|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学号：  **Chapter 1 緒論**  □級數   1. 算術級數與末項平方同階：1+2+…+n=O(n2) 2. 幂方級數比冪次高一階：Σ0nkd=O(nd+1) 3. 幾何級數與末項同階：Σ0nak=O(an) 4. 調和級數：1 + 1/2 + 1/3 + … + 1/n = log(n) 5. 對數級數：Σ1nln(k)=O(nlog(n)) 6. 卡特蘭遞推關係：T(n) = Σ0n-1 T(k)·T(n-k+1) 7. 卡特蘭數：Catalan(n) = (2n)! / (n+1)! / n!   □估算   1. 1天=105 sec；1生=1世紀=3x109 sec 2. 三生三世=3x1010 sec；宇宙大爆炸至今：4x1017 sec   **Chapter 2 向量**  分攤分析：足夠多次，總時間/總次數；平均分析：概率加權  □向量   1. 複雜度   ⋄遞增策略：擴容m次，複製m次需O(n)  ⋄加倍策略：分攤成本O(1)   1. 唯一化：重複者跳過，不重複者向前移，一次刪除尾部。   ⋄Ω(nlog(n))，可歸約為序列中是否有重複元素的命題。   1. 二分查找：O(log(n)), 劃分方式：[lo,mi-1],[mi,hi-1]   ⋄實際為O(1.5log(n))，這是右探需要兩次比較。  ⋄記成功查找次數為S，失敗為F。則有(S+1)n=F(n+1)   1. Fibonacci查找：Φ=0.618（漸近複雜度O(log(n))） 2. 插值查找：猜測軸點，O(loglog(n))   ⋄mi=lo+(hi-lo)(e-A[lo])/(A[hi]-A[lo])  □排序   1. 起泡排序：穩定，最好O(n)，最差O(n2) 2. 歸併排序：穩定，O(nlog(n))，佔空間   □Bitmap   1. 小集合，大數據去重。（如篩法） 2. 快速初始化(Hopcroft)：校驗環T[F[k]]=k, F[T[k]]=k   ⋄初始化只需top=0即可，O(1)。  **Chapter 3 列表**  □插入排序   1. 不斷將元素插入排好序的前綴中，最好O(n)，最壞O(n2)   若逆序對間距不超過k，則運行時間O(kn)   1. 後向分析：平均比較次數。   ⋄E=1+Σ0r k/(r+1)=1+r/2, 總次數：Σ0n-11+r/2=O(n2)  ⋄若有I 個逆序對，關鍵碼比較次數不超過O(I)  □逆序對   1. Bubble Sort：操作數＝逆序對數，插入：O(n+I) 2. 歸併排序計序逆序對數。   □列表的游標實現   1. 同一行中，link指示其後繼，elem指示其數值。   **Chapter 4 棧與隊列**  □遞歸的分類   1. [[[[尾遞歸]線性遞歸]二分遞歸]多分支遞歸] 2. 消除遞歸的目的：空間複雜度取決於遞歸深度。 3. 顯示的調用棧可以常數意義上優化空間，但有用。   □棧用途   1. 進制轉換：矩除法由低到高壓入棧中，反向輸出。 2. 括號匹配：遇「(」入棧，遇「)」出棧。 3. 棧混洗：S(n)= ΣS(k-1)S(n-k)=Catalan(n)=(2n)!/(n+1)!/n!   禁形：i<j<k，則無{k..i..j}，稱為915禁形（其它：615禁形）   1. 中綴表達式求值：棧＋線性掃描，棧頂可優先計算時，先退棧計算，再入棧：當前字符入棧，轉下一字符。 2. 逆波蘭表達式求值：（J. Lukasiewicz）   RPN>中：手動加括號，中>RPN：運算符替換「) 」，消「(」。  給出x個左括號，棧規模最大為4x+1（組合：\0+\*^(…+\*^!）  □隊列   1. 與堆結合：Stack+Heap - push, pop, getMax O(1)   P中每個元素都是S對應後綴最大者：P.push(max(e,P.top()))  Queue+Heap – enqueue, dequeue, getMax 每個都放O(n)   1. 雙棧當隊：若上棧為空，下棧pop，順序push到上棧   分析：Accounting：一個元素最多4次操作，O(n)=4n  Aggregate：d+e, T≤4d+3(e-d)=3e+d, O(n)≤3n  Potential: 設Φ=R-F（下規模減上規模，則2n=ΣT+Φn-Φ0）   1. 直方圖最大矩形：只需確定左側第一個比r小的值和右側第一個比r小的值即可確定以H(r)為高的最大矩形。   棧：保留前綴最小元素, S(r), T(r) 各掃一遍，2\*O(n)  One-Pass Scan: 棧必遞增，遞減者被pop，每次計算更新最值：H[top](t-s)  **Chapter 5 二叉樹**  □二叉樹實現   1. 更新高度：O(1)，更新祖先：O(h)，高度不變可停止。 2. 先序遍歷：遍歷時右子先入棧，左子後入棧。分攤O(1) 3. 中序遍歴：p次pop，m次push，操作數有(m-p)+2\*p   迭代次數=pop次數, m=pop=n時停止，整體O(n),分攤O(1)  後繼當前驅：右孩左子，左孩右子，suss() 總調用時間O(n)   1. 後序遍歷 2. 層次遍歷：借助隊列，最大規模⌈N／2⌉ 3. 重構：[先|後]＋中，前者找根，後者分解遞歸。   □Huffman 樹   1. 出現頻數越小者越靠左，整體左傾。 2. 構造：左式堆合併，優先隊列。O(n)+merge O(log(n)) 3. 按頻率排列入棧O(nlog(n))+維護有序隊列O(n)   **Chapter 6，7 圖**  □表示方法   1. 鄰接矩陣O(n2)，關聯矩陣O(ne)=O(n3)，空間利用率2e/ne 2. 鄰接表：空間O(n+e) 有向O(n+2e) 時間尚可   □搜索   1. 廣度優先：初始化O(n+e)，内循環ΣO(1+deg(v))=O(n+2e)。 2. 深度優先：活躍期fTime[u]-dTime[u], 完全包含為爺孫關係   DFS 森林的向前邊、向後邊和跨越邊都可能不同。   1. 拓撲排序：零入度算法：順序輸出零入度（用隊）   零出度算法：逆序輸出零出度（用棧）   1. 優先級搜索：O((n+e)\*log(n))   □應用   1. Prim求最小支撐樹：極短跨邊（但可左右横跳） 2. Kruskal求最小支撐樹：從小至大試邊，安全則加入。   等價於給定互不相交等價類，然後Union-Find合併。   1. Dijkstra求最短路：所有臨時節點找最小。O((n+e)\*log(n)) 2. 雙連通分量：邊雙連通具有傳遞性，但點雙連通沒有。 3. 並查集：路徑壓縮（矮者接入高者）O(mlogHm/nn) | **Chapter 8 二叉搜索樹**  □BST   1. 中序遍歷單調非降。 2. 插入、刪除、查找操作，線性正比於深度。 (不超過樹高） 3. 刪除：與直接後繼交換，刪除葉節點 4. SearchAll：等價於搜索區間 (e+ε, e-ε) 分別向左(右)深探 5. 有n個節點的BST：   ⋄有n! 種排序等概率出現，平均高度Θ(log(n))  ⋄種數為對應的卡特蘭數，等概率出現平均樹高Θ(sqrt(n))  □BBST   1. 任一 BST 轉為最左分支：n-1次旋轉（最左路有右子,zag） 2. 至多經過2n-2次操作，即可等價轉換兩棵BST   □AVL   1. 平衡因子絕對值不大於1，高度不超過O(log(n))，每種操作複雜度均為O(log(n))，儲存空間O(n) 2. 插入：單旋 / 雙旋（先旋轉父再到祖父），O(1)調整。   ⋄可能使多個祖先失衡（最低者不低於祖父)   1. 刪除：最多 log(n) 次（旋轉後高度可能降低）   ⋄至多一個祖先失衡（決定節點高度的孩子高度不變）   1. 任一葉節點深度不小於⌈h/2⌉（數學歸納法證明） 2. 順序加入2h+1-1個關鍵碼得到高度為h的滿樹。   ⋄考慮：{0}, {2h+1-1}, ⋄3\*2h-1⋄, {3\*2h}, {2h+2-1}  **Chapter 9 BST Application**   1. 區間查找：輸出敏感（不可忽略輸出數r）  |  |  |  |  | | --- | --- | --- | --- | |  | 1D-Tree | KD-Tree | Multilevel ST | | 建樹 | O(nlog(n)) | O(nlog(n)) | O(nlog(n)) | | 查找 | O(log(n)) | O(r+n(1-1/k) ) | O(r+log2n) | | 規模 | O(n) | O(n) | O(nlog(n)) |   □KD-Tree   1. 記Q(n)規模為n的子樹中與查詢區域邊間相交的子區域的節點總數，則有Q(n)=2+2Q(n/4)=O(sqrt(n))   ⋄考察2x2情況，每一條直線最多穿過其中兩個區間。   1. 由1知，Kd search運行時間為 O(r+sqrt(n))   □Interval Tree, Segment Tree, Priority Tree 都是O(n)空間的。  **Chapter 10 高級搜索樹**  □Splay Tree（R. E. Targan & D. D. Sleator）   1. 單層伸展樹Ω(n^2)，分攤複雜度O(n) 2. 雙層伸展，zig-zig/zag-zag 時先轉祖父後轉父（樹高減半） 3. 插入/刪除：尋找節點後只需要在根節點操作即可。 4. 所有接口分攤複雜度O(log(n))   ⋄勢能分析法：取勢能函數Φ=Σ(v∈S) log|v|，|v|後代數目  ⋄A=T+Φ, 每步調整不超勢能變化3倍（A≤3[Φ’(v)- Φ(v)]）   1. 適用於局部性強、命中率高，此時O(log(k)) 2. 連續m次查找：O(mlog(k)+nlog(n))   □B-Tree（R. Bayer & E. McCreight）   1. 內存速度：磁盤/內存>10^5；磁盤讀1B與1KB一樣快。 2. 外部節點深度相同(h)，葉節點深度(h-1) 3. (n,m)-樹，（m階b-樹：(⌈m/2⌉, m)-樹）對每一個節點：   n-1≤關鍵碼數≤m-1，n≤分支數≤m（根節點至少為2）   1. 樹高：logm(n+1)≤h≤log⌈m/2⌉⌊(n+1)/2⌋+1 2. S 次分裂 M 次合併，則S-M=n-h恆成立。 3. 查找：順序查找/下探 ，複雜度O(log(n)) 4. 插入：分裂上溢，最壞情況有Ω(logm(n))次分裂。 5. 刪除：與直接後繼交換（最底層），旋轉調整   ⋄兄弟可借節點，則旋轉。否則父節點下移，兄弟合併。  ⋄B-樹分裂與合併總次數是O(n)的，故分攤意義下O(1)   1. B\*-Tree：聯合分裂，k個兄弟共同分擔溢出值，空間利用率由50%提升至k/(k+1)。   □紅黑樹（L. Guibas & R. Sedgewick）   1. 併發性：插入刪除不超過O(1)，持久性：支持歷史版本 2. 紅黑樹等價於4階B-樹（(2,4)-樹）。   樹高：⌊log2n⌋≤h≤log⌈4/2⌉⌊(n+1)/2⌋+1=log2(n+1)  黑高度：⌈log4(N+1)⌉≤d≤⌊log⌈4/2⌉⌊(n+1)/2⌋⌋+1   1. 規則：(1)樹根、(2)外部節點必黑，(3)紅節點父子皆黑。   (4)外部節點：黑深度相等，等於全樹黑高度。   1. 插入：插入節點v必為末節點，非根則染紅（根為黑）   ⋄雙紅修正：考察叔父節點u, 父節點p，祖父節點g。  ⋄RR-1: u黑, 顏色改為RBR再拆開。  ⋄RR-2: u紅, 則p, u 染黑，g染紅，向上繼續修正。   |  |  |  |  | | --- | --- | --- | --- | | u | 旋轉 | 染色 | 後續 | | RR-1 黑 | 1-2 | 2 | 調整完成 | | RR-2 紅 | 0 | 3 | 可能雙紅，上升兩層 |   時間O(log(n))，重染色O(log(n))，旋轉O(1)。   1. 刪除：交換，然後刪去直接前驅或後繼。不一定滿足(3)(4)   先考察被刪除節點x, x的右子r。（父-子-孫排列：p-x-r）  若x, r其一為紅，令r變黑接入原樹即可（即p-r）  若均為黑（等價於B-樹下溢），此時變為：  ⋄雙黑修正：考察p的另一孩子s（x的兄弟）。  ⋄BB-1: s黑，若s有紅孩t，則旋轉令s為p, t 之父。  ⋄BB-2R: s及其子皆黑, p紅，則p轉黑，s轉紅。  ⋄BB-2B: s及其子皆黑, p黑，則s轉黑，向上修正。  ⋄BB-3: s紅，旋轉令s為p之父。變為BB-1, BB-2R   |  |  |  |  | | --- | --- | --- | --- | |  | 旋轉 | 染色 | 後續 | | BB-1黑s 紅t | 1-2 | 3 | 調整完成 | | BB-2R 紅p | 0 | 2 | 調整完成 | | BB-2B 黑p | 0 | 1 | 雙黑，上升一層 | | BB-3 紅s | 1 | 2 | 變為(1)或(2R) |   時間O(log(n))，重染色O(log(n))，旋轉O(1)。   1. 分攤意義下染色節點數不超過O(1)。   ⋄勢能函數Φ(s)=2\*BRR(S)+BBB(S)，初值為零，恆為非負。  （BRR 表示狀態 S 下有兩個紅孩的黑節點總數）  ⋄插入或刪除可使Φ(s)至多增大c，每次染色使Φ(s)減少1 | 姓名： | **Chapter 11 詞典**  □散列表   1. 保持查找速度，減少空間。 2. 裝填因子：λ=N/M，數值越大衝突越多，效率降低。   □散列函數   1. 評價準則：確定、快速、滿射、均匀。 2. 除餘法：hash(key)=key%M（M為素數最均匀？）   ⋄問題：不動點0，高階均匀性。   1. MAD 法：hash(key)=(a\*key+b)%M   ⋄需要使 M, a互素，保正散列的隨機和均匀性。  ⋄Hack: 從空開始插為M 個間隔T 的值，則每個關鍵碼與 g=gcd(M, T) 個關鍵碼衝突，空間利用率不超過1/g。  ⋄所以取M=2k不好，相當於二進制取後k位。   1. 其他：數字分析，平方取中，折疊，位異或，多項式，循環移位，（偽）隨機數法   □衝突排解策略   1. 開放散列：   多槽位法：桶拆分為多個槽位，槽位較少時仍為O(1)。  獨立鏈：空間不連續，系統緩存失效。  公共溢出區：衝突者順序存入，效率正比於規模。   1. 封閉散列（開放定址）   線性試探：直至命中或抵達空桶，刪除需加懶標記。  平方試探：前⌈M/2⌉個桶必互異（證明02,… 模M同餘類）  雙向平方試探：取素數M=4k+3，則前M個桶必互異。  ⋄利用恆等式：(u2+v2)(s2+t2)=(us+vt)2+(ut-vs)2   1. 再散列：hash1衝突，則取[hash1(key)+hash2(key)\*n]%M 2. 重散列：裝填因子過大，移動至新表。   □桶排序   1. 時間O(n)，空間O(m)，空桶為0，否則為1。 2. 輸出需時O(n)，允許值重複時空間為O(m+n)。 3. MaxGap：左閉右開n-1個桶，記錄區間左右點，O(n)遍歷找出相鄰非空桶距離的最大者。   □基數排序   1. 從低到高位桶排序，若鍵值有t位則算法為O(t\*(n+m))，其中m = max{m1, m2, …, mk} 為第k位取值範圍。 2. 整數排序：給定n個[0,dn]的數。元素轉換為n進制，則複雜度為 O(d\*(n+n))=O(n) 。   □計數排序   1. 小集合、大數據：n個數，m個桶，n>>m，O(n+m+n)。   O(n)遍歷所有數，對應桶計數器加1 。  O(m)遍歷所有桶，以前綴和記數。（即H(M+1)+=H(M)）  O(n)由後而前遍歷數組輸出。（相應計數器減一）  □跳轉表（William Pugh）   1. 橫為層，縱為塔。塔高符合幾何分布P(h=k)=pk-1(1-p) 2. 期望空間：E(|Sk|)=n\*2-k, E(S)=nΣk2-k<2n=O(n) 3. 縱向跳轉次數/ 查找時間累計不過expected-O(log(n))。   **Chapter 12 優先級隊列**  □完全二叉堆   1. 只查找極值元，無需維護全序關係，只需保留偏序關係。 2. 邏輯等價於完全二叉樹，實現上偏向緊湊排列的向量。 3. 插入：逐層上濾，O(log(n))。（期望上升1層，幾何分佈）   ⋄先把祖先下移再插入值，可以減少swap的次數。  ⋄O(loglog(n))次比較：路徑上祖先有序，可二分查找位置。   1. 刪除：與堆尾元素交換，由根開始下濾。O(log(n)) 2. 批量建堆：蠻力，自上而下的上濾，O(nlog(n))。   ⋄Floyd: 自下而上的下濾，成本正比於高度，O(n)。  □堆排序（利用大根堆實現選擇排序，O(nlog(n))），不穩定。   1. 就地初始化，不斷調用delMax函數。   □錦標賽樹   1. 勝者樹：空間O(n)，構造O(n)，更新勝者祖先即可。 2. 敗者樹：内部節點記錄敗者，根的父節點為冠軍。   □多叉堆（d-heap）   1. 優先級隊列：適用於優先級搜索，O((n+e)\*log(n)) 2. 上濾O(logd(n))，下濾O(d\*logd(n)) 3. PFS效率：O((nd+e)logd(n)), d=e/n+2最優   注：三叉堆比二叉堆更快。  □Fibonacci堆/ 左式堆（Crane）   1. 效率與右側藤長成正比。⋄引入外部節點，真二叉樹。 2. Null Path Length(NPL)：到外部節點最近距離。   ⋄滿足 npl(lc)≥npl(rc)，npl(x)=1+npl(rc)，子堆也是左式堆   1. 右側鏈：終點為全堆中最淺節點。   若npl(rc)=d，則至少有2d-1個內部節點，d≤O(log(n))   1. 合併：沿右藤合併，確保左子堆npl不小，右边值不小。 2. 插入：相當於左式堆與單節點組成的左式堆合併。 3. 刪除：合併堆頂刪除後剩餘的兩個堆即可。 4. 任給高度g和h的兩棵AVL樹S和T，S值均不大於T，在O(max(g,h))時間内合併：找出T中最小元素m，記摘去m的樹為T’。找出S中高度不低於h’的樹S’，S\S’中最右節點連接m，m連接S’與T’，然後上濾m。   **Chapter 13 串（核心：模式匹配）**  □KMP（Knuth, Morris, Pratt）   1. next表：next[j] 表示 [0,j-1] 中最長匹配前後綴。   ⋄再改進：若P[j]=P(next[j]), 則next[j]=next[next[j]]   1. 分攤分析：O(m+n)，觀測量：k=2i-j, i為做過的成功比對數, i-j不少於做過的失敗比對數。注意有k≤2n-1 2. 適用範圍：單次匹配概率大，字符集小（二進制串）   □BM算法（Boyer, Moore）   1. BC（Bad Character）策略：從末字符開始，自後向前掃描：每次或得到匹配的後綴，或因失配得到壞字符，bc策略可以找到能跟壞字符匹配的位置，否則後移一位。   ⋄為何不找左側最右的相同字符？徒增麻煩，邏輯更複雜  ⋄BC 相當於找出每個字符在字串最後出現的位置。   1. BC性能：單次匹配概率越小，Patten越長效能越高。   ⋄最好O(n/m)，後綴第一個字母就不匹配。  ⋄最壞O(nm)，匹配到最前才不匹配。   1. GS（Good Suffix）策略：   ss[j] 表示P[0,j] 與P 的某一後綴匹配的最長者。  ⋄從後至前掃描只需O(m)時間。  ss[j]=j+1, 對任意i<m-j-1，m-j-1必是gs[i]的一個候選。  ss[j]≤j，m-j-1必是gs[m-ss[j]-1]的一個候選。   1. BM-GS性能：最好O(n/m)，最壞O(n+m)，空間O(|Σ|+m)   □Karp-Rabin 算法   1. 利用散列壓縮Patten，比較複雜度降為O(1)。   □Trie 樹（鍵樹）  1. 利用指針快速定位，匹配字符。 | 班级： |  |

**Chapter 14 排序**

□快速排序

1. LUG版: 勤擴展懶交換
2. 隨機選取一個軸點，整體O(n)時間，O(1)空間。

⋄收縮L,G，移動非法點，此時排序不穩定。

1. 性能：平均划分O(nlog(n)), 取最值為軸點O(n2)。
2. 比較次數：

遞推T(n)=(n-1)+1/n\*Σ T(k)·T(n-k+1)

後向分析：Σ Σ 2/(d+1) = Σ 2(ln(j)-1)≤2nln(n)

⋄ai, aj 接受比較，當且僅當其中一個被確認。

1. DUP版: 大量重複元素，勤交換懶擴展。（左右向中心）
2. LGU版: 也不穩定。

□選取

1. 眾數：若有一半以上元素同為m，謂之眾數，減而治之。⋄若在向量前綴P中, x出現次數恰佔一半，僅當對應的後綴有眾數m。
2. 歸併向量的中位數：O(log(min{n1,n2}))

⋄m(S1∪S2)=m(S1.suf⌈n/2⌉)∪m(S2.pre⌈n/2⌉)

1. 第k小元素: 選取第k元素為軸點，遞歸查找。

⋄T(n)=(n-1)+1/2Σ0n-1max{T(k),T(n-k-1)}

≤(n-1)+2/n\*Σn/2n-1T(k) ≤(n-1)+2/n\*Σn/2n-1(4k)<4n

1. LinearSelect: T(k) =cn+T(n/Q)+T(3n/4)

□希爾排序（D. L. Shell）

1. 序列視作矩陣，按列排序。
2. 輸入敏感性排序：插入排序。
3. 步長序列H={h1=1, h2, …, hk}，矩陣寬度逆向排列而成。 Shell’s Sequence {2k}: Ω(n2/4)

PS {2k-1}: 外循環O(log(n)), 排序O(n(3/2))

Pratt {2p3q}: O(nlog2(n))

Sedqewick {9\*4k-9\*2k+1}∪{4k-3\*2k+1}: O(n(7/6)), O(n(4/3))

**題目**

1. TM，RAM的加法操作都是常數級別的。[F]

□**二叉樹**

1. 完成n個節點的二叉樹層次遍歷需要（⌈n/2⌉）的輔助隊
2. 相同的2019個節點真二叉樹個數對應1009對括號表達式。[T]
3. 二叉樹葉子節點在先序、中序、後序遍歷次序一致。[T]
4. 後序遍歷中D生命期覆蓋A當且僅當D為A祖先。[T]

□**排序**

1. 插入排序後，逆序對不致增多，循環節不致減少。[F]
2. 選擇排序後，逆序對不致增多，循環節不致減少。[T]
3. 起泡排序每經過一輪掃描交換，相鄰的逆序對必然減少。[F, 如2, 3, 1]
4. 只要是基於比較的排序算法（CBA），對任何輸入序列都至少需要運行Θ(nlog(n))。[F, 插入排序有序下只需O(n)]

□**查找**

1. 不存在CBA式算法能夠經過少於2n-3次比較即找出最大和次大者。
2. 存在CBA式算法可以在O(n)時間内找到前10%
3. 對有序向量做Fibonacci查找，最壞情况下成功與失敗的次數相等。[T]
4. 無論有序向量或有序列表，最壞情况下都可在O(log(n))完成一次查找。[F]
5. 對於同一有序向量，每次拆半查找絕不會慢於順序查找。[F, 查找第一個]
6. 即便借助二分查找確定每個元素的插入位置，向量的插入排序最壞情况下仍然是Θ(n2) 。[F]
7. 如果有序向量中元素的分佈滿足獨立均匀分佈，插值查找的平均時間複雜度為：O(loglog(n))。
8. V={2, 3, 5, 7, 11, 13, 17}。V.search(16, 0, 7)需要進行多少次比較？5次。
9. V={1, 2, 3, 4, 5, 6, 7}，在V中用Fibonacci查找元素1，被選取為軸點mi的元素依次是：{5,3,2,1}
10. {2, 3, 5, 7, 11, 13, 17, 19, 23}中用插值查找搜索7，則mi=1
11. 長度為4的序列共有多少個不同的棧混洗？14

□**棧**

1. 調用棧中多幀可能對應同一函數的調用，且不一定緊密相鄰。
2. RPN中各操作數的相對次序與中綴表達成一致。[T]
3. 對不含括號的中綴表達式求值時，操作法棧的容量可以為某一固定常數。

□**複雜度**

1. 即便f(n)=O(g(n))，也未必2^f(n)=O(2^g(n))。

□**BST**

1. 若調用BSTremove刪除x，則需時為x的深度。[F]
2. 在BST中刪除兩個節點（7-B3），則無論先刪除哪個節點，最終BST的拓撲結構均相同。[T]
3. 由同⼀組共n個詞條構成的任意兩棵BST，經O(logn)次zig或zag旋轉之後，必定可以相互轉換。[F]
4. 對BST進行插入操作，對待插入的目標元素e進行查找後，若查找失敗，\_hot指向的節點為：e被插入後的父親。
5. 由5個互異節點構成的不同的BST共有（42）種。
6. 當欲刪除的節點v在BST中的度為2時，實際被刪除的節點為：v的右子樹中左側分支的最後一個節點。
7. 如果元素理想隨機，那麼對⼆叉搜索樹做平衡化處理，對改進其漸進時間複雜度沒有什麼作⽤。

□**Splay**

1. 0-2018插入splay樹，樹高2018，則詞條必單調順序插入。[F]
2. 伸展樹單次查找操作的最壞時間複雜度比AVL樹大。[T]
3. 最底層葉節點一旦被訪間並做過splay調整後，伸展樹高度必然下降。[F]
4. 訪問序列具有較強的局部性時，splay才能保證分攤O(log(n))。[F]
5. 在任何情况下，伸展樹總能保持每次操作O(log(n))的平均複雜度。[F]
6. 伸展樹每次訪問過某節點後都會把該節點：移動到根。
7. 伸展樹採用雙層伸展策略，可避免最壞情況發生。[F]
8. 所訪問的節點及其父親都是右孩子，則雙層伸展要執行的操作是：zag-zag。
9. 任意一顆伸展樹中，按節點值的大小以升序依次訪問完所有節點，最後樹變為一條僅含左孩子的單鏈。[T]
10. 即便訪問序列不滿⾜局部性（⽐如完全理想的隨機），伸展樹依然能夠保證分攤O(logn)的性能。[T]

□**AVL**

1. 在某節點被刪除後avl樹高即便下降，其間也未必做過旋轉調整。
2. AVL樹插入元素過程中發生旋轉操作則樹高不變。[T]
3. 規模n的AVL做一次插入，最壞情况下引發Θ(log(n))次局部重構。[F, 1次]
4. 若AVL樹插⼊元素的過程中發⽣了旋轉操作，則樹⾼必不變。[T]
5. 設在某新節點插⼊AVL樹後（尚待平衡化時），最低失衡節點為g。若此時g的左、右孩⼦的平衡因⼦分別為-1，0，則應通過（zag-zag）旋轉使之重新恢復平衡。
6. AVL樹中插入一個節點後失衡節點個數最多為O(logn)。
7. AVL樹中刪除一個節點後失衡節點個數最多為O(1)。
8. 高度為3的AVL樹至少包含幾個節點？7個。
9. AVL樹中插入節點引發失衡，經旋轉調整後重新平衡，此時包含節點g,p,v的子樹高度不變。
10. AVL樹中刪除節點引發失衡，旋轉調整後重新平衡，此時包含節點g,p,v的子樹高度：可能不變，也可能減小1 。
11. 經過3+4重構後的AVL樹[中序遍歷序列]不變。

□**RB-Tree**

1. 紅黑樹的插入或刪除都可能導致Θ(log(n))個節點顏色變化。[T]
2. 若紅⿊樹插⼊⼀個元素後⿊⾼度增加，則雙紅修正過程中沒有拓撲結構變換，只有重染⾊操作。[T, RR-2]
3. 紅黑樹相比於AVL樹的特點是：每次插入/刪除後拓撲結構的變化不超過O(1)。
4. 所有AVL樹可以染成紅黑樹。[T]
5. 當叔父節點u為紅色時，修正雙紅缺陷導致的紅黑樹拓撲結構的變化為：有變化，但是不超過O(1) 。

□**B-Tree**

1. 4階B樹中每個節點的分支數為：2~4 。
2. B樹查找算法若最終失敗，返回值為：NULL。
3. 若B樹的階m=128，則它的高度是對應BBST的1/6 。
4. B樹高度的增加一定伴隨著：分裂到根。
5. B樹高度的減少只會發生於：根節點的兩個孩子合併。
6. 將N個關鍵碼按隨機次序插⼊B樹，則期望的分裂次數為O(log2N)。[T]
7. BT::solveOverflow()和solveUnderflow()在最壞情況下均需下界(logn)的時間，然⽽在B-樹任⼀⾜夠長的⽣命期內，就分攤意義⽽⾔⼆者都僅需要O(1)時間。[T]
8. B樹在不發生上溢和下溢的情況下，那麼單次刪除和插入操作的時間花費大致相同。[F, 刪除需查找後繼]
9. ⼈類擁有的數字化數據總量，在2010年已經達到2B（2^20 = 10^21）量級，若每個字節⾃成⼀個關鍵碼，⽤⼀棵16階B-樹存放，則可能的⾼度為（20）。

□**PFC樹**

1. 最優PFC樹交換深度不同的節點及其子樹後必然不是最優PFC樹。[F]
2. 帶權重的最小PFC編碼樹不僅未必唯一、拓撲結構未必相同，樹高也可能不等。

□**樹-應用**

1. （B-樹，伸展樹）在插⼊元素後都可能導致O(logn)次局部結構調整。
2. 對大規模的數據（不能全部放於內存中）的存取：B-樹

易於實現，而且各接口的分攤複雜度為O(logn)：伸展樹

處理和幾何有關的問題：kd樹

擴充後可支持對歷史版本的訪問：紅黑樹

1. 對於任何⼀顆⼆叉樹T，其右、左⼦樹的規模之⽐“λ=T.re().size()/T.le().size()”稱作右偏率。對於（常規）⾼度同為h的AVL樹（A），紅⿊樹（R），左式堆（L），若分別考察其λ所能達到的最⼤值，則在h⾜夠⼤之後，三者按此指標的排列次序應是（L<A<R）。
2. 為從2014個隨機元素中挑選出最⼤的5個，（⼤頂的錦標賽樹）在最壞情況下所需的⽐較操作次數最少。

□**KD-Tree**

1. 在kd-search中，查找區間R與任⼀節點的4個孫節點（假設存在）對應區域最多有兩個相交。[T]
2. 我們知道了利用BBST來進行2D Range Search時，空間複雜度是O(nlogn)。如果將這個數據結構推廣到3D Range Search的話，空間複雜度是？O(nlog2n)
3. 我們知道了利用BBST來進行2D Range Search時，查詢時間複雜度是O(r+log2n)。如果將這個數據結構推廣到3D Range Search的話，時間複雜度是？O(r+log3n)
4. 對於樹套樹的x-tree與y-trees的結構來說，它們所佔的空間複雜度是多少？x-tree所佔的空間複雜度為O(n)，y-tree所佔的空間複雜度為O(nlogn)。

□**圖**

1. 圖dfs算法default分支，將dTime(v)<dTime(u)改為dTime(v)<fTime(u)同樣可行。[T]
2. 有向圖經dfs後若有k條邊被標記為backward，則它恰有k個環路。[F]
3. G是有向無環圖，(u, v)是G中的一條由u指向v的邊。對G進行DFS的結果是：fTime(u) > fTime(v)。
4. 對於同一無向圖，起始於頂點s的DFS儘管可能得到結構不同的DFS樹，但s在樹中的度數必然固定。[T]
5. 如果把朋友圈視為一無向圖，那麼即使A君看不到你給B點的讚，你們仍可能屬於同一雙連通分量。[T]
6. 設在有向圖G中，存在一條自頂點通往u的路徑，於是，若在某次dfs中有dTime(v)<dTime(u)，則這次生成的dfs森林中，v必是u的祖先。[F]
7. 先序遍歷的順序是:先自上而下訪問左側鏈上的節點,再自下而上訪問它們的右子樹。
8. 左右都無路可走的節點是後序遍歷第一個被訪問的節點
9. 一筆劃問題即要找出：歐拉路徑。
10. 在n個頂點的圖其中加入一個新的頂點後鄰接矩陣增加了多少項？2n+1
11. 設在有向圖G中，存在⼀條⾃頂點v通往u的路徑。於是，若在某次DFS中有dTime(v) < dTime(u)，則這次DFS所⽣成的DFS森林中，v必定是u的祖先。[F]
12. 在無向連通圖G中選定⼀個頂點s，並將各頂點v到s的距離記作dist(v)（特別地，dist(s)=0）。於是在G.Bfs(s)過程中，若輔助隊列為Q，則dist(Q.front()) + 1>= dist(Q.rear()) 始終成⽴。[T]
13. 我們知道，因同⼀頂點的鄰居被枚舉的次序不同，同⼀有向圖G所對應的DFS森林未必唯⼀。然⽽只要起始於G中某頂點s的某次DFS所⽣成的是⼀棵樹，則起始於s的任何⼀次DFS都將⽣成⼀棵樹。[T]

□**堆與優先級搜索**

1. 在圖的優先級搜索中，每次可能調⽤多次priorUpdater，但累計調⽤次數仍為O(e)。[T]
2. 完全二叉堆實現pfs，則各頂點出堆之前深度只增不減。[F]
3. 使用自上而下的上濾建立規模為n的完全二叉堆，最壞時間複雜度為：O(nlgn) 。
4. 完全⼆叉堆刪除元素在最壞情況下時間複雜度為O(logn)，但平均情況下僅為O(1)。[F]
5. 與⼆叉堆相⽐，多叉堆delMax()操作時間複雜度更⾼。[F, 三叉堆⽐⼆叉堆快]
6. 勝者樹根節點是冠軍，敗者樹根節點是亞軍。[T]
7. 相對於⼆叉堆，儘管多叉堆的⾼度更低，但⽆論是下濾⼀層還是整個下濾過程，時間成本反⽽都會增加。[F]
8. 在使⽤Heapify批量建堆的過程中，改變同層節點的下濾次序對算法的正確性和時間數常數無影響。[T]

□**左式堆**

1. 左式堆中每一對兄弟節點高度未必左大右小，但左兄弟至少不低於右兄的一半。[F, 與深度有關]
2. 對於左式堆A和B，合併後所得⼆叉堆的右側鏈元素⼀定來⾃A和B的右側鏈。[F，可能會交換左右⼦堆]
3. 相對於完全二叉堆，左式堆存在的意義是：高效的合併
4. 左傾的意義是：任何節點右孩子的NPL不超過左孩子。
5. 合併左式堆A和左式堆B，其中A的最大元素比B中所有元素都大，則遞歸的步驟為：合併A的右子堆和B。
6. 有2015個節點的左式堆，左⼦堆最⼩規模為（1個）。

□**散列**

1. 採用單向平方策略的散列表，只要長度M 不是素數，則每一組同義詞在表中都不會超過⌊m/2⌋個。[F]
2. n 個詞條插入一個容量為M，採用線性試探策略初始為空的散列表，n<M。則無論次序，平均成功查找長度必然一樣。[F, 若同餘類衝突但同餘鏈不變]
3. 在存在n個詞條的跳轉表中，塔高期望值為2。[T]
4. 在n個節點的跳轉表中，塔⾼的平均值為O(logn)。[F]
5. 我們知道，採取雙向平⽅試探策略時，應該將散列表取作素數M = 4k + 3。盡管這樣可以極⼤降低查找鏈前M個位置發⽣衝突的概率，但仍不能杜絕。[F]
6. 用哪種數據結構可以解決多槽位法的不足：列表
7. 當表的長度為素數時，為了使平方試探總是成功，裝填因子需要少於：50%。
8. S為所有可能詞條的空間，A為所有可用地址的空間（A< S），h是散列函數，則：從S映射到A，不可能是單射。
9. 若元素理想隨機，則⽤除餘法作為散列函數時，即使區間長度不是素數，也不會影響數據的均匀性。[T]
10. 相對於除餘法，MAD法在（⾼階均匀性、不動點）⽅⾯有所改進。

□**排序**

1. 左式堆中每一對兄弟節點高度未必左大右小，但左兄弟至少不低於右兄的一半。[F, 與深度有關]
2. 與勝者樹相⽐，敗者樹在重賽過程中需反復將節點與其兄弟進⾏⽐較。[F]
3. 若序列中逆序對個數為O(n^2)，則使⽤快速排序(12-A1)進⾏的交換次數為O(nlogn)。
4. shellSort最後的1-sorting都只需要O(n)時間。[F, Shell否]
5. shellSort按照某個增量做逐列排序，序列中逆序對總數不致增加。[T]
6. 無論g和h互素與否，已經h-有序的序列再經g-排序之後，必然繼續保持h-有序。[T]
7. 只要底層排序算法正確且穩定，則radixSort必然正確且穩定。[T]
8. 從規模為n的向量中選取中位數，quickselect算法的最壞時間複雜度是：O(n2)
9. N個待排序元素的取值範圍是[1, M]，計數排序的時間複雜度為：O(M+N)。
10. 反覆比較相鄰元素，逆序則交換，直至有序：冒泡排序

將原序列以軸點為界分為兩部分，遞歸排序：快速排序

將序列前半和後半部份分別排序，再合併：歸併排序

不斷從原序列中取出最小元素：堆排序

原序列中的元素是一定範圍內的整數，計算每個可能的整數在其中出現的次數，便可得排序結果：計數排序

抓撲克牌時人們常用的排序方法：插入排序

序列{5, 1, 3, 7, 8, 19, 13}中有2個元素滿足軸點的性質。

1. 針對不同的軸點選取策略，估計其發生不平衡的概率

從n個元素中等概率隨機選取一個作為軸點：0.2

從n個元素中等概率選取三個元素，以它們的中間元素作為軸點：0.056

1. 利用最壞時間複雜度為O(n)的中位數算法於快速排序的軸點選取，得到的快速排序最壞時間複雜度為O(nlogn)

[不可行，因為O(n)的中位數選取算法實際效率非常低]

□**KMP**

1. 在BM算法中，對於任⼀模式串P，0 < gs(j) <= j 對於每個0 <= j < |P| 都成⽴。[T]
2. 相較KMP算法，BM更適合大字符集的應用場合。[T]
3. 若KMP算法不使⽤改進版的next表，最壞情況下時間複雜度可能達到O(mn)。[F]
4. 對⼩寫字母集的串匹配，KMP算法與蠻⼒算法在（最好，平均）情況下漸進時間複雜度相同。
5. 對隨機⽣成的⼆進制串，gs表中gs[0] = 1的概率為（1/2^(m-1)）。注：所有元素相等
6. 對於長度為n的文本串和長度為m的模式串，KMP算法的時間複雜度為：O(m+n)。
7. 給定一個進行串匹配的算法，如何衡量它的效率？對於成功匹配和失敗匹配兩種情況分別討論其時間複雜度。
8. KMP算法查詢表為next[]，模式串P，若P[0,j)與文本串匹配，而在P[j]處失配，則：P[0, next[j]) = P[j - next[j], j)
9. 使用BC+GS策略的BM算法，最好和最壞情況下的時間複雜度分別為：O(n/m), O(m+n) 。