

Time complexity

Sorting algorithms

Course: Object Oriented Programming (OOP)
CTU, FS, U12110
Matouš Cejnek

Contents

- Time complexity
- Insertion sort
- Selection sort
- Bubble sort
- Quick sort
- Merge sort
- Heap sort

Time complexity

Time complexity

The time complexity is the computational complexity that describes the amount of computer time it takes to run an algorithm.

Time complexity is commonly estimated by **counting the number of elementary operations** performed by the algorithm, supposing that **each elementary operation takes a fixed amount of time** to perform.

Time complexity

Since an algorithm's running time **may vary among different inputs of the same size**, one commonly considers the **worst-case time complexity** (exactly the case of sorting algorithms)

Big O notation is used as a tool to describe algorithm time complexity.

Time complexity

Big O notation:

- Only the most difficult expression is kept
- Multiplications are removed

Example:

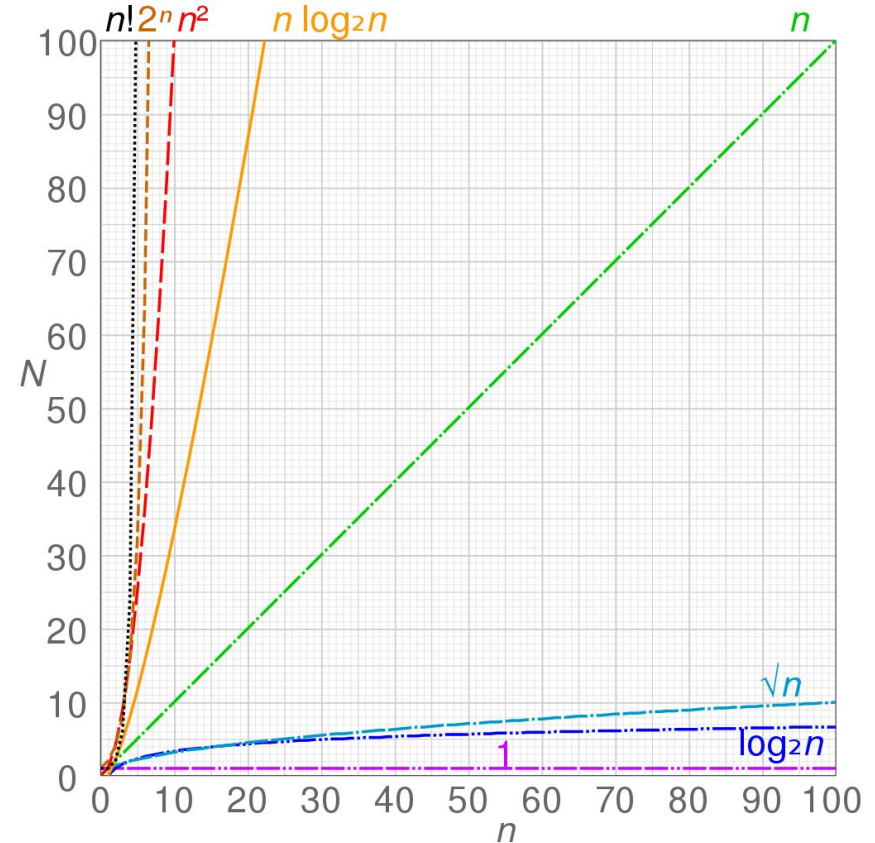
$2n + 3n!$ is just $O(n!)$

Name	Running time (T(n))
constant time	$O(1)$
logarithmic time	$O(\log n)$
linear time	$O(n)$
quadratic time	$O(n^2)$
cubic time	$O(n^3)$
factorial time	$O(n!)$



Time complexity

Number of operations N as
the result of input size n for
selected functions.



Space complexity

Space complexity

- The amount of memory or storage space that an algorithm or program uses in relation to the size of the input.
- Space complexity is crucial in situations where memory is limited or expensive.

Complexity classes

Complexity classes

- **polynomial time** - P
- **nondeterministic polynomial time** - NP
- zero-error probabilistic polynomial time - ZPP
- randomized polynomial time - RP
- bounded-error probabilistic polynomial time - BPP
- bounded-error quantum polynomial time - BQP

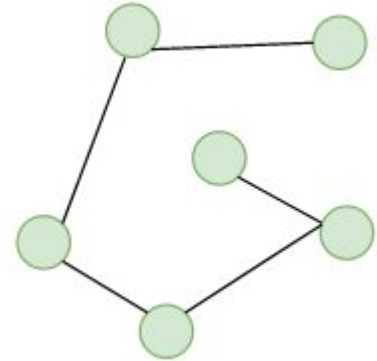
Popular complexity classes

- **P**: The complexity class of decision problems that can be solved on a deterministic Turing machine in polynomial time
- **NP**: The complexity class of decision problems that can be solved on a non-deterministic Turing machine in polynomial time

Polynomial time class

- A polynomial-time algorithm runs in an amount of time proportional to some polynomial value of input size.
- Example of P time problem is sorting of values.

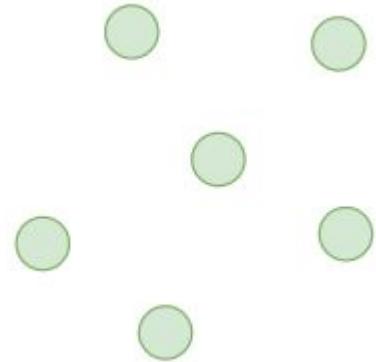
Example: measure the trajectory connecting the points



Non-polynomial class

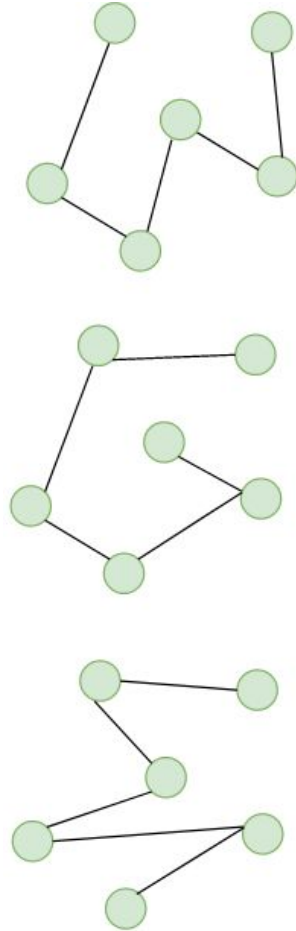
- A problem is called NP (nondeterministic polynomial) if its solution can be guessed and verified in polynomial time.
- Example problem - Traveling salesman

Example: find the shortest trajectory connecting all points



P vs NP problem

- The P versus NP problem is a major unsolved problem in theoretical computer science.
- it asks whether every problem whose solution can be quickly verified can also be quickly solved.
- P vs NP problem is one of the seven **Millennium Prize Problems**



Sorting algorithms

Sorting algorithms

- Sorting algorithm is an algorithm that puts elements of a list into an order.
- Sorting is fundamental task in computer science. It is required for many other tasks.
- Sorting algorithms use different approaches and possess different features.

Selection sort

Selection sort is a simple sorting algorithm that switch elements.

Initial state	44	55	13	43	90
	44	55	13	43	90
	13	55	44	43	90
	13	44	55	43	90
	13	43	55	44	90
sorted	13	43	44	55	90

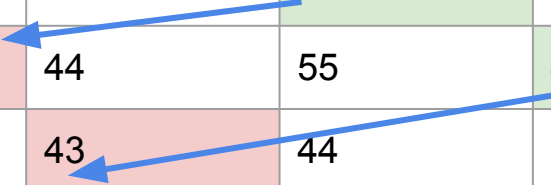
Selection sort features

- Number of comparisons: $\sum_{i=1}^{n-1} i = \frac{(n-1)+1}{2}(n-1) = \frac{1}{2}n(n-1) = \frac{1}{2}(n^2 - n)$
- Worst complexity is $O(n^2)$
- does not change the relative order of elements with equal keys (it is **stable**)
- only requires a constant amount of additional memory (it is **in-place**)

Insertion sort

Insertion sort is a simple sorting algorithm that builds the final sorted array

start	44	55	13	43	90
i = 2	44	55	13	43	90
i = 3	44	55	13	43	90
i = 4	13	44	55	43	90
i = 5	13	43	44	55	90



Insertion sort features

- Worst time complexity is $O(n^2)$. It is $O(kn)$ when each element in the input is no more than k places away from its sorted position.
- It does not change the relative order of elements with equal keys (it is **stable**).
- It is **in-place**.

Bubble sort

Bubble sort is swapping adjacent elements.

Initial state	44	55	13	43	90
First pass	44	55	13	43	90
	44	13	55	43	90
Second pass	44	13	43	55	90
	13	44	43	55	90
Third pass	13	43	44	55	90
Sorted	13	43	44	55	90

Two swaps during pass

One swap during pass

No swap, it is done

Bubble sort features

- It is $O(n^2)$
- does not change the relative order of elements with equal keys (it is **stable**)
- only requires a constant amount of additional memory (it is **in-place**)

Quick sort

- Quicksort was developed by British computer scientist Tony Hoare in 1959 and published in 1961.
- It is still a commonly used.
- Worst complexity is $O(n^2)$, average is $O(n \log n)$
- Memory complexity is $O(n \log n)$.

Quick sort

Quicksort is a divide-and-conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot.

Quick sort

Example

Creating lists of
smaller and bigger
numbers than pivot

6	8	3	8	4	3	2	3	7	4	7	9
3	4	3	2	3	4	6	8	8	7	7	9
3	4	3	2	3	4		8	8	7	7	9
3	2	3	3	4	4		8	7	7	8	9
3	2	3		4	4		8	7	7		9
2	3	3		4	4		7	7	8		
2	3						7	7			
2	3						7	7			
2	3	3	3	4	4	6	7	7	8	8	9

Quick sort

Example

Using only swaps
(comparison left
and right)

6	8	3	8	4	3	2	3	7	4	7	9
6	8	3	8	4	3	2	3	7	4	7	9
6	4	3	8	4	3	2	3	7	8	7	9
6	4	3	3	4	3	2	8	7	8	7	9
4	3	3	4	3	2	6	8	7	8	7	9
4	3	3	4	3	2		8	7	8	7	9
4	3	3	2	3	4		7	7	8	8	9
3	3	2	3	4	4		7	7	8		9
3	3	2	3		4		7	7	8		
2	3	3	3					7	8		
2	3		3					7	8		
2	2										
2	2	3	3	4	4	6	7	7	8	8	9

Quick sort

Pivot selections:

- Commonly is used the most left (or right value).
- Also pivot can be selected randomly.
- Better is to choose (guess) the median

Merge sort

- Merge sort is a divide-and-conquer algorithm that was invented by John von Neumann in 1945.
- Worst case time complexity of merge sort is $O(n \log n)$
- Memory complexity is $O(n)$.
- It is **stable**.

Merge sort

Idea behind:

1. Divide the unsorted list into n sublists, each containing one element (a list of one element is considered sorted).
2. Repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining. This will be the sorted list.

Merge sort

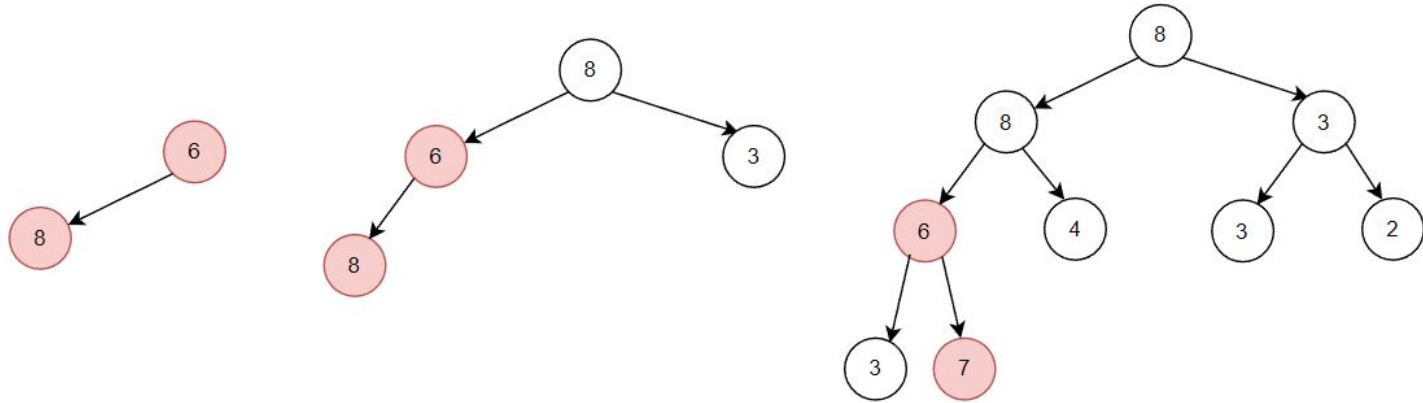
- Sorting is done during merging.



Heap sort

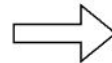
- Heap sort was invented by J. W. J. Williams in 1964 (same year as heap structure)
- Worst-case performance is $O(n \log n)$.
- It is **in-place**, but it is **not stable**.

Heap sort - example - step 1



Initial array

6	8	3	8	4	3	2	3	7	4	7	9
---	---	---	---	---	---	---	---	---	---	---	---



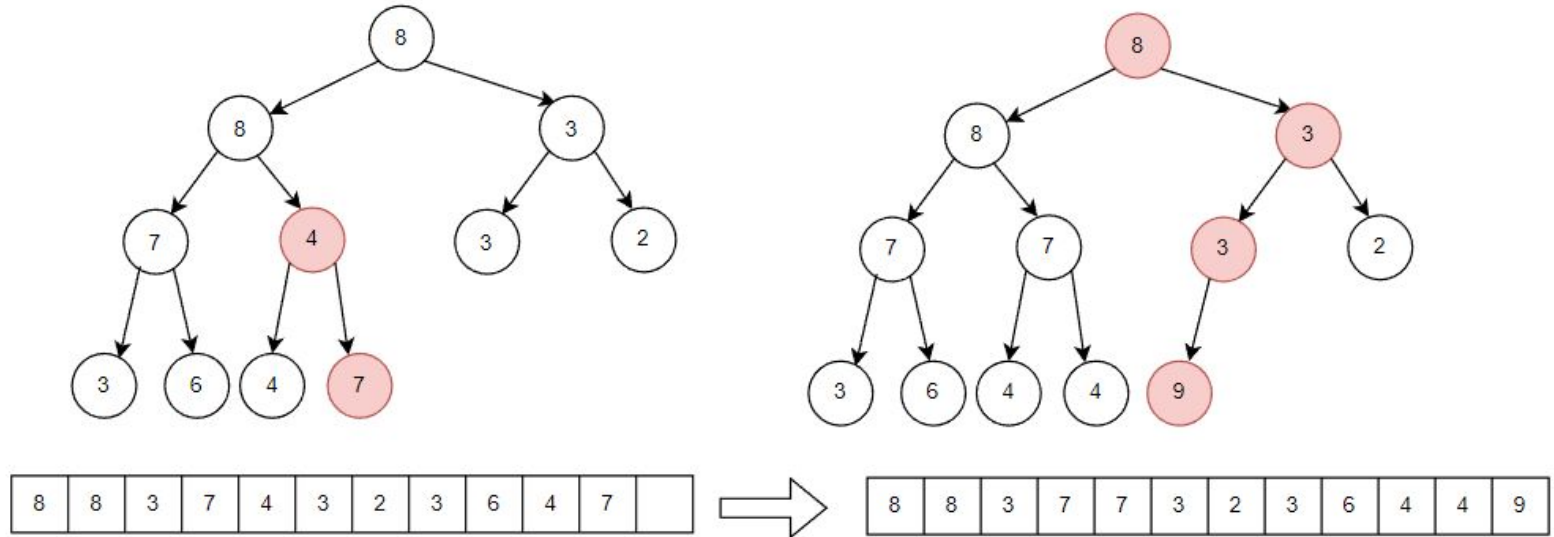
8	8	3	6	4	3	2	3	7			
---	---	---	---	---	---	---	---	---	--	--	--



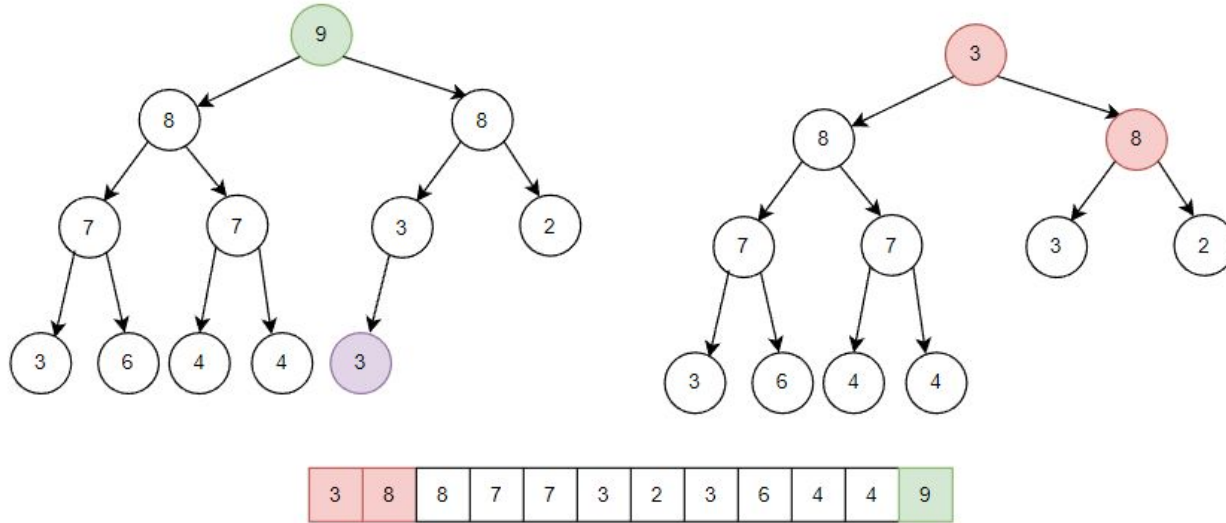
CTU

CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

Heap sort - example - step 2



Heap sort - example - step 3

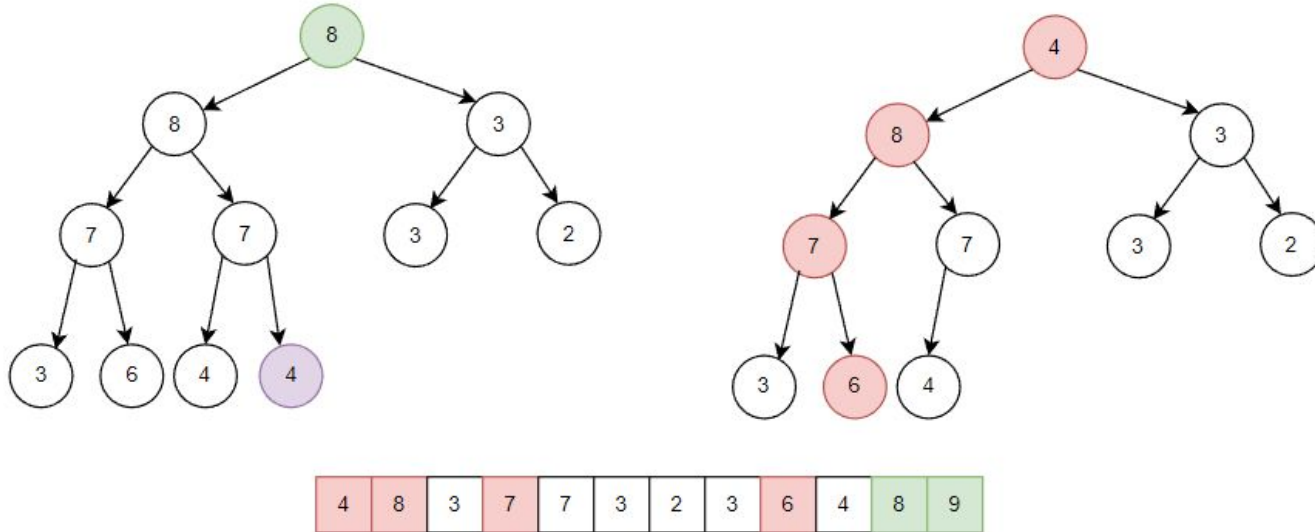




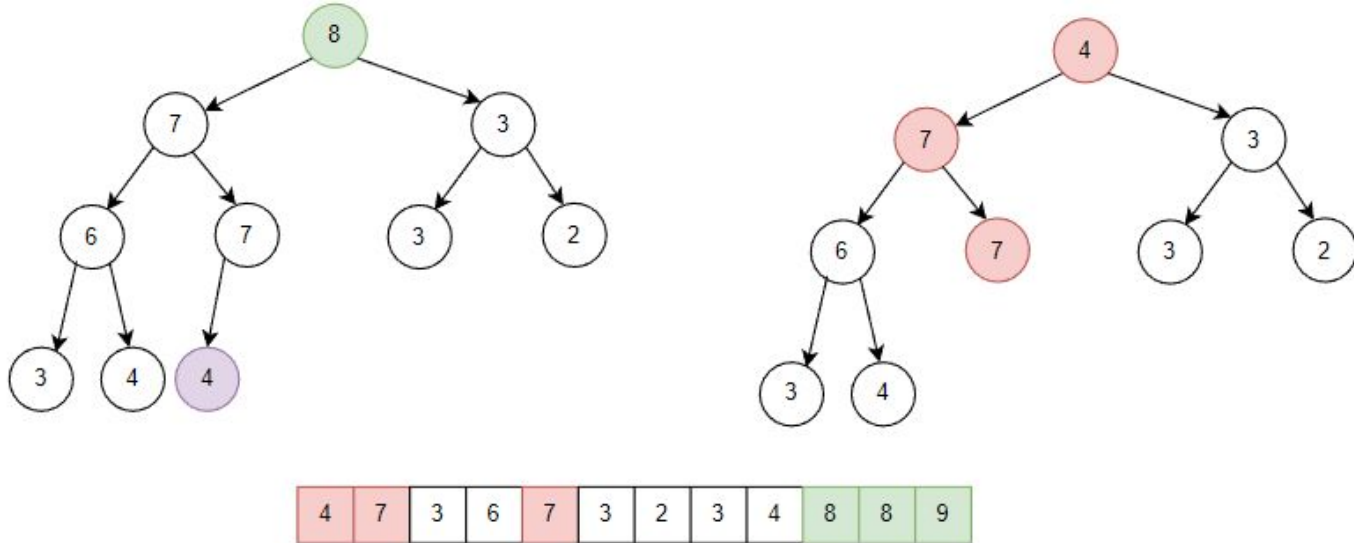
CTU

CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

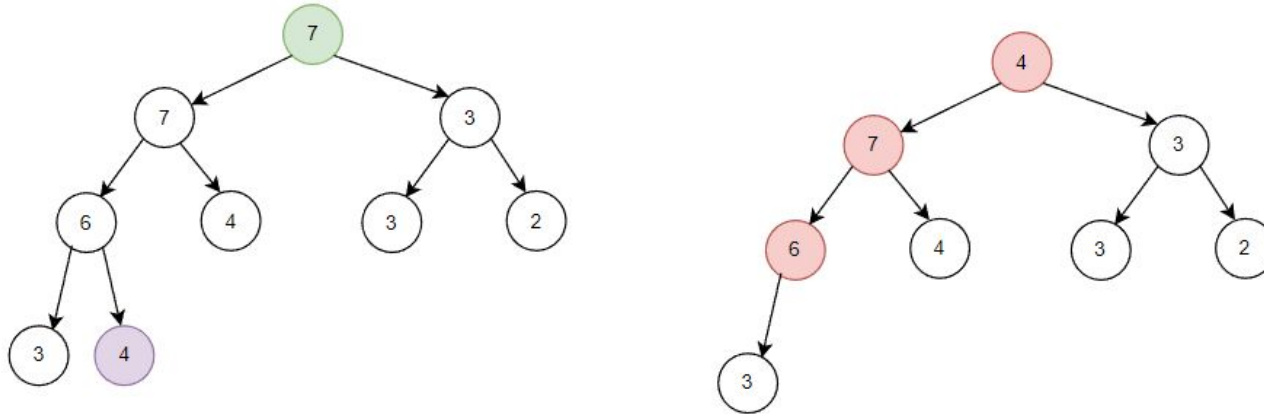
Heap sort - example - step 4



Heap sort - example - step 5

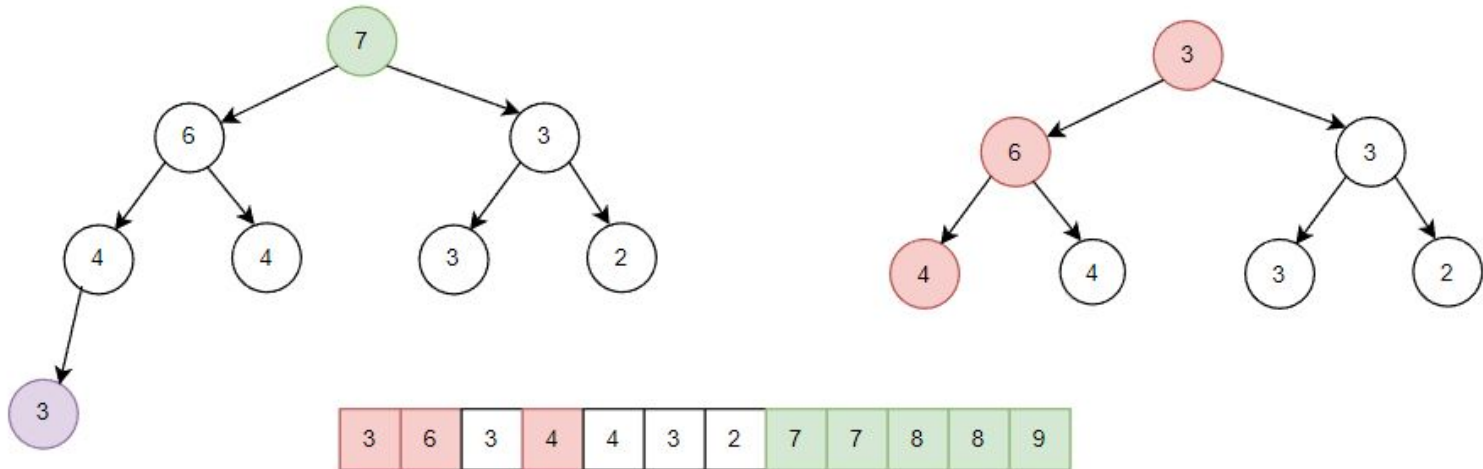


Heap sort - example - step 6

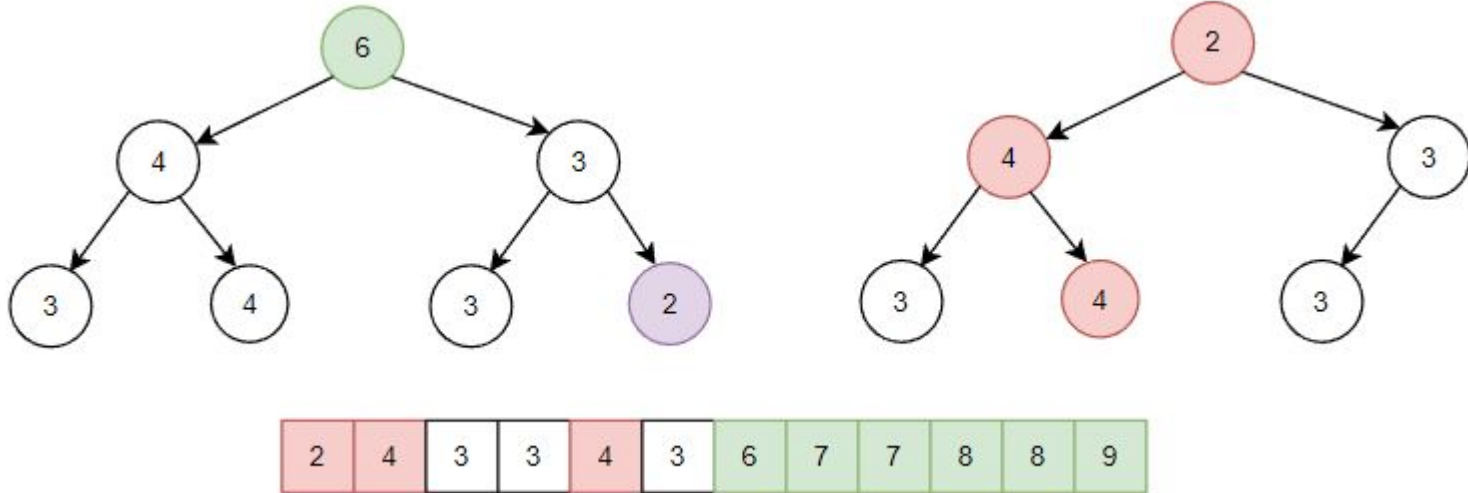


4	7	3	6	4	3	2	3	7	8	8	9
---	---	---	---	---	---	---	---	---	---	---	---

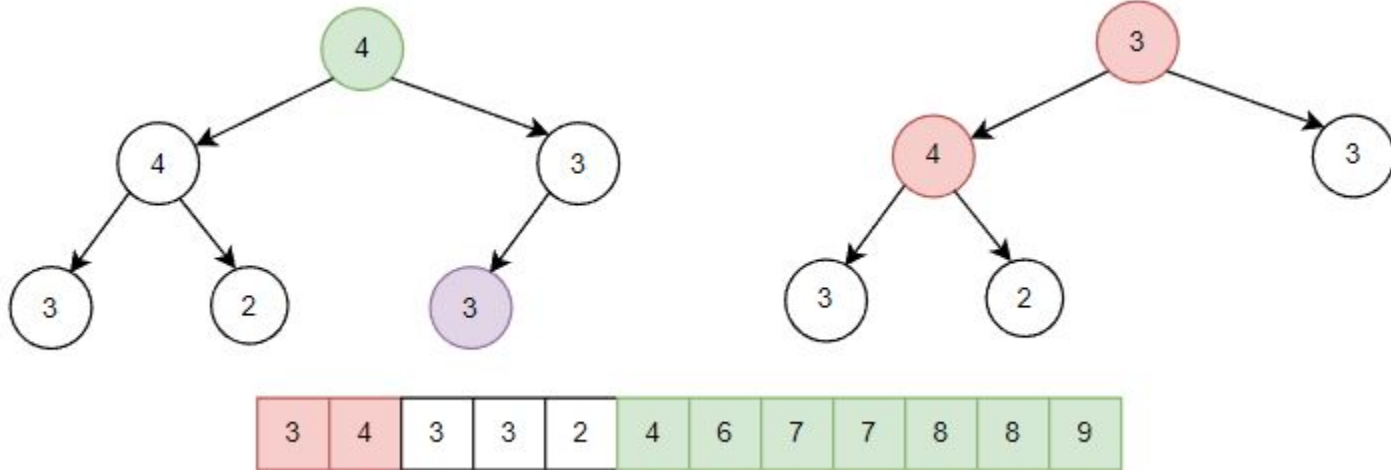
Heap sort - example - step 7



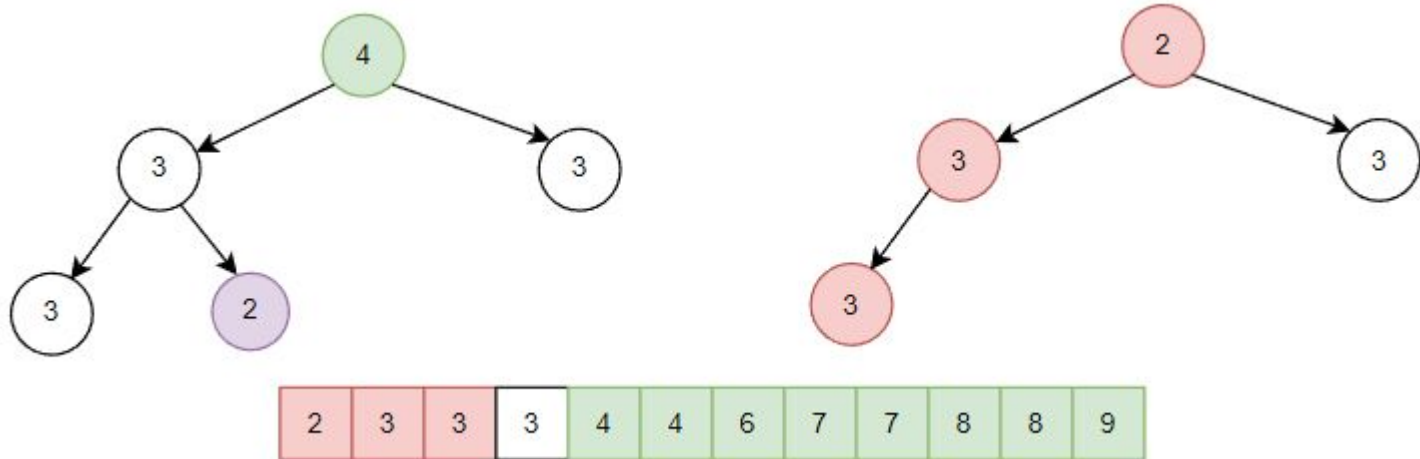
Heap sort - example - step 8



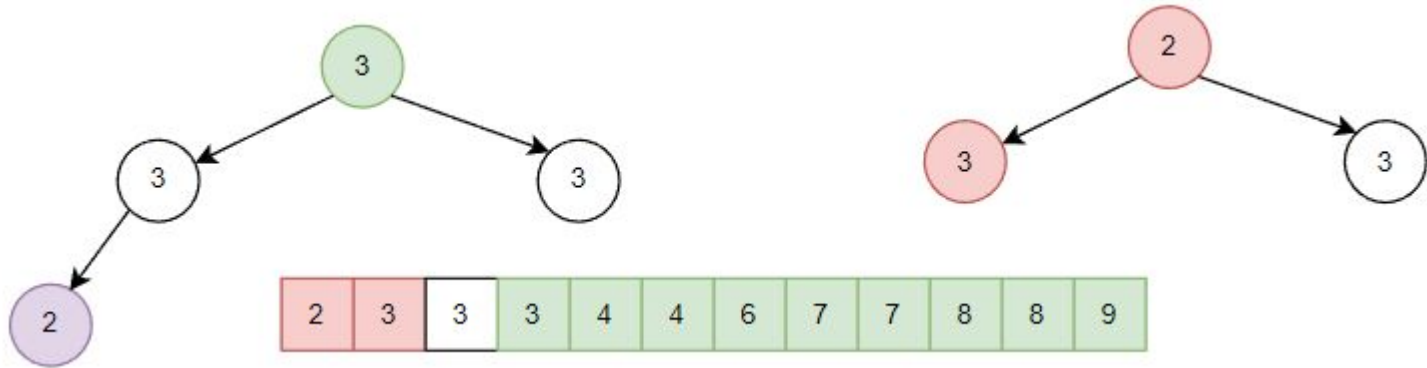
Heap sort - example - step 9



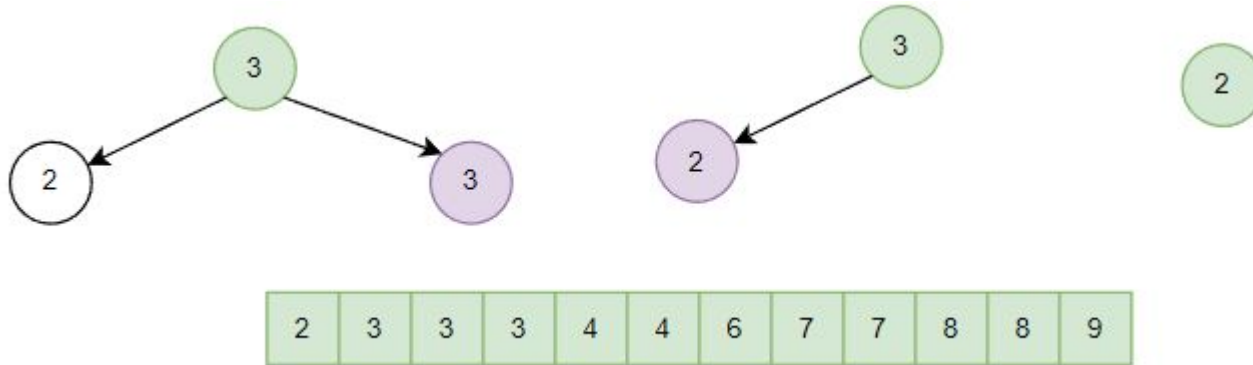
Heap sort - example - step 10



Heap sort - example - step 11



Heap sort - example - step 12



Bonus content

15 sorting in 6 minutes video

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

Bogosort

Also known as permutation sort, stupid sort, or slowsort.

The Bogosort generates permutations of its input until it finds one that is sorted.

Sleep sort

Sleep sort create different threads for each of the elements in the input array and then each thread sleeps for an amount of time which is proportional to the value of corresponding array element.

**Time complexity
of sleep sort is
hilarious**

