1. Spiral matrix -> see more
2. **Jagged** arrays ->see more
3. **Rest**, **spread** operators
4. More about **this** and **arguments** in functions -> see more
5. 🡪Math.random(); -> checked
6. Not using function expression on the old way => use fat arrow functions
7. Do not use function declarations => use expressions
8. There is no point to give exact sort way in .sort() method
9. .filter() -> see how it works
10. Direct recursion -> only 1 function
11. Indirect recursion -> 2 functions calls each other
12. Exit criteria of recursion -> bottom ( prevent infinite recursion)
13. Merge sort / Binary search -> see more
14. Going through DOM tree with recursion
15. Call stack -> see more – коя функция след коя се извиква – исторя на извикванията
16. << bit shifting -> see more
17. 1 << 32 – max used value of integer
18. В програмирането , когато имаме интервал , края му е отворен (т.е. той се изйлючва)
19. Next permutation -> see more
20. Memoization -> making recursion not being so harmful, keep calculated elements
21. Use recursion only when it makes sense
22. Duck typing -> see more
23. Inheritance, encapsulation, abstraction, polymorphism -> 4 principles of OOP
24. Inheritance -> reusing the code , razshirqvame edin klas s dr I go nadgrajdame
25. Inheritance -> extensibility, reusability, provides abstraction, eliminates redundant piece of code
26. Квадрата е правоъгълник , но не наследява правоъгълника , защото квадрата е вариация на правоъгълника
27. Class inheritance -> is-substitutable-for
28. Getting all from the base class
29. Private variables -> Symbols , weakmap structures
30. If something is private our class can’t get it
31. Super() -> method calls the parent properties

**Date structures**

1. **Lists ->** lineary sequence of elements , using resizable array, linked implementation -> operations like splice , shift , unshift are slow, (pop and push are fast enough) -> resizable. Dostupvaneto na elementi e s konstantna slojnost. O(1) !!! Pazi se mqstoto v pametta I sled tova se umnojava indexa po bitovete na dannite **ex. 123(mqsto v pametta) 4(poziciq na elementa) \* 8(bite -> razmer na data type). If we do not know the exact size of the data type we can get it with key -> value (constant complexity) -> the keys in the objects can be only strings.** In JS arrays are objects -> functions are first class objects. Dostupvaneto na elementi po indeksi tuk e s lineina slojnost zashtoto trqbva da trugnem ot 0 element do iskaniq element ‘x’. Nqma smisul da se pazqt indexite -> ste stane otnoto s lineina slojnost. Gonim constant complexity. Vseki list.node , koito dobavqme se zakacha v pametta , no ne ni interesuva tochno kude , nqmame nujda da sa edin sled drug. V lista nie pazim referenciqta na prev, next … Kogato imame promenliva koqto sochi kum mqstoto v pametta , ne se pazi samiqt OBJECT !!! this.next = new Node() -> ne pazi objecta , a negovoto mqsto v pametta. Premahvaneto v kraq na ednosvurzan spisuk e bavna operaciq, zahsoto ne mojem ra razrushim posldnata referenciq I slojnostta veche ne e constant -> zatova ima dvoino-svurzan spisuk.
2. **Stack ->** nqma koncepciq za dostupvane na element po index , imame dustup samo do elementa na vurha (posledniq) .Rabotim samo v ediniq krai. Samo push() I pop(). Polzva se pri algoritmi nujdaeshti se ot stack. **Linked stack** -> top element is first element because in linked list remove perform better at the beginning.
3. **Queue -> FIRST IN FIRST OUT** -> enqueue , dequeue -> like printing lists on basic printer. Elements removed from the head -> dequeue. Elements insterted at the tail -> enqueuer.
4. **Generators ->** gives us the opportunity to use the new loop in JS forOF -> we can give the generator to forOF. It is NOT needed to talk about generators only in infinite sequence, we can use with combination of destructuring assigments.
5. **Array ->** object with properties -> indexes -> [0, 1, 2, 3 … n] indexes!!!
6. **Hash Tables ->** guaranteed constant complexity O(1). They adding values , removing values , finding values
7. **Maps and Sets -> MAPS -> KVP collections , indices can be not-only-number** , they are like arrays ; Sets -> only unique values there are no indices , there is no order, they can be implemented with maps !!! If we need order we use another structure like threes and others. Maps are like switch () -> dinamicly adding cases. The important is how they actualy workd

**Tree and Graphs**

1. **Trees -> Can be easily approached -> with small complexity O(logn) -> typical example is DOM TREE !!! (not only ofc). Между всеки 2 върха има само 1 път**
2. **Tree traversal algorithms -> DFS , BFS**
3. **Depth-First Search -> recursive -> going to the leaf on single branch -> Visit node’s successors first**
4. **Breadth-First Search -> nearest nodes visited first ->implemented by a queue**
5. **If we change queue with stack -> BFS go to DFS !!! (good to know)**
6. **Red-black tree- sorted**
7. **Binary heap -> binary tree -> Binary heap implementation with array.**
8. **Graphs ->** добър пример е картата на България -> пътищата са неговите разклонения, не е задължително , да има само 1 път между 2 върха, може да има цикли. Ребрата ни казват възможните пътища. Distance or shortest path between two points. Има насочени графи , ненасочени и претеглени.
9. **Насочена – казва се накъде да ходи**
10. **Ненасочена -> можем да отидем от София до Бургас и може да се вънем по същия път -> ребрата и посоките са много важни**
11. **Претеглена -> София-Бургас е 370 км (слага се оценка) \*тежест**
12. **Google Maps -> работи с графи**
13. **Графите могат да са свързани и несвързани -> свързаните -> от всеки връх да можем да стигнем до всеки връх.**
14. **Търсим прости пътища -> пътища без цикли ;**
15. **Predstavqne na graphs :**

**spisuk na susedite -> vseki znae koi mu e sused -> pazi spisuk na negovite susedi (sobstvenite si susedi)**

**matrici na susestvo -> kvadratna matrica -> broq na vurhovete -> rebrata koito izlizat ot dade vruh**

**spisuk na rebrata -> vzimame vs rebra I gi sortitrame po tejest …. Proverqvame dali shte se poluchi cikul**

1. **BFS -> nai-kratkiq put po broi vurhove**
2. **DFS -> obhojda v dulbochina**

**Document Object Model**

1. **API - Application programming interface ->**
2. **Ethernet -> inside connection (mini net)**
3. **JS -> imperative language**
4. **HTML -> decorative ‘language’ something will happen but we do not care how it will happen**
5. **Dom API -> we can change what user see , can apply styles, remove elements, add objects that represent HTML**
6. **The power is that we can change what user see**
7. **We can change the whole context**
8. **Css must not use id, it is saved for JS**
9. **document.getElementsByName is used for input elements**
10. **getElement -> return array like object and HTML collection -> live list**
11. **querySelector -> return Node List -> select elemets with CSS selectors -> static list**
12. **NodeList -> array like object ->collection of nodes -> has indexes but it’s not an array and it is very slow , querySelectiorAll –Statick -> if it is selectet can’t be changed**
13. **HTML collection -> live -> after selecting it can be changed -> real time changes**

**Promises and callbacks**

1. **Promise -> we build new code and we have a promise that we will get some result from the function above**
2. **Do not nest promises**
3. **They can be : resolved, rejected, pending states**
4. **When we create new Promise the constructor of the function can get 3 parameters of the callback : p2!!! Resolved, rejected, pending**
5. **Result from catch and then is new promise**
6. **Give us a promise that something will happen and we build code after that expectation**
7. **If a promise reject automatic goes to catch(), and if it resolve goes to then()**
8. **!!! SEE MORE ABOUT CALLBACK**