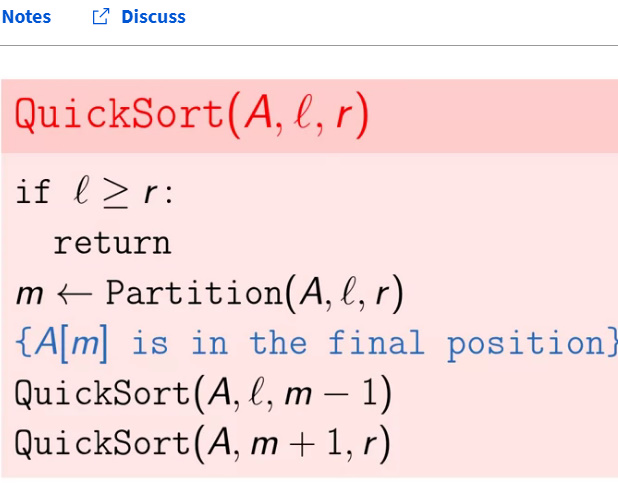
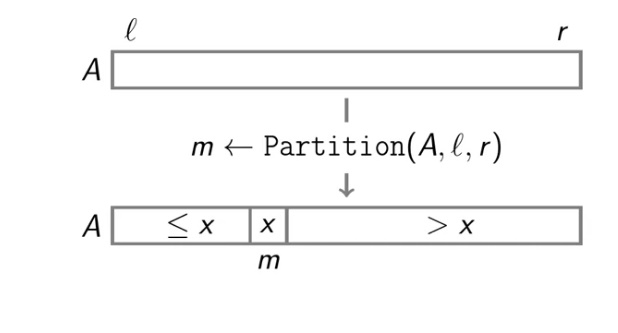
* O(n) = nlog n in medie
* 
* l si r arata pozitia initiala si finala
* initial l va fi 0 si r lungimea la array
* m va fi pozitia elementului pivot ales care a fost situat dupa aranjare asa incat tot ce e in stang sa fie mai mic sau maxim egal ca el si ce in dreapta mai mare
* apoi sortam iar subsirul din stanga lui si apoi din dreapta lui

Asta trebuie sa obtinem:



* Quick Sort se bazeaza pe faptul ca alegem un oricare element pivot, deobicei ultimul, si facem in asa fel incat sa il aducem la pozitia in care toate elementele din stanga lui sunt mai mici si cele din dreapta sunt mai mari. De ex:

5 2 1 8 9 6 2 3

Alegem 3 ca element pivot, si deci va trebui sa obtinem dupa prima iteratie asa ceva:

2 5 1 8 9 6 2 3

2 1 5 8 9 6 2 3

2 1 2 8 9 6 5 3

Final:

2 1 2 **3** 8 9 6 5

Vedem ca toate elementele din stanga sunt mai mici si din dreapta mai mari.

* Cum functioneaza:

5 2 1 8 9 6 2 **3**

* **1.** Alegem ca pivot ultimul element, adica 3
* Vom avea un for care va incepe de la pozitia initiala, in cazul dat 0, pana la cea finala, adica r, dar fara ultimul element, deci r-1 si o variabila j care va memora care a fost ultimul element mai mic ca 3 pus
* Acum trebuie facem cumva ca elementele mai mici ca 3 sa fie cat mai in stanga si cele mai mari ca el cat mai in dreapta, ca el sa poata sta intre ele
* for i = 0 si j = 0

5 > 3, deci nu avem ce schimba, lasam asa unde e

* for i = 1 si j = 0

2 < 3

Deci, inseamna ca trebuie trecut cat mai in stanga posibil, de aceea il schimbam cu 5

swap(array[i],array[j])

++j

2 5 1 8 9 6 2 **3**

* for i = 2 si j = 1

1 < 3

swap(array[i],array[j])

++j

2 1 5 8 9 6 2 **3**

* for i = 3 si j = 2

5 > 3, nu avem ce impinge

* for i = 4 si j = 2

8 > 3

....

* for i = 6 si j = 2

2 < 3

swap(array[i],array[j])

j++

2 1 2 5 8 9 6 **3**

* am ajuns la final, si j = 3, deci j trebuie sa stea pe a 3 pozitie, si asa va fi in mijloc, adica in stanga vor fi termeni mai mici ca el,si in dreapta mai mari, deci

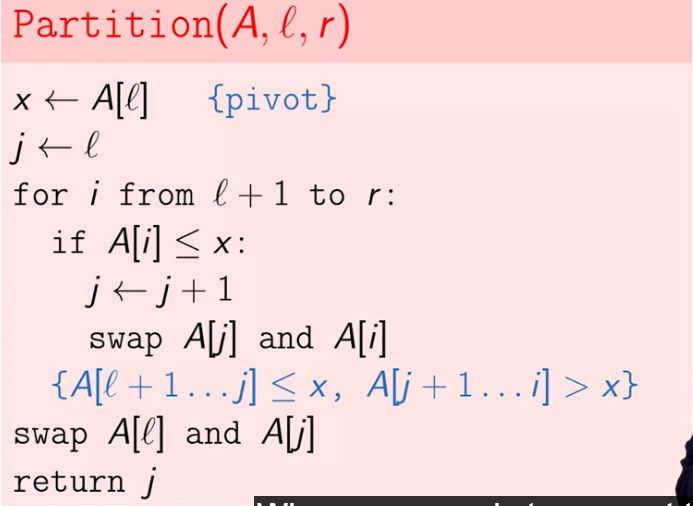
swap(array[last],array[j])

2 1 2 **3** 5 8 9 6

* **2.** Acum cream 2 array, adica 2 1 2 si 5 8 9 6, si iar luam ca elemente pivot ultimul element, deci respectiv 2 si 6

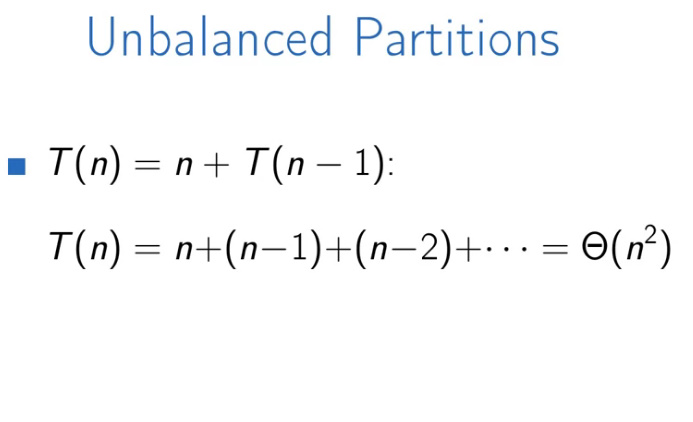
**si acum iar facem recursiv acelasi proces, si tot asa pana vom ajunge ca nu mai putem crea subsiruri.**

* In acest algoritm, ca element pivot e ales primul element, nu ultimul ca mai sus
* l – pozitia initiala
* r – pozitia finala



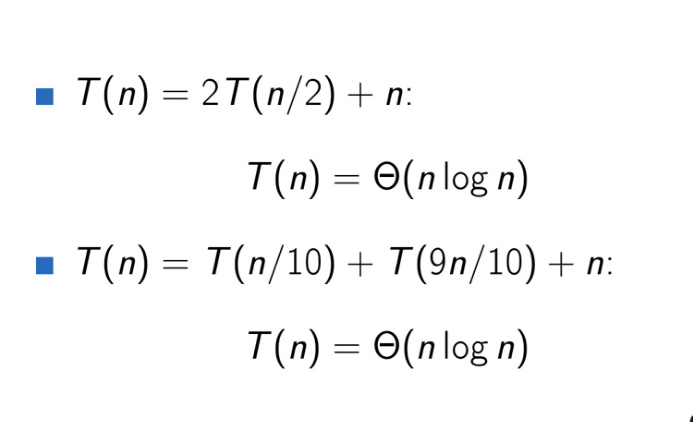
**Running Time**

* stabilirea la running time este destul de complicata si anevoioasa, asa cum nu putem sti nicidecum cate elemente pivor ar urma sa fie selectate.
* Totusi, putem estima relatia de recurenta in multe feluri, in dependenta de cat de balansate vor fi subarray create, de exemplu putem lua un caz cam asa:

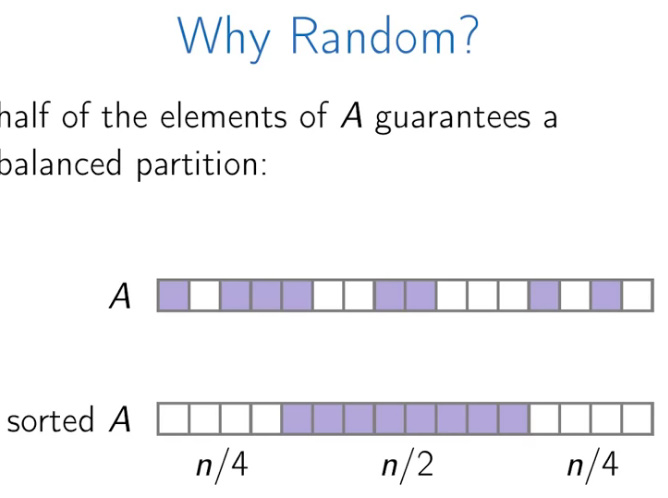


Deoarece, am avea ca T(n) = n\*(n+1)/2 care e egal cu (n^2+n)/2 si deci ramane doar n^2

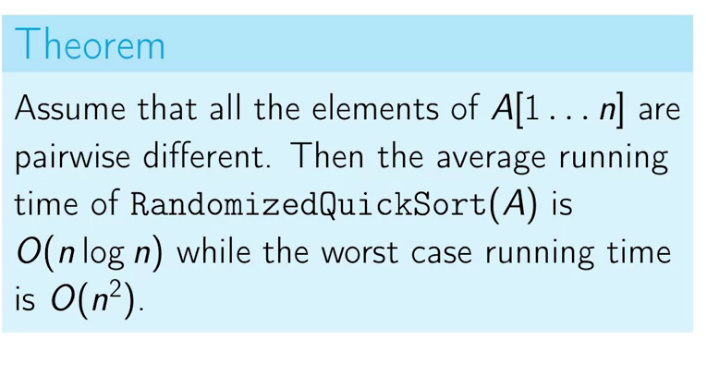
* Observam ca obtinem n^2, care nu e prea rapid chiar
* Se pot intampla totusi si situatii cand array se imparte mereu in jumatate si apoi iar in jumatate, deci cand e balansat:

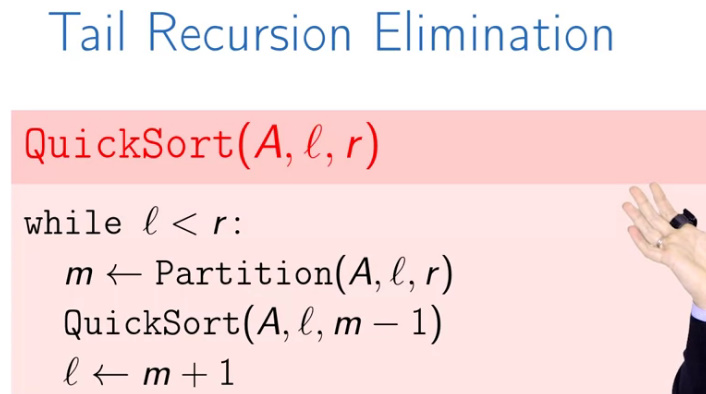


* Totusi, situatiile cand e balansat pot fi mai rare.
* Cea mai rapida varianta e considera cand alegem totusi elementul pivot random si il schimbam cu elementul de la urma, sau de la final, cum ne place.



Probabilitatea ca elementele alese random sa permita sa avem siruri balansate e mai mare decat cea ca alegand mereu ultimul element vom obtine asa caz.



****