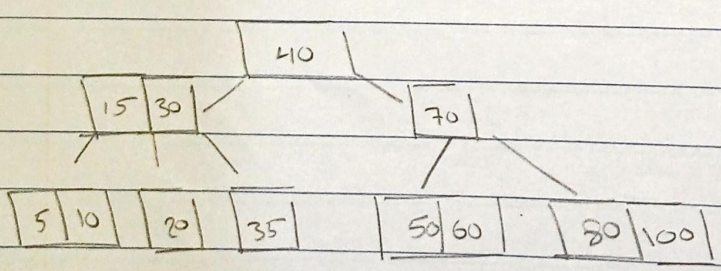












Chapter 5

Hashing

* هون كنا نشتغل بال array

والـ array ← Fixed size

- linear search ⇒ $O(N)$
- Binary search ⇒ $O(\log_2 N)$
- if we know index ⇒ $O(1)$

* عشان نوصل لـ $O(1)$ بدنا كل رقم في الـ index تاعه في الـ array
يعني 1 ← يكون في index 1 بس لو كان حجم الـ array مثلاً
10 يعني من 0 ← 9 وبدي اضيف الرقم 10 رح يصير عندي
مشكلة.

عشان احلها بنستخدم مفهوم الـ Hashing وبنستخدم هاي المعادلة
[X % table size]

insert 10 ⇒ 10 % 10 = 0

insert 13 ⇒ 13 % 10 = 3 مليانة !!

فشو بنعمل بروح على اللي بعده لحتى الاقي مكان فاضي

هاي المشكلة اسمها collision

0	10
1	
2	
3	3
4	4
5	5
6	6
7	13
8	8
9	

* هاي بعد وقت رح تلاقي يعني: out of array size

عشان أحل مشكلة ال collision

في 2 solve

close

1- Separate chaining : array of linked list بستخدم فيها

2- oppen addressing :

1- Linear probing ⇒ linear search
2- Quadratic probing ⇒ nonlinear search
3- Double hashing ⇒ use two hash function.

أشهر ال ٣

Separate chaining ⇒ Close.

Ex 23, 13, 21, 14, 7, 8, 15

table size 7

h(key) = key % table size

h(23) = 23 % 7 = 2
h(13) = 13 % 7 = 6
h(21) = 21 % 7 = 0
14 % 7 = 0
7 % 7 = 0
8 % 7 = 1
15 % 7 = 1

0 → 21 → 14 → 7
1 → 8 → 15
2 → 23
3
4
5
6 → 13

* time Complexity > O(1) but less than O(N).

* هاي ما بتشتكي وتعب لأنه بستخدم Linked list.

* هاي بس مشكلتها كيف بدي اتعامل مع ال Pointer و ال memory.

* ال Hashing مهم جداً في عملية ال insert
searshing

* بيستخدم في تحليل ال DNA و incretion أو decretion.

Example:

Use the hash function to load the following commodity items into a hash table of size 13 using separate chaining:

1- Onion ⇒ O = 111   n = 110   i = 105

hash (onion) = (111 + 110 + 105 + 111 + 110) % 13

= 1

2- Salt ⇒ S = 115   a = 97   l = 108   t = 116

has (salt) = (115 + 97 + 108 + 116) % 13

= 7

0 → Okra → Potato
1 → onion → carrot
2
3
4 → cabbage
5
6 → mushroom
7 → salt
8
9 → cucumber
10 → tomato → mellon → olive
11 → banana
12 → orange.

open addressing :-

* $h_i(key) = [h(key) + f(i)] \% n$ → لكل الحالات
  - ↳ linear
  - Quadratic
  - double

* $f(0) = 0$ بغض النظر عن النوع

$f(0) \% n$, $f(1) \% n$ ~~ $f(n-1) \% n$ → n = table size

value ⇒ between [0 and n-1]

* linear $f(i) = i$
* Quadratic $f(i) = i^2$
* double $f(i) = i * h_p(key)$

بالعادة بنختار ال table size يكون Prime (عدد أولي)

~~عشان حال مشكلة كده حل ال collision~~

لازم بعد حل عملية ال collision أطلع ال load factor

ولازم يكون بين (0.6 , 0.7)

λ = Number of element / table size.

وإذا كان أكبر من 0.7 ← undesirable ~~under~~

فبعمل rehash

Type of open addressing :-

[1] linear

Example

insert (18), (26) (35), (9), find(15) (48)
delet (35), (40) find (9), insert (64), (47)
find (35).

$f(i) = i$, table size = 13.

$h(key) = key \% \text{table size}$

$h_i(key) = (h(key) + f(i)) \% 13$

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

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

$h_0(35) = ((35 \% 13) + 0) \% 13 = 9 \% 13 = 9$

$h_0(9) = (9 \% 13 + 0) \% 13 = 9 \% 13 = 9$ ← collision

↳ $h_1(9) = (9 \% 13 + 1) \% 13 = 10 \% 13 = 10$

$h_0(15) = (15 \% 13 + 0) \% 13 = 2 \% 13 = 2$ (notfound)

$h_0(48) = ((48 \% 13) + 0) \% 13 = 9 \% 13 = 9$

لقى في رقم بس هيش 48 ف بروح بدور على اللي بعد

$h_1(48) = (48 \% 13 + 1) \% 13 = 10 \% 13 = 10$

ما لقى 48 وبرد بدل زي فوق

$h_2(48) = (48 \% 13) + 2 \% 13 = 11$ (Not found)

$h_0(35) = 9$ index of 9 has a value 35

D ← E وبتتحول من Change Status فبدل

$h_0(40) = (40 \% 13 + 0) \% 13 = 1 \% 13 = 1$ (Notfound)

Delete بدل ما بدل status E كانت ال ←

$h_0(9) = (9 \% 13 + 0) \% 13 = 9 \% 13 = 9$ (D) برجع

$h_1(9) = 9 \% 13 + 1 \% 13 = 10$ وبروح لـ h_1

D ← O من status بدل ال

$h_0(64) = (64 \% 13 + 0) \% 13 = 12$

$h_0(47) = (47 \% 13 + 0) \% 13 = 8$

$h_0(35) = 9$ Found

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

E Empty
D Delete
O Oquay

2 Quadratic

$f(i) = i^2$

$h_i(key) = (h(key) + i^2) \% \text{table size}$

* على نفس [هنا] المثال السابق

insert(13), (39), (26), (52)

$h_0(13) = (13 \% 13 + 0^2) \% 13 = 0$

$h_0(39) = (39 \% 13 + 0^2) \% 13 = 0$ collision

$h_1(39) = (39 \% 13 + 1^2) \% 13 = 1$

$h_0(26) = (26 \% 13 + 0^2) \% 13 = 0$ collision

$h_1(26) = 1$ collision

$h_2(26) = (26 \% 13 + 2^2) \% 13 = 4$

* كل ما زاد الـ collision بتزيد المشاكل
الحجم أكتر من 70% مشكلة فبنحل
بننشئ Array جديدة ونضاعفها X2
ونخزن العناصر في مواقع جديدة.

Primary Clustering ← دا متجمعة حوالين بعض بسبب كترة الـ collision.

عشان اضمن انه الـ table size هو Prime نحطه في هاي المعادلة

* $f(x) = x^2 + x + 41$

* في بالسلايدات الفرق بين الـ open & الـ close

Secondary clustering

* تجمعات بس مش بنفس ~~الـ~~ المكان (بيكون في تفرعات) وهو أهون من الـ Primary cluster

3| Double Hashing :-

$h_i(key) = [h(key) + f(i)] \% \text{ table size}$

* $f(i) = i * hp(key)$

hp is prime →

$hp = 1 + key \% (\text{table size} - 1)$

$= q - (key \% q)$

$q * (key \% q)$

q is prime #

Example :

insert (~~26~~, ~~70~~, ~~18~~, 9, 47, 35, 96, 64)

* table size 13

* $hp = 1 + key \% 12$

$hi(key) = (h(key) + i * hp(key)) \% \text{ table size}$

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

$h_0(70) = (70 \% 13) + 0) \% 13 = 5$

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

$h_1(18) = (18 \% 13 + (1 * (1 + 18 \% 12)) \% 13 =$

$(5 + (1 * 7)) \% 13 = 12$

$h_0(96) = (96 \% 13 + 0) \% 13 = 5$ collision

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

$= (5 + 1) \% 13 = 6$

	Double
0	26
1	
2	
3	
4	
5	70
6	96
7	
8	
9	
10	
11	
12	18

Exercise :-

* if the hash table after using linear probing is as shown below?

1- insert ~~[illegible]~~ 29.

2- is there a problem is hashing table? if there is state it and give a solution.

0	
1	15
2	37
3	25
4	29
5	
6	13

1- $h_0(29) = (29\%7 + 0)\%7$

$= 1\%7 = 1$ collision

$h_1(29) = (29\%7 + 1)\%7$

$= 2\%7 = 2$ collision

$h_2(29) = (29\%7 + 2)\%7$

$= 3\%7 = 3$ collision

$h_3(29) = (29\%7 + 3)\%7$

$= 4\%7 = 4$ inserted.

* هنا مشكلة (كثرة ال colision) وشو بحل ا- بضاعف ال array 2x
بصير حجم ال array = 14 فـ اقرب prime# هو ال 17
وبعمل insert جديدة باستخدام ال Double hash

table size = 17

$hp = 1 + key\%16$

$h_0(15) = (15\%17 + 0)\%17 = 15$

$h_0(37) = (37\%17 + 0)\%17 = 3$

$h_0(25) = (25\%17 + 0)\%17 = 8$

$h_0(29) = (29\%17 + 0)\%17 = 12$

$h_0(13) = (13\%17 + 0)\%17 = 13$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
			37					25				29	13		15	

* حل مشكلة الـ String

* $hash = \sum_{i=0}^{keysize-1} key[keysize - i - 1] \cdot 32^i$

Array

بوفر الـ ASCII Code للحرف

وبلاش كل مرة الـ Method

```
Public Static int hash(String key, int table_size){
    int hashVal = 0;
    for (int i=0; i< key.length(); i++)
        hashVal = hashVal <<5 * key.charAt(i);
        hashVal %= table_size;

    if(hashVal<0)
        hashVal += table_size;

    return hashVal;
}
```

بدل الضرب ف 32 كل مرة shiftting

عشان ما يحصلش over flow.
ونرجع لو الصيغة الـ positive

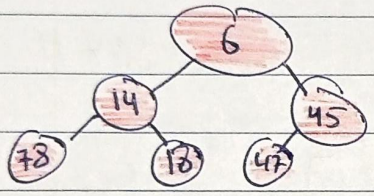
HEAP

Chapter 6

Binary Heap

1. Complete binary Tree ⟵ شجرة من فوق لتحت ومن اليسار لليمين

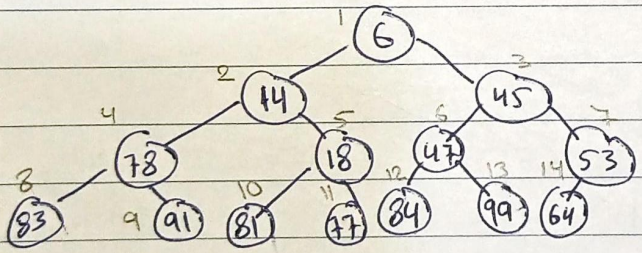
2. Min-heap ordered. ⟵ الأولاد لازم يكونوا أكبر أو يساوي الأب



⟵ مهم اللي على اليمين أكبر أو اللي على اليسار وكمان مش مهم التكرار

Minimum element is in the root

Heap with N elements has height = $\log_2 N$



⟵ Tree هيك بدي الـ

بس بترجمها

Parent (i) = i/2

left (i) = 2i

Right (i) = 2i + 1

Insert

insert into next available solt
Bubble up until it's heap ordered

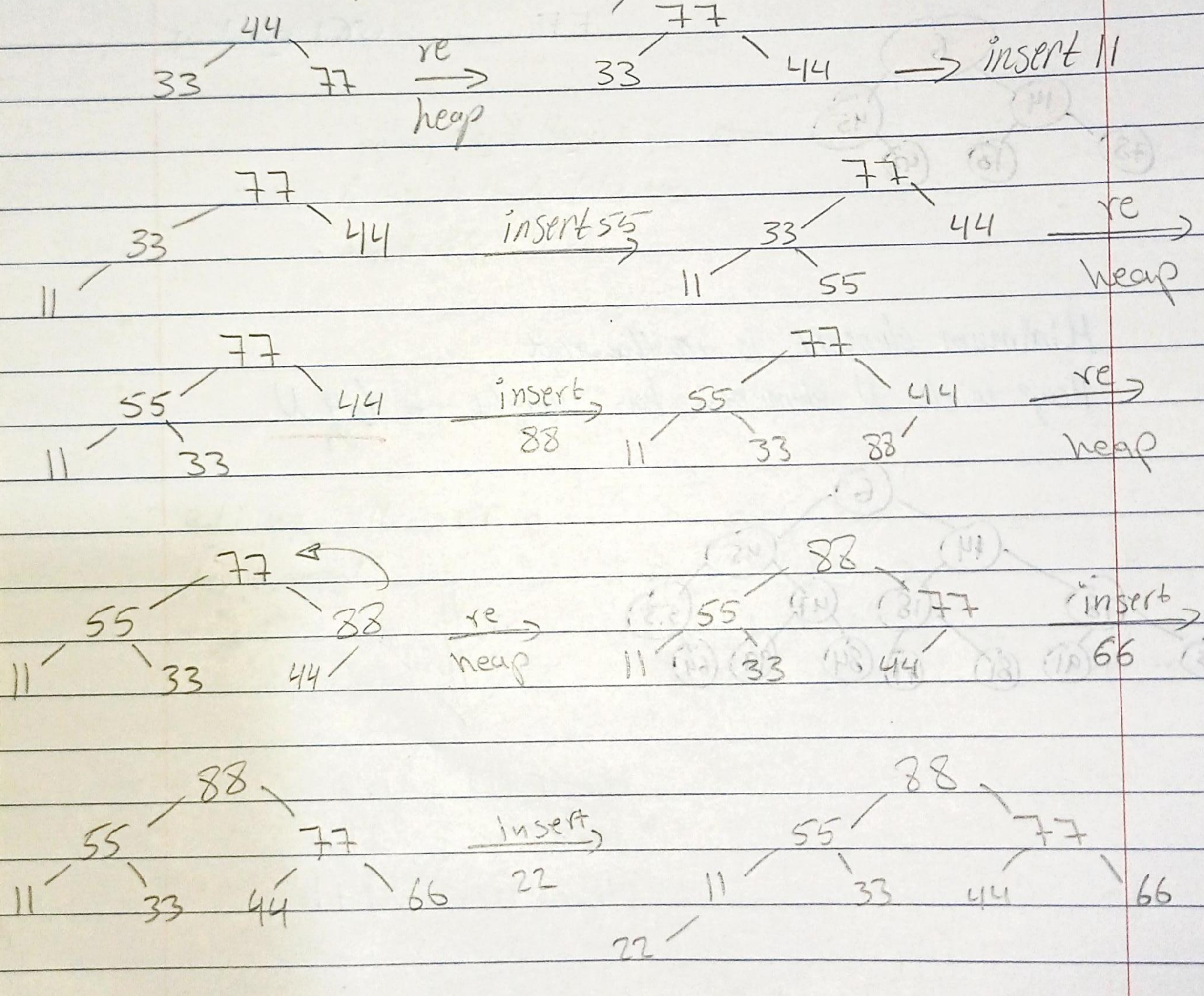
وبنطلع فوق بنوقف
ونبقى بنحط هو اذا هو heap ordered
(الاكبر اعلى من الاب)

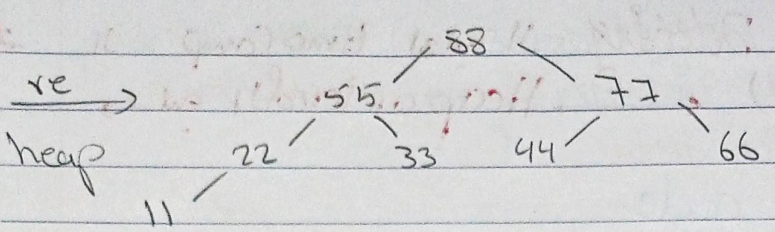
Swape ← انا و ابدل

Time Complexty O(log N)

Example

Bulid Maximium heap tree?
44, 33, 77, 11, 55, 88, 66, 22





Done

Delete minimum element from heap.

exchange root with rightmost leaf.

* أخذ ابن ذلك اليمين

Bubble root down until it's heap ordered.

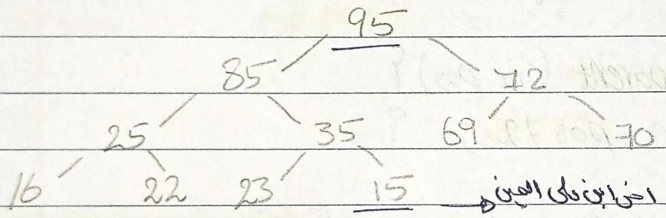
* يتحقق اذا كان الابن اصغر من الآباء.

* ويتوقف من حجم الواحد !

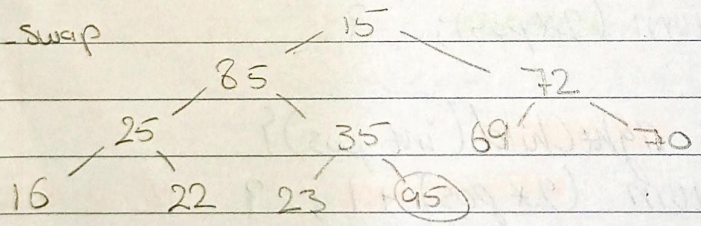
Example

Delete root 95

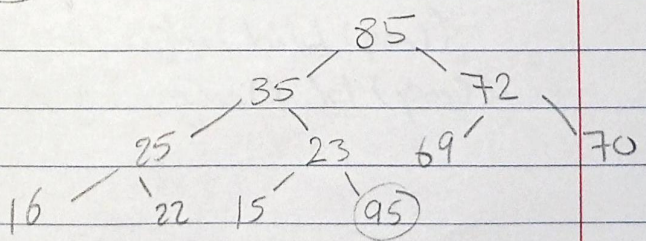
maximum Heap



1- Swap



2- Heap ordered



Done

O(log N) Deleting الـ time Comp الـ
O(n log N) ← Heap ordered الـ

Min Heap code

```
public class MinHeap {
    privet int [] Heap;
    privet int size;
    privet int maxsize;
    privet static final int FRONT = 1;

Constructor → public MinHeap (int maxsize) {
        this.maxSize = maxsize;
        this.size = 0;
        Heap = new int [this.maxsize +1];
        Heap [0] = Integer.MIN_VALUE; }

    privet int parent (int pos) {
        return pos/2; }

    privet int leftChild (int pos) {
        return (2*pos); }

    privet int rightChild (int pos) {
        return (2*pos) +1; }
```

```
public void insert (int element){
    if (size >= maxsize){
        return; }  → insert اطلع فش ع بال نكمل
    Heap [++ size] = element;
    int current = size;  → موشر للـ size

    while ( Heap [current] < Heap [Parent (current)]) {   ← يقارن الاب مع الابن
        Swap (current, parent (current));   ← ميثود
        Current = parent (current); }
    }

public void minHeap(){
    For (int pos = (size/2); pos>=1; pos --){
        minHeapify (pos); }

~~public~~ privet void minHeapify(int pos){
    if (! isleaf (pos)){   ← فش اله اولاد
        if ( Heap [pos] > Heap [leftChild (pos)] ||   or
            Heap [pos] > Heap [rightChild (pos)]) {
        if (Heap [leftChild (pos)] < Heap [rightChild (pos)])
        {
            swap (pos, leftChild (pos));          اللي على اليسار اصغر
            minHeapify (leftChild (pos));         اعمل swap
        }
        else {   ← اللي على اليمين اصغر
            swap (pos, rightChild (pos));
            minHeapify (rightChild (pos)); }
        }
    }   → اله اولاد فش داعي اعمل اشي
}
```

```
private boolean isLeaf (int pos){
    if (pos >= (size/2) && pos <= size){
        return true;
    }
    return false ; }

private void swap (int fpos, int spos){
    int temp;
    temp = Heap [fpos];
    Heap[fpos] = Heap [spos];
    Heap [spos] = temp;
}

public int remove () {
    int popped = Heap [ FRONT ];
    Heap[FRONT] = Heap[size--];
    minHeapify (FRONT);
    return popped;
}
```

بدل اول element بـ element اخر

Heap Sort

1. Bulid a max heap from the input data.
2. At this point, the largest item is sorted at the root of the heap. Replace it with the last items of the heap followed by reducing the size of heap by 1.
3. Repeat above steps while size of heap is greater than 1.

اكبر عنصر

Time Complexity :

Heapify ⇒ $O(\log N)$

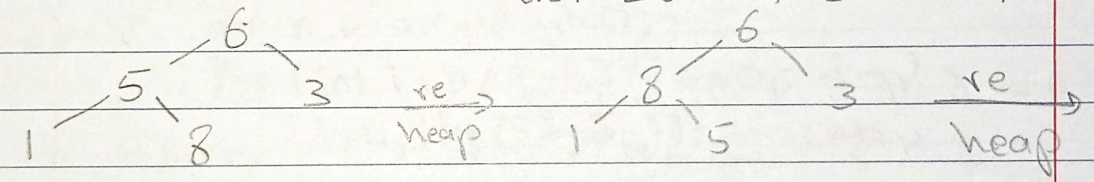
Bulid Heap ⇒ $O(N)$

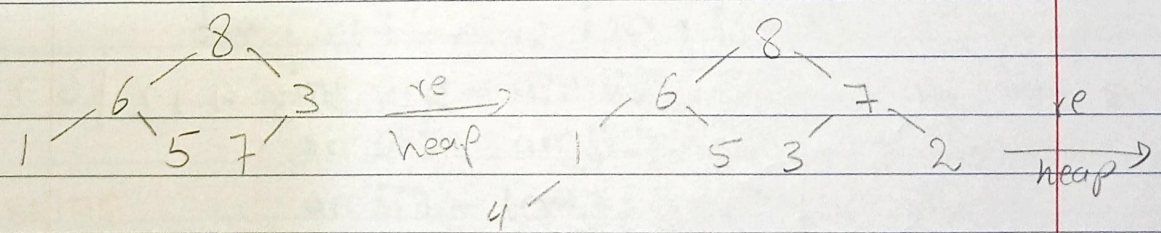
Heap sort ⇒ $O(N \log N)$.

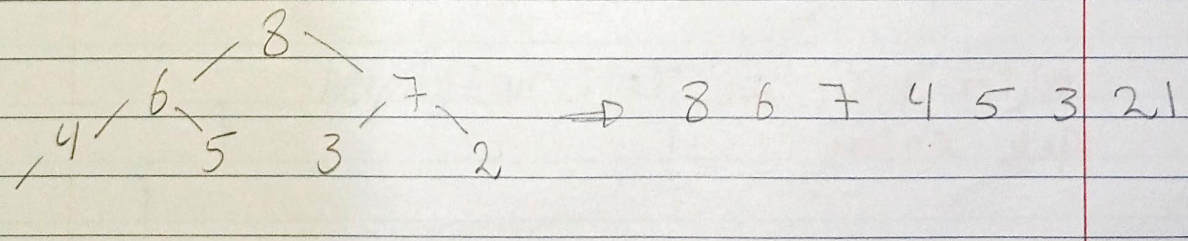
Example

A=[6 5 3 1 8 7 2 4] full Array. size = 8

1- (Heapify) بدل كود ال Array بنوظف ال size ، ال
وبقترح الداتا اللي جوا الاراي وبعينها







Front

A = [8 6 7 4 5 3 2 1]

8 بدل remove بس

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

size = 7

وبعدين بدل Bulid لل Heap

re → re → re →

A = [7 5 6 1 4 3 2 | 8 |]

remove 7

[2 5 6 1 4 3 | 7 8 |]

size = 6

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

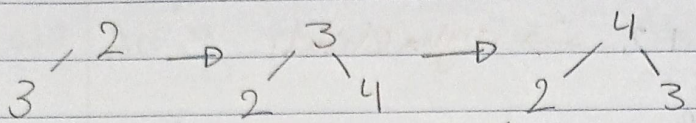
3 4 5 1 2 | 6 7 8 |

remove 6

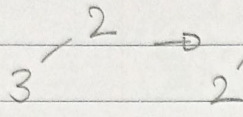
size = 5

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

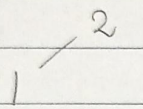
A= 5 3 4 1 2 | 6 7 8 |
2 3 4 1 | 5 6 7 8 | removes 5



A= 4 2 3 1 | 5 6 7 8 |
2 3 1 | 4 5 6 7 8 | remove 4



A= 3 2 1 | 4 5 6 7 8 |
2 1 | 3 4 5 6 7 8 | remove 3



A= 2 1 | 3 4 5 6 7 8 | remove 2

A= 1 | 2 3 4 5 6 7 8 | remove 1

A= 1 2 3 4 5 6 7 8 is sorted. ^^

```
public void sort (int arr[]){
    int n = arr.length();
    for (int i = n/2-1 ; i>=0; i--)
        heapify (arr, n, i);        → Data

    for (int i = n-1; i>0 ; i--){
        int tmp = arr[0];
        arr[0] = arr[i];            } swap
        arr[i] = temp;

        heapify (arr, i, 0);        → لاقراي مع ال index
    }
}
```

```
public void heapify(int arr[], int n, int i){
    int largest = i        → root
    int l = 2i+1           → left Child
    int r = 2i+2           → right Child.

    if(l < n && arr[l] > arr[largest])
        largest = l;

    if(r < n && arr[r] > arr[largest])
        largest = r;

    if (largest != i) {
        int swap = arr[i];
        arr[i] = arr[largest];
        arr[largest] = swap;

        heapify(arr, n, largest);   → Recursive
    }
}
```

اليمين أكبر

swap أعمل

Radix Sort

هدفين شغالين في ال Sorting بشكل عام

1- No extra space ← اي ما احتاج Array جديدة

2- Time Complexity.

Radix Sort → $O(N \log N)$ with space

Selection and Bubble Sort → $O(N^2)$ No space

Heap Sort → $O(N \log N)$ No space 👍

Quick Sort

هذا النوع مبني على ال Pivot العمود

الداتا بتتكون حول عمود معين

من عمود الاربعة ⟸

bad 1- first element pivot

bad 2- last // //

bad 3- random // //

good 4 median // //

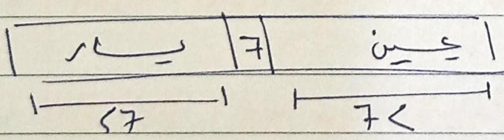
Example:

A = [7 6 10 5 9 2 1 15 7] (l at 7, h at 7)

Pivot first element ⟹ Pivot = A[l];

صياغة Put pivot in proper (plase) so that all data less than Pivot goes on left, and all data great than pivot goes to Right.

بنحط ال pivot مكان بحيث يصير كل الداتا الا اصغر منه على اليسار والاكبر على اليمين.



divide & conquer

* نقسم ونجمع

7	6	10	5	9	2	1	15	7

Pivot = 7

swap

Start, s, e, end

* Move start to right ⇒ till find a number greater than or equal to the pivot
* Move end to left ⇒ till find a number less than the pivot.

* وبعدين نعمل swap

7	6	7	5	9	2	1	15	10

s, s, swap, e, e

7	6	7	5	1	2	9	15	10

s, e, e, s

No swap because
s > e (crose)

swap between element end with pivot.

Pivot

2	6	7	5	1	7	9	15	10

Pivot = 2

less, great, proper place

* هيا بقا كل جزء لحاله.
اللي على اليمين لحاله واليسار لحاله

Algorithim

```
Partion (a, l, h){
    int start, end, pivot;
    pivot = a[l];
    while (start < end){
        while (a[start] <= pivot){      لما نلاقي رقم أكبر من ال Pivot وقف
            start ++; }
        while (a[end] > pivon){
            end --; }                   اذا لقيت رقم أصغر وقف
        if (start < end){
        swap(a[start], a[end]);
    }
    swap(a[l], a[end]);   ← اذا صار تقاطع * بدل بين ال end مع ال index تاع البداية
    }
    return end;   عشان أعرف وين مكان ال Pivot غالبًا
}
```

```
Qsort (a, l, h){
    int loc = Partion (a, l, h);
    Qsort (a, l, loc-1);     أخذ الأراي (اليسار)
    Qsort (a, loc+1, h);     (اليمين)
}
```

* Best Case ⇒ $O(N \log N)$

* Worest Case ⇒ $O(N^2)$

حطوات ال Pivot ←

الاول Pivot = a[L];

الاخير Pivot = a[h];

Pivot = randome (a);
عشوائي

Pivot = Median3 (a, L, h);

```
int Median3 (int a[], int L, int h){
    int center = (L+h)/2;

    if( a[L] > A[center])
        swap(a, L, center);

    if( a[L] > a[h])
        swap(a, L, h);

    if(a[center] > a[h])
        swap(a, center, h);

    Swap(a, center, h-1);
    return a[h-1];
}
```

بدل الرقم اللي بالنص ← مع اللي قبل الاخير (Swap(a, center, h-1))

رجعنا ال Pivot (return a[h-1])

Insertion Sort

بتشتغل شغل لعب الشدة .

Exercise

$A = \overset{0}{5} \mid \overset{1}{4} \quad \overset{2}{10} \quad \overset{3}{1} \quad \overset{4}{6} \quad \overset{5}{2}$

sort — unsort

الفكرة ← بقسم الارراي ل
1- sorted list
2- unsorted list

temp [4]

index 1 بياخد
يعني بيبقا 4 جواه

وبقارن الي جوا ال temp مع ← الي موجود في sorted list.

* هل الي بال sort list أكبر من الي بال temp ؟
إذا أكبر بعمل shift to the right .
وبقارن باقي العناصر الي ضلت مع ال temp
اذا لقيت عنصر أصغر من ال temp
بوقف .

$A = 4\cancel{?} \quad 5 \mid 10 \quad 1 \quad 6 \quad 2$

sort — unsort

temp [10]

وبرجع بقارن 10 مع 5 & 4
مش أكبر من 10 بدخل جواها

$A = 4 \quad 5 \quad 10 \mid \cancel{1} \quad 6 \quad 2$

sort — unsort

temp [1]

4	5	?	10	6	2	هل 10 > 1 ← shift
4	?	5	10	6	2	هل 5 > 1 ← shift
?	4	5	10	6	2	هل 4 > 1 ← shift
1	4	5	10 |	~~6~~	2	وصلت index 0 ومش

sort — unsort

جواه اشي بحط ال 1

1	4	5	?	10	2
1	4	5	6	10 |	2

sort — unsort

temp [6]

هل 10 > 6 ← shift
هل 5 < 6 ← ☒ بحط ال 6

A = 1 4 5 6 10 | 2

sort | unsort

temp = 2

1	4	5	6	?	10	shift ← 2<10
1	4	5	?	6	10	shift ← 2<6
1	4	?	5	6	10	shift ← 2<5
1	?	4	5	6	10	shift ← 2<4
1	2	4	5	6	10	left ← 2<1

A = 1 2 4 5 6 10

sorted

Algorithim

n ← 1 من تبدأ ← for

1 ← n من تبدأ ←

decrement

```
for (i=1 ; i<n ; i++) {
    temp = A[i];
    j = i--;
    while (j>=0 && A[j] > temp) {
shifting بدل  A[j+1] = A[j];
        j--; }
    A[j+1] = temp;
}
```

Time Complexity ⇒ worst $O(N^2)$

Best $O(N)$

Shell sort

خاصة على مبدأ ال movement

* تقليل عدد الحركات اللي بعمل فيها shifting.

away جاي ال

gap = n/2

A= 23 29 15 19 31 7 9 5 2

i No swap j

23 29 15 19 31 7 9 5 2

i swap j

23 7 15 19 13 29 9 5 2

i swap j

23 7 9 19 31 29 15 5 2

i swap j

23 7 9 5 31 29 15 19 2

i swap j

A= 23 7 9 5 2 29 15 19 31

i j

gap = 9/2 = 4

بقارن إذا إلي أنا فيه من i

بعمل swap

وبعدين i++ و j++

بطلع Backword و forword

وبنا بطلع Backword لما يحقق هاد الشرط

i - gap >= 0

i - gap >= 0

4 - 4 = 0

∴ بطلع Backword على ال index 0

هيا بقارن ال i ع Backword.

phase 0 ⇒ صار جاهزة.

هيا ال phase اللي بعدها بتبقى ال gap القديم تقسم 2

New gap = Old gap / 2

* Newgap = 4/2 = 2

A= 2 7 9 5 23 29 15 19 31

i j No swap

2 7 9 5 23 29 15 19 31

i j swap and No need to look Backword

2 5 9 7 23 29 15 19 31

i j No swap

2-2 = 0

need to look Backword

~~2 5 9 7~~

2 5 9 7 23 29 15 19 31

look ← i j No need swap

2 5 9 7 23 29 15 19 31

look ← i j swap i and j

2 5 9 7 15 29 23 19 31

look ← i j swap i and j

2 5 9 7 15 19 23 29 31

look ← i j No swap

A= 2 5 9 7 15 19 23 29 31

phase 2 ✓

new gap = 2/2 = 1

A= 2 5 9 7 15 19 23 29 31

i j No swap No look Back word

2 5 9 7 15 19 23 29 31

look ← i j No swap and need to look Back word

2 5 9 7 15 19 23 29 31

look ← i j swap

2 5 7 9 15 19 23 29 31

← i j No swap

2 5 7 9 15 19 23 29 31

← i j No swap

2 5 7 9 15 19 23 29 31

← i j No swap

2 5 7 9 15 19 23 29 31

← i j No swap

2 5 7 9 15 19 23 29 31

← i j No swap

A= 2 5 7 9 15 19 23 29 31

Sorted ☺

Code

```
void shellSort ( A[]) {
    int i, j, increment;
    int temp;
    int N = A.length;
    for ( gap = N/2 ; gap>0 ; gap/=2)
        for ( j=gap ; j<N ; j++ ){
            temp;
            for( i=j-gap ; i>=0 ; i-=gap)
                if ( A[i] < A[i+gap]) {
                    temp = A[i+gap]);
                    A[i+gap]= A[i];
                    A[i] = temp;
                }
                else
                    break;
        }
}
```

هان إذا كانت أ أكل من ن نفذ السؤال.

Swap ، بنفس الوقت بشوف إذا كان في Backword بتطلع.

Time Complexity => $O(N^2)$.

Question

Suppose we have 5 GB of data using olnly 1 GB of Ram, what is the Best sorting algorithim could you use ?

* عندي Data حجمها كبير و الرام عنا قليلة
وبدي اورتب هاي الداتا شو اعمل ؟

⇓ من هان طلعت عندي فكرة

External Sort.

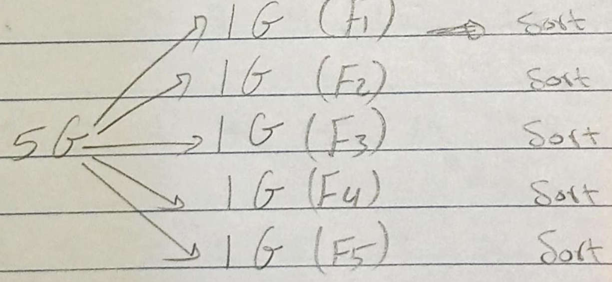
بقسم وبعمل Sort
اي نوع سورت بدي ياه
بس اذا كان عندي اقل من 10 بعمل insertion Sort
واذا كانت الداتا كثير كبيرة استعمل Quick sort.

Example
5 GB of data (file) sorted in H.D.

بعمل Divid لل file

1G	1G	1G	1G	1G

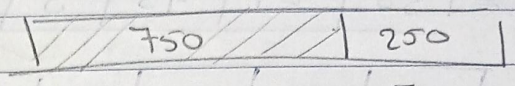
load in to Memory ← بوخذ كل 1G وبعمله
sort use quick sort



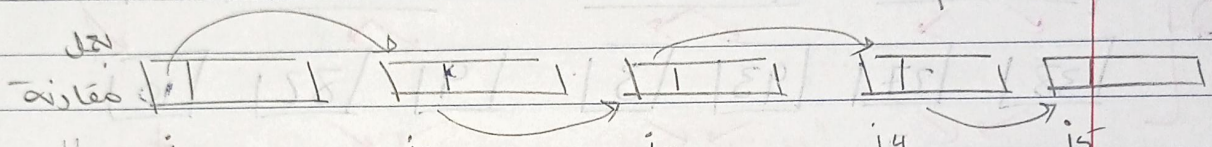
سميناه external
لانه انا استعنت بمساعدات
خارجية اللي هي H.D
و صرت اعمل loaded
على ال Memmory.

طيب ربنا كل واحد كان بس أنا بدي كل الداتا تكون مرتبة !؟

* بوخذ من كل 1G ← 150 MG
5 X 150 = 750 ميغابايت
وهدول الـ 750 بحطهم loaded كل من الـ 1G من الـ RAM

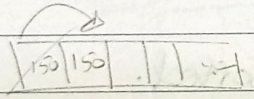


* وبعدين بدل Merge sort ← K-way Merge sort
بدي اقسم لعدة شغلات



Compare ⇒ وبترتيبهوا الـ Free 250
ولما تخلص الـ 250 بطلع out 1
وبرد بعبي الـ 250 وهكذا.

وفي مرات اشي بكون قبل مثلاً i4 قبل i3
فشو بدل ← بدل load another 150 MG
وبضل بطلع عندي out حجمها 250 M



$\frac{5 \times 1024}{250}$ = عدد الـ out اللي هي الـ Files

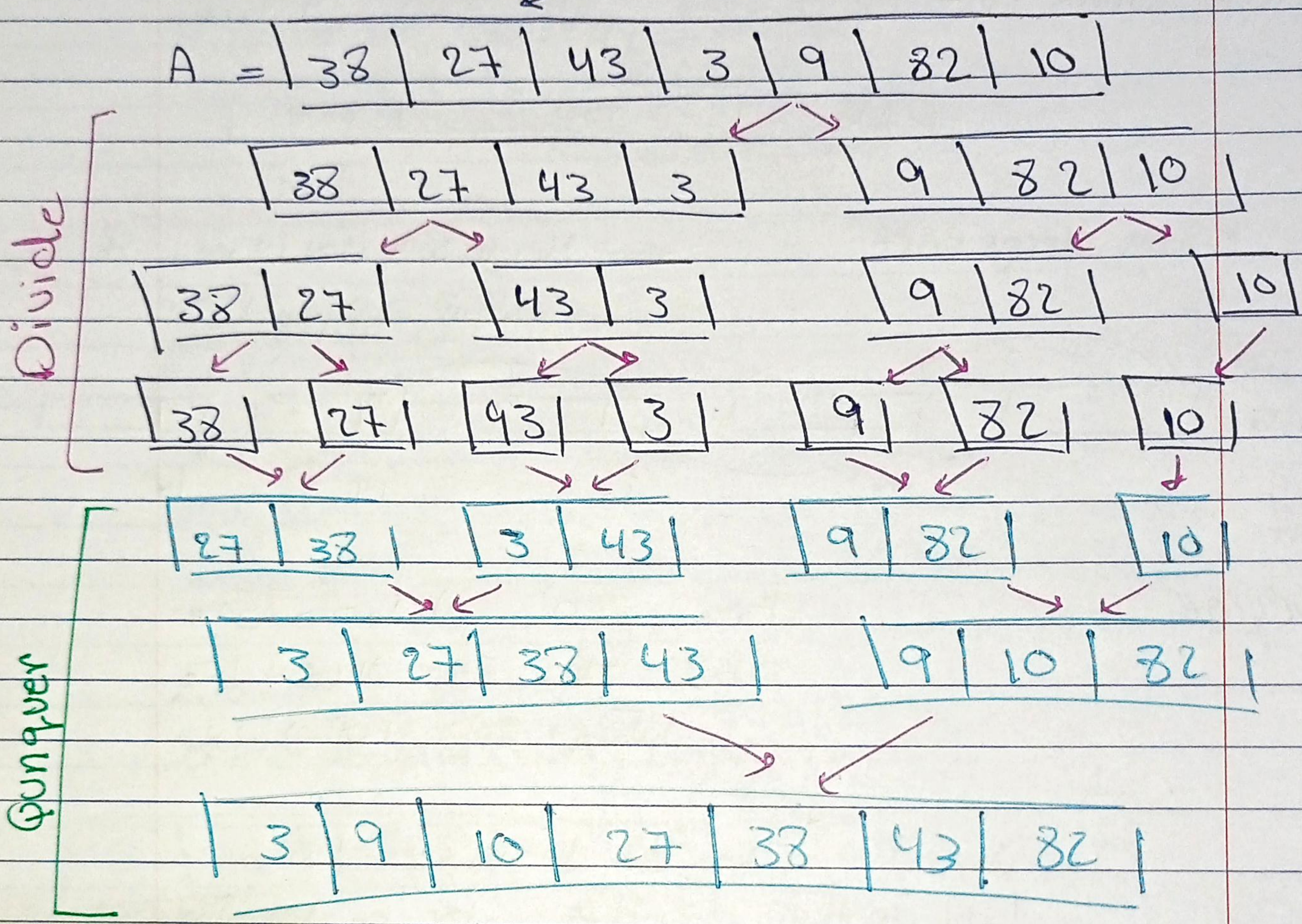
هاي الفايلات بطلع sorted

out 1
out 2
out 3
!

Merge file it's 5G sorted.

Merge Sort

فكرته انه يقسم الاراي ويجمعها مرتبة
بعدين عشان هاي ال Array



Sorted.

Algorithim

```
void mergesort (int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        Recursion {
            mergesort (low, mid);
            mergesort (mid + 1, high);
            merge ~~sort~~ (low, mid, high);
        }
    }
}
```

Time Complixity ⇒ O(n log n).

$$T(n) = \begin{cases} c & , n = 1 \\ 2T(n/2) + cn & , n > 1 \end{cases}$$

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

$T(n/2) = 2T(n/4) + cn/2$

$T(n) = 2(2T(n/4) + cn/2) + cn$

$= 2^2 T(n/4) + 2cn$

$\vdots$

$= 2^k T(n/2^k) + kcn$

$T(n) = k\,T(1) + cn \log n$

$\boxed{T(n) = n \log n}$

Let

$2^k = n$

$\log_2 2^k = \log n$

$\boxed{k = \log n}$