Sorting Algorithms & Convex Hull

Elias MOONS

April 15, 2015

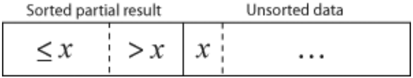
Table of Contents

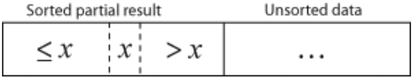
Table of Contents

Table of Contents

Insertion Sort

Idea

- each iteration, pick one element from the unsorted part of the data and put it in the right place in the sorted part of the data.
- 

- becomes
- 

Insertion Sort

6	5	3	1	8	7	2	4

Insertion Sort

6	5	3	1	8	7	2	4

6	5	3	1	8	7	2	4

Insertion Sort

6	5	3	1	8	7	2	4

5	6	3	1	8	7	2	4

5	6	3	1	8	7	2	4

Insertion Sort

5	6	3	1	8	7	2	4

5	3	6	1	8	7	2	4

3	5	6	1	8	7	2	4

3	5	6	1	8	7	2	4

Insertion Sort

3	5	6	1	8	7	2	4

3	5	1	6	8	7	2	4

3	1	5	6	8	7	2	4

1	3	5	6	8	7	2	4

1	3	5	6	8	7	2	4

Insertion Sort

1	3	5	6	8	7	2	4

1	3	5	6	8	7	2	4

Insertion Sort

1	3	5	6	8	7	2	4

1	3	5	6	7	8	2	4

1	3	5	6	7	8	2	4

Insertion Sort

1	3	5	6	7	8	2	4
1	3	5	6	7	2	8	4
1	3	5	6	2	7	8	4
1	3	5	2	6	7	8	4
1	3	2	5	6	7	8	4
1	2	3	5	6	7	8	4
1	2	3	5	6	7	8	4

Insertion Sort

1	2	3	5	6	7	8	4

1	2	3	5	6	7	4	8

1	2	3	5	6	4	7	8

1	2	3	5	4	6	7	8

1	2	3	4	5	6	7	8

1	2	3	4	5	6	7	8

Insertion Sort

Complexity

Average	Best	Worst	Memory
$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$

Table of Contents

Merge Sort

Idea

- recursively divide the unsorted list into sublists until all the sublists have length 1
- repeatedly merge sublists to produce new sorted sublists until there is only 1 list remaining
- this will be the sorted list

Merge Sort

6	5	3	1	8	7	2	4

6	5	3	1

8	7	2	4

6	5

3	1

8	7	2	4

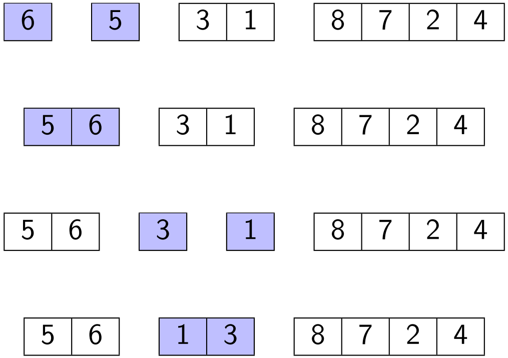
6

5

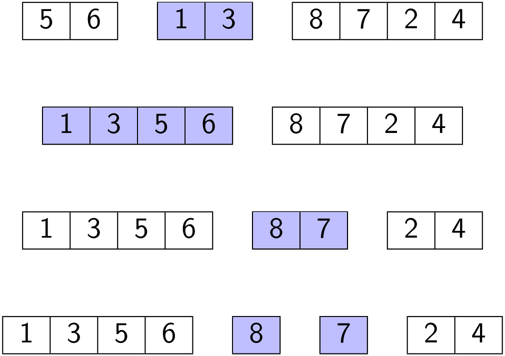
3	1

8	7	2	4

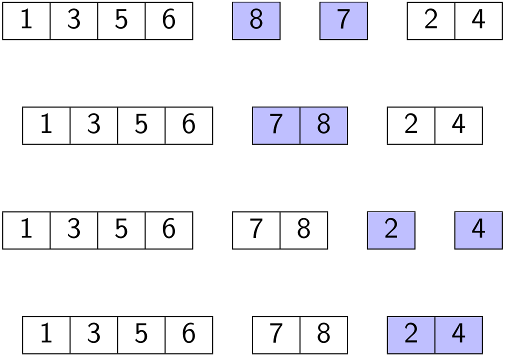
Merge Sort



Merge Sort



Merge Sort



Merge Sort

1	3	5	6		7	8		2	4

1	3	5	6		2	4	7	8

1	2	3	4	5	6	7	8

Merge Sort

Complexity

Average	Best	Worst	Memory
O(n log(n))	O(n log(n))	O(n log(n))	O(n)

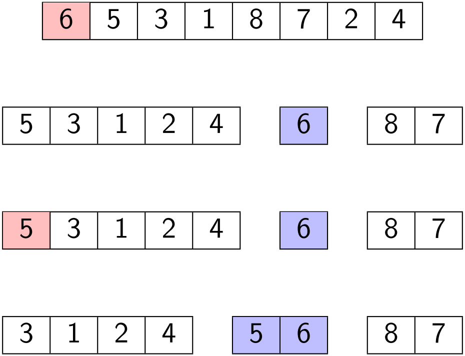
Table of Contents

Quicksort

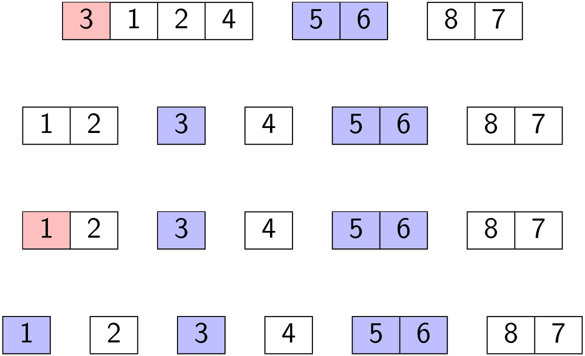
Idea

- pick an element from the array (the pivot)
- partitioning: reorder the array such that all elements smaller than the pivot come before the pivot and all the elements with a value greater than the pivot come after the pivot (so the pivot is already in its correct position)
- recursively apply these steps to the subarray of smaller elements and to the subarray of the bigger elements

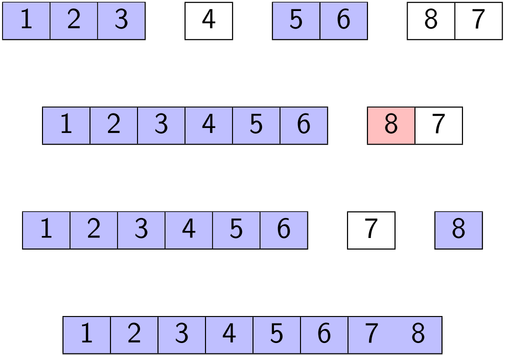
Quicksort



Quicksort



Quicksort



Quicksort

Complexity

Average	Best	Worst	Memory
$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$	$O(1)$

Table of Contents

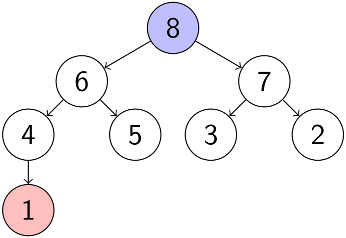
Heapsort

Idea

- put all (unsorted) elements in heap structure
- extract elements from the heap one by one (from high to low, or vice versa)
- is a variation of selection sort with extraction of elements in logarithmic rather than linear time

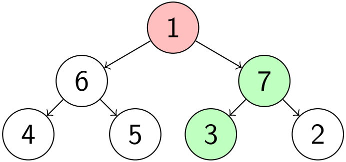
Heapsort

8	6	7	4	5	3	2	1



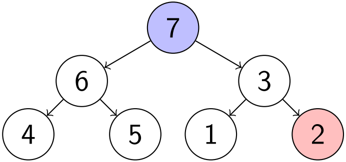
Heapsort

1	6	7	4	5	3	2	8



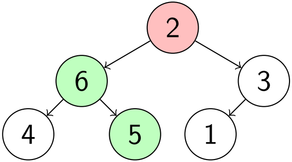
Heapsort

7	6	3	4	5	1	2	8



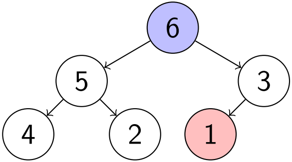
Heapsort

2	6	3	4	5	1	7	8



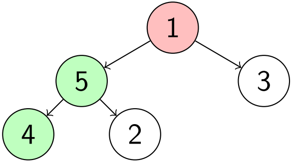
Heapsort

6	5	3	4	2	1	7	8



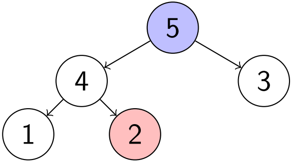
Heapsort

1	5	3	4	2	6	7	8



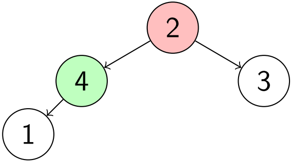
Heapsort

5	4	3	1	2	6	7	8



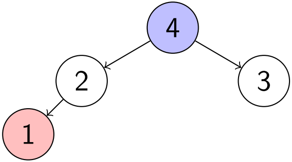
Heapsort

2	4	3	1	5	6	7	8



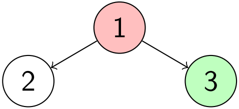
Heapsort

4	2	3	1	5	6	7	8



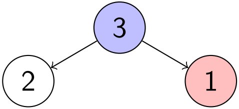
Heapsort

1	2	3	4	5	6	7	8



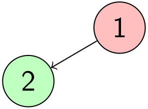
Heapsort

3	2	1	4	5	6	7	8



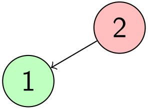
Heapsort

1	2	3	4	5	6	7	8



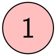
Heapsort

2	1	3	4	5	6	7	8



Heapsort

1	2	3	4	5	6	7	8



Heapsort

1	2	3	4	5	6	7	8

Heapsort

Complexity

Average	Best	Worst	Memory
O(n log(n))	O(n log(n))	O(n log(n))	O(1)

Table of Contents

Table of Contents

Convex Hull

Given a set of points in the plane, compute the smallest convex polygon in the plane that contains all the points.

Table of Contents

Graham Scan

Method of computing the convex hull of a finite set of points in the plane with time complexity O(n log(n)). The algorithm finds all vertices of the convex hull ordered along its boundary.

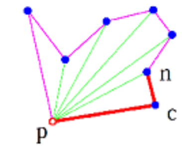
Graham Scan

Idea

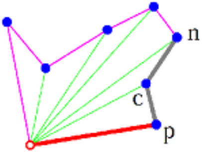
- find the point with the lowest y-coordinate. if there are multiple points with the lowest y-coordinate, the pick that one of them with the lowest x-coordinate. call this point P
- now sort all the points in increasing order of the angle they and point P make with the x-axis
- consider each point in the sorted array in sequence. For each point determine whether coming from the 2 previous points it makes a left of a right turn.
- if it makes a left turn, proceed with the next point
- if it makes a right turn, the second-to-last point is not part of the convex hull and should be removed from the convex hull, continue this removing for as long as the last 3 points make up a right turn

Graham Scan

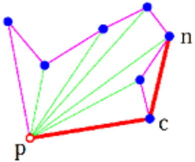
p: previous c: current n:next



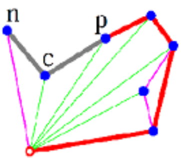
(p, c, n) is a left turn, accept c



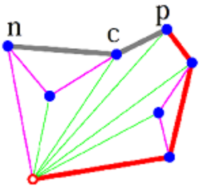
(p, c, n) is a right turn, reject c



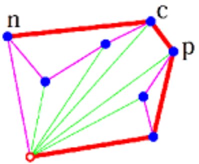
(p, c, n) is a left turn, accept c



(p, c, n) is a left turn, accept c



(p, c, n) is a right turn, reject c



(p, c, n) is a left turn, accept c

In the above algorithm and below code, a stack of points is used to store convex hull points. With reference to the code, p is next-to-top in stack, c is top of stack and n is points[i].

Graham Scan

Direction of the turn

To determine whether 3 points constitute a left or a right turn we do not have to compute the actual angles but we can use a cross product.
Consider the 3 points (x_1, y_1), (x_2, y_2) and (x_3, y_3), which we will call P_1, P_2 and P_3.
Now compute the z-component of the cross product of the vectors P_1P_2 and P_1P_3. Which is given by the expression $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$.
If the result is 0, the points are collinear. If the result is positive, the 3 points constitute a left turn. If the result is negative, the 3 points constitute a right turn.