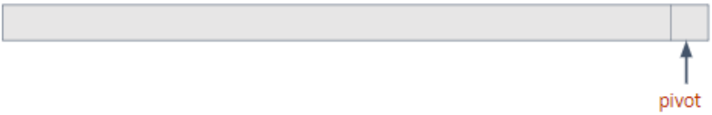
Quick Sort

Please refer to chap7 from CSLR

Sakeena Shahid | 21-01-2022

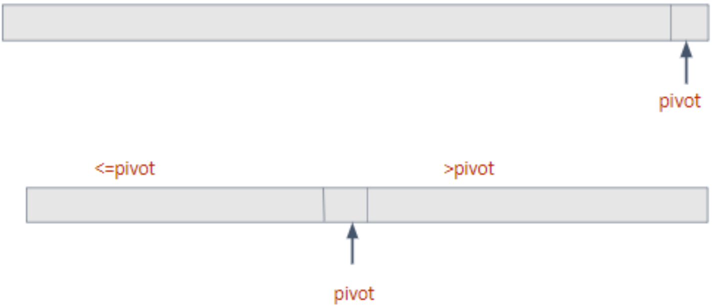
Quick Sort



Diff ways to select Pivot :-

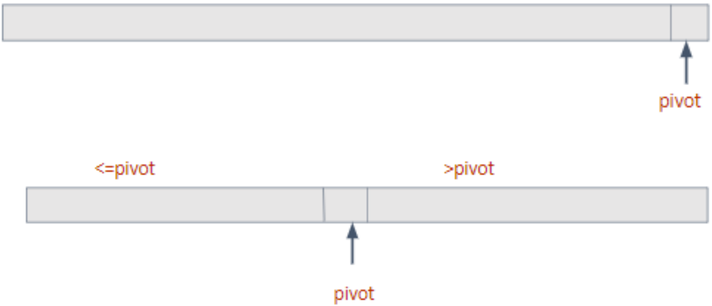
- first element
- last element
- middle element
- random element - later
- median element (adds to complexity)

Quick Sort



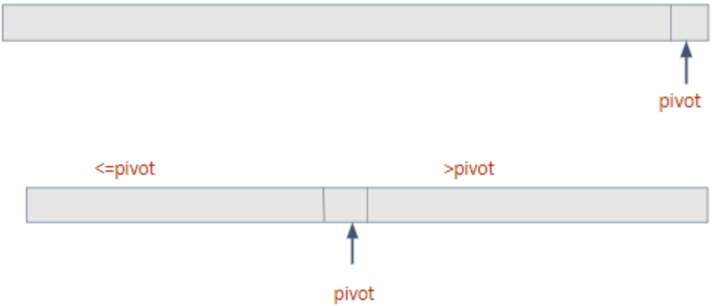
after partitioning

Quick Sort



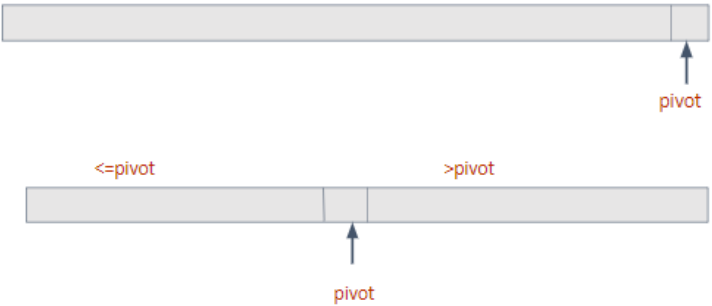
10, 25, 8, 9, 21, 15, 10 $\quad$ $pivot = 10$

Quick Sort



10,	25,	8,	9,	21,	15,	10	$pivot = 10$
10,	8,	9,	10,	25,	21,	15	(after partition)

Quick Sort



10, 25, 8, 9, 21, 15, 10 $\quad$ *pivot* $= 10$

10, 8, 9, 10, 25, 21, 15 $\quad$ (after partition)

8, 9, 10, 10, 15, 21, 25

(after sorting the partitions recursively)

Quick Sort

input : Array: $p, r, A[p \ldots r]$
output: Sorted Array: $A[p] \leq A[p+1] \leq .... \leq A[r]$

```
QuickSort(A,p,r)
/* Performs sorting on the input array */
if p < r then
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end
```

returns the index of pivot element

leaves pivot in the next iteration (Why?)

Quick Sort

input : Array: $p, r, A[p \dots r]$
output: Sorted Array: $A[p] \leq A[p+1] \leq .... \leq A[r]$

QuickSort(A,p,r)
/* Performs sorting on the input array */
if $p < r$ **then**
  q=Partition(A,p,r) → returns the index of pivot element
  QuickSort(A,p,q-1)
  QuickSort(A,q+1,r)
end

leaves pivot in the next iteration (Why?) Because it is in its correct position.

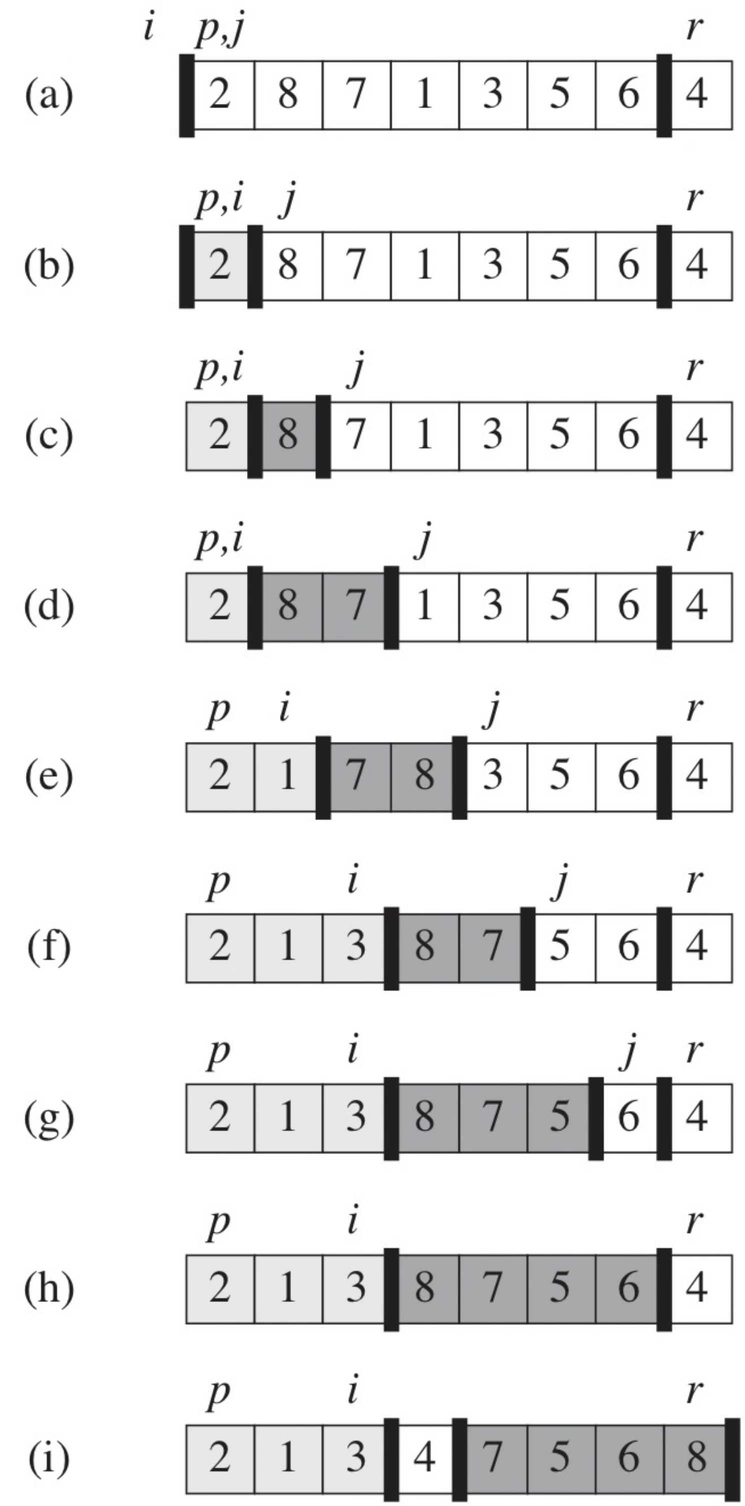
Partition Pseudo-code

input : Array: $p, r, A[p \dots r]$
output: q: the Index of the pivot

```
Partition(A, p, r)
/* "p" and "r" are the first and the last indices, respectively,
 of the array A */
x=A[r]
i=p-1
for j : p to r − 1 do
    if A[j] ≤ x then
        i=i+1
        exchange A[i] with A[j]
    end
end
exchange A[i+1] with A[r]
return i+1
```

Partition Algorithm

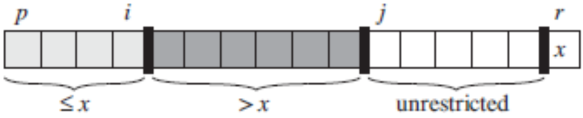
```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



Stop and think about the loop invariant.

Partition

INITIALISATION



[1]

pivot $= x$

Invariance:

- $A[p \dots i] \leq pivot$
- $A[i+1 \dots j-1] > pivot$
- $A[r] = pivot$

Initially, if $i = p - 1$ and $j = p$, first two invariance properties are satisfied vacuously.

[1] Figure from Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms

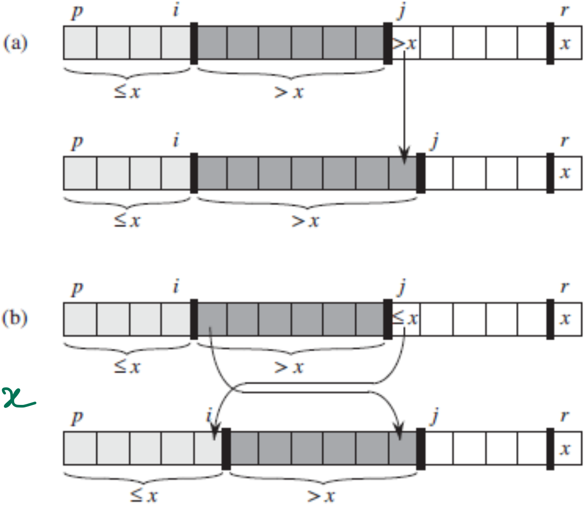
Partition contd..

MAINTENANCE

Invariance: $A[p \dots i] \leq pivot$, $A[i+1 \dots j-1] > pivot$, $A[r] = pivot$

case 1:
$A[j] > x$

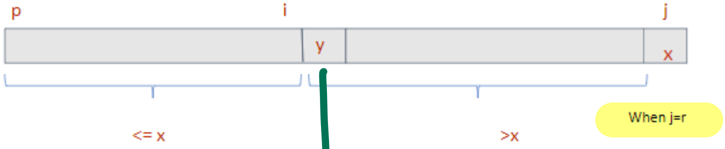
case 2:
$A[j] \leq x$



[2]Figure from Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms

Partition contd..

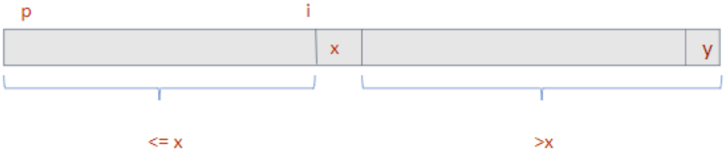
TERMINATION



$\geq x$ with the earliest index

Partition contd..

(last 2 statements : outside of loop)



Analysis
of
Quick Sort

Analysis of Partition

We will no longer refer to the pseudo-code for analysis. We will do it at abstract level. We know that we don't need to count the statements executed constant number of times. And we don't count the number of times loop control variable changes its value.

Analysis: After every comparison at least one key is in the right place.

Analysis of Partition

We will no longer refer to the pseudo-code for analysis. We will do it at abstract level. We know that we don't need to count the statements executed constant number of times. And we don't count the number of times loop control variable changes its value.

Analysis: After every comparison at least one key is in the right place.

After at most how many comparisons, all the keys will be in their right place?

Analysis of Partition

We will no longer refer to the pseudo-code for analysis. We will do it at abstract level. We know that we don't need to count the statements executed constant number of times. And we don't count the number of times loop control variable changes its value.

Analysis: After every comparison at least one key is in the right place.

After at most how many comparisons, all the keys will be in their right place?

n......Worst Case

Analysis of Partition

We will no longer refer to the pseudo-code for analysis. We will do it at abstract level. We know that we don't need to count the statements executed constant number of times. And we don't count the number of times loop control variable changes its value.

Analysis: After every comparison at least one key is in the right place.

After at most how many comparisons, all the keys will be in their right place?

n......Worst Case

What is the best case for Partition?

Analysis of Partition

We will no longer refer to the pseudo-code for analysis. We will do it at abstract level. We know that we don't need to count the statements executed constant number of times. And we don't count the number of times loop control variable changes its value.

Analysis: After every comparison at least one key is in the right place.

After at most how many comparisons, all the keys will be in their right place?

n......Worst Case

What is the best case for Partition?

It does exactly n (plus minus 1) comparisons in every case.

Analysis of Quick Sort

input : Array: $p, r, A[p \ldots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

```
QuickSort(A,p,r)
/* Performs sorting on the input array */
if p < r then
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end
```

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

$$T(n) = ? \quad + \quad +$$

Analysis of Quick Sort

input : Array: $p, r, A[p \dots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

```
QuickSort(A,p,r)
/* Performs sorting on the input array */
if p < r then
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end
```

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

$$T(n) = T(q-p) + T(r-q) + n$$

Analysis of Quick Sort

input : Array: $p, r, A[p \dots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

```
QuickSort(A,p,r)
/* Performs sorting on the input array */
if p < r then
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end
```

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

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

in best case when the partition is almost perfectly balanced

Analysis of Quick Sort

input : Array: $p, r, A[p \dots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

```
QuickSort(A,p,r)
/* Performs sorting on the input array */
if p < r then
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end
```

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

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

in best case when the partition is almost perfectly balanced

Analysis of Quick Sort

input : Array: $p, r, A[p \ldots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

QuickSort(A,p,r)
/* Performs sorting on the input array */
if $p < r$ **then**
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

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

in best case when the partition is almost perfectly balanced

☆. Why is this the best case? Reason discussed in class. Make sure you understand it.

Analysis of QuickSort Contd..

And, in the worst case

$$T(n) = T(n-1) + T(0) + n$$

when the partition is completely imbalanced

Analysis of QuickSort Contd..

And, in the worst case

$$T(n) = T(n-1) + T(0) + n$$

$$T(n) = T(n-1) + T(0) + n = \theta(n^2)$$

when the partition is completely imbalanced

☆. Why is this the worst case? Reason discussed in class. Make sure you understand it.

What will the average case look like?

Few recurrences which could resemble our average case :

- $T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$

- $T(n) = T\left(\frac{2n}{3}\right) + T\left(\frac{n}{3}\right) + n$

solve using any method you like.
- CLRS Ch-3 Recursion Tree method Pg 91

Complexity?

Is it closer to best case or worst case?