6/2/2020

Data structure.

* Performance metrics :
① Runing time
② Memory usage

* Scalable : when the size of input increases, the performance decreases with acceptable ratio (is not an option)
يعني بطاقة البرنامج بين كيف حجم الinput لما يكبر شو بصير من ناحية الـ time والـ Memory
* عشان أضمن إنك الـ System تاعي Scalable :

* To prove Scalability, we should derive a Formula for complexity :
① Experimental : Try different sizes of input and observe the performance → equation. (result, output يعني أجرب أزيد حالي input وأشوف شو بصير بالـ)
② Mathemetical notation : analyse the code and estimate each instruction how long it takes. (أخير أشوف line line وأحلله واشوف قديش بناخد ومنتب استخدامه)
→ tidy process → Approximate estimate
  - → most time consuming operations
  - → most frequent operations
  - → ignore others

* % more 8ints.text → Show the contents of the file 8ints.txt.
javac. → Compile the code
java Three sum 8ints.txt → execution
Space بينهم ← → Parameter at execution time ممكن أحط أي ملف تاني جنب
* Main accepts parameters as any other method. these parameters are passed to the main at execution time. /arg[0] = "8ints.txt" File اسم الـ
* Note : ① noise → اشي بخرب الداتا ② bugs : أخطاء

Slide 7 :

```
for (i=0 → N-1)
    for (j=i+1 → N-1)
        for (K=j+1 → N-1)
            if (array[i] + array[j] + array[K] = 0)
                count ++;
```

Slide 9 : Stopwatch a = new Stopwatch();

↳ it initialise a variable to the current time (start = current time)

```
Stopwatch sw = new stopwatch();
Count (a)
Stdout.println (sw.elapsed time());
```

→ بحسب كم أخذ وقت لما نفذها

9/2/2020

running time = $a \times N^{b}$ ← b: Slope

↳ Plug one of the values in the equation.

execution time = # of instructions * execution time of each instruction → هاي في assemply

execution time = $\sum$ Cost of each statement * frequency of that statement → هاي اللي بنستخدمها

Slide 19 : لو حليت الـ code بالتجربة وهو أمر مزعج بإني أشوف واحسب كل عملية بنفسي

↳ int dec → 2

int assign → 2

less than compare → N+1

increment → (N → 2N) (0 → N)

equal to compare → N

array access → N

* Slide 20 : Note : less than compare : $\frac{1}{2}(N+1)(N+2)$

هاي ل i ↓ , هاي ل j ↓

* Slid 20 : Note : equal to compare : $\frac{1}{2} N(N-1)$

1 → N-1 ← i = 0 ⇒ N-1
2 → N-1 ← i = 1 ⇒ N-2
هاي ال j من ⋮
وين لوين بتلف . i = N-1 ⇒ ∅

من وين اجت ؟ ←

→ $\frac{(N-1)(N)}{2}$ = مجموع = $\sum_{i=0}^{N-1} i$

"N-1+1"

حسب القانون

* Finding the frequency of each statement is a tidy process, therfore we will use some kind of approximation :

① Most time consuming operation.

② Most frequency operation.

* Tilda approximation :

~F is the approximate value of F ⇒ $\lim_{N\to\infty} \frac{\sim F}{F} \simeq 1$ → لو N كبيرة

, when N is small ⇒ $\lim \frac{\sim F}{F} \neq 1$, but we don't care. → لو N صغيرة

11/2/2020

* $N^2 - N \longrightarrow N^2$

* Tilda notation : ignore lower order terms.

Slide 26 : in 3-sum ⇒ $\frac{1}{2} N^3$ array accesses

- How many 3 combinations out of N elements

$$\binom{N}{3} = \frac{N!}{(N-3)! * 3!} = 3 * \frac{N(N-1)(N-2)}{3!} \sim \frac{1}{2} N^3 \text{ array accesses}$$

* Mathmatical → Tilda approximation → order of complexity

we removed lower order terms. ↲

we remove constant ↲
↳ is dependent on many factors and we are interested in approximate

Ex : $f(N) \sim C * g(N)$
↓ for any constant

⇒ order of complexity = $g(N)$
⇒ to find order : find the most frequency statement and find its order.

Slide 30 : ① Most operations take constant time.

$C5 \gg C2$

* power : 2^{15} → order = 1
* log N : when we have a loop and in each iteration the size is decreased by half.
* N : loop that iterates over all elements !
* T(2N) / T(N) doubling factor : as you double the input size, what is the effect on running time?

Ex : $\frac{N \log N}{2N \log 2N} \simeq N$ / $\frac{(2N)^2}{(N)^2} \simeq 4$ / $\frac{2N}{N} \simeq 2$ / $\frac{\log N}{\log 2N} \simeq N$

* 2^N : exponential complexity : NP hard problem non polynomial time

Slide 40 : أحل الـ 3-Sum بطريقة أسهل من "3 loop"

```
For ( i=0 ; i<N ; i++ )
    For ( j=i+1 ; j<N ; j++ )
        log N  if → (Search in array for -(a[i] + a[j]) != -1 )
                        Count++
```

N^2 (for the two loops); * log N (for the search)

⇒ $N^2 * \log N$.

هيك الحل مش صح فبنعملها بـ

بطريقة ← Search in array for - ((j+1) → (N-1))*

أو * (Search in array for- (a[i]+a[j] != -1 && k>j)

* هذا الحل لو كانت الـ array sorted لو مش Sorted بنعملها Sorted

↳ $N^2 + N^2 \log$ ⇒ $N^2 \log N$

* بنعمل Sorting لما بدي أحسب 3-Sum لإنها بتقرب الحل أما لو عملت Search لـ Array

لما نعمل Sorting. ⇐ الـ Search (log N if sorted / N if not sorted)

13/12/2020

* order of growth :
Ex : $\frac{1}{2}N^3 + N^2 + 3N + 5$
∴ Telda : $\frac{1}{2}N^3$
∴ growth order : N^3

* Big Theta $\Theta(N)$: right bound (average complexity) : The complexity of all inputs have specific range.
Ex : $\Theta(f) = N$ average → $a*N$ ____ $b*N$ شو أقصى اشي بنحتاجه.
* Big oh $O(N)$: on the worst case it takes N times complexity. ↑
* Big Omega $\Omega(N)$: lower bound will take N operations whatever the iput is. (difficult to find accurately). → إنه هاي الشغلة ما بقدر أحكي هاي المعلومة ؛ مثلا أفضل حل شو ما كان.
Ex : $\Omega(3sum) = N^2 \log N$ طالبين أحكي هيك
Ω → Theoritical lower bound
Ex : $\Omega(3Sum) = N$ (Not achievable yet) " ولكونها الأقرب ، إذا تكون الـ Ω
* if $O = \Omega$ So $\Theta = O = \Omega$

Slide 52 :
char[] array = new char[N]; جوا → 2*N + 24 bytes overhead

* The Size of any object should be multiple of 8, if not ⇒ Padding (1-7 bytes) → 8 بضيفلك ليصير يقبل القسمة على

16/12/2020

Stack : To hold the return address.

Ex : ① in web browsing.
② code : Main()
push the address on the stack { Square(value);}
Square(var) {
sum(7);
return; }
sum(var);
return;
* بيناديلي الأحدث ويعملك دائما

*Note : The size in queue and stack is flexible.

* Stack API Slide 5 :

→ Interface : Just name of functions that should be defined along with their signatures.

→ Implementation : The real code that implements the functions in the interface.

→ Client : Client code is the code that calls the implementation code.

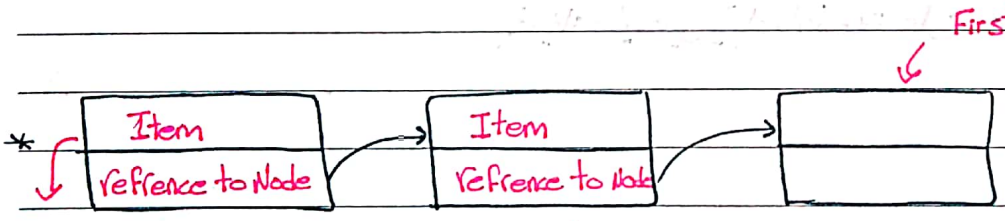
* Stack will be defined in 2 ways :

① Linked lists

② Array → Fixed size array
→ * resiable array

*In stack implementation : We have a linked list, and a pointer that refers to the top of the stack.

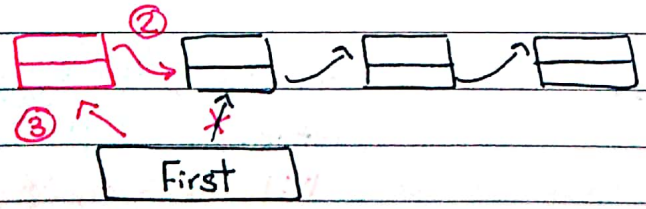
*to push an item on the stack : we push it before the top (first) and to pop from the stack, read the first item.



any data : String / double / object

* To add an element to a Stack (push) : 3 steps :

① create a new item (old first)



* Ideal Scenarios :

① Any insert or remove take constant time regardless the size of the collection.

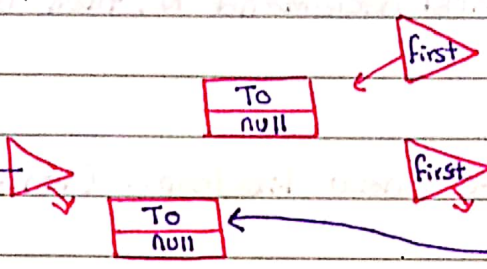
② The allocated size = a * Number of elements.

18/2/2020

Slide 11 :

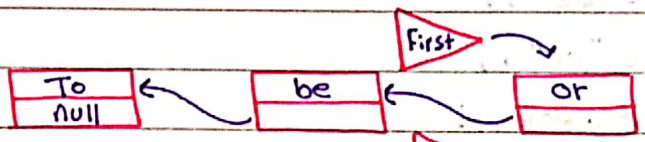
LinkedStackOfStrings Lss = new LinkedStackOfStrings();

Lss.Push("to");

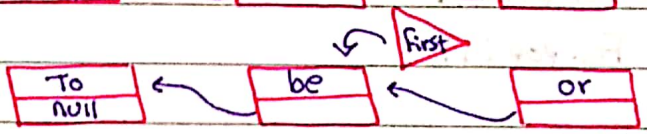


Lss.Push("be");

Lss.Push("or");



String Str = Lss.pop();



→ G.C. بيشال خلاص.

Ex : Class Cs {
int x,y;
double z;
String v;
funct1 { int c,d; }
funct2 { double x,b; }
}

what is the size of any object from class Cs ?

⇒ Sol : x→4 , y→4 , z→8 , v→8,
overhead → 16 bytes
∴ Σ = 40

* Note : array overhead : 24 bytes
object overhead : 16 bytes
innerclass overhead : 8 bytes

*For N nodes of LinkedStackOfStrings:

40 byte/Node * N Nodes = 40 N bytes.

Slide 16:

N → refers to the position after last element.

[x, y, z, v, null]

To push an item: we push it at location N and then increment N

To pop an element: first decremend N, then return the element at position N.

Slide 17:

```
FixedCapacit_____ FC = new FixedCap--(10);
  FC.push("to");
  Fc.push("be");
  FC.push("or");
String st = FC.pop();
String st = FC.pop();
```

*loitering: when there is an element with a refrence refers to it and you don't need it. Garbage collection will clear it.

*Size of stack of N Strings:

16 + 4 + 24 + N*8 = 8N ~ 8N

"object overhead" — 16; N — 4 + 24 + N*8

*problems with fixed size array stack:

① Fixed size and going out of the bound of the array.

② you reserve the worst case of the size even for one element.

*Advantages of fixed size array stack:

① fixed time push and pop() (constant)

② Memory usage for N element is ~ 8N

⟹ we do ∴ hybrid structure ⇒ resizable array.

20/12/2020

* Fixed capacity stack ∴
- Execution time → constant time
- Memory usage → ~ 8 N

* problems ∴ ① Fixed size

arr [N]

Throw exeption ← -1 N > S.length ⇒ resizing array

* Ideal Scenarios ∴

① Insert and remove take constant time that is not related to the size of collection

② Memory usage to store N elements must be ~ a*N

* String [] s = new String [1];

push ∴ + Size by 1 ~ N^2

pop ∴ - Size by 1

* Push N elements ∴

1 → 1 (array access)

2 → 1 + 2 (copy)

3 → 1 + 4

4 → 1 + 6

i → 1 + 2 (i-1)

N → 1 + 2 (N-1)

* To push element # N ⟹ 1 + 2 (N-1) array accesses, ignoring array ~ 2 N

« Constant time هذا ليس منطقي لأنه ما حيطلبني »

* increase by 10 when the array is full → 100,000 resize

* double size when the array is full → 20 resize

Five Apple

• resizing_array implementation:

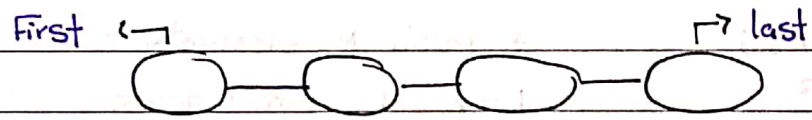
Push N elements

1 → 1 + 0
2 → 1 + 2
3 → 1 + 4
4 → 1
5 → 1 + 8
6 → 1
7 → 1
8 → 1
9 → 1 + 16

$\Rightarrow \sum = \sim 3N$

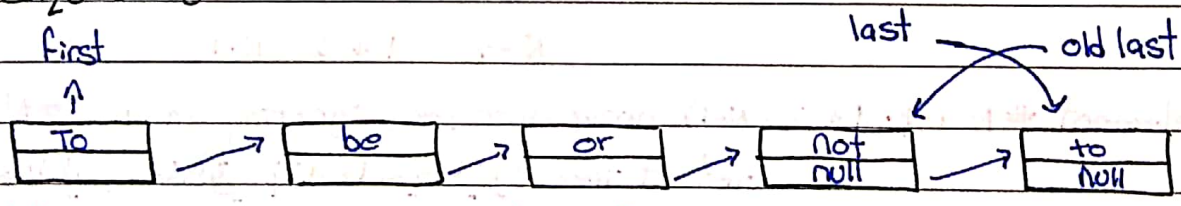
*Note: resizable array: most pushes and pulls take constant time, Some operations which need resizing for the array take time related to the size of the stack.

23/2/2020



enqueue ≡ Push last
dequeue ≡ Pop first

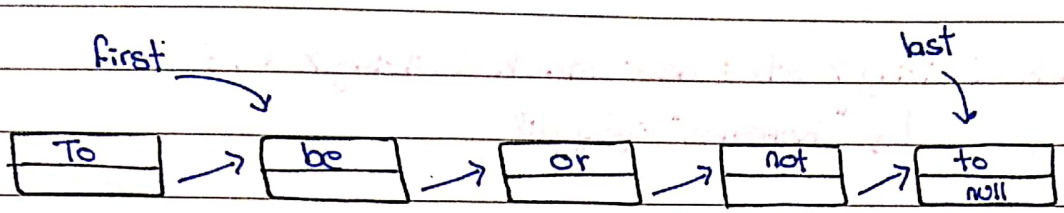
* enqueue:




```
Public void enqueue (String item ) {
  Node oldlast = last ;
   last = new node ;
   last. item = item;
   last. Next = null ;
   oldlast.next= last ; }
```

* dequeue :

```
public String dequeue () {
   String item = first. item ;
   first = first. next ;
    return item ;
}
```



* والكود الـ 2 codes زي كلمين الـ 2 codes الكاملين "Slide 37" .

Ex Slide 37 :

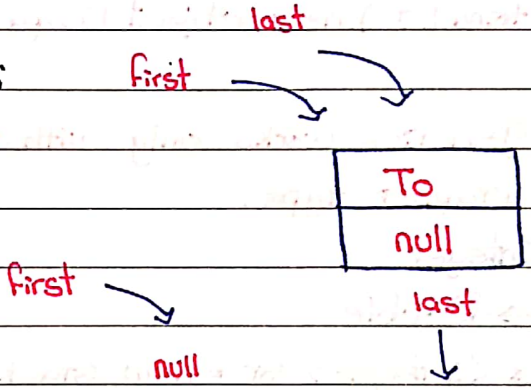
Linked Queue -- LQ = new Linked-- ();

LQ. enqueue ("To") ;

LQ. enqueue ("be") ;

String str = LQ.dequeue(); item = "To"

str = LQ. dequeue(); item = "be"



* Slide 41 :

{ null empety }

first last

enqueue

```
arr [last++] = item ;
```

dequeue

```
return arr [first++]
```

* 100 elements "(Note)"

25/2/2020

Slide 46: Stack <String> Str = new stack <String> ();
↳ "Refrence" يؤشر

Slide 48: S = new Item[capacity]; X, error in Java because general arrays are not allowed in Java.
S = (Item[]) new object [capacity]; ✓ (الصحيح)

*Note: generics works only with objects; not primitive data types
=> use wrapper type.
int → Integer
double → Double

Ex: Stack <Integer> sr = new stack <Integer>();
Sr.push(15) ↳ automatically converted into Integer (autoboxing)
int j = Sr.pop(); → auto unboxing

Slide 66: import java.util.*; {
LinkedList <string> LS = new LinkedList <string>();
LS.add ("To");
LS.add ("be");] → مثال لي استخدامها.
:
}

Slide 67: Java.util.Stack extends Vector, Vector implements List.
* Vector is a resizable array.

* Stack <Integer> st = new Stack<Integer>(); {
st.push(15); → "متاحة لي و تستطيع استخدامها"
}

*The existing data structure in Java may own un-needed and un-suitable Functions.
* Some functions will not take constant time.
* Slide 68 & Queue<String> q = new LinkedList<String>
Interface هي "new Queue" لي ماينفع أعمل
"Implementation" هاي و

		resize	size	# of elements
push	1	No	1	1
	2	re	2	2
	3	re	4	3
	4	No	4	4
	5	re	8	5
	6	No	8	6
	7	No	8	7
	8	No	8	8
	9	re	16	9
	10	No	16	10
	11	No	16	11
	12	No	16	12
	13	No	16	13
	14	No	16	14
	15	No	16	15
	16	No	16	16
	17	re	32	17

Ex8 O(N)
20 resize
10^6 - 20 No resize
→ cost = constant time.

• Arrays Sorting in Java :

- Arrays.Sort (a);
- Collections.Sort (c);

* Sorting is usually based on key.
* key can be all member variables or part of it.

Slide 11 :

Paper > Stone
Stone > Scissor ⟹ مفروض حسب هاي تكون Paper > Scissor و لكن لم
Scissor > Paper يحدث . إذا ليس مثال على Arrays Sorting .

* To do Sorting on user defined types, we should define the natural order (≤) by implementing compareto function.
* Interface Comparable : any class that implements this interface must define "CompareTo"

=> The array of objects or collection of objects from this class can be sorted

* all wrapper types and many other types implement comparable interface.

Ex : obj1.CompareTo (obj 2);

returns : +1 → obj1 > obj2
0 → obj1 = obj2
-1 → obj1 < obj2

```
Ex : class person {                    → implements Comparable <person>
     Date DoB;
     Long NationalID;
     double weight;
     String name;
Public int compareTo (Person that) { if this.NationalID < that.NationalID
 return +1  else return -1 }
}
  Void main ( ) {
Person [ ] ar = new person [50];
   = we added 50 persons to they array.
   array.Sort (ar); } → "كشان يقدر يعمل Sort لازم أعمل اللي بالأبرز"
```

• Elementary sorts :

① Selection Sort ("تقنية")

② insertion sort

③ Shuffling

① Selection sort :

① Ex : { N elements }

$i = 0 \rightarrow N-1$

```
for ( i=0 ; i<N; i++ ) {
    min = i
  for (int j= i+1 ; i<N ; j++ ) {
        if (a[j] < a[min])
            min = j ;
    }
    exchange (a[i], a[min])
}
```

Slide 50 : * Note : less (obj 1 , obj 2)

↓ class implements comparable.

Slide 51 : Selection. Sort (Static لإنها class قدرت أناديها باسم ال

↳ because Sort is a Static method .

* Selection St= new Selection ();

St . Sort ( ) ; X → class ما بصير، بناديها باسم ال

Slide 51 : * less => N^2

↓ * exchange => N

* i : 0 → N-1 j : i+1 → N-1

i = 0 → j : N-1

i = 1 → j : N-2

i = 2 → j : N-3

i = 3 → j : N-4

i = N-2 → j : 1

i = N-1 → j : 0

$$\sum_{i=0}^{N} i = \frac{N*(N+1)}{2} = \frac{(N-1)*N}{2} = \frac{1}{2}N^2 - \frac{1}{2}N$$

$$* \sim \frac{1}{2}N^2$$

$$* G_1.O = N^2$$

3/3/2020

* Selection Sort : average time = worst case time = best case.
* Selection Sort : has no additional memory overhead.

② insertion sort : → "بس لو ضلت صفر عادي ما بتفرق" ← الأفضل تكون "1"

Ex : for(i=0 ; i<N ; i++)
for(j=i ; j>∅ , j--)

= insertion sort : it exchanges only adjacent elements if a[j] < a[j-1]

∴ The code :

```
for(i=1 ; i<N ; i++)
   for( j=i ; j>∅ , j-- )
      if ( less(a[j] , a[j-1] )
         exch (a, j, j-1 )
      else
         break;
```

* if the array is totally sorted ⇒ less : N
exch : ∅

* if the array is in reverse order (worst case) ⇒ less : $\frac{1}{2}(N-1)*N \sim \frac{1}{2}N^2$
exch : $\frac{1}{2}(N-1)*N \sim \frac{1}{2}N^2$

* insertion sort : on average $\frac{1}{4}N(N-1) \sim \frac{1}{4}N^2$.
G.O : N^2 .

③ Shuffling :

Ex :
```
for ( int i = 0 , i<N ; i++ )
{    int r= Math.random (0, i) ;→ int r= StdRandom.uniform (i+1)
     exch (a[i], a[r] );
}
```

⇒ complexity : N

5/3/2020

* Merge Sort : is divide and conquer method.

- Merge code :

```
merge( )
{
   Copy array to auxiliary array ;
   [ lo (low) .  Mid / Mid+                hi (high) ]
     ↑           ↑
     i           j
   merge code
}
```

* Note: Arrays.sort, Collections.sort ⇒ they use merge sort for objects and quick sort for primitives.

* when a method takes a parameter of type comparable [ ] : in the client code you can pass to this method an array of objects from class that implements the interface comparable.

```
int [ ] a = new int [1000];
* merge.sort (a);
```

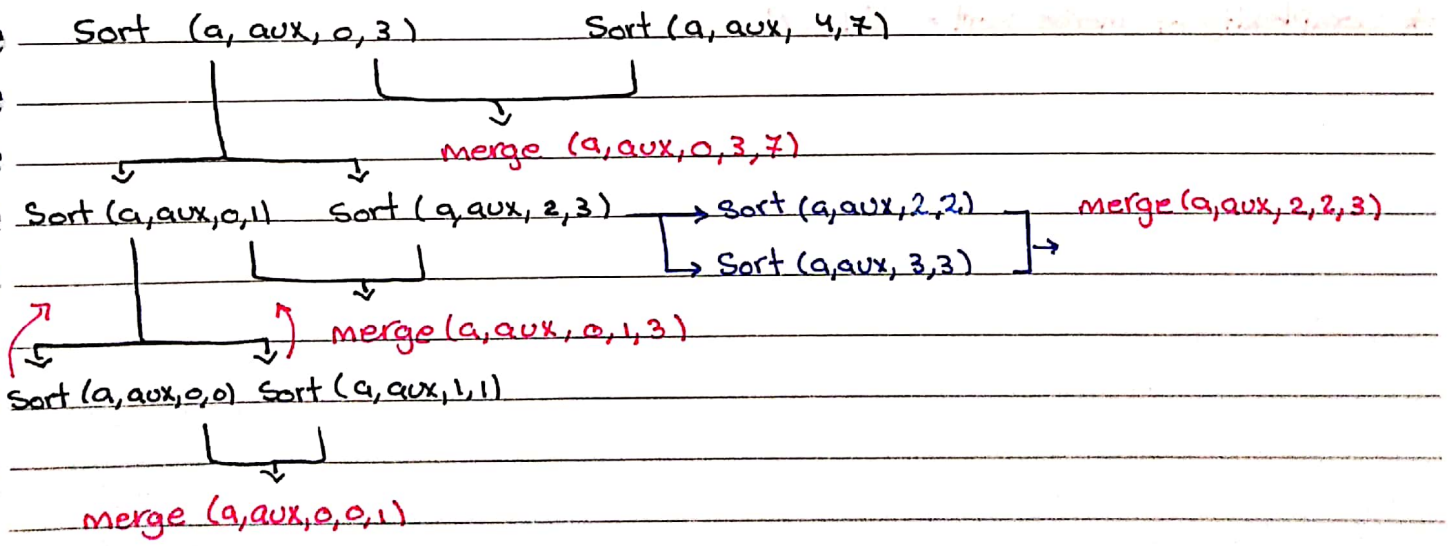
array لازم يكون متلوا

انا بنشتوا حشان ما ينفع أرتبهم بنفس مكانهم".

Ex: Merge of 8 elements :

[0 , 1 , 2 , 3 , 4 , 5 , 6 , 7]

↑ lo ↑ mid ↑ hi



```
* Public static void (comparable [] a, comparable [] aux, int lo, int mid, int hi )
{   for (int v = lo ; v <= hi ; v++ )
        aux [v] = a [v] ;                                   ] "copy"
    int i = lo ;          // index for first element in 1st half
    int j = mid + 1 ;     // index for first element in 2nd half
    for ( int k = lo ; k <= hi ; k++ )
    {   if (less aux[i], aux[j] )   a[k] = aux [i++];       ] "merge"
        else                        a[k] = aux [j++];
    }
                                        :عشان يكمل ال code لازم هون نحط
}
        if ( i > mid ) a[k] = aux[j++]; // if first half exausted
        else if ( j > hi ) a[k] = aux[i++]; // if second half exausted
        else if - - - - -
```

* Note: [,] ⇒ N → N/2, N/2

⇒ mid

* less (compare) = N-1 = worst case ~N

* best case = N/2

N
N/2 N/2 — N <
N/4 N/4 N/4 N/4 — N = 2 * (N/2) <
N/8 N/8 N/8 N/8 N/8 N/8 N/8 N/8 — N = 4 * (N/4) <

log N levels ⇐

N * log N comparisions

* array access = 6 * N log N .

" * additional memory : ~N "

* Complexity of merge sort = N log N .

8/3/2020

Slide6 : if two keys are equal, the merge copies the element from the left half.

* EX : [16] ((حل التحسين من ال Memory ولكنه بسيط))

k ↑ ↓ ↑ j

[ ] aux : $\sim \frac{1}{2} N$ / complexity : N

i ↑

* when we do recursive function calls, all local variables are pushed into the memory ⇒ high cost.
* If the cost of calling and returning from a function is higher than the cost of the function it self, try to avoid recursive.

10/3/2020

* Note : [↓y , ↓x]
↳ if y < x you don't have to do merge.

* Comparators.

Ex : UniversityRoom [] ua = new UniversityRoom [100];
and code to add 100 objects of university Rooms.
Arrays. Sort (ua); ⟶ هنا الـ code حيشتغل في حال : ↓

↳ . sort ⇒ use natural order defined by compareTo.

```
Public class universityRoom implements Comparable <university Room> {
    double area;
    double Size;
    int capacity
Public int compareTo (university Room that) {
    Return this. capacity - that capacity ;
}
}
```

→ لو كنت بدي أرتب بطريقة غير رح نستخدم Comparators اللي حنشرحه.

* Comparable → compareTo (one object)
Comparator → Compare (o1, o2)

If you want to do Sorting according to any of X fields
⇒ X inner class that inplement Comparator.

* if the inner class is static, you can crate objects from it with referring to an object from the outer class.
* if static → Student.Byname bn = new Student.Byname ;

* if not static → Student st = new student ();
Student.Byname bn = st.new Byname ();

* Arrays.sort (a , object from the inner class you want to sort according to inner class .)

12/3/2020

If you have a collection of objects orderd in any format, if after sorting the collection using a specific key , if the relative items with the same key value has changed order of the ⇒ Sorting algorithm is not stable .

Note : long distance exchange most Probably ⇒ not stable

* Selection
* not stable ⇒ because the relative order of elements of the same key value was not maintained.

Insertion
* stable

Merge
* will be stable if the merge fonction is stable (keeps relative order of keys with the same value) ⇒ Stable حسب المبنود
اللي عنا لو كان المبنود هيك
↓ not stable بصير

else if (less(aux[i], aux[j])
a[k] = a[i++]
else a[k] = a[j++]
↳ Stable because when two keys are equal ⇒ we take the element from the first half.

* Quicksort :
Shuffle : to avoid worst case. $\frac{1}{2}N^2 \longrightarrow N \log N$

* insertion sort : worst $\sim \frac{1}{2}N^2$
⇓ Shuffle ⇒ $\sim \frac{1}{4}N^2$

* Code بيل كي استخدام الـ Comparators &

```
public class UniversityRoom Implements Comparable <UniversityRoom> {
        Public static int attribute ;

public int compareTo (UniversityRoom that) {
        if (attribute == 1)
            return this.area - that.area;
    else if (attribute== = 2)
            return this.Size - that.size ;
    else    return this.capacity - that.capacity ;
    }
}
```