1. 二叉查找树 (Binary search tree)[logn]: 左子树上的所有结点小于根节点小于右子树，插入的结点一定是叶结点。

2. 红黑树 (Red-black tree)[logn]: 特殊的二叉查找树，接近平衡二叉树，2倍，1.红黑2.根节点黑色3.叶子结点黑色4. 结点红色，则子节点必须黑色5.路径上的黑结点数相同。

3. 平衡二叉查找树(Balanced binary search tree)[logn]: AVL, degeneration(有序), 结点的左右子树高度高度差不超过1. 左旋，右旋以平衡。(特殊: 先左旋后右旋，或者反之)

4. B 树 (B tree)[mlog_m^n]: 平衡多路查找树 balanced multi-way search tree, 2-3和2-3-4是特例。m阶表示最多m个子节点。更矮为内外存数据交互，减少访存次数。B树是将各种信息保存在所有节点中的。B+树是将各种信息保存在叶子中的。所有叶子结点都位于同一层。m/2-1~m-1个key。以结点中间的key分裂并上移。

5. B+树[O(logN)]: 关键字数=子结点数, 添加子结点之间的连接。m/2*log(m)n=log(m^1/m)n=logn, 对于n来说，m是常数。层级更低，因为只存储最大值, IO 次数更少; 每次都会查到叶子结点，更稳定; 叶子结点形成有序链表，范围查询更方便，减少内存跳跃。

6. 2-3树，2-3-4树[O(log_3/4^n)~O(logn)]: 每个中间结点都有2-3或者2-3-4个子结点。插入需要升级父结点。

7. 堆 (heap)[建堆O(n), 调整O(logn)]: 节点不大于或者不小于父结点的值。使用向上调整策略创建堆（只需将子结点中较大的那个和父结点比较）。

8. 冒泡排序(bubble sort)[O(n^2)]: 比较相邻元素，交换位置, 类似选择排序, 后者不需要额外的交换位置

9. 插入排序(insertion sort)[O(n^2)]: 不断比较未排序部分的第一个与排序部分的内容，插入到合适位置，费时在交换位置。与冒泡排序不同于不需要找到最大值，只需要找到合适位置即可。

10. 希尔排序(shell sort)[O(n^1.3)]: 一种插入排序，不过是进行了分组，利用分治思想，将数据按照gap分组

11. 选择排序(selection sort)[O(n^2)]: 从未排序部分选择最大的，放到排序部分的顺次的位置上。

12. 快速排序(quick sort)[O(nlogn)]: 选择一个pivot，以它为标准划分list，将小的放左边，大的放右边，递归地完成pivot两边的sublist（选择new pivot）

13. 归并排序(merge sort)[O(nlogn)]: 递归地将list分解到最小, merge and sort adjacent elements

14. 堆排序(heap sort)[O(nlogn)+O(n)]: 构建大顶堆，删除堆顶(heap top)，更新堆，进行迭代

15. 计数排序(counting sort)[O(n+k)]: 有确定范围的整数，设置一个counting list，以出现的数字为index的elements进行计数

16. 基数排序(radix sort)[O(d(n+k))]: 按照位数places of the integer，完成计数排序，注意按照FIFO恢复数据

17. 桶排序(bucket sort)[O(n)+O(m*n/n*log(n/m))]: 使用function将数据map到k个桶中，每个桶按照comparison method 排序

18. K路归并败者树(K-way merge loser tree)[O(N*logK)]: 应用于external sorting，只有叶子结点是数据，内部节点是比赛的败者，使用胜者与下个进行比赛。根结点记录败者，额外的空间记录最终胜者。每个叶子结点连接一个外部数据sequence (已经排序好的)，每个sequence最后数据是较大的。

19. 二分查找(binary search)[O(logn)]: 分治法，就是完全二叉查找树，需要有序的列表, 查中间点。

20. 大整数乘法(large integer multiplication)[O(n^log3)]: 分治法，二分两个integer，使用三个小规模乘法代替原本的乘法

21. Strassen 矩阵乘法(Strassen matrix product)[O(n^log7)]: 分治法，四分两个matrix，使用七个小规模矩阵乘法代替原本的乘法

22. 最接近点对(the planar closest pair)[O(nlogn)]: 分治法，划分plane为两个部分，考虑左边、右边以及横跨(cross)两个部分这三种情况

23. 最长公共子序列(the longest common subsequence)[O(mn)+O(m+n)]: 考虑两个小子段，两个的最后一个元素相同则计数+1，否则考虑两个子段分别减1之后的common subsequence中的最长的那个，使用二维表完成记录。动态规划需要找到1.目标: 价值、和等，2. 自变量: 容量、长度等，1随着2的变化而变化

24. 最大子段和(maximum interval sum)[O(n)]: Enumerate all intervals, 分治法包含三种情况(第三种的左部分以中间节点结尾，右部分以中间节点开始)，动态规划生长interval，只是需要判断当前sum是否小于0，如果真的话舍弃，从下一个重新sum，否则继续。dp[i] = max(dp[i−1] + num[i], num[i])

25. 背包问题(knapsack problem)[O(nC)]: 递归，放置后一个object会减少前一个物体的可行的容量，选择两者中最大的那个作为当前的值(memory search)。动态规划，对于中间的任意一个物体状态转移方程(state transition equation)与递归相同，自底向上直到容量为C

26. 计数问题(counting problem): 解决这个问题有多少种方法

27. 斐波那契数列(Fibonacci sequence)[O(n)]: f(N)=f(n-1)+f(n-2)，直接按照状态转移方程计算下一个数值

28. 凑零钱(Collect change)[O(n)]: f(x)=min{f(x-coin[1]), f(x-coin[2])…}+1，按照状态方程计算下一个目标钱数

29. 爬楼梯(climb stairs)[O(n)]: f(x)=f(x-1)+f(x-2)，直接按照状态转移方程计算下一个阶梯数的可行方式

30. A-B path[O(mn)]: 总数C_(m-1+n-1)^(m-1)，f(m,n)=f(m,n-1)+f(m-1,n)，两种方式到达某个点，二维爬楼梯

31. A-B path with obstacle[O(mn)]: 判断是否存在obstacle，不存在就正常，若存在有obstacle的位置f(i,j)=0

32. 二叉搜索树(different binary search tree)[O(n^2)]: 考虑不同的数字作为根节点，然后考虑不同的左右子树，f(n) = f(0)f(n-1)+f(1)f(n-2)…

33. Dijkstra algorithm[O(n^2), O((m+n)logn)]: 要有松弛函数(relax function)，每个权重必须为正数，计算未标记节点到起点的可达距离，取最短点并标记，标记后的结点无法更新。Core: 部分最短路径是较长的最短路径的一部分，删和改最小堆均需要logn调整。父结点为前驱结点 (predecessor, successor)。对于n-1个未标记结点，均需要选出最近点、删除最近点、更新与最近点邻接的节点的值(k=m/n)

34. Bellman-ford algorithm[O(nm)]: 可以有负值权重，但不能有负回路，对边集进行逐个判断，添加这个边是否可以使结点的距离更新（松弛），进行n遍，Core: 每次松弛迭代都能找到一个最优的边加到最短路径集合中。

35. Floyd-warshall algorithm[O(n^3)]: 可以有负权边，但是不能有负回路；使用两个同样大小的矩阵，记录距离以及中间节点。动态规划，对于每个节点，判断这个结点是否能够加入到任意两个节点对之间能否让两个结点距离变短，遍历所有结点。Core:部分最短路径是较长的最短路径的一部分

36. Kruskal algorithm[O(E(logV判断边对应的两个点是否在同一个树上并合并+logE弹出最短边)]: minimum spanning tree 定点选边，贪心地选最短的, 并查集(union find)(查：logV，并: logV，路径压缩: logV)

37. Prim algorithm[O((V+E)logV) 弹出并修改距离最近点,类似dijkstra]: 贪心地按边选点，距离当前生成树最短的点，稠密图有优势

38. N后问题(n-queen problem)[O(n!) factorial剪枝优化]: 回溯(backtracking algorithm)，定义并找到易于搜索的解空间(solution space)、dfs+剪枝。Row、column、diagonal，逐行放置、判断是否冲突、并回溯

39. 图的着色问题(graph coloring problem)[O(m^n)]: 点着色，边着色要求最大度+1色，回溯法，找到解空间，尝试每个点的着色，不通则剪枝。回溯法相当于返回父结点dfs

40. 旅行售货员(travelling salesman problem)[O(n!)]: 经过每个顶点正好一次，总权值最小。回溯法，构建解空间，dfs遍历，不通则剪枝

41. 匈牙利算法(Hungarian algorithm)[O(VE)]: 二分图最大匹配(bipartite graph maximum matching)、最小点覆盖(minimum vertex cover)。Core: 增广路径(augmentation path)，分成两组，尝试连接，成功下一个，不成功回溯修改上一个连接。不在匹配中的右边的vertex构建交错路(staggered path)，左边在交错路中的点+右边不在交错路中的点。

42. 图的深度优先遍历(Depth first search)[O(V^2)邻接矩阵, O(V+E)栈+邻接表]: 右手原则(right-hand principle)，利用栈(stack)与递归(recursive)，树的先序遍历、中序遍历、后序遍历都是深度优先遍历。理解为图可以变形为树，利用先序遍历。同下，每个结点只需要判断一次邻接边。

43. 图的广度优先遍历(Breadth first search)[O(V^2)邻接矩阵, O(V+E)队列+邻接表]: 与树相同，按层遍历，利用队列，访问入队，出队时入队子结点

44. 无向图的连通性(Connectivity of undirected graph): 任意两个点都连通(connected graph/unconnected graph)，(最小)点割集(minimum vertex-cut set)=点连通度(vertex connectivity)，(最小)边割集(minimum edge-cut set)=边连通度(edge connectivity)

45. 有向图的连通性(Connectivity of directed graph): 基图(base graph)去除方向的图连通-弱连通(weakly connected graph)，单向联通(unidirectional connected graph)(存在通路)-任意两个顶点之间至少能可达(reach)一个，一定是弱连通图，强连通图(strongly connected graph)(存在回路)-均互相可达。

46. 次小生成树(subminimum spanning tree): m次kruskal算法，每次删除最小生成树的一个边，找到新的最小生成树，次小生成树与最小生成树只差一条边

47. 第K短路径(k-th shortest path): 贪心算法，找到每个节点到终点最短路径，使用BFS找到起点到中间点的路径，计算所有到中间点的路径+中间点到终点的最短路径，完成所有中间点的判断即可

48. 朴素字符串匹配(naïve string matching)[O(MN)]: 直接逐个匹配，brute-force

49. RK算法(Robin-karp algorithm)[O(M+N)]: 改进BF算法，hash字符串，取长度=M的子串并hash，比较，减少了逐个比较字符的复杂度，优化hash函数，将子串的hash降低到线性时间，M为hash相等时比较字符

50. KMP algorithm[O(n+m)]: 最长公共前后缀(the longest common substring of prefix and suffix)，next array对应pattern string 的每个位置，表示下次需要和主串指针所指的位置进行比较的pattern string的位置=the length of the longest common substring of prefix and suffix.

51. BM 算法(boyer-moore algorithm)[最好O(N/M)最坏O(MN)]: 从后往前，1.查找坏字符，对齐2.查找好后缀对齐3.查找最长好后缀对应的前缀，对齐。

52. Hash: hash function, hash conflict (chaining, open addressing, linear probing, quadratic probing, double hashing, re-hashing), loader factor，扩容为两倍, hash size，Capacity expansion mechanism，key-value-bucket

53. Bit map: 大规模数据，状态不多，可以用于排序查找

1. 导数: derivative
2. 链式法则: chain rule
3. 驻点: Stationary Point
4. 极小值点: minimum point
5. 极大值点: maximum point
6. 鞍点: Saddle point
7. ∂: partial
8. ∇: nabla
9. 泰勒公式: Taylor formula
10. 梯度下降: gradient descent，使用一阶泰勒展开来近似值
11. 一阶泰勒展开: first order Taylor expansion
12. 一元函数: function of one variable
13. SGD: Stochastic 有最简单的SGD，然后加入了动量的思想的Momentum，然后是在Momentum中加了小聪明的Nesterov，再然后动态调节学习率的AdaGrad， 然后把前面的历史信息逐步遗忘的RMSprop，然后结合了Momentum和动态调节学习率同时把前面的历史信息逐步遗忘(forget)的Adam。优点是相对于BGD，无需考虑整个数据集，只需要一个minibatch，但是容易震荡(fluctuate)，增大batch
14. 损失函数: loss function
15. 交叉熵: cross entropy (sigmoid, softmax)
16. 前向传播: forward propagation
17. 反向传播: backward propagation，通过链式法则构建一个反方向的网络：先求中间节点梯度，然后分别计算Wb的梯度，W的梯度=线性中间节点的梯度*上一层的激活有关，b=线性中间节点的梯度
18. 卷积: convolutional operation, full, valid, same
19. 池化: max pooling, average pooling
20. 浅层神经网络: shallow neural network
21. 残差: residual /ri' zidjual/
22. 批归一化: batch normalization, μ, sigma, alpha, gamma, 多个样本对应的channel进行BN, 在激活之前
23. 方差: variance, 标准差 standard deviation
24. 退化: degeneration
25. 锚点: anchor，卷积核中与输入中的左上角的数据第一次进行计算的位置
26. valid卷积核的梯度: the convolutional gradient of kernel, F对输出求导，并用它对输入做卷积
27. valid输入的梯度: the convolutional gradient of input, F对输出求导, 并用旋转卷积核对它做卷积
28. same卷积核的梯度: F对输出求导，使用部分输入与它做卷积
29. same输入的梯度: F对输出求导，并用旋转卷积核对它做卷积
30. 平均池化的梯度: 线性梯度
31. 最大池化的梯度: 只有池化输出选择的那个值有梯度，其余为0
32. BN层的梯度: 将BN的计算展开，然后按照反向顺序计算梯度
33. 中间变量: intermediate variable

Good afternoon, professors. It is my great honor to take part in this interview. Let me start the introduction of myself. My name is Wen Xiangyu, 24 years old. I received my bachelor's degree from the University of Electronic Science and Technology of China (UESTC) in 2019. And I am now studying at the same university as a master's student in grade three and will receive my master's degree in June of this year. During my master's degree, my major is software engineering, my research direction is about deep learning, neural networks, and AI security. And I won the China national scholarship and some other awards during this period. If I get the chance to study at CUHK, I will try my best to gain more professional knowledge to enrich myself, forming a comprehensive view of my research field and laying a solid foundation for my future career. Thanks for your attention! Thank you!