	Bi- O Notation	<ul> <li>If element is in array then A[begin] = key</li> </ul>	Worst complexity		
Things to know for each searching/sorting algos	Big-O Notation	Loop invariant	infinite	o O(1)	
Worst case time complexity (when) Expected time complexity (when)	Function Name  5 Constant	A[begin] <= key <= A[end]			
Best case time complexity (when)	loglog(n) double log		Space complexity     O(1)	<ul> <li>Stability</li> <li>Not stable → swap changes order</li> </ul>	
Invariant	Order or size: Reg(n) logarithmic Polylogarithmic	Binary search algorithm <	U O(1)	Not stable → swap changes order	
good in which case Bad in which case	n linear nlog(n) log-linear	Sorting algorithm	<ul> <li>Stability</li> </ul>	Loop invariant	
Implementation (at least in pseudo code)	m <sup>1</sup> polynomial	In computer science, binary search, also known as	Not stable	<ul> <li>After j iteration → the smallest j elements are sorted</li> </ul>	
Runtime	nting(a)  nt	half-interval search, logarithmic search, or binary chop, is a search algorithm that finds the position of a	Bubble sort	Insertion sort	
Space usage Stability	a* exponential	target value within a sorted array, Sinary search compares the target value to the middle element of	Implementation	Implementation	
Check List	n! factorial	the array Wikipedia	<ol> <li>If A[j] &gt; A[j + 1] → swap</li> </ol>	<ul> <li>J from 2 to n - 1 insert A(j) into sorted array</li> </ul>	
Big-O notation	O(log(n!)) = O(nlog(n))	Worst complexity: O(log n)	<ol> <li>Repeat n times, each time with one less element (the last element of each</li> <li>Recurrence</li> </ol>		
Definition: Upper bound	O(log(n:)) = O(nlog(n))	Average complexity: O(log n) Best complexity: O(1)	○ T(n) = T(n - 1) + n	<ul> <li>Recurrence</li> <li>T(n) = T(n - 1) + n</li> </ul>	
■ n0> 0; c > 0 ■ For all n > n0 -> T(n) <= cf(n)	Recurrences	Space complexity: O(1)			
	T(n) = 1 + T(n - 1) + T(n - 2) Big O will be O(2*n)	Data structure: Array Classe: Search algorithm	Time complexity     O(n^2)	Time complexity	
Definition: Lower bound	Example would be the recurrence relation of fibonacci algorithm	Class: Search argorithm	o O(II-2)	o O(n^2)	
■ n0>0; c>0 ■ For all n>n0-> T(n) >= cf(n)			Best complexity	Best complexity	
Recurrences	SEARCHING Binary Search	Peak Finding (Key idea is binary search)	<ul> <li>O(n) → when all are sorted</li> </ul>	<ul> <li>O(n) → when it's already sorted</li> </ul>	
Edge cases of algos	Implementation	To find local maximum	Worst complexity	Worst complexity	
Binary Search     Conditions	Start at the middle	Implementation     Start at the middle	<ul> <li>O(n^2) → when it's reverse sorted</li> </ul>	<ul> <li>O(n^2) → when it's inversely sorted</li> </ul>	
■ While begin < end	<ol> <li>If that is the search element → return element</li> <li>If the search element less than middle → search recursively on left (update new high</li> </ol>	<ol> <li>If Afmid1 is peak → return Afmid1</li> </ol>		40 00	
■ Key <= begin + (end - begin)/2	<ol> <li>If the search element more than middle → search recursively on right (update new right).</li> </ol>	h) 3. Else if left > A[mid] → search left ow) 4. Else if right > A[mid] → search right	Space complexity     O(1)	Space complexity	
<ul> <li>Begin = mid + 1 → no array out of bounds because division always round down</li> <li>End = mid</li> </ul>		The Last Hight - April - Search Light	~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	o O(1)	
<ul> <li>Return arr[begin] == key ? begin : -1</li> </ul>	<ul> <li>Things to take note</li> <li>Middle is low + (high - low) / 2</li> </ul>	<ul> <li>Invariant</li> <li>If we search right then peak is in right</li> </ul>	Stability	<ul> <li>Stability</li> </ul>	
BST	<ul> <li>Stop when low &gt; high</li> </ul>	<ul> <li>If we search left then peak is in left (can prove using induction)</li> </ul>	<ul> <li>Yes → only swap elements that are different</li> </ul>	o Yes	
o Insert o Delete: 3 cases	Binary Search	<ul> <li>There exits a peak in the range [begin, end] and the peak in [begin, end] is also in the peak in [begin, end].</li> </ul>	peak in [1, n]  Loop invariant	Loop invariant	
Delete: 3 cases     Find min	Sorted array: A[0n-1]	<ul> <li>T(n) = T(n/2) + O(1)</li> </ul>	<ul> <li>At the end of j iteration → last j element is sorted</li> </ul>	First j elements are sorted	
Find max		<ul> <li>O(1) is comparing the middle with the left and right element</li> </ul>	<ul> <li>Last j element are also the biggest j elements</li> </ul>		
Find successor	2 4 4 5 6 7 8 9 n 17 23 28		Selection sort	Note: Insertion sort is very fast on sorted array!	
Find predecessor     inorder/preorder/post-order	int search(A, key, n) begin = 0	SORTING Bogosort	<ul> <li>Implementation</li> </ul>	Merge sort  implementation	
o search	end = n-1	Implementation	Find minimum element A[j] im A[1n]	Split array into 2 halves	
	while begin < end do:	Choose random permutation of A     Return that permutation if A is sorted	Swap A[j] with A[k]	<ul> <li>Recursively sort the 2 halves</li> </ul>	
Common Recurrences  1. $T(n) = 2T(n/2) + n \rightarrow O(nlogn)$ (e.g merge sort)	mid = begin + (end-begin)/2; if key <= A[mid] then		Recurrence	Combine both     Recurrence	
<ol> <li>T(n) = 2T(n/2) + 1 → O(n) (e.g in-order traversal)</li> </ol>	end = mid	Time complexity     O(n * nf)	→ T(n) = T(n - 1) + n	<ul> <li>T(n) = 2T(n/2) + cn (c is a fix constant)</li> </ul>	
<ol> <li>T(n) = T(n/2) + 1 → O(logn) (e.g binary search)</li> </ol>	<pre>else begin = mid+1 return (A[begin]==kev) ? begin : -1</pre>	○ O(n*n!)	Time complexity	Time complexity	
<ol> <li>T(n) = T(n/2) + n → O(n)</li> <li>T(n) = T(n - 1) + T(n - 2) + 1 → O(2<sup>n</sup>n) (e.g. fibonacci)</li> </ol>	return (x(begin)=-key) 7 begin : -1	Best complexity	• O(n^2)	O(nlogn) Best complexity	
0. 1(1) - 1(1 - 1) - 1(1 - 2) - 1 - 0(2 1) (0.9) INDURADO)	Preconditions	o O(n)		O(nlogn)	
The more common Big O Notation	Array of size n	Recurrence	Best complexity     ○ O(n^2)	Worst complexity	
	Array must be sorted	○ General: T(n - k) + T(k) + n	o O(ii-2)	o O(nlogn)	
Control Control Control	Postcondition	Time complexity	Worst complexity	<ul> <li>Space complexity</li> <li>O(n) → put the elements into new array</li> </ul>	
Summary	· restoricinon	o O(nlogn)	<ul> <li>O(n^2) → when it's inversely sorted</li> </ul>	<ul> <li>Stability</li> </ul>	
	Quick sort	Best case	Space complexity	o Stable	
	Implementation (Partition)     Choose pivot	<ul> <li>O(nlogn) → when the partition always divide the element into 2 equal halves</li> <li>T(n) = T(n/2) + T(n/2) + n</li> </ul>			
Name of Acad Cons.   Language   Words.   Salary   Statute	<ol> <li>Find element that is less than pivot → move to left side of pivot</li> </ol>	n=cost of partition on n elements			
Name Best Case Average Worst Extra Stabler Case Case Memory	<ol> <li>Find element that is greater than pivot → move to right side of pivot</li> </ol>	Worst case			
Bubble Sort $O(n)$ $O(n^2)$ $O(n^2)$ $O(1)$ Yes		<ul> <li>O(n^2) → when each partition only reduce the number of element by one</li> </ul>	Successor Queries		
Selection Sort $O(n^2)$ $O(n^2)$ $O(n^2)$ $O(1)$ No	Time complexity     Runtime partition; O(n)	T(n) = T(n - 1) + T(1) + n     n=cost of partition on n elements	Background: when you search for a key that is not in the tree, you	ou will either get its successor or	
	Invariant partition	T(1) = cost of quicksort on one element	predecessor		
Insertion Sort $O(n)$ $O(n^2)$ $O(n^2)$ $O(1)$ Yes	<ul> <li>For all i &gt; high, A[i] &gt; pivot</li> </ul>	Space complexity	•		
Merge Sort $O(n \log n)$ $O(n \log n)$ $O(n \log n)$ $O(n)$ Yes	○ For all i < j< low, A[j] < pivot	<ul> <li>O(logn) → quicksort calls itself logn time. Each time allocation a new space</li> </ul>	If you want to search for successor:		
		Bad pivot	Search for that key that is not in the tree		
	partition(A[1.m], n, plades) — # Assume no deplicates, and	First element	Check if the obtained key is successor		
	pixer = A[pixelex]; If pladex is the index of pivet  swapt ([1], A[pixelex](; If store pixer in A[1])	Last element     Middle element	If it is → return that key		
		Basically every single thing that is the same position	If it is → return that key     Else, search for successor of that key		
	while (low < high)	Runtime: O(n*2)	4. Eise, search for successor of that key		
	while (Allow) < pivot) and (low < high) do low++; while (Allogi) > pivot) and (low < high) do high;	Good pivot		San 144507	
	If thee < high) them snop(d)hee), d[high];; snop(d[f],d[him-1]);	Median element     Random element	Background: If you want to search for successor and the key is i	n that tree	
	return Arm-1;	Runtime: O(nloan)			
	. —	QuickSelect     Runtime: O(n) → select stuff using the idea of partition	<ol> <li>Just search for that node normally</li> </ol>		
	Implementation for Quick sort		<ol> <li>After finding the node, if the node has right child → search</li> </ol>	h rightchild.min()	
		Order Statistics (Search for kth element)  Implementation	<ol><li>Else search parent of that node until the child is not the p</li></ol>	arent's right tree (the child is the parent's	
	Try Quick Sort, on example array (27, 38, 12, 39, 27, 16), We shall elaborate the first	Implementation     Choose random pivot	left tree)		
	partition step as follows:	<ol> <li>Do partition</li> <li>If the kth element is on the left → recursive left find the kth element in left</li> </ol>			
	We set p = a(0) = 27. We set a(1) = 38 as part of \$2 so \$1 = () and \$2 = (38).	<ol> <li>If the kin element is on the left → recursive left find the kin element in left</li> <li>Else if kth element is on the right → recurse right, find the kin he element on the right</li> </ol>			
	We swap a(1) = 38 with a(2) = 32 to 51 = (12) and 52 = (38). We set a(3) = 59 and later a(4) = 27 as part of 52 so 51 = (12) and 52 = (38,59,27).	Time complexity			
	We stort alp(= 92 and start a(q) = 22 as part or 22 to 51 = (12) and 52 = (34,59,27). We sawap a(2) = 38 with a(5) = 16 = (51 = (12,6) and 52 = (39,27,38). We swap p = a(0) = 27 with a(2) = 16 to 51 = (16,12), p = (27), and 52 = (39,27,18).	O(n)			
		Note: the main idea is we just do recursion on just one side instead of 2 sides. This is a lot faster			
	After this, all 1 = 27 is guaranteed to be sorted and now Quick Sort recursively sorts the	,			
	left side a(01) first and later recursively sorts the right side a(35)				
	After thin, 4(2) = 22 is guaranteed to be serted and row Quick Stre recursively sorts the left side 4(0,1) first and later recursively sorts the light side 4(3,5).	TREES Binary Tree			
	_	Binary Tree  Factors that determine the order of BST			
	Stability	Binary Tree  Factors that determine the order of BST  Order of insertion			
	_	Binary Tree  Factors that determine the order of BST			

		Augmented Trees	Hash table		Time	Rest		Space	0.120	Invarian	Good	Rad	Recurre	
Deleting node     Case1: No children		Updating weight takes constant time (when rebalancing)	Type Unordered associative array		Comple		Worst Time	Space	Stability	t	when?	when?	nce	
<ul> <li>Just delete</li> </ul>		2. You only update the weight of the 2 affected nodes (blue and orange). The rest remains	he same Invented 1963 Time complexity in big O notation		xity									
Case 2: one child			Algorithm Average Worst case	Bubble	O(n^2)	O(n)	O(n^2)	O(1)	Stable	Last j	Small	Reverse	T(n) =	
<ul> <li>Delete that node</li> <li>Then, connect the child to the pa</li> </ul>	rent of that node	Augmented Trees	Space O(n) <sup>(1)</sup> O(n)							element s sorted	dataset	d array	T(n-1) +	
Case 3: 2 children		minutening minutening	Search O(1) O(n) Insert O(1) O(n)							3 Joilea	Almost	Smalles		
<ul> <li>Find successor of the node that</li> <li>Delete that node</li> </ul>	ou want to delete		Delete C(1) O(n)								sorted	t element		
Remove successor											unuy	in last		
<ul> <li>Successor take that position inst</li> </ul>	ead	w1 w2 w3 w1 w2 w3	How to deal with collision									position		
		Interval Search	<ul> <li>Chaining</li> <li>Put both items in the same bucket</li> </ul>	Selectio	O(n^2)	O(n^2)	O(n^2)	O(1)	Unstabl	First j	Check if	Large	T(n) =	
		Runtime	■ Form a linked list  ○ Open Addressing	ln l					e	sorted	already	array	T(n-1) +	
Binary Search Tree	Binary Search Tree	○ O(logn)	Open Addressing							First i	Least	Inversel		
delete(v) Running time: O(height		<ul> <li>If the thing we are searching is more than the interval then go right. Else go left.</li> </ul>	Implementation (for hashing with chaining)     Insert							also the	swap	y sorted		
1. No children:	- insert: O(h) - delete: O(h)	Orthogonal Range Searching (One Dimension)	Calculate h(key)							smallest				
2. 1 child:		Implementation	Look up h(key) and add (key, value) to the linked list     Runtime: O(1 + cost(h))							J	_			
- retrieve v - connect child(v) to parent(v) 3. 2 children	Query Operations: - search: O(h)	<ol> <li>Find split node (take note that the circle nodes contain the maximum element on subtree)</li> </ol>	the left Runtime (expected/worst): O(1)/O(1)	Insertio	O(n^2)	O(n)	O(n^2)	O(1)	Stable	First j element	Almost sorted	Inversel y sorted	T(n) = T(n-1)	
3. 2 children - x = successor(v) - children	- predecessor, successor: O(h)	Do left traversal	<ul> <li>Search</li> </ul>	l"						sorted	array	l'	+n	
<ul> <li>remove v = connect x to left(v), right(v), parent(v)</li> </ul>	<ul> <li>findMax, findMin: O(h)</li> <li>in-order-traversal: O(n)</li> </ul>	Do right traversal	Calculate h(key)									Large		
	= IPO de l'avesar. O(II)	Time complexity     Finding split node: O(logn)	<ol> <li>Search for (key, value) in the linked list Runtime; O(n + cost(h))</li> </ol>		011	011	011	011				Almost	mr. s	
		<ul> <li>We recurse at most O(logn) time</li> </ul>	Runtime (expected/worst): O(1)/O(n)  Java Hash Functions	Merge	O(nlogn )	O(nlogn )	O(nlogn )	U(n)	Stable	Subarra v sorted	Sorting linkedlis	sorted	T(n) = 2T(n/2)	
Height-balanced Tree (AVL)		<ul> <li>If k is the number of queries found → overall time complexity: O(k + logn)</li> <li>Note:</li> </ul>	<ul> <li>Always returns the same value, if the object hasn't changed</li> </ul>								t	array	+ cn	
Note	and at amount had what he a con-	<ol> <li>do all-leaf traversal-right if key &gt;= low</li> </ol>	<ul> <li>If two objects are equal, then they return the same hashCode</li> <li>Must redefine equals to be consistent with the hashCode</li> </ul>								When			
A height-balanced tree with n nodes i     A height-balanced tree with height h i		Do all-leaf traversal-left if key <= high									we want			
		Complet Table	Java Hash Functions								to maintai			
Trie.		Symbol Table  Key should be immutable	Every object supports the method:								n the order of			
		Using <key, value=""> pair</key,>	boolean equals(Object o)								order of array			
	Trie Tradeoffs	Direct Access Tables	noslean equalstRejeon #11 if op == noill return falses if op == thing returns transp	Quickso	O(nloar	O(nlogn	O(p^2)	O(logn)	Unstabl	Pivot is	When	When	T(n) =	
Trie Tradeoffs	Time:	Direct Access Tables  Time complexity:	if (1 p instance) Pairs) return falses Pair pair * (Pair)	rt	)	)	O(II 2)	O(logil)	e	correct	pivot	pivot	T(n-k)	
Time:  - Trie tends to be faster: O(L).	<ul> <li>Trie tends to be faster: O(L).</li> <li>Does not depend on size of total text.</li> </ul>	Search: O(1)	if qualries   first  return false if qualries accord := second return false;							position	will divide	only remove	+ T(k)	
The tends to be raster: O(L).     Does not depend on size of total text.	Does not depend on number of strings.	Insert: O(1) Problems	return truer							Element	element	one		
<ul> <li>Does not depend on number of strings.</li> </ul>		Too much space								on left less	s into 2 equal	element		
	Space:  — Trie tends to use more space.	<ul> <li>If keys are not integers then it's a problem (how to arrange them in a table)</li> </ul>	<ul> <li>Properties of good hash function</li> </ul>							than	size			
	- BST and Trie use O(text size) space.	Hashing	<ol> <li>h(key, i) should enumerate all the buckets → if not it'll return table full when there still space left</li> </ol>	8						pivot Element	groups			
Even faster if string is not in trief	<ul> <li>But Trie has more nodes and more overhead.</li> </ul>	павния	<ol><li>Simple uniform hashing assumption → every key is equally likely to map to every</li></ol>							on right				
			permutation, independent of other keys							greater than				
Dunamia Order Statistics		Depth-First Search	<ul> <li>Each edge is decrease once → O(ElogV)</li> </ul>							pivot				
Dynamic Order Statistics  1. If you search kth item and you are going	to the right		<ul> <li>Each node is deleted once → O(VlogV)</li> </ul>	Quickse	O(nlogn	O(n)	O(n^2)	-	-					
<ol><li>When you search right, you are searching</li></ol>	ig the k - (left_weight + 1)	DFS(Node[] nodeList)(	○ Total: O(ElogV)	lect	)			l	l		I			
<ol><li>If you are searching left, you continue to</li></ol>	search the kth item		Can be used:	Store Tree	in an Arra	ıy		Store Tree	e in an Arr		Kruskal's A	gorithm		
		boolean[] visited = new boolean[nodeList.length];	<ul> <li>Find shortest path when there is no negative weight cycle</li> </ul>	left(x) = right(x) =	2x+1		-	parent (a)	- floor(ix-1		// fort edges and Edge() sortedEdge	initializa - mart(0.E(1))		
Merge sort is more efficient and works faster t		Arrays.fill(visited, false);	<ul> <li>Find the longest path by negating the edges. (Only when no cycle)</li> </ul>								Acception-oldges of December of - se		Lat-Hidgen () /	
datasets. Quick sort is more efficient and work size or datasets	s faster than merge sort in case of smaller a			printly 24 I		:::		priority 24	1 1 3 10 10 10		// Itemate through	h all the edges, in oriedliges, lengths i	order re) (	
size or datasets		for (start = i; start <nodelist.length; (<="" start++)="" td=""><td>Shortest Path Summary:</td><td></td><td>8</td><td>_</td><td></td><td></td><td>0</td><td></td><td>State 4 -</td><td>ortedbiges, benging in sortedbiges[133 // e.see()/ // get no e.te(d)</td><td>de endociena</td><td></td></nodelist.length;>	Shortest Path Summary:		8	_			0		State 4 -	ortedbiges, benging in sortedbiges[133 // e.see()/ // get no e.te(d)	de endociena	
Insertion Sort is preferred for fewer elements, nearly sorted because it skips the sorted value				0	•	<u>~</u>			•		AE 174E	Eindov.wol ( // in : etilique.add(so); // s Lunionfv.wl; // com	the same tree?	
of both algorithm we can say that Merge Sort is		if (!visited[start]) {	Graph Type Algorithm Time			0 0		0				Custom(r.w): // com	bin trees	
efficient in terms of space.		visited[start] = true;	No negative weight cycles Bellman-Fond O(VE)											
Insertion sort is faster for small n because Ou	ick Sort has extra overhead from the recursion	DFS-visit(nodeList, visited, start);	No negative edges Dijkstra D(E log V)								<ul> <li>Use union</li> <li>If poll from</li> </ul>	n queue and	still not unio	n → add the node in the MST then union
function calls. Insertion sort is also more stable		Steps	No directed cycles TopoScrt + Relax O(E)	Sorted-lis     Ke	it to heap ep extract	max					Queue so	rt edges by v	veight	
GRAPHS		Start at a node Go all the way until cannot go further	No cycles DFS + Relax O(V)	o run	time : O(nl	ogn) → each	extract ma	ax is logn			Find: eac	ich O(logV) h O(logV)		
Adjacency list		<ul> <li>Backtrack until find a node that can go further</li> </ul>	Planar	Unsorted     Use	-list to hea						Sort the e	dges O(Elog	E) = O(Elog	n
Adjacency matrices		Do recursively until there is no more nodes.				because the	e height dec	creases			<ul> <li>Total: O(E</li> </ul>	logV)		
	reible path				iume. O(n									E)
*Note: Both DFS and BFS do not explore all po E.g. cannot use DFS/BFS to find longest path	ssible path	Runtime  ○ O(V+E) (adjacency list) → each node is visited once, each edge also visited once		nion-find						• Bor		uvka's sten	cost : O/V+	r for all nodes O(V)
E.g. cannot use DFS/BFS to find longest path	ssible path	Runtime  O(V+E) (adjacency list) — each node is visited once, each edge also visited once O(V*2) (adjacency matrices) — each node is visited once. Need to check all the								• Bor	Each Bo ■ St		ent identifie	
E.g. cannot use DFS/BFS to find longest path BFS	ssible path	Runtime  O(V+E) (adjacency list) — each node is visited once, each edge also visited once O(V*2) (adjacency matrices) — each node is visited once. Need to check all the possible registrouring nodes Topological order		nion-find						• Bor	Each Bo ■ St	ore compone nd the min e	ent identifie	th connected components using BFS/DFS
E.g. cannot use DFS/BFS to find longest path  BFS  frontier = {s}  while frontier is not empty:	ssible path	Runtime  O(V+E) (adjacency list) — each node is valided once, each edge also visited once  O(V*D) (adjacency matrices) — each node is visited once. Need to check all the property of the control of the		nion-find						• Bor	■ Each Bo ■ St ■ Fir + I ■ Sc	ore compone nd the min en i) an all nodes	ent identifie dges for eac to merge c	th connected components using BFS/DFS omponents O(V)
E.g. cannot use DFS/BFS to find longest path  BFS  frontier = {s} while frontier is not empty: next-frontier = {}		Runtime  O(V+E) (adjacency list) — each node is visited once, each edge also visited once  O(V+E) (adjacency matrices) — each node is visited once. Need to check all the possible neighbouring nodes  Topological order  O(p) two forward object.	Baseled advisory.  Wear free Grafts.	Union-Find S Fath Compression willhoo	ummary at recipited union?	miles 000				• Bor	■ Each Bo ■ St ■ Fir + I ■ Sc	ore component and the min ent i) an all nodes at most O(lo	ent identifie dges for eac to merge c	th connected components using BFS/DFS omponents O(V)
E.g. cannot use DFS/BFS to find longest path BFS  frontier = {s} while frontier is not empty: next-frontier = {} for each node u in the front	ier:	Rustime  O(V+E) (adjacency list) — each node is valided once, each edge also visited once  O(V+2) (adjacency matrices) — each node is valided once. Need to check all the  Topological cert any libraries modes  Only have forward edges  Only InDA  Frost coder DFS  SSSP	Bandod a Harring.  Were Free Graph.  Heap	nion-find	ummary at relighted union?					Steiner	Each Bo	ore component of the min end of the min end of the min end of the min end of the min end of the min end of the min end of the min end of the min end of the	ent identifie dges for eac to merge c gV) Boruvk	ch connected components using BFS/DFS omponents O(V) a's step
E.g. cannot use DFS/BFS to find longest path  BFS  frontier = {s} while frontier is not empty: next-frontier = {} for each node u in the front for each edge (u,v) in the if v has not yet bean	ier:	Runtime  O(V4E) (adjacency list) — each node is visited once, each edge also visited once  O(V*2) (adjacency matrices) — each node is visited once. Need to check all the postable neighboring nodes  Topological order  Only has browned edges  Only in DAG  Post color DFS  SSP  Bellman Ford	Secretal intensity.  More free Graphs.  Heap  Insert	Union-Find S  Feth Compression without  guite the guite test guite	ummary at relighted union?	the o				Steiner	Each Bo     St     Fit     Sc     There is     Total: O(E     ps (This doe     For every	ore component the min end the	ent identifie dges for eac to merge c gV) Boruvk	th connected components using BFS/DFS omponents O(V)
E.g. cannot use DFS/BFS to find longest path  BFS  frontier = {a} while frontier is not empty: next-frontier = {} for each node u in the front for each edge (u, v) in the	ler: graph:	Rustime  O(V+E) (adjacency list) — each node is valided once, each edge also visited once  O(V+2) (adjacency nations) — each node is valided once. Need to dreak all the  Topological core in englishcency nodes  Only have forward edges  Only have forward edges  Only have forward edges  Only in DAD  Prote color DFS  SSSP  Bellman Ford  Current edde  Current edde  Occurrent edde  Current edde  Occurrent edde  Occur	Becelet intends.  More free Graphs.  Heap  Insert  Add at left-most leaf  Then bubble up until the condition is satisfied  Increase key	Union-Find S  Feth Compression without  guite the guite test guite	ummary at relighted union?	the o				Steiner	Each Bo  St Fit Fit Sc There is Total: O(E ps (This doe For every use Dijks	ore component the min end the min end the min end of the min end o	ent identifie dges for each to merge c gV) Boruvk intee optima calculate the	th connected components using BFS/DFS omponents O(V) is step  I Steiner Tree)  In minimum distance between those vertices —
E.g. cannot use DFS/BFS to find longest path  BFS  frontier = {s} while frontier is not empty: next-frontier = {} for each node u in the front for each edge (u,v) in the if v has not yet bean	ler: graph:	Runtime  O(V=E) (adjacency list) — each node is visited once, each edge also visited once O(V=2) (adjacency matrices) — each node is visited once. Need to check all the propological order Topological order O(V=1) (ADD O(V=	Bacoled Interville.  Where Free Couple.  Heap  Insert  Add at left-most leaf  Then bubble up until the condition is satisfied  Increase key  Just increase then bubble up	Union-Find S Path Compression without  quals tel quals t	ummary at recipited union?  Find  033  000  000  000  000  000  000  0	(hq r) (h, r)				Steiner • Stei	Each Bo  St Fin Fin Sc There is Total: O(E ps (This don For every use Dijks Form an	ore component the min end the	ent identifie dges for ear to merge c gV) Boruvk ntee optima calculate the	th connected components using BFS/DFS omponents O(V) is step  I Steiner Tree)  In minimum distance between those vertices —
E.g. cannot use DFS/BFS to find longest path BFS frontier = {s} while frontier is not empty: next-frontier = {} for each node u in the front for each edge (u,v) in the if v has not yet been frontier = next-frontier  • Steps • Look at the current frontier	ler: graph: visited, add v to next-frontier	Runtime  O(V+E) (adjacency fat) — each node is valided once, each edge also visited once  O(V+E) (adjacency fath-rose) — each node is valided once. Need to check all the  Topological core imagilizationing nodes  Only have forward edges  Only have forward edges  Only have forward edges  Only in DOC  Special core of E/FS  Bellinant Ford  Core to see delle edges  Core to see to see in registre weight cycle — if something still change at the sea the season check in registre weight cycle  Runtime to the see in registre weight cycle  On the see the see in registre weight cycle  On the see the see in registre weight cycle  On the see the see in registre weight cycle	Heap  Insert  Add at left-most leaf  Then bubble up until the condition is satisfied  Increase key  Just increase then bubble up  Decrease key  M	Union-Find S Fish Compression without such that such tha	ummary to weighted usion?  find 031 001 001 0001 00001 00001 000001 000001	exactly V-1	edges)			Steiner  • Stei	Each Bo  St Fit Fit So There is Total: O(E  ps (This doe For every use Dijks Form a n Run MST	ore component the min end the min end of the min en	ent identifie dges for ear to merge c gV) Boruvk ntee optima calculate the	th connected components using BFS/DFS omponents O(V) is step  I Steiner Tree)  In minimum distance between those vertices —
E.g. cannot use DFS/BFS to find longest path  BFS  frontier = {s}  while frontier is not empty:     next-frontier = {}      for each edge (u, v) in the         if v has not yet been     frontier = next-frontier      Steps      Steps     Once the queue in the current frontier      Once the queue in the current frontier	ler: graph: visited, add v to next-frontier	Rustime  O(V=E) (adjacency list) — each node is valided once, each edge also visited once O(V=2) (adjacency matrices) — each node is visited once. Need to check all the provious exploration registrocency nodes  Foliopis have forward edges Only in Nod Only have forward edges Only in Nod Post order DFS  SSP Bellman Ford  Nogative weight edges Can deside edswittence of negative weight cycle — if something still change at the last fuestion, there is negative weight cycle ON ONE ONE	Heap  Insert  Add at left-most leaf  Then bubble up until the condition is satisfied  Increase key  Just increase then bubble up  Concrease key  Just decrease and then bubble down	Union-Find S  Fish Compression without spike and path orders well path orders with path ord	ummary at resigned union?  To ontains see contains see contains cycle	exactly V-1 s v - 1 edger	s			Steiner • Stei	Each Bo  St Fir St There is Total: O(E  ps (This dor For every use Dijks Run MST	ore component the min end the min end of the min en	ent identifie dges for ear to merge c gV) Boruvk ntee optima calculate the n those verti	th connected components using BFS/DFS- omponents O(V) it's step   Il Steiner Tree) minimum distance between those vertices — ces
E.g. cannot use DFS/BFS to find longest path  BPS  frontier = {s} while frontier is not empty: next-frontier = {} for each node us in the front for it has not yet been frontier = next-frontier  • Steps  • Look at the current frontier  • Once the queue in the current for Discovery list)	ier: graph: graph: visited, add v to next-frontier tter is empty, go to the next frontier	Runtime  O(V=E) (adjacency last) — each node is valided once, each edge also visited once O(V=2) (adjacency last) — each node is visited once. Need to check all the Proviological provides registering nodes Transport of the control	Heap  Insert  Add at left-most leaf  Then bubble up until the condition is satisfied  Increase key  Just increase then bubble up  Decrease key  Just decrease and then bubble down  Bubble down to the side that has greater root	Union-Find S  Fish Compression without guide first gash and guide first graphs or grammer and gram	ummary at weighted union?  To ontains are contains are contains cycle x weight et	exactly V-1	s			Steiner • Stei	Each Bo  St  Fin  St  Total: O(E  ps (This do  For every  use Dijks  Form a n  Run MST  topological  sue stores n	ore component of the min end of the	ent identifie dges for ear to merge c gV) Boruvk ntee optima calculate the n those verti	th connected components using BFS/DFS- omponents O(V) it's step   Il Steiner Tree) minimum distance between those vertices — ces
E.g. cannot use DFS/BFS to find longest path BFS frontier = { s} while frontier is not empty: next-frontier = { } for each mode u in the front for each edge (u, v) in the if v has not yet been frontier = next-frontier  • Steps • Look at the current fronter • Once the queue in the current for Bruitme adjacency ist) • O(V=E) • Each vertex is explored on	ier: graph: visited, add v to next-frontier tier is empty, go to the next frontier	Rustime  O(V+E) (adjacency list) — each node is valided once, each edge also visited once O(V+2) (adjacency list) — each node is valided once. Need to discit all the Topological cert imeglibrouring nodes Only have forward edges Only in DAO Prost codes DF3 SSSP  Bellman Ford Och Combine State of the Stat	Heap  Insert  Insert  Insert  Insert  Insert  Insert  Insert  Increase key  Insert  Increase key  Insert  Inse	Union-find Union-find S  And Compression without seasons are seasons as a season are season are season as a season are season are season as	ummary  the registed usion?  To contains see contains see contains cycle  x weight er cut  the registed usion?	exactly V-1 s v - 1 edger dge is not in t	s the MST in the MST	r		Steiner • Stej  Kahn's • For • Que  DYNAMIC 1. Opt	Each Bo  St File File Total: O(E  St Form an Run MST  Topological Sue stores n  PROGRAM imal substr	ore componed the min ere of the min ere of the min ere of the most O(lo log/) as not guarar vertex (v, w) are s'y times are graph from on new graph ordering ode that has MING acture	ent identifie dges for ear to merge c gV) Boruvk intee optima calculate the in those verti h	th connected components using BFSIDFS impropends O(V) is step?  I steiner Tree)  I steiner Tree)  In rimmum distance between those vertices—  ces
E.g. cannot use DFS/BFS to find longest path  BFS  frontier = {a} while frontier is not empty: next-frontier = {} for each edge (u, v) in the if v has not yet been frontier = next-frontier  Steps  Look at the current fronter Once the queue in the current front (note the current fronter)  Authority of the current fronter  Conce the queue in the current front  Runtime (assume adjacency list)  CO 4**  Each vertex is explored on  Can be used to the dipp in being gone the	ier: graph: visited, add v to next-frontier tier is empty, go to the next frontier	Rustime  O(V-E) (adjacency list) — each node is visited once, each edge also visited once O(V-E) (adjacency list) — each node is visited once. Need to check all the provious provides registrocency nodes  Policy in the control of th	Heap  Insert  Add at left-most leaf  Then bubble up until the condition is satisfied  Increase key  Just decrease then bubble up  Decrease key  Bubble down to the side that has greater root  Delete  Swap with the latest leaf node	Union-find Union-find S  Path Compression without the Authority of the Compression without the Authority of the Compression without the Authority of the Author	ummary  the registed usion?  To contains see contains see contains cycle  x weight er cut  the registed usion?	exactly V-1 s v - 1 edger	s the MST in the MST	「 ≋ part of M:	ST	Steiner • Stej  Kahn's • For • Que  DYNAMIC 1. Opt	Each Bo  St File File Total: O(E  St Form an Run MST  Topological Sue stores n  PROGRAM imal substr	ore compone of the min er i) an all nodes at most Ollo clogV) is not guarar vertex (v, w) ra's V times aw graph fron on new grap ordering ode that has MING ucture solution can	ent identifie dges for ear to merge c gV) Boruvk intee optima calculate the in those verti h	th connected components using BFS/DFS- omponents O(V) it's step   Il Steiner Tree) minimum distance between those vertices — ces
E.g. cannot use DFS/BFS to find longest path BFS frontier = {a} while frontier is not empty: next-frontier = {} for each node u in the front for each edge (u,v) in the if v has not yet been frontier = next-frontier  • Steps • Look at the current frontler o Once the queue in the current for Runtime (assume adjacency list) • Q(V+E)  • Each vertex is explored on • Can be used to • Find shortest path	ier: graph: visited, add v to next-frontier titer is empty, go to the next frontier ze crough once	Rustime  O(V+E) (adjacency list) — each node is valided once, each edge also visited once O(V+E) (adjacency list) — each node is valided once. Need to check all the O(V+E) (adjacency matrices) — each node is valided once. Need to check all the Propological continued and propological continu	Heap  Insert  Add at left-most leaf  Then bubble up until the condition is satisfied  Increase key  Just increase then bubble up  Decrease key  Bubble down to the side that say greater root  Delete  Swap with the latest leaf node	Union-find Union-find S Petit Compression without space of the space o	ummary  to respite to second  on open on open on open on open open ope	exactly V-1 s v - 1 edger dge is not in t gge is always outgoing edg	the MST in the MST ge is always	s part of M:  → O(Vlog)	V)	Steiner  Steiner  Steiner  Steiner  Gue  DYNAMIC  Option	Each Bo  St File File So There is Total: O(E  For every use Dijks Form a n Run MST  Topological sue stores n  PROGRAM simal substra a. Optimal sub-prot	ore compone of the min et i) an all nodes at most Olo- licg(V) ra's V times wortex (v, w) ra's V times wordex in min on new grap ordering ode that has MING acture solution can elems	ent identifie dges for ear to merge c gV) Boruvk intee optima calculate the in those verti h	th connected components using BFSIDFS impropends O(V) is step?  I steiner Tree)  I steiner Tree)  In rimmum distance between those vertices—  ces
E.g. cannot use DFS/BFS to find longest path  BFS  frontier = {a} while frontier is not empty: next-frontier = {} for each edge (u, v) in the if v has not yet been frontier = next-frontier  Steps  Look at the current fronter Once the queue in the current for Runtime (assume adjacency list) O(V <sup>4</sup> Each vertex is explored on Each edge is being gone the Can be used to	ier: graph: visited, add v to next-frontier titer is empty, go to the next frontier ze crough once	Rustime  O(V-E) (adjacency list) — each node is visited once, each edge also visited once O(V-E) (adjacency list) — each node is visited once, Need to check all the O(V-E) (adjacency matrices) — each node is visited once. Need to check all the possible registrous nodes  O(V-E) (adjacency list) — each node is visited once. Need to check all the O(V-E) (adjacency list) — each node is visited once. O(V-E) — for some (V-E)  Baltiman Ford  O(V-E) — Registrous de registrous verigits of occurs — if something still change at the last literation, there is negative weight of ock — if something still change at the last literation, there is negative weight of occurs — if occurs on the last literation, there is negative weight of ock — if something still change at the last literation, there is negative weight of ock — if something still change at the last literation of the last literation occurs of the literation occurs	Buobled intervals.  Where Free Graphs.  Heap  Insert  Add at left-most leaf  Then bubble up until the condition is satisfied Increase key  Just increase then bubble up  Decrease key  Bubble down to the side that has greater root  Delete  Swap with the latest leaf node Bubble down the leaf node Extract max	Union-find Union-find S Path Compression without substance of the Compression of the Compres	ummary  A resplict uses  F contains  Be contain  Cycle  X weight et  Be infilmem  Ch node is  Ch node	exactly V-1 s v - 1 edger dge is not in t ge is always outgoing edg added/removisited onco	the MST in the MST ge is always	s part of M:  → O(Vlog)	V)	Steiner  Steiner  Steiner  Steiner  Steiner  For  Que  DYNAMIC  1. Opt	Each Bo  St File There is Total: Q(E  Tota	ore compone of the min et 3) an all nodes at most Olo logV) as not guara- vertex (v, w) ra's V times we graph from on new grap ordering ode that has MING acture solution can dems	ent identifie dges for ear to merge c gV) Boruvk intee optima calculate the in those verti h	th connected components using BFSDFS amounts of the control of the
Eg. cannot uso DFS/BFS to find longest path BFS frontier = {#} while frontier is not empty; next-frontier = {} for each node u in the front for each node u in the front for the first the front for the first the first for the first the first for the first for the first for the first for the first separate separate leach vertex is explored on leach each first is explored on leach each each each each each each leach each each each each each leach each each each each each leach each each each leach each each each leach each each each leach each each leach each each each leach each l	ier: graph: visited, add v to next-frontier titer is empty, go to the next frontier ze crough once	Rustime  O(V+E) (adjacency fat) — each node is valided once, each edge also visited once  O(V+E) (self-cency matrices) — each node is valided once, need to dreck all the  Topological cent are registering nodes  Only have forward edges  Only have forward edges  Only have forward edges  Only in NAC  Post-color Off  Belliams Ford  Can the used with  Can the used with  Can the used with  Radic assistance of negative weight cycle — if something still change at the last feateroic. Only in the innegative weight cycle  Radic Section (In the section of the section	Heap  Insert  Add at left-most leaf  Then bubble up until the condition is satisfied  Increase key  Just increase then bubble up  Decrease key  Bubble down to the side that say greater root  Delete  Swap with the latest leaf node	Union-find Union-find S Path Compression without substance of the Compression of the Compres	ummary  a registed usion?  out  out  out  out  out  out  out  ou	exactly V-1 s v - 1 edger dge is not in t ge is always outgoing edg added/removisited onco	the MST in the MST ge is always	s part of M:  → O(Vlog)	V)	Steiner  Steiner  Steiner  Kahn's  For  Que  DYNAMIC  1. Opt	Each Bo  St File File Sc There is Total: O(f For even use Dijks Form a n Run MST topological substress PROGRAM simal substr so portional substr sub-prot sub-prot portional substr sub-prot su	ore compone of the min et 3) an all nodes at most Olo logV) as not guara- vertex (v, w) ra's V times we graph from on new grap ordering ode that has MING acture solution can dems	ent identifie siges for ear to merge c gV) Boruvk  ntee optima calculate the n those verti h  be constru  be constru	th connected components using BFSDPS improperate O(V) is step?  I steiner Tree)  I steiner Tree)  I steiner Tree)  I steiner Tree  I steiner T