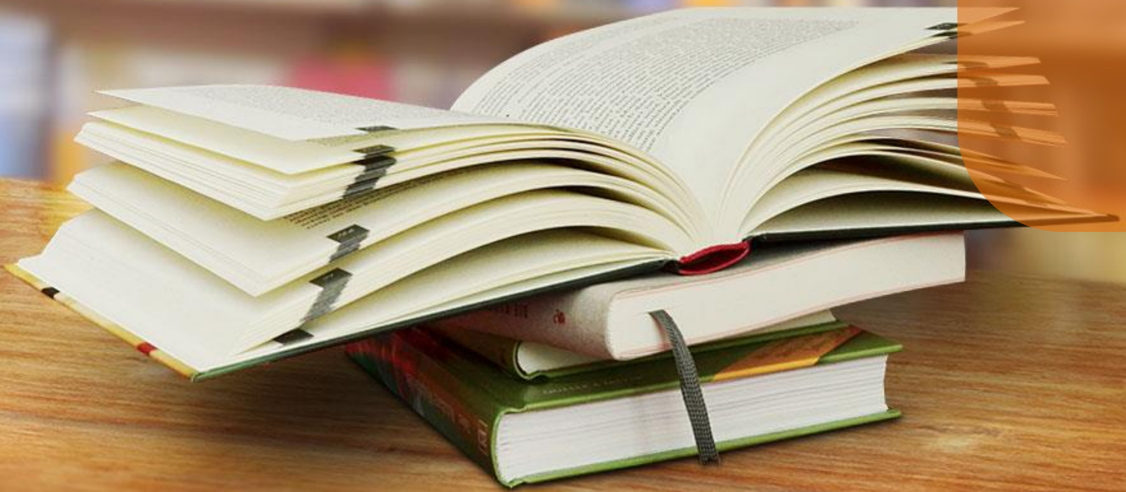
QUICK SORT



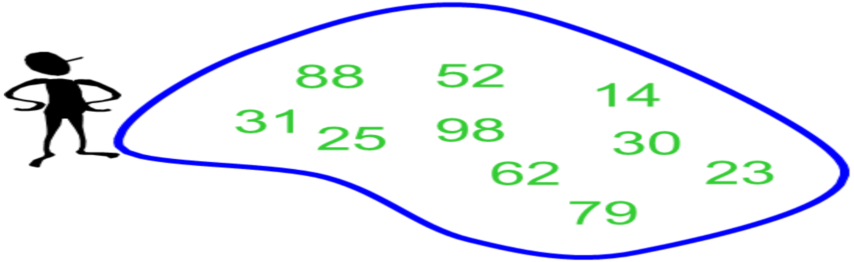
Identify the problem


ComputerHope.com

Sorting

Problem Example

Quick Sort

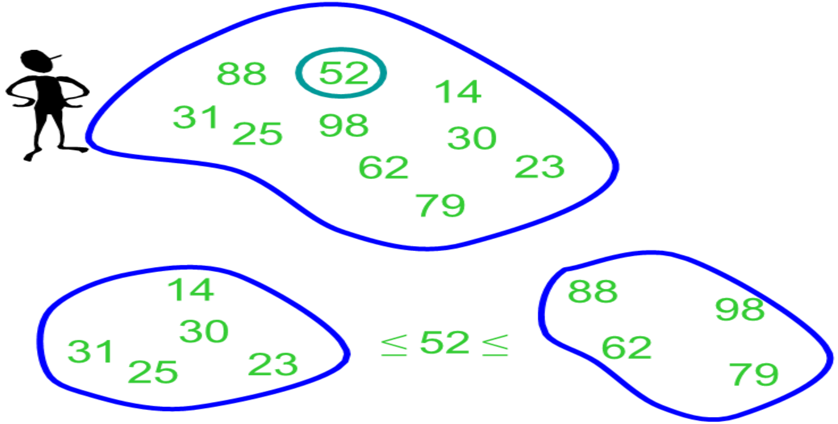


Divide and Conquer

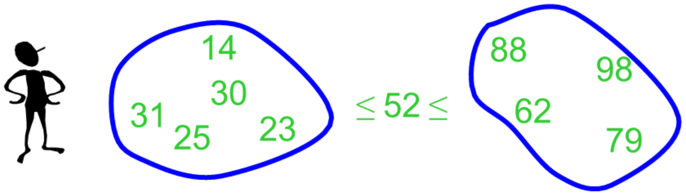
Problem Example

Quick Sort

Partition set into two using randomly chosen pivot



Quick Sort



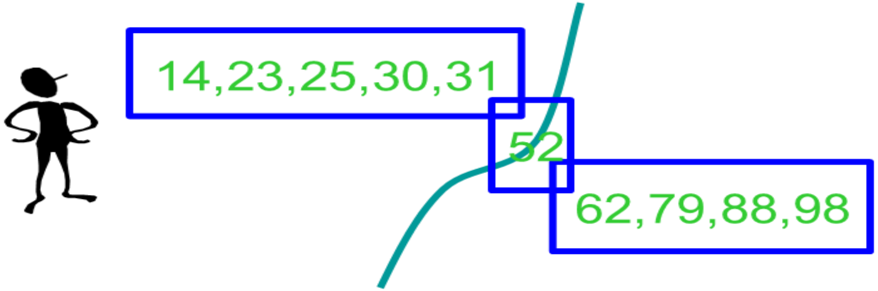
sort the first half.

14,23,25,30,31

sort the second half.

62,79,98,88

Quick Sort



Glue pieces together.

14,23,25,30,31,52,62,79,88,98

Quicksort

- Quicksort pros [advantage]:
  - Sorts in place
  - Sorts $O(n \lg n)$ in the average case
  - Very efficient in practice , it's quick

Quicksort cons [disadvantage]:

- – Sorts $O(n^2)$ in the worst case
  - And the worst case doesn't happen often … sorted

Quicksort

Another divide-and-conquer algorithm:

Divide: **$A[p…r]$** is partitioned (rearranged) into two nonempty subarrays **$A[p…q$-1]** and **$A[q$+1…$r]$** s.t. each element of $A[p…q-1]$ is less than or equal to each element of $A[q+1…r]$. Index q is computed here, called **pivot**.

Conquer: two subarrays are **sorted by recursive calls** to quicksort.

Combine: unlike merge sort, no work needed since the subarrays are sorted in place already.

Algorithm

ALGORITHM *Quicksort(A*[*l..r*]*)*

//Sorts a subarray by quicksort

//Input: Subarray of array *A*[0..*n*   1], defined by its left and right

//indices *l and r*

//Output: Subarray *A*[*l..r*] sorted in nondecreasing order

if $l < r$

s ←*Partition(A*[*l..r*]*)* //*s* is a split position

Quicksort(A[*l*….*s* − 1]*)*

Quicksort(A[*s* + 1…..*r*]*)*

Algorithm

ALGORITHM *HoarePartition(A[l..r])*

//Partitions a subarray by Hoare's algorithm, using the first element
//as a pivot
//Input: Subarray of array *A*[0..*n*-1], defined by its left and right
//indices *l* and *r (l < r)*
//Output: Partition of *A*[*l*..*r*], with the split position returned as
//this function's value

```
P=A[l]
i=l;j=r+1;
repeat
            repeat i=i+1 until A[i]>=p
            repeat j=j-1 until A[j]<=p
swap(A[i], A[j ])
Until  i>=j
swap(A[i], A[j ])  //undo last swap when i>= j
Swap (A[l], A[j ])
return j
```

Complexity Analysis of Quick Sort

Worst Case Time Complexity [Big-O]: **$O(n^2)$**

Best Case Time Complexity [Big-omega]: **O(n*log n)**

Average Time Complexity [Big-theta]: **O(n*log n)**

Space Complexity: **O(n*log n)**

Thank you!