Top 10 Algorithms in the 20th century

QuickSort

Shunyao Li

10/22/2014

Outline

- Introduction
- QuickSort
- Complexity Analysis
- Optimization
- Application

Outline

- Introduction
- QuickSort
- Complexity Analysis
- Optimization
- Application

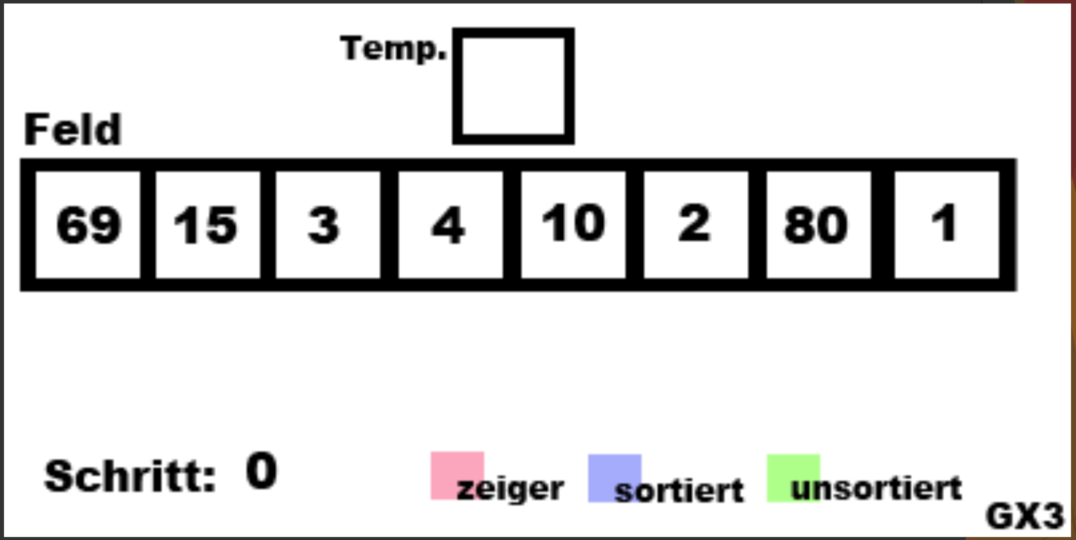
What is sorting

What is sorting

History

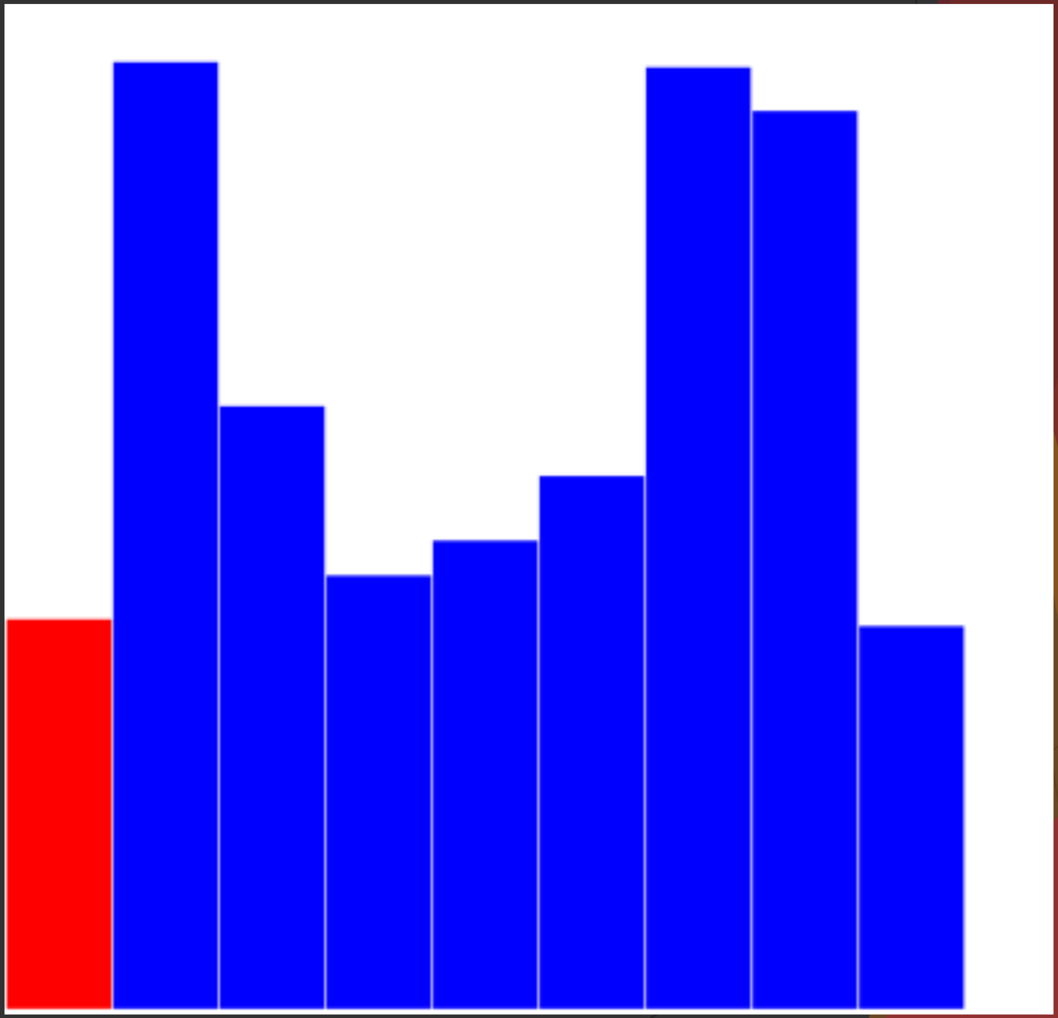
History

Insertion Sort



History

- Insertion Sort
- Selection Sort



History

- Insertion Sort
- Selection Sort
- 1948 Merge Sort
- 1956 Bubble Sort
- 1960 Quick Sort — most widely used
- 1964 Heap Sort
- 21st century Hybrid Sorting algorithms

Concepts

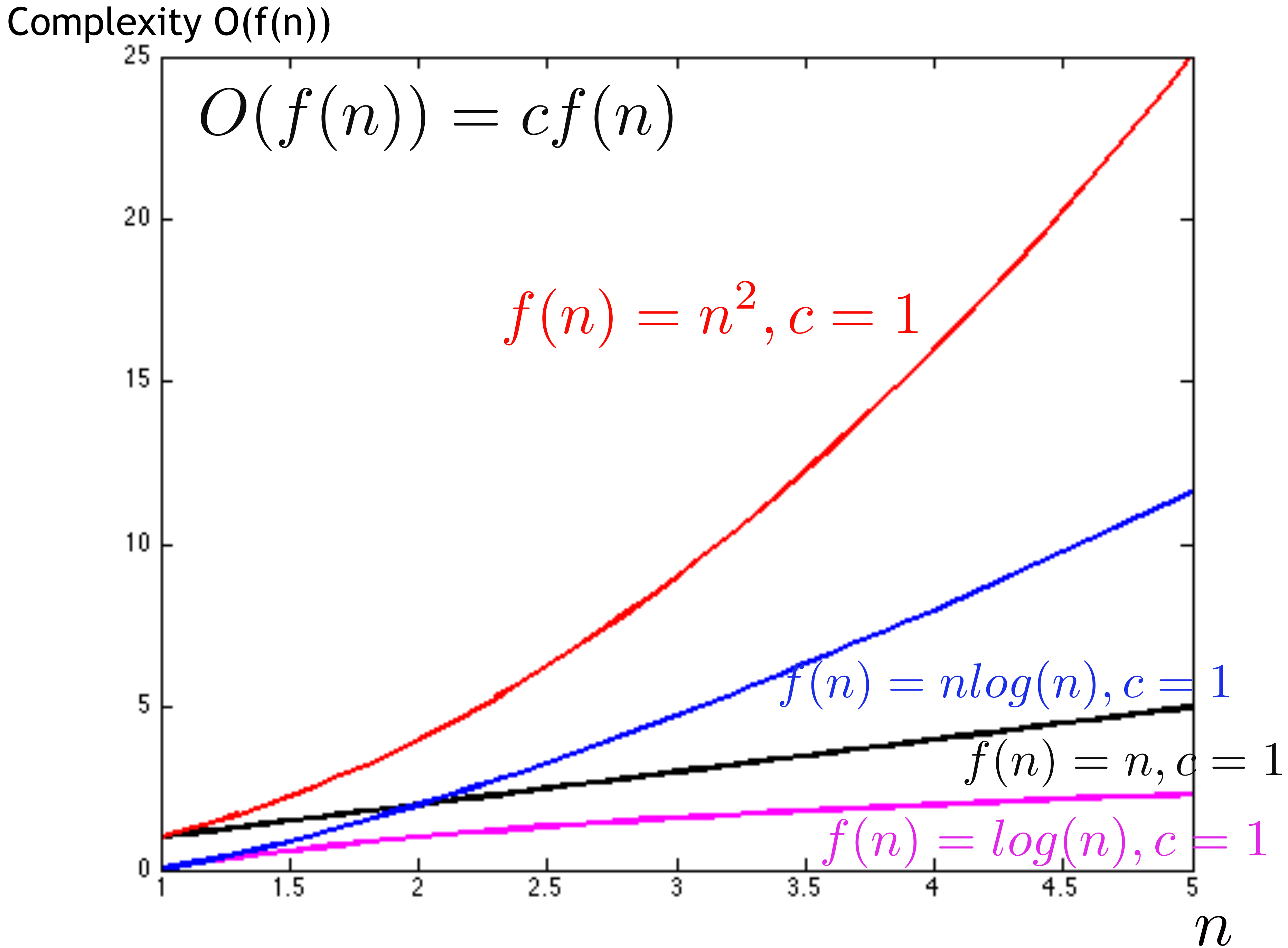
- Time Complexity
  - number of comparisons
- Space Complexity

function of n (scale of the problem)
best/worst/average cases

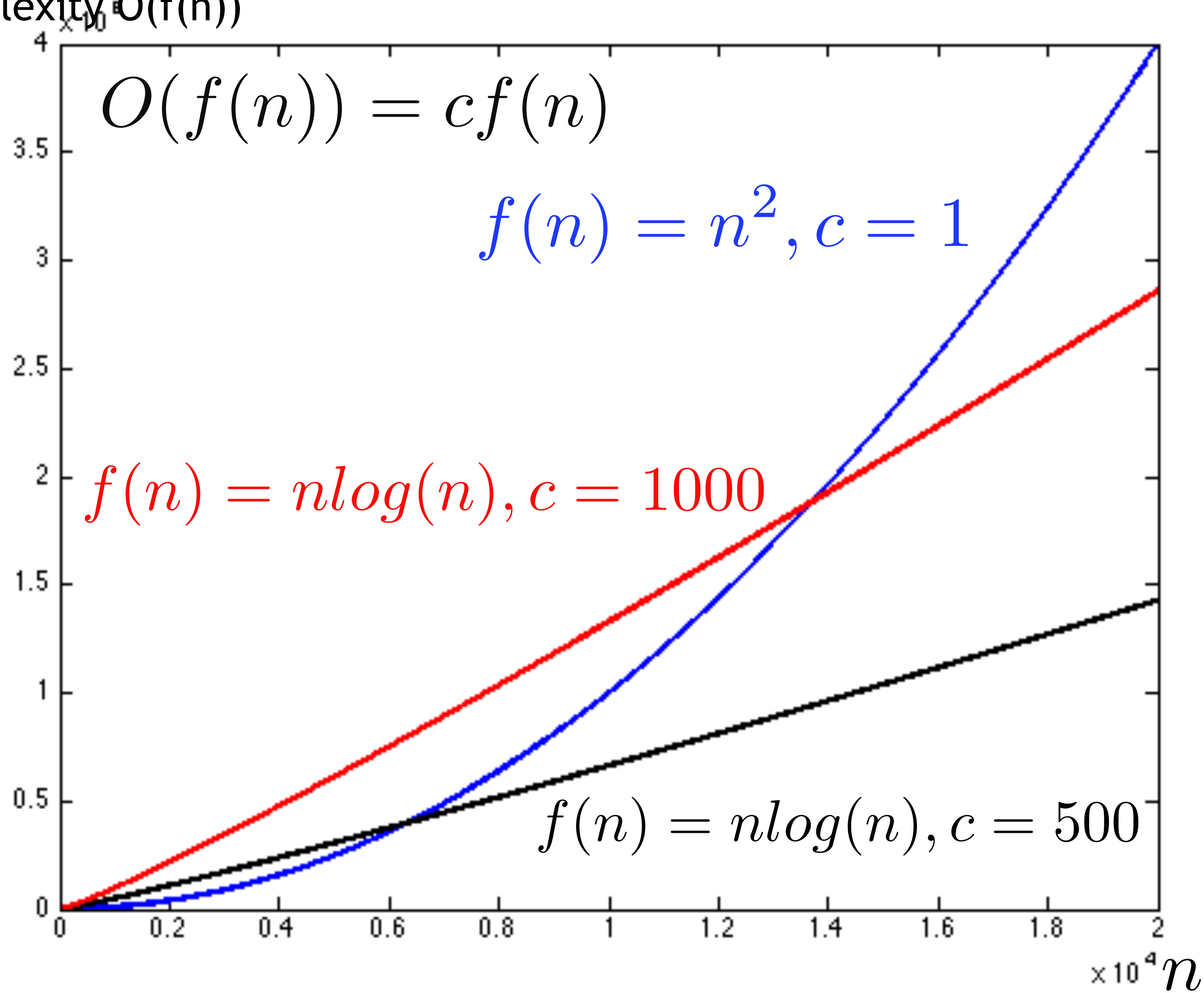
- Big O Notation　　n: scale of the problem

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

$O(f(n)) = cf(n)$

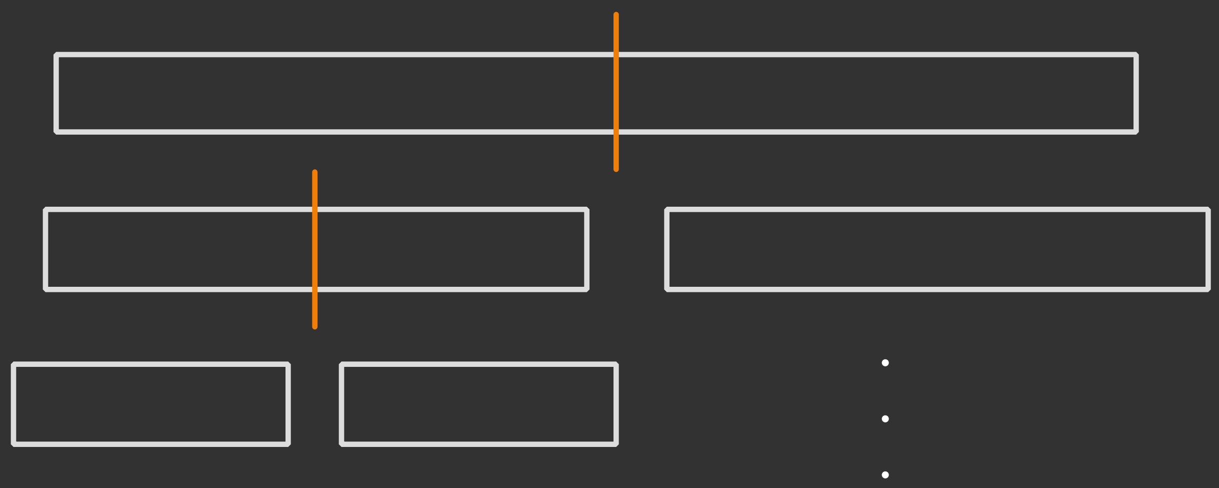

Complexity O(f(n))
$O(f(n)) = cf(n)$
$f(n) = n^2, c = 1$
$f(n) = nlog(n), c = 1$
$f(n) = n, c = 1$
$f(n) = log(n), c = 1$
25
20
15
10
5
0
1
1.5
2
2.5
3
3.5
4
4.5
5
n

Complexity O(f(n))



Divide and Conquer

- Divide a big problem into several small problems



Eg: Binary Search, Fast Fourier Transform…

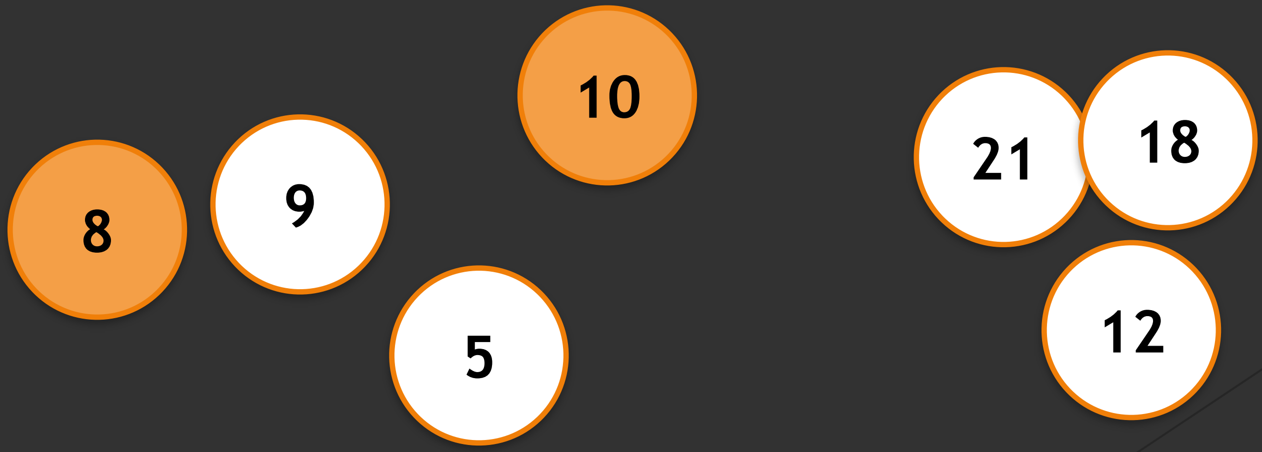
Outline

- Introduction
- QuickSort
- Complexity Analysis
- Optimization
- Application

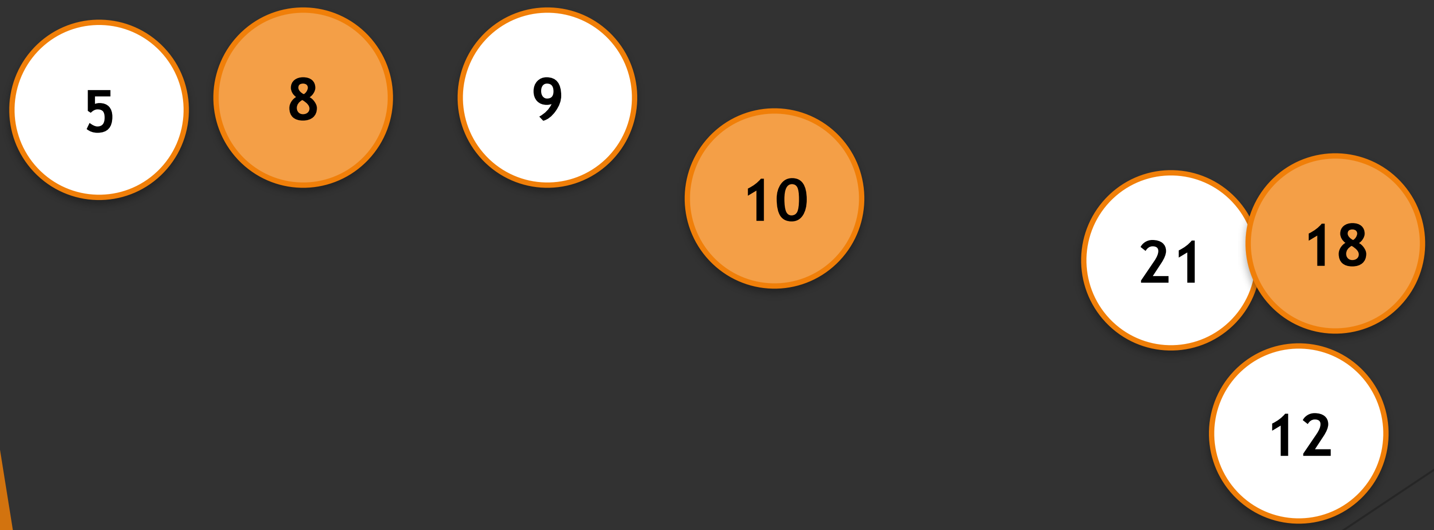
Quick Sort

Quick Sort

Quick Sort



Quick Sort



Quick Sort

- Step One: Select the first element, called pivot
- Step Two: Re-order the array so that all elements less than pivot come before the pivot, values greater than pivot come after it. **How to re-order the array?**

Values less than Pivot	Pivot	Values greater than Pivot
Sub-array A		Sub-array B

- Step Three: Apply the above steps over and over again on each sub-array until there is only one element in the sub-array.

Quick Sort Example

23	10	49	34	12	19	24

pivot

Select the first element as Pivot

Quick Sort Example

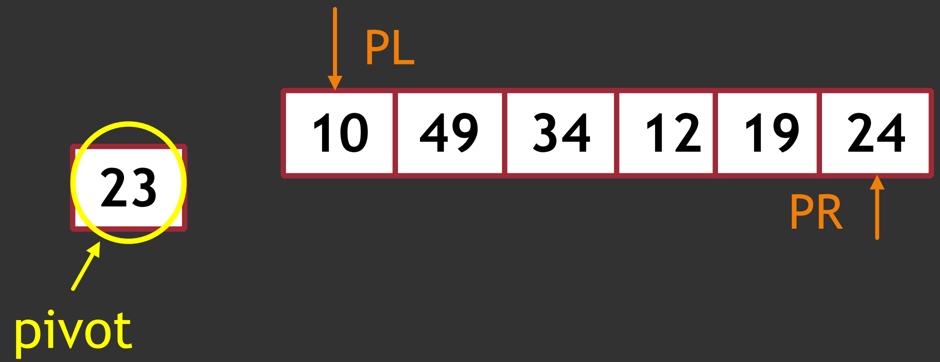
10	49	34	12	19	24

23

pivot

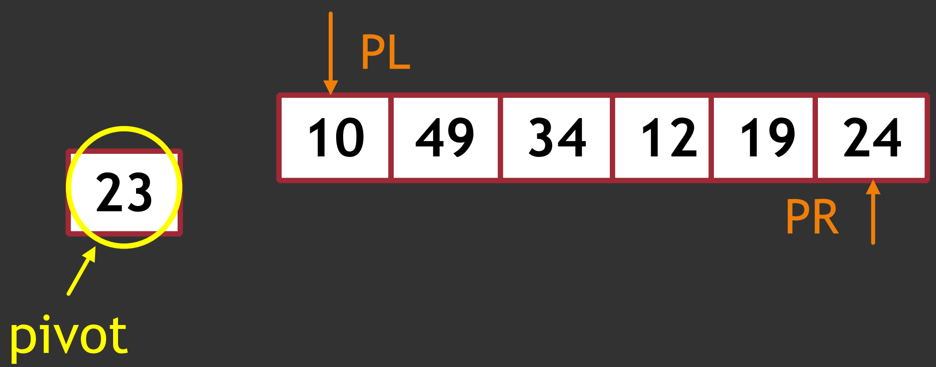
Select the first element as Pivot

Quick Sort Example



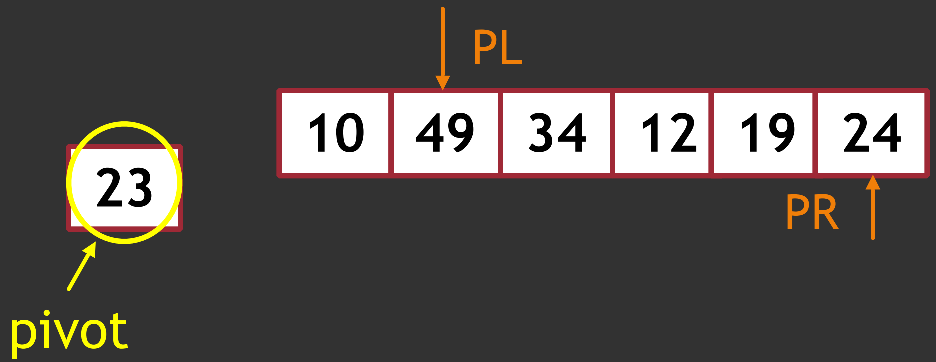
PL points to the element on the left;
PR points to the element on the right

Quick Sort Example



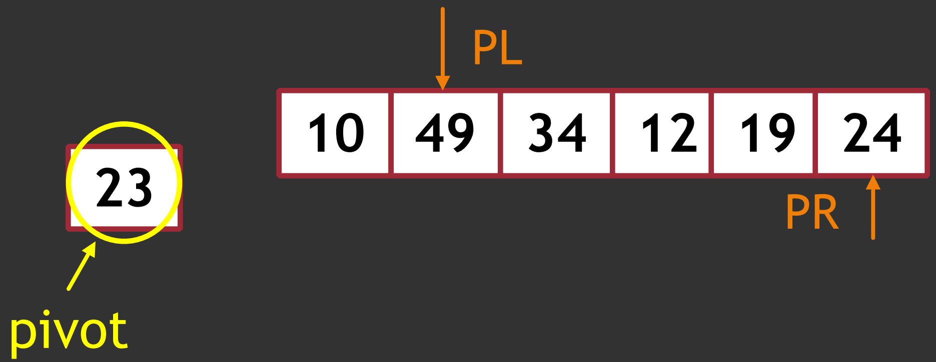
If the element in PL is less than pivot, pass it

Quick Sort Example



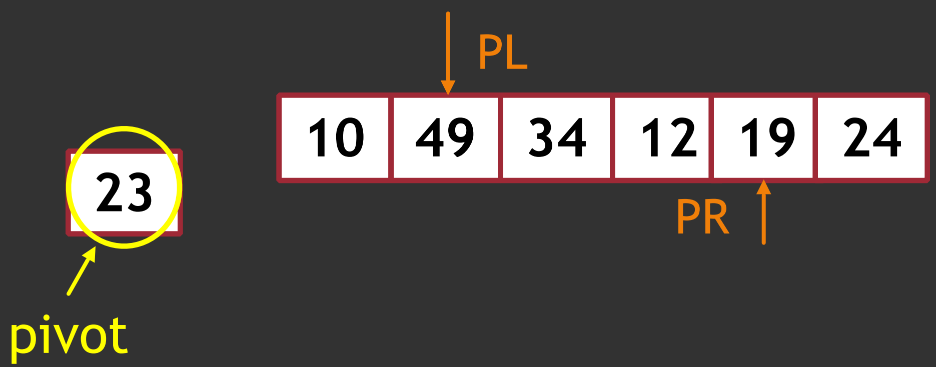
If the element in PL is greater than pivot, stop

Quick Sort Example



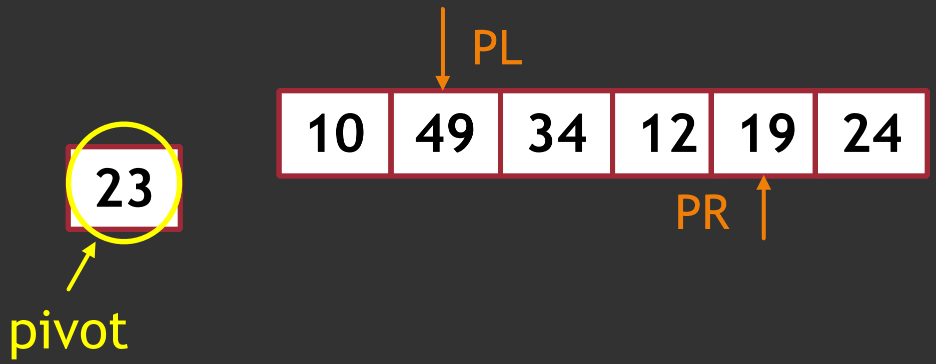
If the element in PR is greater than pivot, pass it

Quick Sort Example



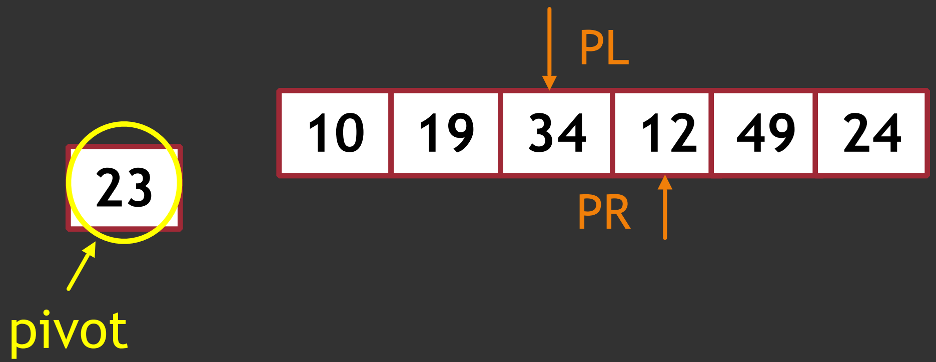
If the element in PR is less than pivot, stop

Quick Sort Example



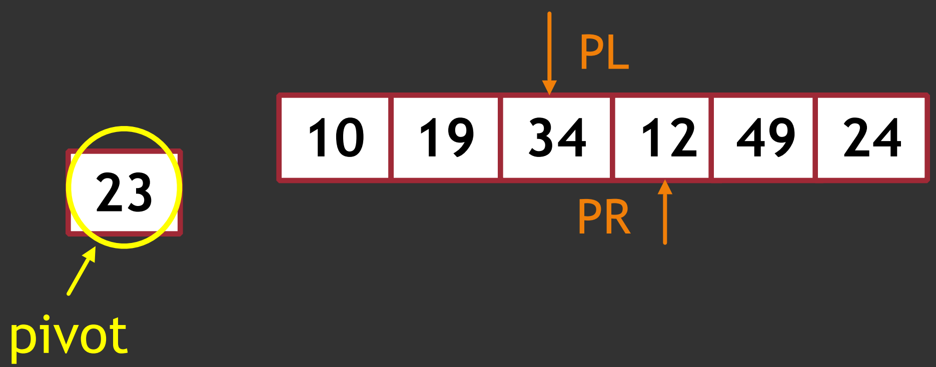
Swap

Quick Sort Example



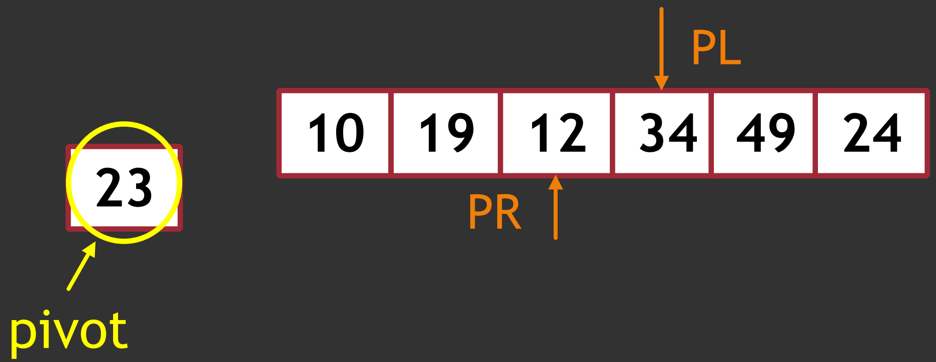
Continue Moving

Quick Sort Example



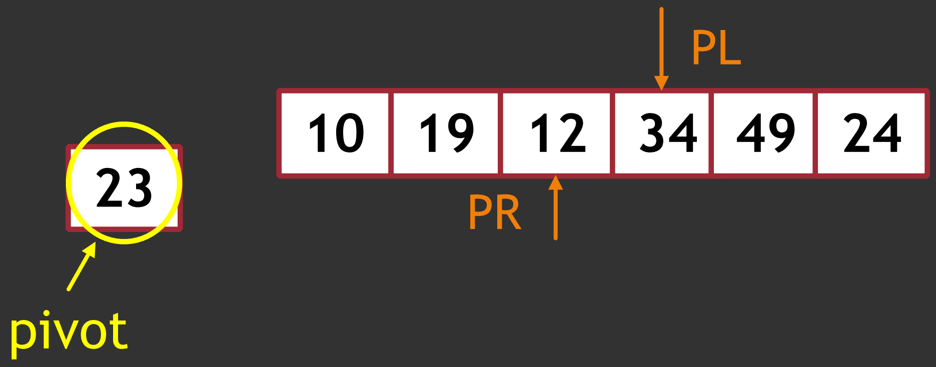
Swap

Quick Sort Example



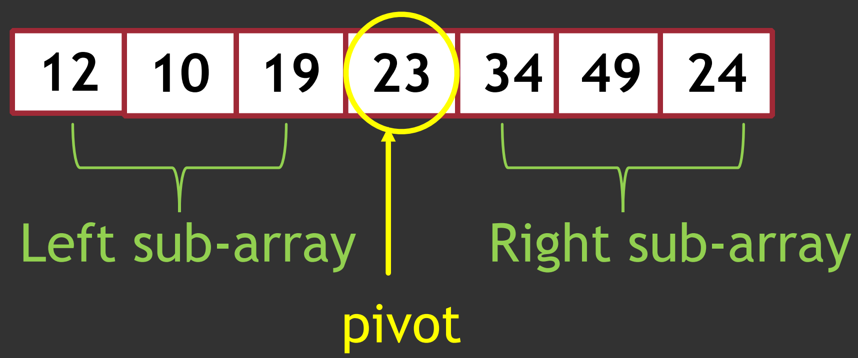
Continue Moving until PL is behind PR
Have scanned over the whole sequence

Quick Sort Example



swap pivot with PR

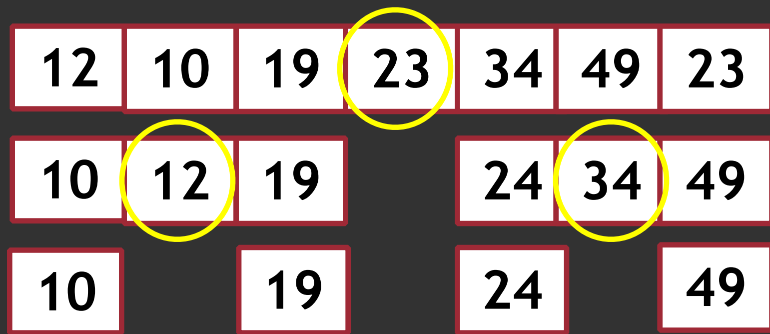
Quick Sort Example



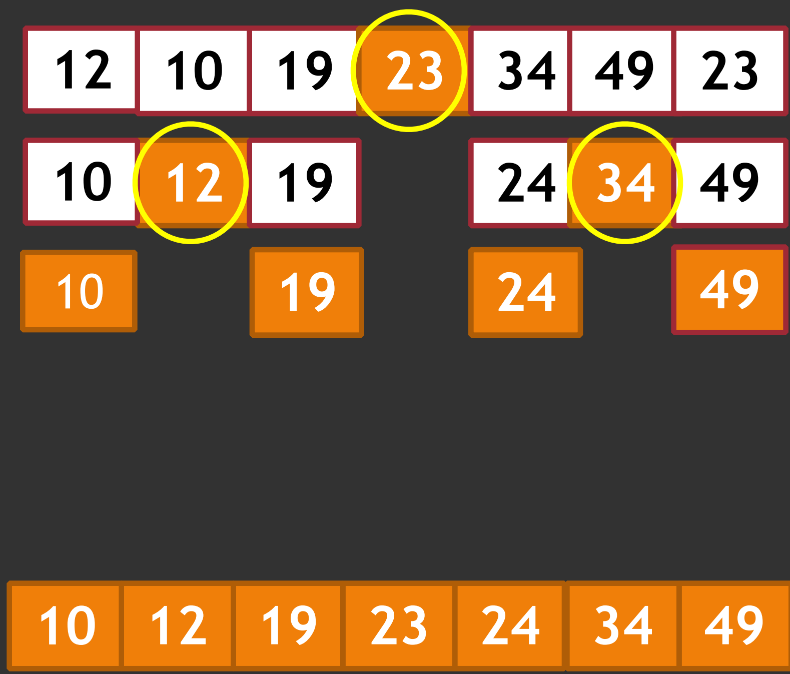
Quick Sort Example

12	10	19	23	34	49	24
10	12	19		24	34	49

Quick Sort Example



Quick Sort Example



Outline

- Introduction
- QuickSort
- Complexity Analysis
- Optimization
- Application

Complexity Analysis

- Time/Space Complexity
  - for best/worst/average cases

	Time	*Space*
Best		
Worst		
Average		

Best Case

length of the array: n=7

Level=1	12	10	19	23	34	49	23	n-1=6 comparisons
Level=2	10	12	19		24	34	49	n-3=2+2 comparisons
Level=3	10		19		24		49	0 comparison

Depth=3

Tree

10	12	19	23	24	34	49

6+(2+2)=10 comparisons

	Time	*Space*
Best		
Worst		
Average		

Best Case

length of the array: n=7

Level=1	12	10	19	23	34	49	23	n comparisons
Level=2	10	12	19		24	34	49	n comparisons
Level=3	10		19		24		49	n comparison

Depth=3 Minimum Depth = log(n) (See reference)

10	12	19	23	24	34	49

6+(2+2)=10 comparisons

	Time	Space
Best		
Worst		
Average		

Best Case

length of the array: n=7

Level=1 | 12 | 10 | 19 | 23 | 34 | 49 | 23 | n comparisons

Level=2 | 10 | 12 | 19 | | 24 | 34 | 49 | n comparisons

Level=3 | 10 | | 19 | | 24 | | 49 | n comparison

Depth=3 Minimum Depth = log(n) (See reference)

- Total comparisons = n * Depth = nlog(n)
- Memory usage = Depth = log(n)
- Time Complexity: O(nlog(n))
- Space Complexity: O(logn)

	Time	*Space*
Best		
Worst		
Average		

Best Case

length of the array: n=7

Level=1 | 12 | 10 | 19 | 23 | 34 | 49 | 23 | n comparisons

Level=2 | 10 | 12 | 19 | | 24 | 34 | 49 | n comparisons

Level=3 | 10 | | 19 | | 24 | | 49 | n comparison

Depth=3 Minimum Depth = log(n) (See reference)

- Total comparisons = n * Depth = nlog(n)
- Memory usage = Depth = log(n)
- Time Complexity: O(nlog(n))
- Space Complexity: O(logn)

	Time	*Space*
Best	O(nlogn)	O(logn)
Worst		
Average		

Worst Case

Maximize Depth

	Time	*Space*
Best	O(nlogn)	O(logn)
Worst		
Average		

Worst Case

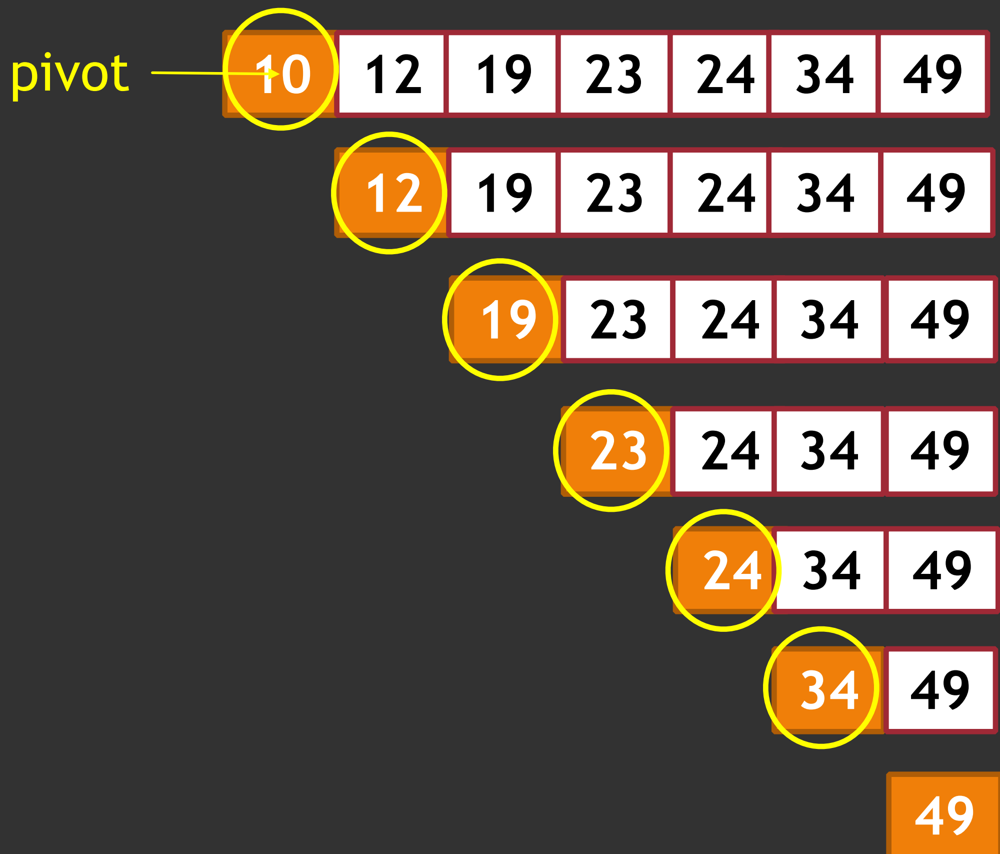
Maximize Depth

49	34	24	23	19	12	10

10	12	19	23	24	34	49

	Time	*Space*
Best	O(nlogn)	O(logn)
Worst		
Average		

Worst Case



	Time	*Space*
Best	O(nlogn)	O(logn)
Worst		
Average		

Worst Case

length of the array: n=7

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

Level=1	10 12 19 23 24 34 49	6 comparisons
Level=2	12 19 23 24 34 49	5 comparisons
Level=3	19 23 24 34 49	4 comparisons
Level=4	23 24 34 49	3 comparisons
Level=5	24 34 49	2 comparisons
Level=6	34 49	1 comparison
Level=7	49	

Depth=n=7

	Time	*Space*
Best	O(nlogn)	O(logn)
Worst		
Average		

Worst Case

length of the array: n=7

$$(n-1)+(n-2)+...+1=O(n^2)$$

Level=1	10	12	19	23	24	34	49	6 comparisons
Level=2		12	19	23	24	34	49	5 comparisons
Level=3			19	23	24	34	49	4 comparisons
Level=4				23	24	34	49	3 comparisons
Level=5					24	34	49	2 comparisons
Level=6						34	49	1 comparison
Level=7							49	

Depth=n=7

- Time Complexity= n * Depth = O(n*n)
- Space Complexity= Depth = O(n)

	Time	Space
Best	O(nlogn)	O(logn)
Worst		
Average		

Worst Case

length of the array: n=7

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

Level=1	10	12	19	23	24	34	49	6 comparisons
Level=2		12	19	23	24	34	49	5 comparisons
Level=3			19	23	24	34	49	4 comparisons
Level=4				23	24	34	49	3 comparisons
Level=5					24	34	49	2 comparisons
Level=6						34	49	1 comparison
Level=7							49	

Depth=n=7

- Time Complexity= n * Depth = O(n*n)
- Space Complexity= Depth = O(n)

	Time	Space
Best	O(nlogn)	O(logn)
Worst	$O(n^2)$	O(n)
Average		

Average Case

Prob(i)

#comparisons at each level

$$T(n) = \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i)) + cn$$

Left sub-array

Right sub-array

i: pivot partition location

By Mathematical induction

$$T(n) = O(nlog(n))$$

	Time	*Space*
Best	O(nlogn)	O(logn)
Worst	$O(n^2)$	***O(n)***
Average		

Average Case

Prob(i)

#comparisons at each level

$$T(n) = \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i)) + cn$$

Left sub-array

Right sub-array

i: pivot partition location

By Mathematical induction

$$T(n) = O(nlog(n))$$

	Time	*Space*
Best	O(nlogn)	O(logn)
Worst	$O(n^2)$	***O(n)***
Average	O(nlogn)	O(logn)

Complexity Summary

	Time	***Space***
Best	$O(n\log n)$	$O(\log n)$
Worst	$O(n^2)$	$O(n)$
Average	$O(n\log n)$	$O(\log n)$

Sorting Algorithm Comparison

	Best	Average	Worst
Insertion Sort			
Selection Sort			
Bubble Sort			
Merge Sort			
Heap Sort			
Quick Sort			

Sorting Algorithm Comparison

	Best	Average	Worst
Insertion Sort	***n***		
Selection Sort	n^2		
Bubble Sort	***n***		
Merge Sort			
Heap Sort			
Quick Sort	***nlogn***		

Sorting Algorithm Comparison

	Best	Average	Worst
Insertion Sort	**n**		n^2
Selection Sort	n^2		n^2
Bubble Sort	**n**		n^2
Merge Sort			
Heap Sort			
Quick Sort	***nlogn***		n^2

Sorting Algorithm Comparison

	Best	Average	Worst
Insertion Sort	**n**	n^2	n^2
Selection Sort	n^2	n^2	n^2
Bubble Sort	**n**	n^2	n^2
Merge Sort			
Heap Sort			
Quick Sort	***nlogn***	***nlogn***	n^2

Sorting Algorithm Comparison

	Best	Average	Worst
Insertion Sort	*n*	n^2	n^2
Selection Sort	n^2	n^2	n^2
Bubble Sort	*n*	n^2	n^2
Merge Sort	*nlogn*	*nlogn*	*nlogn*
Heap Sort	*nlogn*	*nlogn*	*nlogn*
Quick Sort	*nlogn*	*nlogn*	n^2

*MergeSort and HeapSort are faster than QuickSort in the worst case.

Sorting Algorithm Comparison

	Best	Average	Worst
Insertion Sort	**n**	n^2	n^2
Selection Sort	n^2	n^2	n^2
Bubble Sort	**n**	n^2	n^2
Merge Sort	**nlogn**	**nlogn**	**nlogn**
Heap Sort	**nlogn**	**nlogn**	**nlogn**
Quick Sort	**nlogn**	**nlogn**	n^2

*MergeSort and HeapSort are faster than QuickSort in the worst case.

*Practically, QuickSort is still the fastest: constant factor on average case is small

$$O(f(n)) = cf(n)$$

Can it be even better?

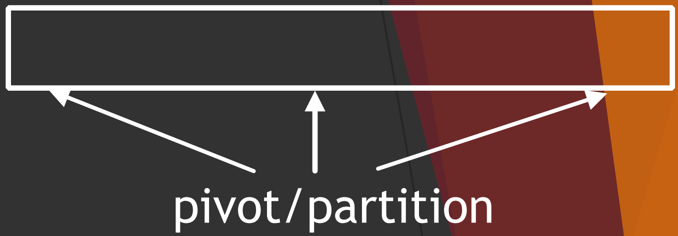
Outline

- Introduction
- QuickSort
- Complexity Analysis
- Optimization
- Application

Optimization

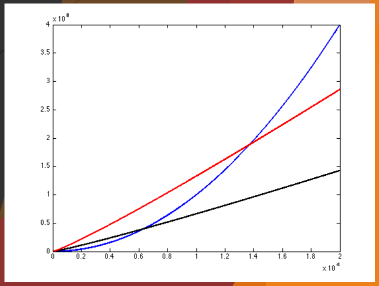
	Time	*Space*
Best	$O(n\log n)$	$O(\log n)$
Worst	$O(n^2)$	$O(n)$
Average	$O(n\log n)$	$O(\log n)$

Avoid the worst case



- Choose pivot randomly
- Choose pivot as a “median-of-three”
- Use insertion sort for small arrays
- Choose median of the array as pivot
  - always be O(nlog(n))
  - Need O(n) for finding the median
  - rarely used because of the large algorithm complexity and the very large constant factor on O(n)

Used in <stdlib> of C

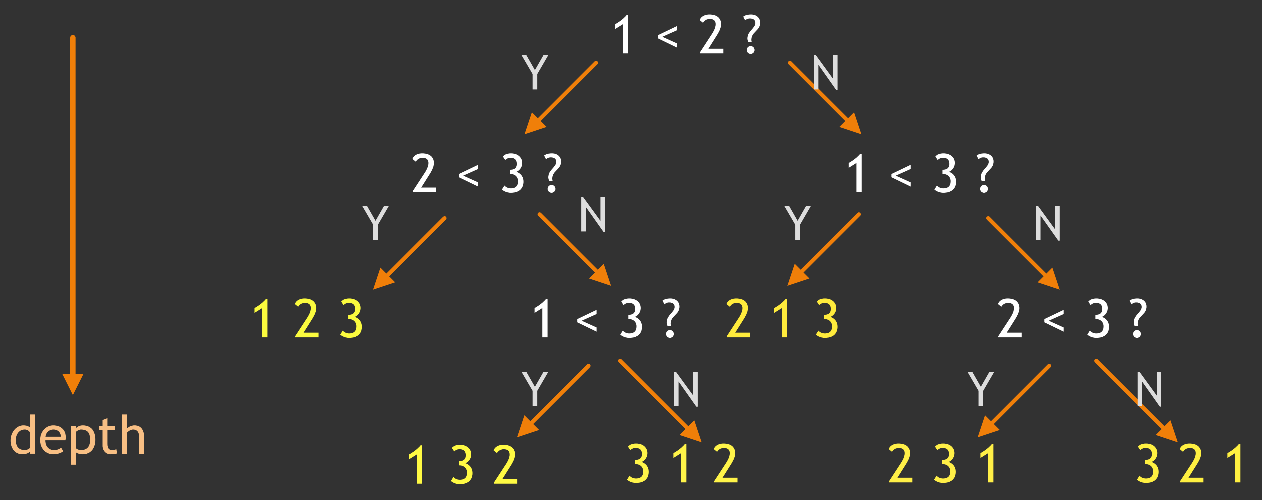


Can we come up with an even better solution?

- There is a lower bound in time complexity

Lower Bound in Time Complexity

- Think of sorting as a decision tree: (eg: {1,2,3})



- Each leaf is a possible order of {1,2,3}
- n numbers -> n! possibilities -> n! leaves
- Fast -> make less decisions -> less levels -> less depth
- minimum depth = log(n!) ~= O(nlogn)

Outline

- Introduction
- QuickSort
- Complexity Analysis
- Optimization
- Application

Application

- C standard library
  - http://repo.or.cz/w/glibc.git/blob/HEAD:/stdlib/qsort.c
  - median of three
  - Use insertion for small arrays
- Java: Dual-pivot quicksort

P1	<P1	P1<=X<=P2	>P2	P2

left L G right

part I part III part II

- The Kth largest number

	pivot	

t < K

Summary

- QuickSort is an divide-and-conquer based algorithm
- QuickSort can be summarized as two steps:
  - Select and place the pivot in its correct place
  - Recursively sort both sides of the pivot
- Optimization and Application

	Time	Space
Best	$O(n\log n)$	$O(\log n)$
Worst	$O(n^2)$	$O(n)$
Averag	$O(n\log n)$	$O(\log n)$

Interesting Visualization

- https://www.youtube.com/watch?v=kPRA0W1kECg

Reference

- Proof for the depth of a binary tree: http://cs.stackexchange.com/questions/6161/what-is-the-depth-of-a-complete-binary-tree-with-n-nodes
- Proof for the best case: http://www.cs.princeton.edu/courses/archive/spr03/cs226/lectures/analysis.4up.pdf
- Discussion of Multi-Pivot QuickSort: http://cs.stanford.edu/~rishig/courses/ref/l11a.pdf
- Proof for average case of QuickSort: http://en.wikipedia.org/wiki/Quicksort

Thanks!