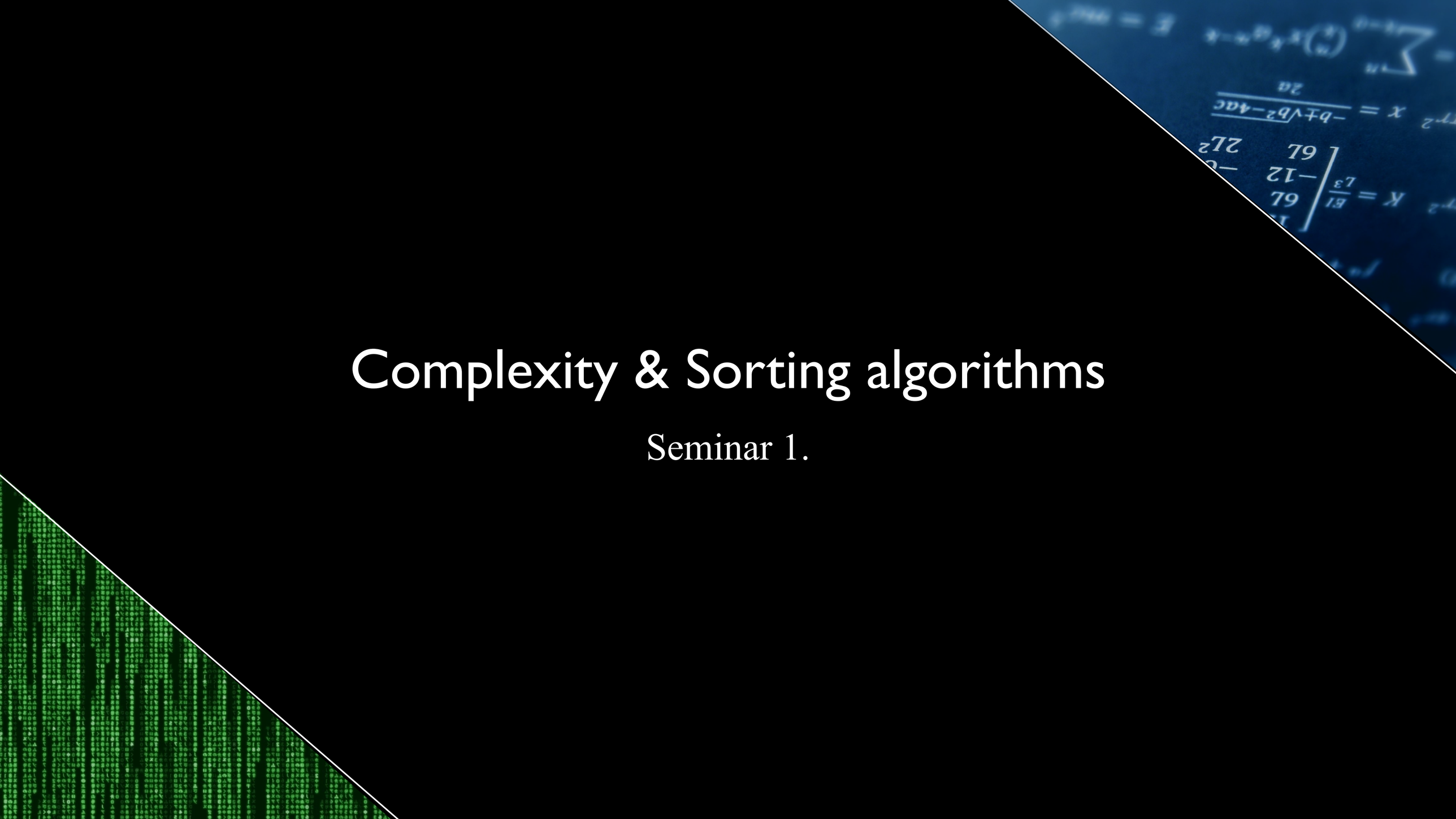


Complexity & Sorting algorithms

Seminar 1.


To-DoList

1. Complexity

- measure of how many operations algorithm will require to proceed an input of size N (or memory)

1. Complexity

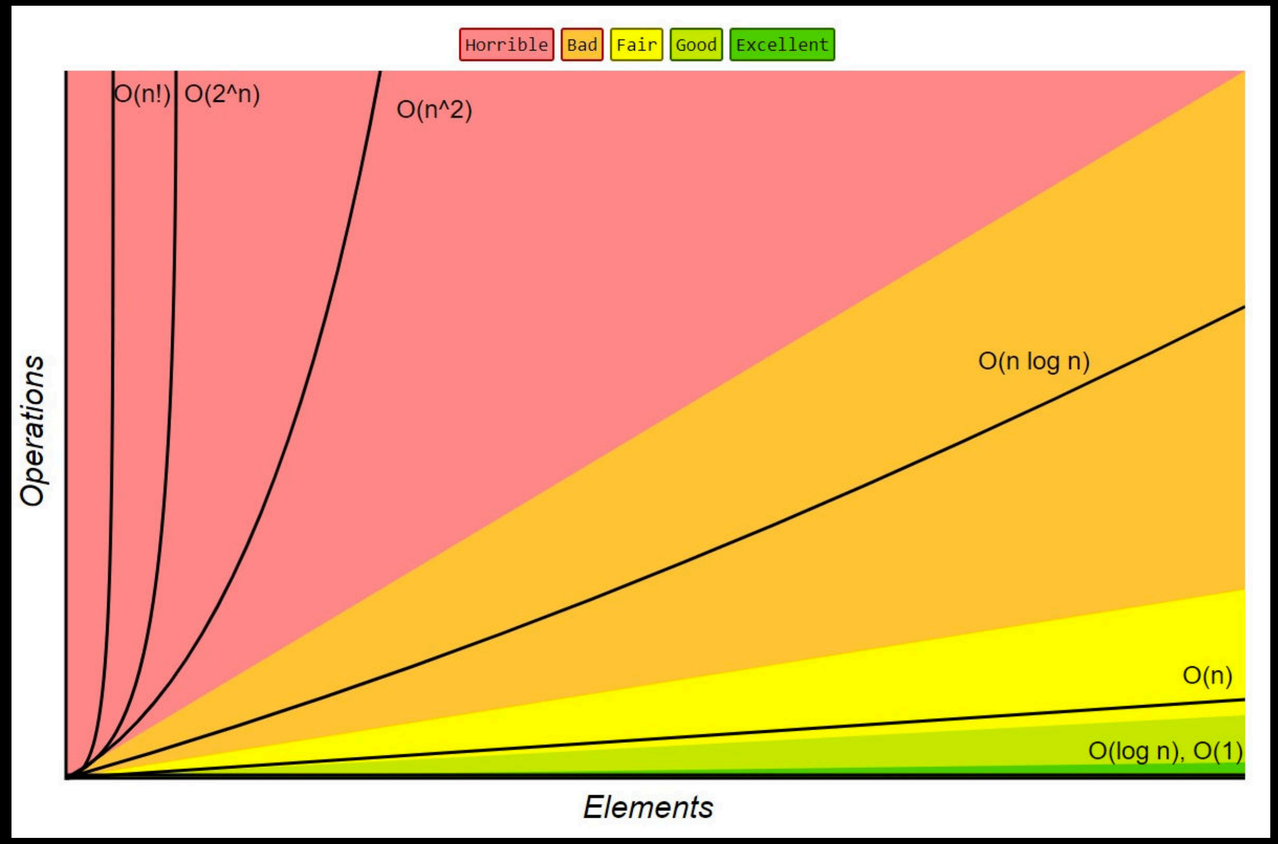
- measure of how many operations algorithm will require to proceed an input of size N (or time)

↓

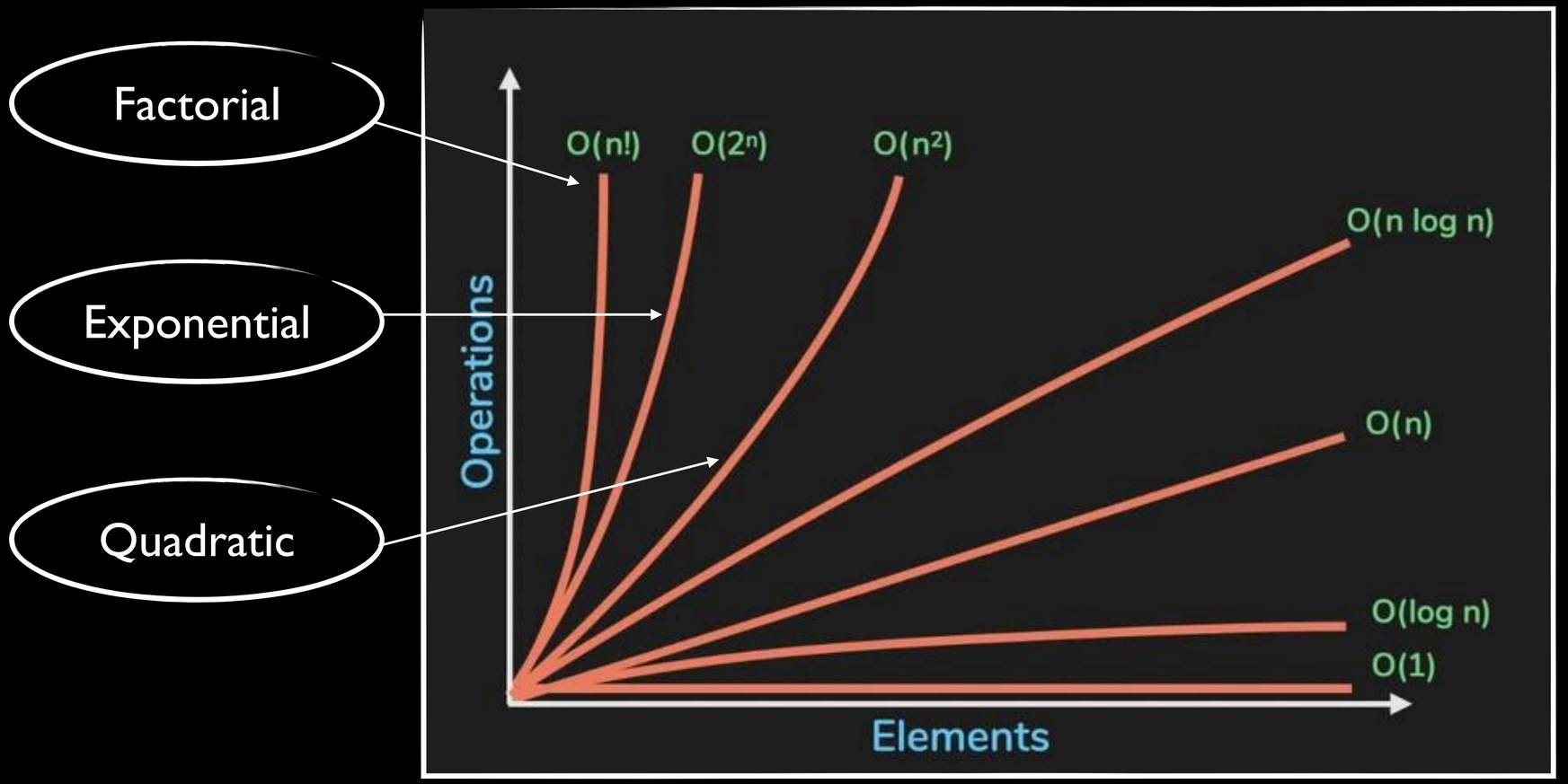
Big-Oh notation

- upper bound of the algorithm's computation time (worst-case)

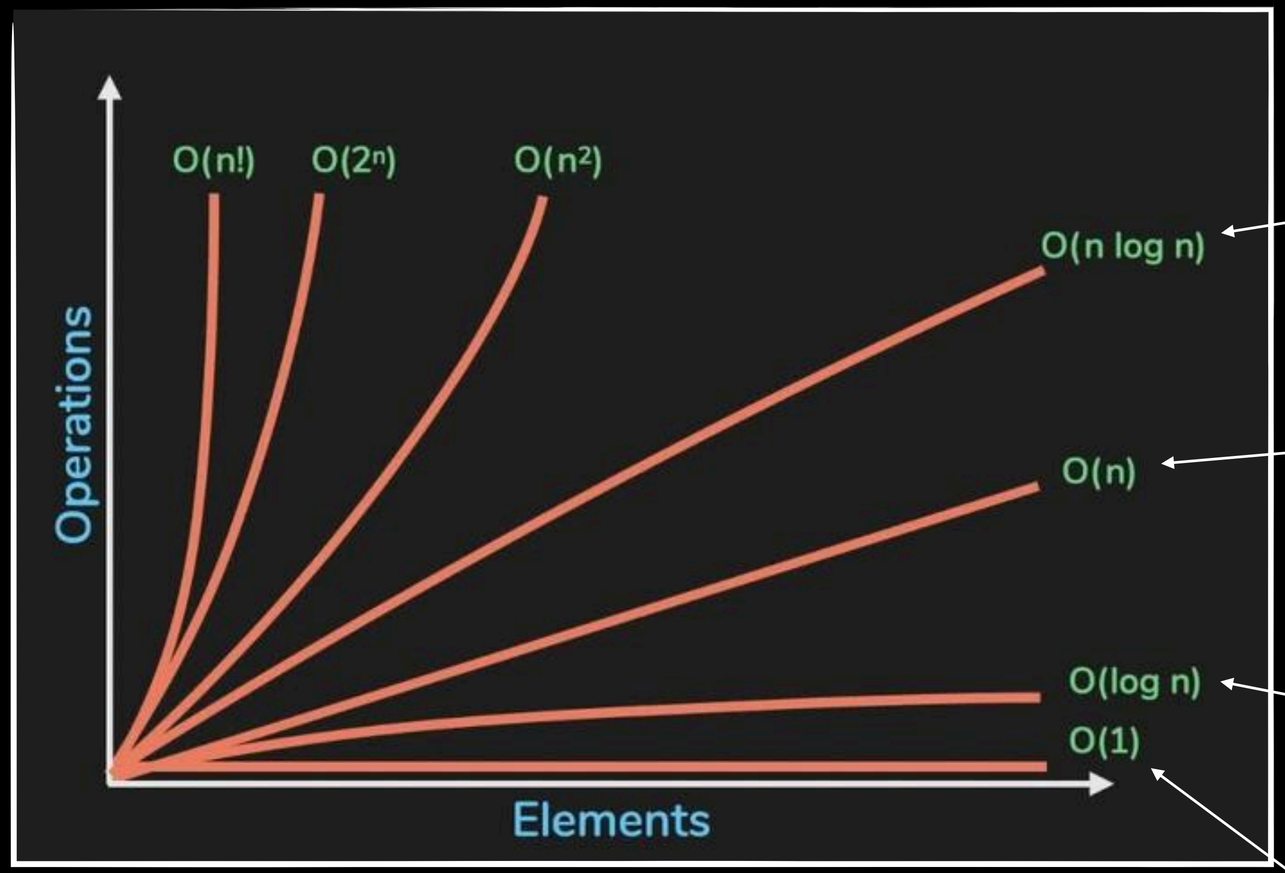
$$f(N) = O(g(N)) \iff f(N) \leq c \cdot g(N)\ \forall N$$



1. Complexity



1. Complexity



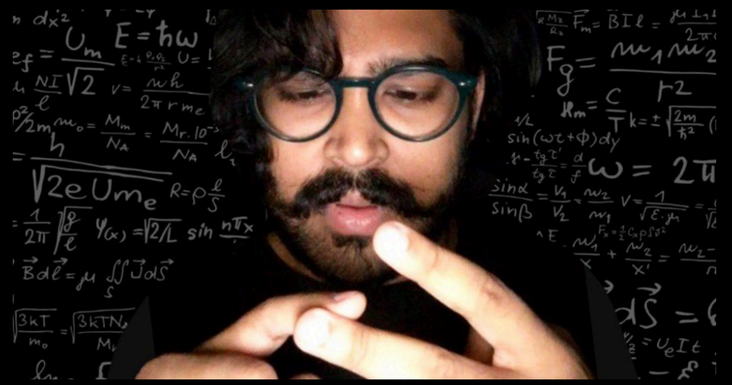
Log-linear

Linear

Logarithmic

Constant

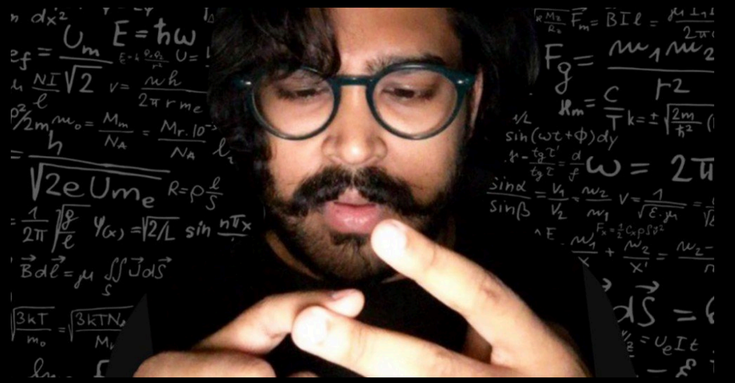
1. Complexity
Practical tasks



Complexity of discovering:

1. if value t is in array A of length N
2. if value t is in array A or array B, each of size N
3. if array A of length N contains duplicates
4. *if $f(n) = a_k n^k + \ldots + a_1 n + a_0$, then $f(n) = O(n^k)$
5. two matrix of size $N \times N$ multiplication
6. all subsets of array A of size N
7. break password of size N, consisting only of digits
8. empty recursion of depth N, called twice each time

1. Complexity
Practical tasks

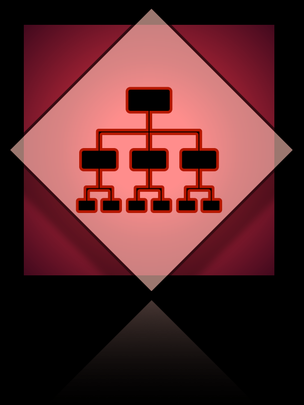


Complexity of discovering:

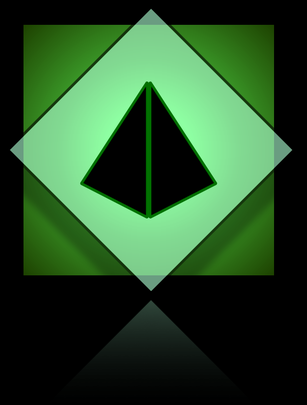
1. if value t is in array A of length N **O(n)**
2. if value t is in array A or array B, each of size N **O(n)**
3. if array A of length N contains duplicates **O(n^2)**
4. *if $f(n) = a_k n^k + \ldots + a_1 n + a_0$, then $f(n) = O(n^k)$
5. two matrix of size $N \times N$ multiplication **O(n^3)**
6. all subsets of array A of size N **O(2^n)**
7. break password of size N, consisting only of digits **O(10^n)**
8. empty recursion of depth N, called twice each time **O(2^n)**

2. Sorting algorithms

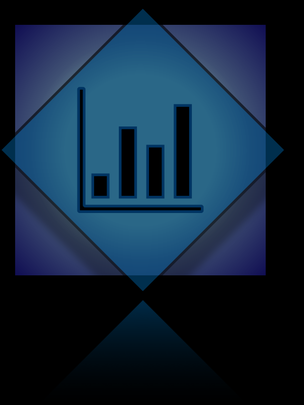
Insertion sort



Merge sort



Heap sort

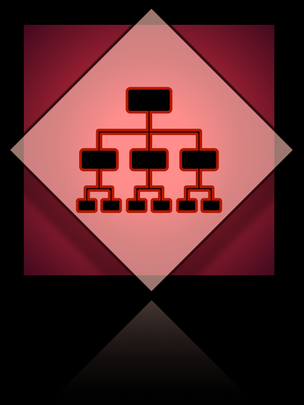


Count sort

2. Sorting algorithms

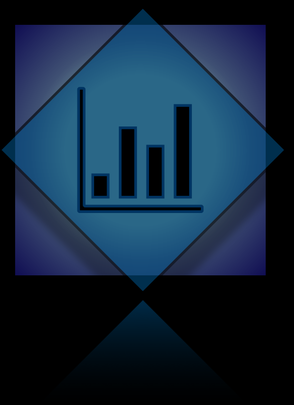
Insertion sort

- divide array into sorted & unsorted regions
- take next lowest-rank object
- find its proper position in sorted region



Merge sort

Heap sort

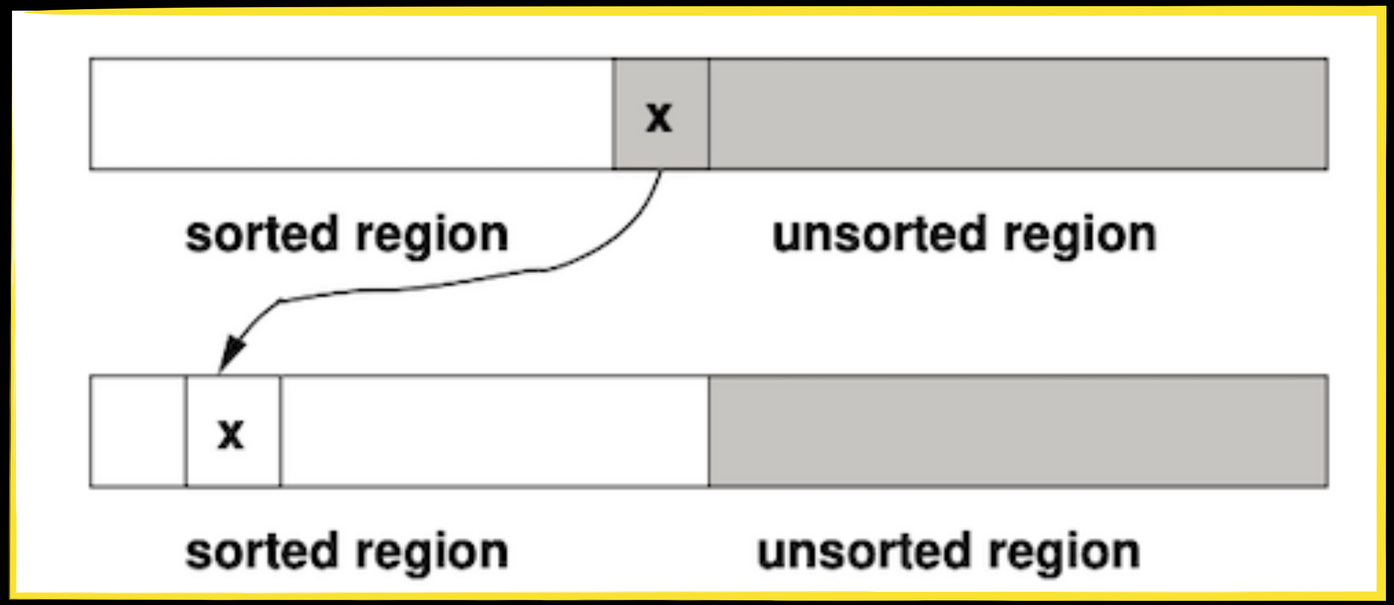


Count sort

2. Sorting algorithms

Insertion sort

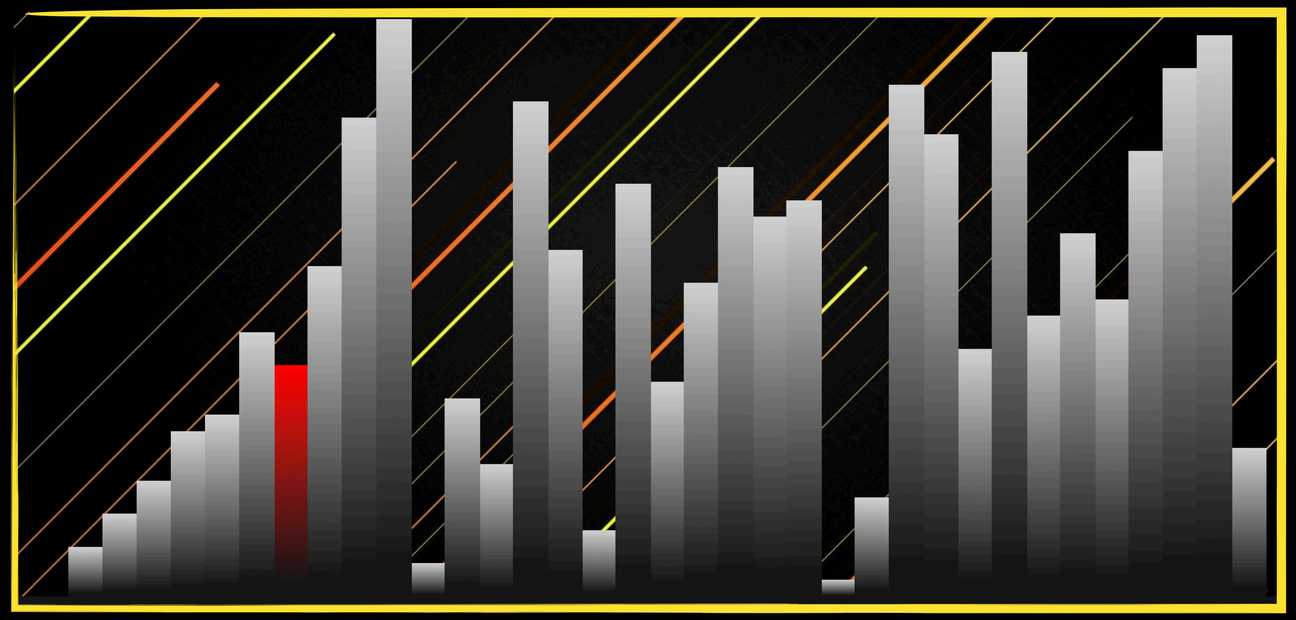
- divide array into sorted & unsorted regions
- take next lowest-rank object
- find its proper position in sorted region



2. Sorting algorithms

Insertion sort

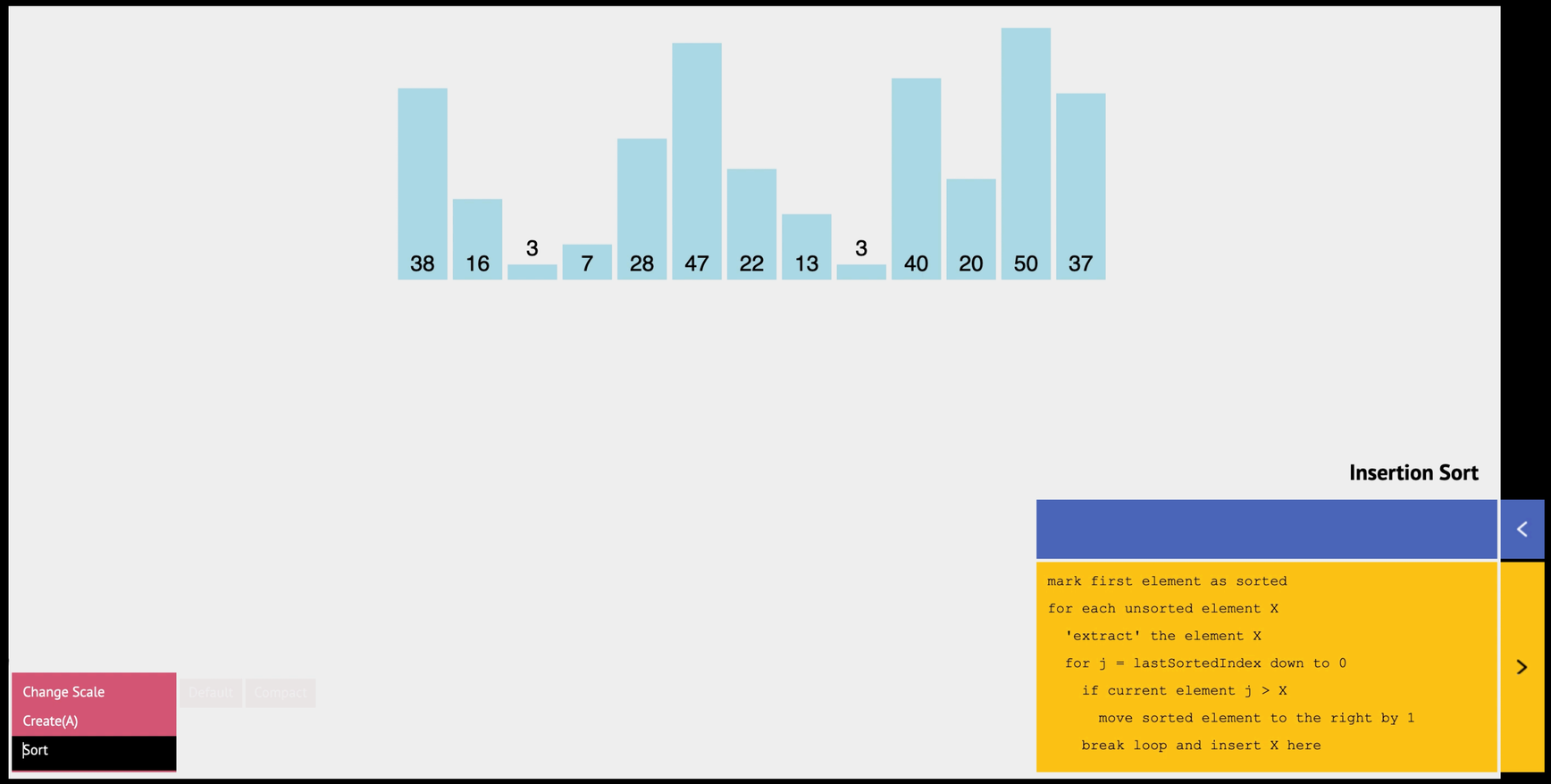
- divide array into sorted & unsorted regions
- take next lowest-rank object
- find its proper position in sorted region



2. Sorting algorithms

Insertion sort

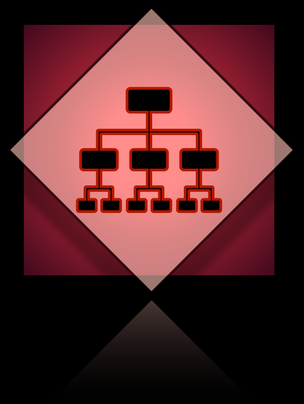
- divide array into sorted & unsorted regions
- take next lowest-rank object
- find its proper position in sorted region



2. Sorting algorithms

Insertion sort

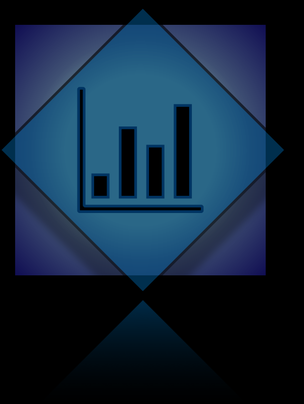
- divide array into sorted & unsorted regions
- take next lowest-rank object
- find its proper position in sorted region



Merge sort

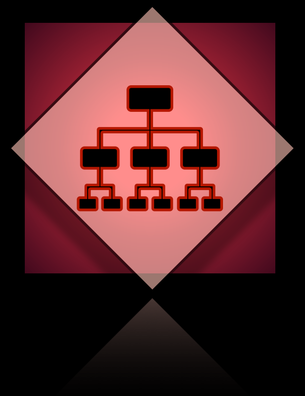
- divide-and-conquer technique
- separate each object into particular bin
- join & sort bins iteratively

Heap sort



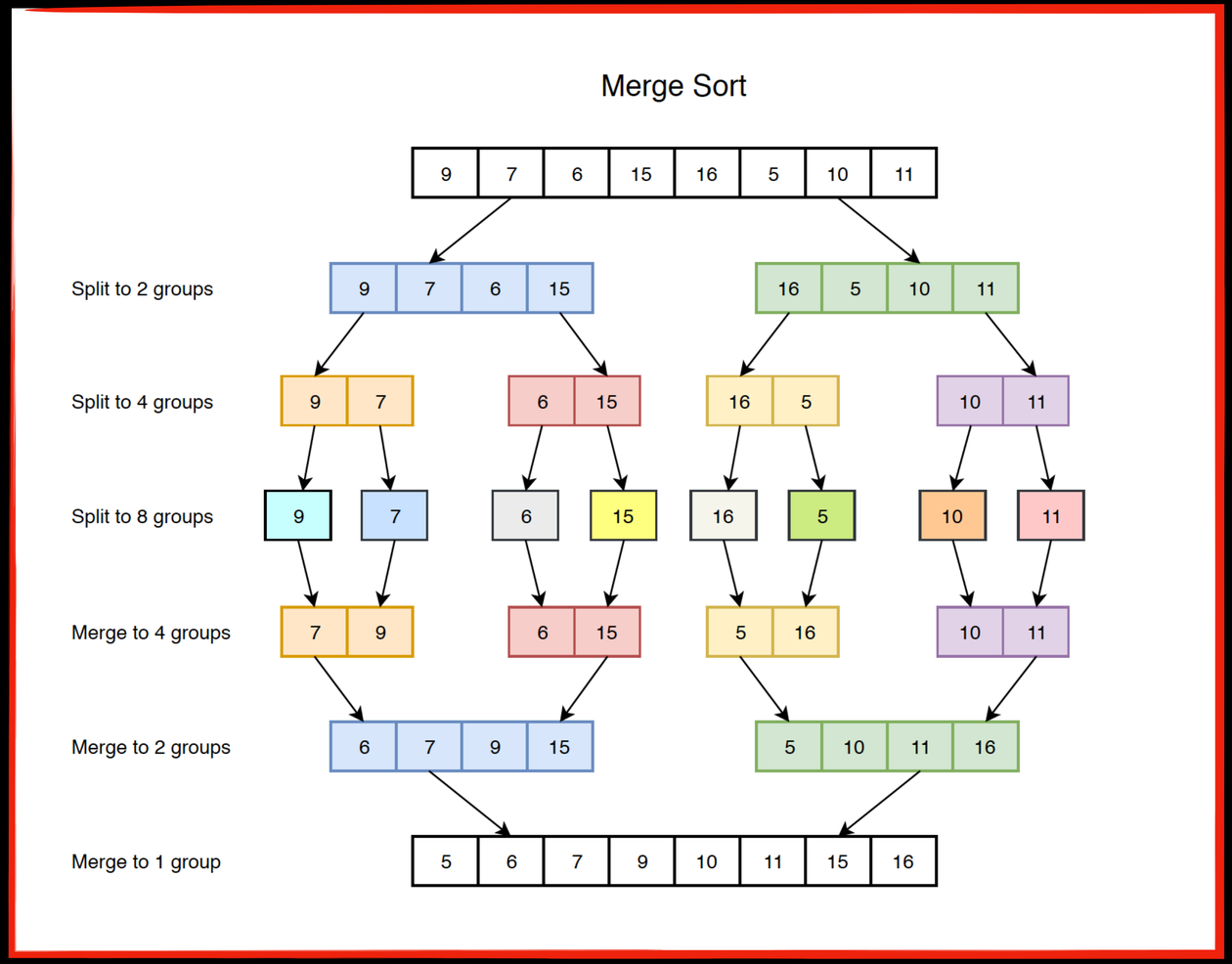
Count sort

2. Sorting algorithms

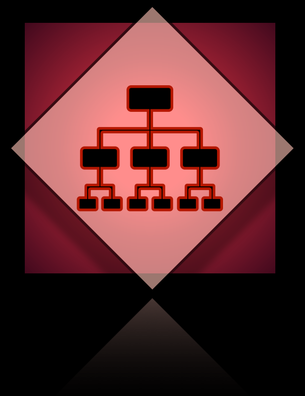


Merge sort

- divide-and-conquer technique
- separate each object into particular bin
- join & sort bins iteratively

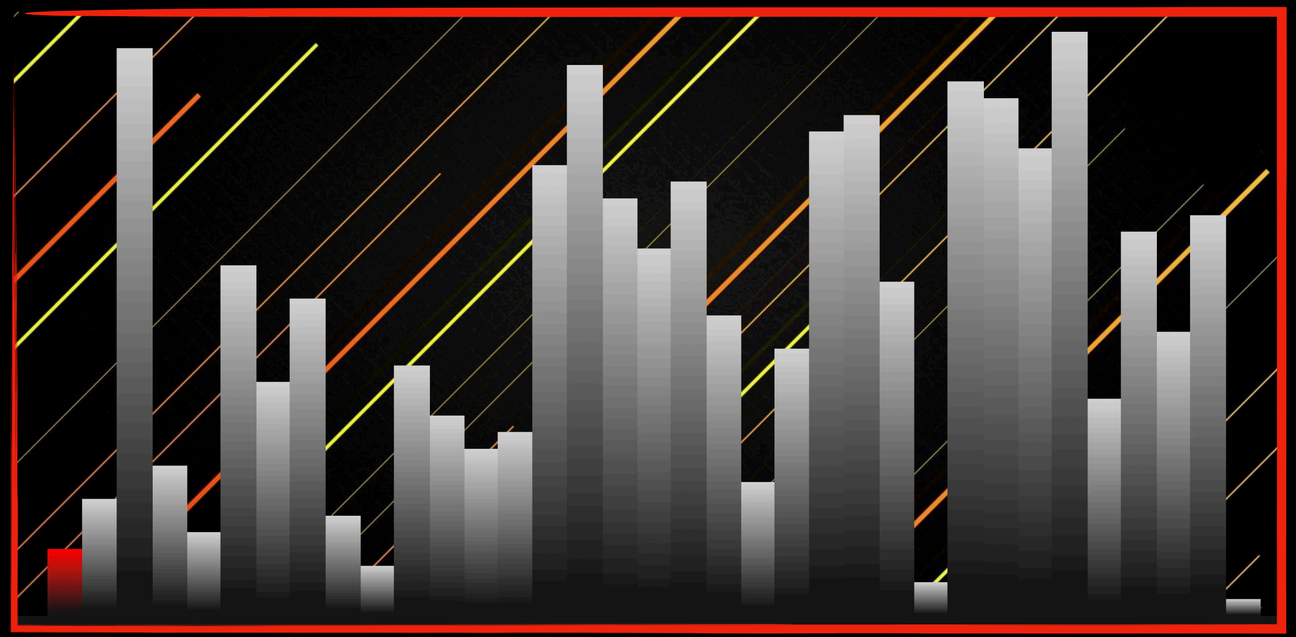


2. Sorting algorithms

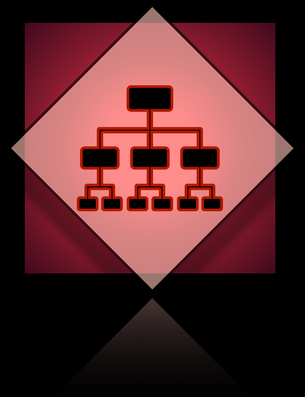


Merge sort

- divide-and-conquer technique
- separate each object into particular bin
- join & sort bins iteratively

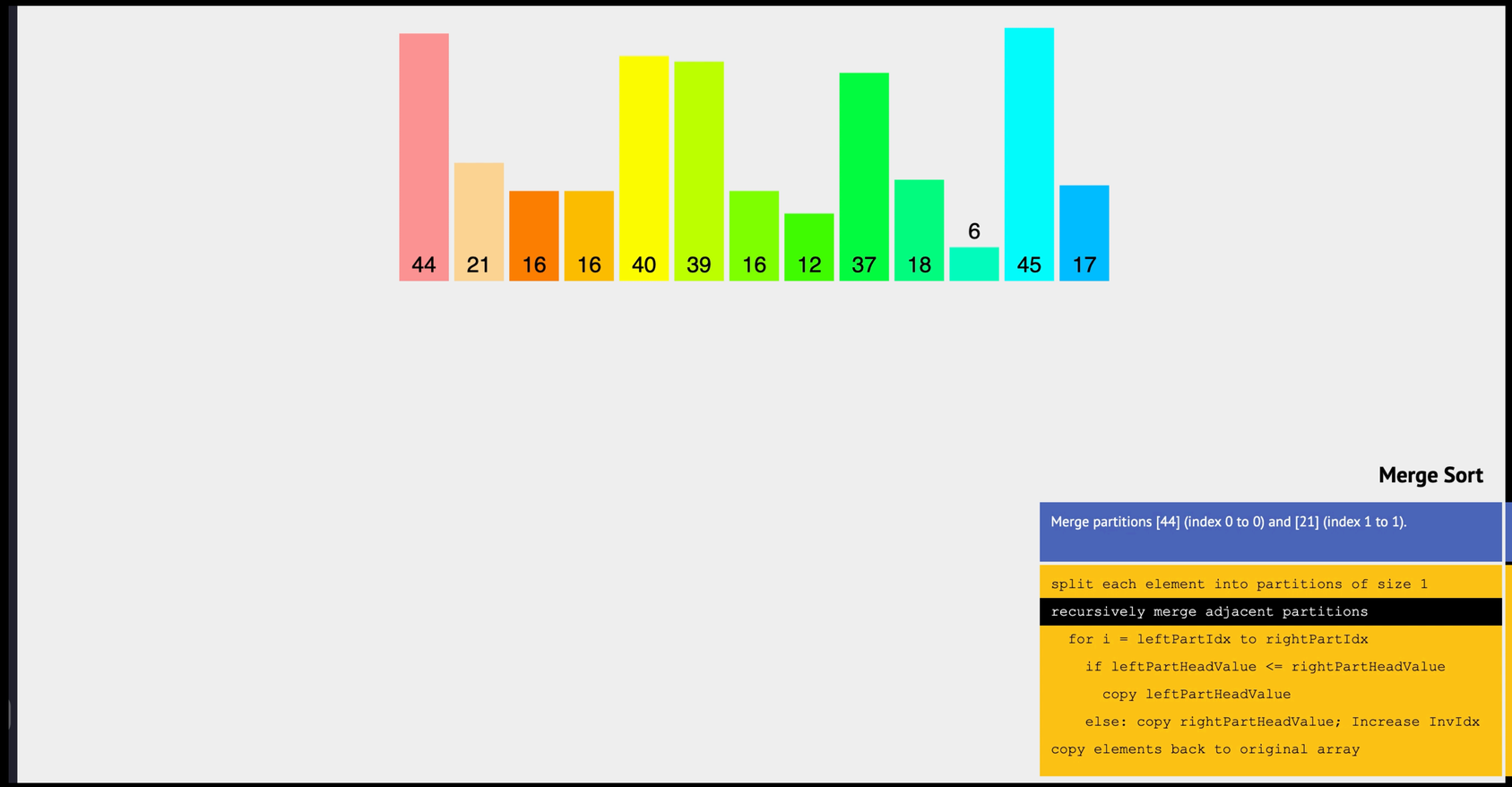


2. Sorting algorithms



Merge sort

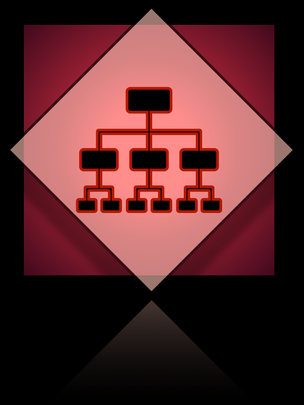
- divide-and-conquer technique
- separate each object into particular bin
- join & sort bins iteratively



2. Sorting algorithms

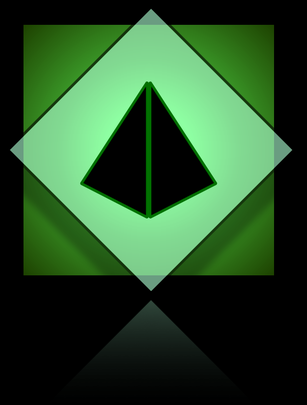
Insertion sort

- divide array into sorted & unsorted regions
- take next lowest-rank object
- find its proper position in sorted region



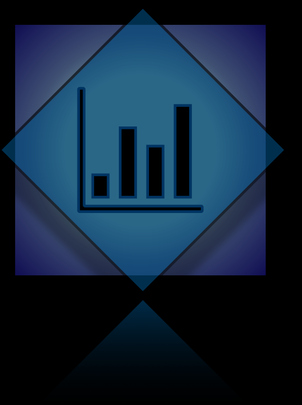
Merge sort

- divide-and-conquer technique
- separate each object into particular bin
- join & sort bins iteratively



Heap sort

- build binary tree from the array
- make each node greater than child
- remove maximum from root

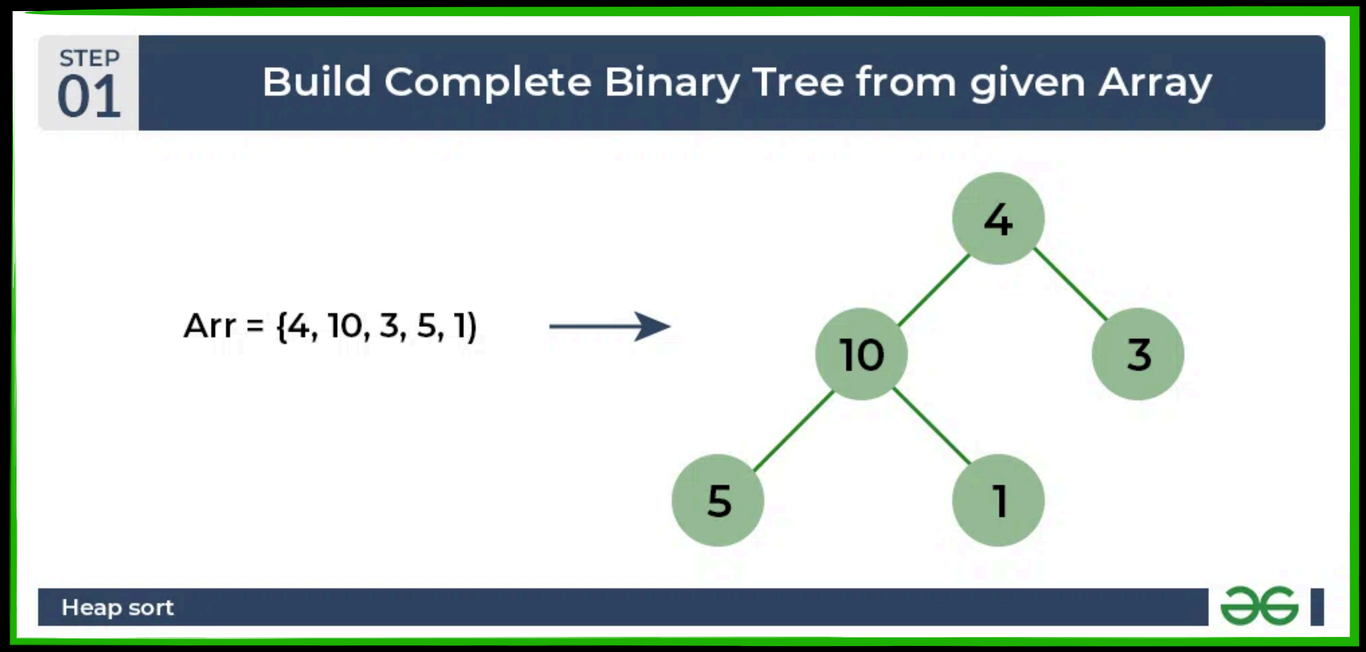


Count sort

2. Sorting algorithms

Heap sort

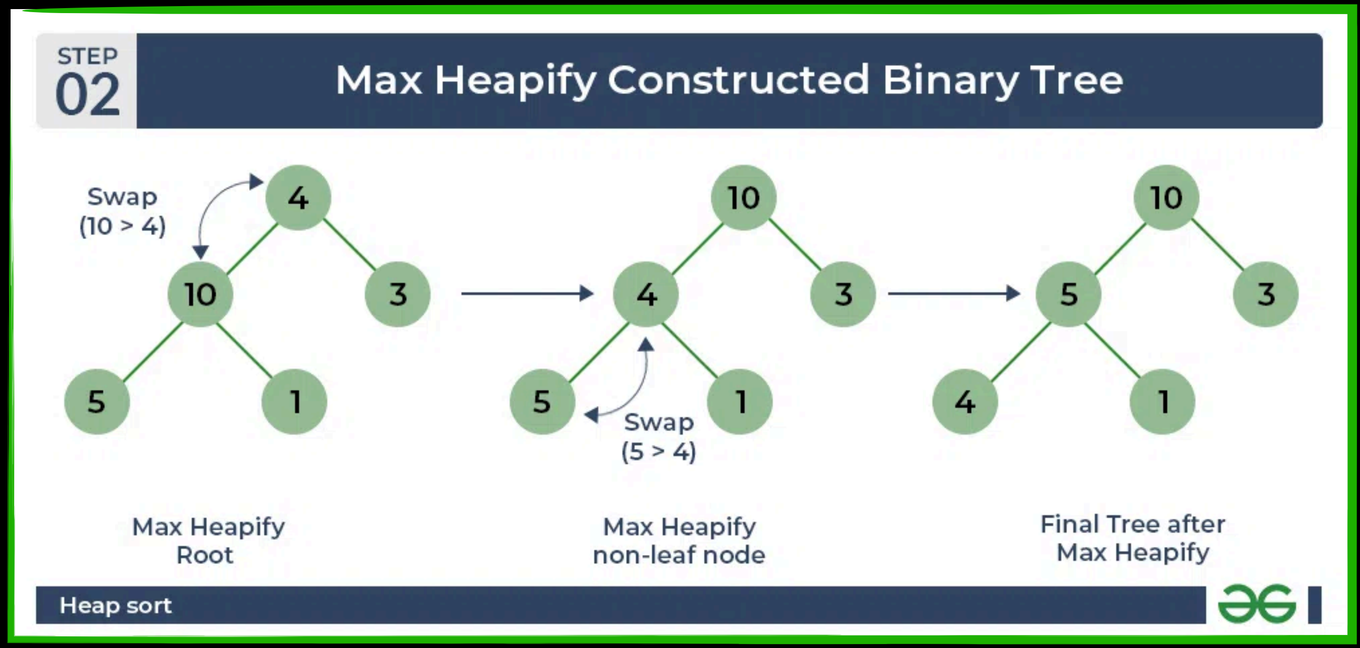
- build binary tree from the array
- make each node greater than child
- remove maximum from root



2. Sorting algorithms

Heap sort

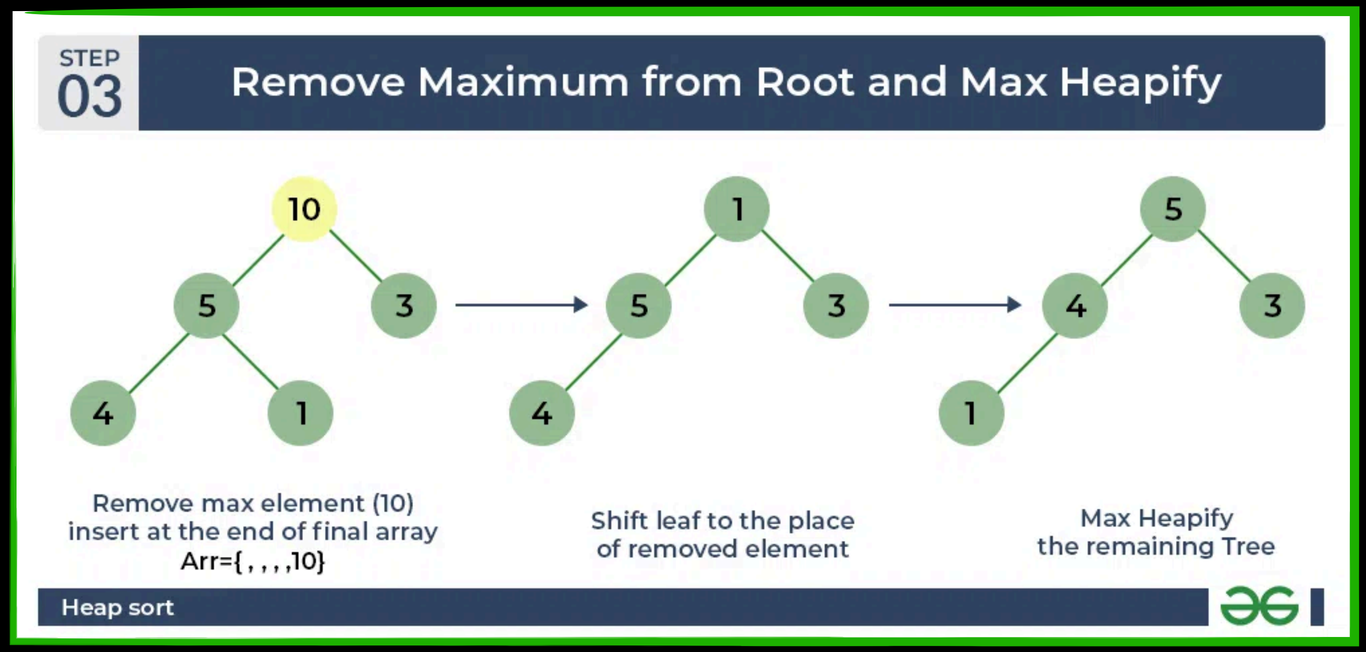
- build binary tree from the array
- make each node greater than child
- remove maximum from root



2. Sorting algorithms

Heap sort

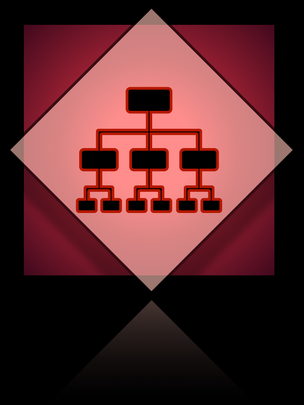
- build binary tree from the array
- make each node greater than child
- remove maximum from root



2. Sorting algorithms

Insertion sort

- divide array into sorted & unsorted regions
- take next lowest-rank object
- find its proper position in sorted region

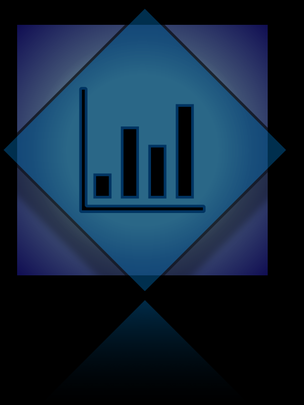


Merge sort

- divide-and-conquer technique
- separate each object into particular bin
- join & sort bins iteratively

Heap sort

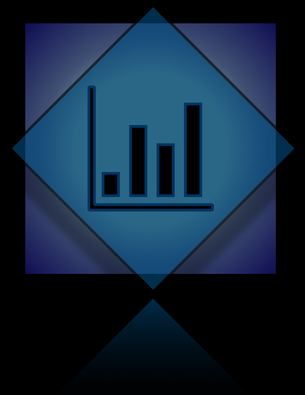
- build binary tree from the array
- make each node greater than child
- remove maximum from root



Count sort

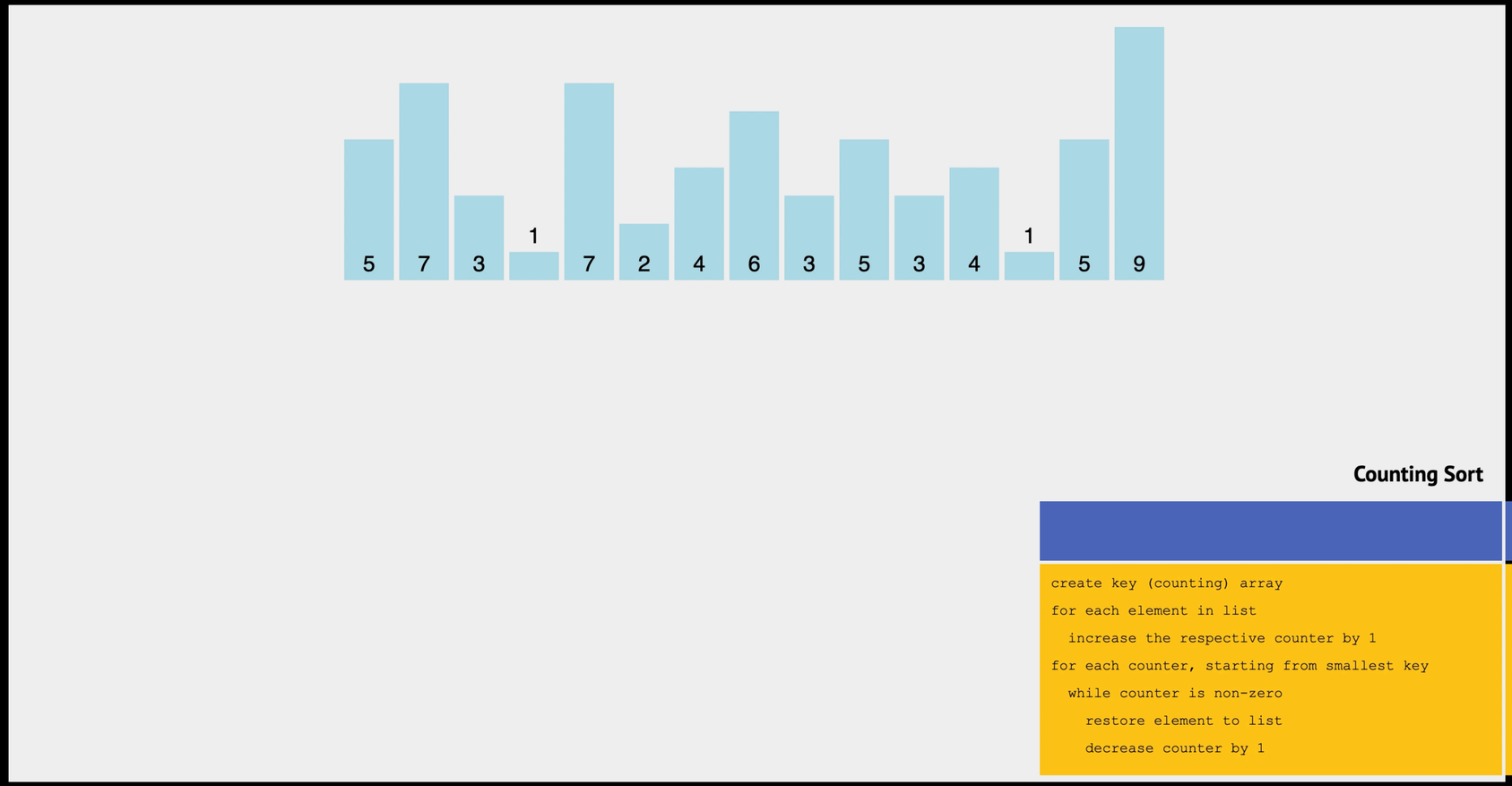
- make bins in range from 0 to K
- group all the elements into bins
- find each bin's index

2. Sorting algorithms



Count sort

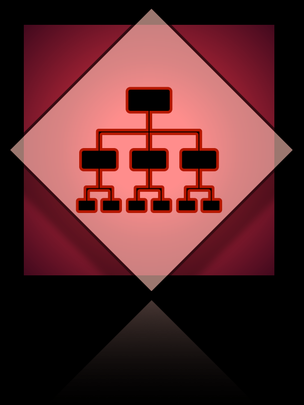
- make bins in range from 0 to K
- group all the elements into bins
- find each bin's index



2. Sorting algorithms

Insertion sort

- divide array into sorted & unsorted regions
- take next lowest-rank object
- find its proper position in sorted region

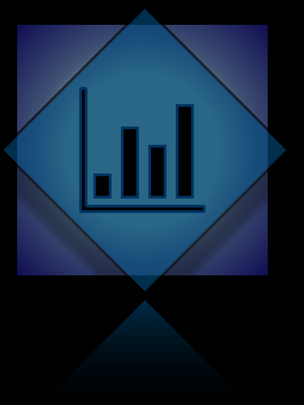


Merge sort

- divide-and-conquer technique
- separate each object into particular bin
- join & sort bins iteratively

Heap sort

- build binary tree from the array
- make each node greater than child
- remove maximum from root



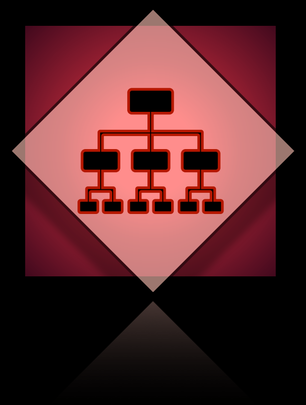
Count sort

- make bins in range from 0 to K
- group all the elements into bins
- find each bin's index

2. Sorting algorithms

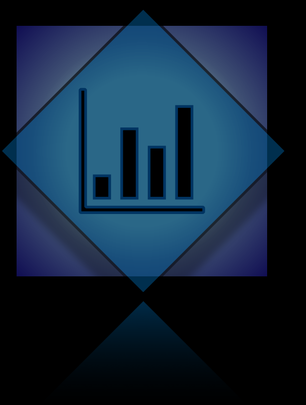
Properties

Insertion sort



Merge sort

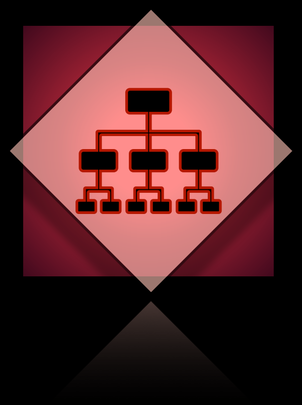
Heap sort

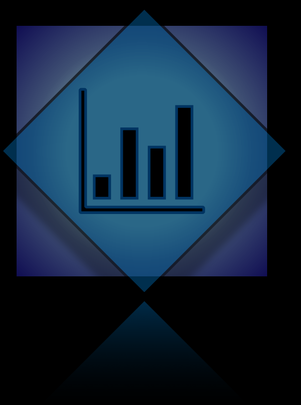


Count sort

2. Sorting algorithms

Properties

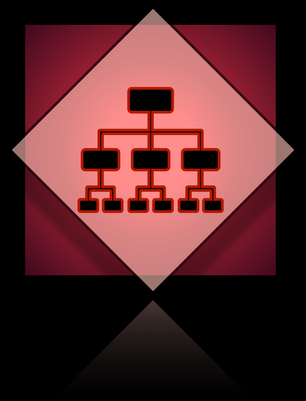


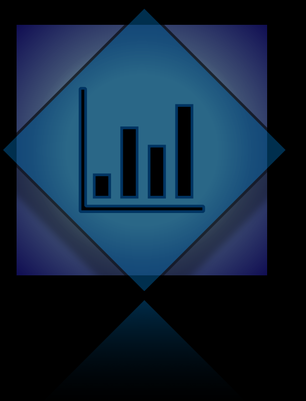


	Insertion sort	Merge sort	Heap sort	Count sort
Inplace				
Stability				
Comparison-based				

2. Sorting algorithms

Properties

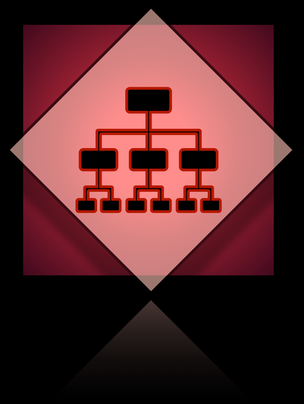



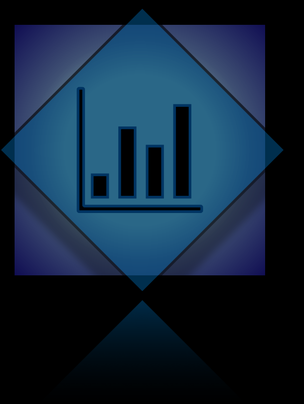


	Insertion sort	Merge sort	Heap sort	Count sort
Inplace	YES	NO	YES	NO
Stability	YES	YES	NO	YES
Comparison-based	YES	YES	YES	NO

2. Sorting algorithms

Complexity

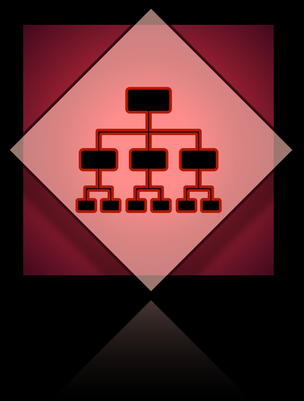


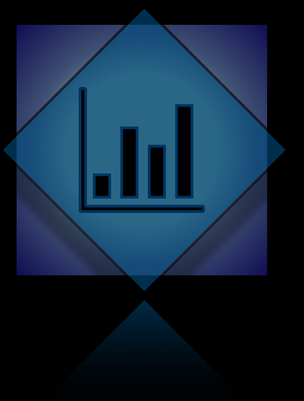


	Insertion sort	Merge sort	Heap sort	Count sort
Time worst				
Time best				
Memory				

2. Sorting algorithms

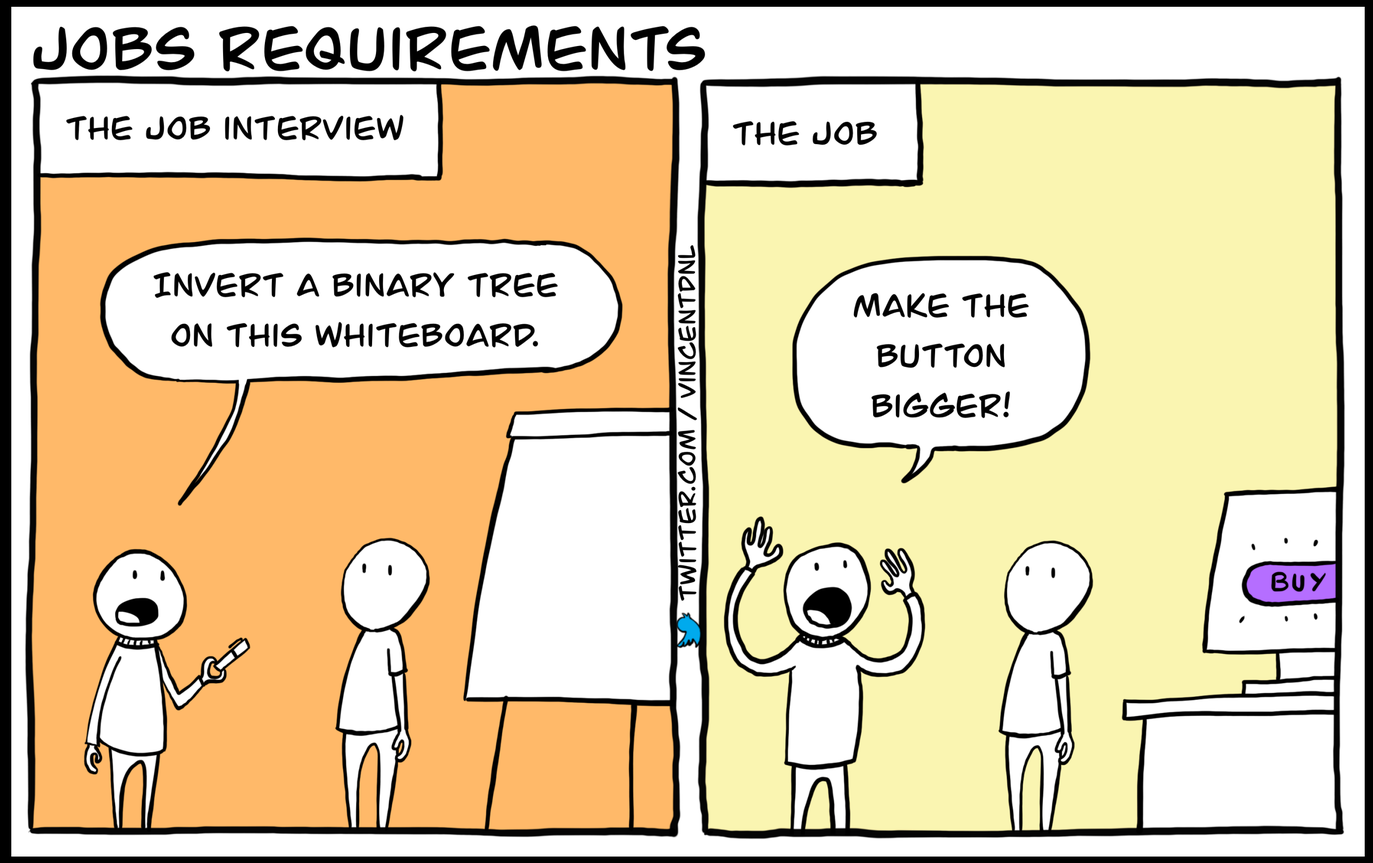
Complexity





	Insertion sort	Merge sort	Heap sort	Count sort
Time worst	O(n^2)	O(n logn)	O(n logn)	O(n+k)
Time best	Ω(n)	Ω(n logn)	Ω(n logn)	Ω(n+k)
Memory	O(1)	O(n)	O(1)	O(k)

3. Coding


JOBS REQUIREMENTS
THE JOB INTERVIEW
INVERT A BINARY TREE ON THIS WHITEBOARD.
TWITTER.COM / VINCENTDNL
THE JOB
MAKE THE BUTTON BIGGER!
BUY

3. Coding
Insertion sort

```
for i = 1 to n-1
    j = i
    while j > 0 AND A[j] < A[j-1]
        swap(A[j],A[j+1])
        j = j-1
```

3. Coding

Heap sort

```
Heapify (A, i)
        lft <– Left(i)
        rt <– Right(i)
        if lft <= heap-size[A] and A[lft] > A[i]
                then largest <– lft
                else largest <– i
        if rt <= heap-size[A] and A[rt] > A[largest]
                then largest <– rt
        if largest != i
                then Swap( A, i, largest );
                Heapify (A, largest);
```