学号：　　　　　　　　姓名：　　　　　　　　班级：

Chapter 1 緒論

□級數
1. 算術級數與末項平方同階：$1+2+...+n=O(n^2)$
2. 冪方級數比冪次高一階：$\Sigma_0^n k^d=O(n^{d+1})$
3. 幾何級數與末項同階：$\Sigma_0^n a^k=O(a^n)$
4. 調和級數：$1 + 1/2 + 1/3 + ... + 1/n = \log(n)$
5. 對數級數：$\Sigma_1^n \ln(k)=O(n\log(n))$
6. 卡特蘭遞推關係：$T(n) = \Sigma_0^{n-1} T(k)\cdot T(n-k+1)$
7. 卡特蘭數：Catalan(n) = (2n)! / (n+1)! / n!

□估算
1. 1 天=10^5 sec；1 生=1 世紀=3×10^9 sec
2. 三生三世=3×10^{10} sec；宇宙大爆炸至今：4×10^{17} 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)
2. 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。
3. 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)
4. Fibonacci 查找：Φ=0.618（漸近複雜度 O(log(n))）
5. 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 $O(n^2)$
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 $O(n^2)$
 若逆序對間距不超過 k，則運行時間 O(kn)
2. 後向分析：平均比較次數。
 ◊$E=1+\Sigma_0^r k/(r+1)=1+r/2$, 總次數：$\Sigma_0^{n-1} 1+r/2=O(n^2)$
 ◊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）
4. 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。
5. 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)
2. 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 $2n=\Sigma T+\Phi_n-\Phi_0$）
3. 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. 中序遍歷：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)
4. 後序遍歷
5. 層次遍歷：借助隊列，最大規模[N / 2]
6. 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 $O(n^2)$，關聯矩陣 $O(ne)=O(n^3)$，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。
3. 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）
4. 優先級搜索：O((n+e)*log(n))

□應用
1. Prim 求最小支撐樹：極短跨邊（但可左右橫跳）
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。
3. Dijkstra 求最短路：所有臨時節點找最小。O((n+e)*log(n))
4. 雙連通分量：邊雙連通具有傳遞性，但點雙連通沒有。
5. 並查集：路徑壓縮（矮者接入高者）$O(m\log^{Hm/n} n)$

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）
3. 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）
4. 任一葉節點深度不小於[h/2]（數學歸納法證明）
5. 順序加入 $2^{h+1}-1$ 個關鍵碼得到高度為 h 的滿樹。
 ◊考慮：$\{0\}, \{2^{h+1}-1\}$, ◊$3*2^h-1$◊, $\{3*2^h\}, \{2^{h+2}-1\}$

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))	$O(r+n^{(1-1/k)})$	$O(r+\log^2 n)$
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。
2. 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)]）
5. 適用於局部性強、命中率高，此時 O(log(k))
6. 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）
4. 樹高：$\log_m(n+1)\le h\le \log_{\lceil m/2\rceil}\lfloor (n+1)/2\rfloor+1$
5. S 次分裂 M 次合併，則 S-M=n-h 恆成立。
6. 查找：順序查找/下探，複雜度 O(log(n))
7. 插入：分裂上溢，最壞情況有 $\Omega(\log_m(n))$次分裂。
8. 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)
9. 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：$\lfloor \log_2 n\rfloor\le h\le \log_{\lceil 4/2\rceil}\lfloor (n+1)/2\rfloor+1=\log_2(n+1)$
 黑高度：$\lceil \log_4(N+1)\rceil\le d\le \lfloor \log_{\lceil 4/2\rceil}\lfloor (n+1)/2\rfloor\rfloor+1$
3. 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。
4. 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)。
5. 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)。
6. 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

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、均勻。
2. 除餘法：hash(key)=key%M（M 為素數最均勻？）
 ◊問題：不動點 0，高階均勻性。
3. MAD 法：hash(key)=(a*key+b)%M
 ◊需要使 M, a 互素，保正散列的隨機和均勻性。
 ◊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 $M=2^k$ 不好，相當於二進制取後 k 位。
4. 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。
2. 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 0^2,… 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：$(u^2+v^2)(s^2+t^2)=(us+vt)^2+(ut-vs)^2$
3. 再散列：hash1 衝突，則取[hash1(key)+hash2(key)*n]%M
4. 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 $m = \max\{m_1, m_2, \cdots, m_k\}$ 為第 k 位取值範圍。
2. 整數排序：給定 n 個$[0,d^n]$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 $P(h=k)=p^{k-1}(1-p)$
2. 期望空間：$E(|S_k|)=n*2^{-k}$, $E(S)=n\Sigma_k 2^{-k}<2n=O(n)$
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。
4. 刪除：與堆尾元素交換，由根開始下濾。O(log(n))
5. 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：內部節點記錄敗者，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 $O(\log_d(n))$，下濾 $O(d*\log_d(n))$
3. PFS 效率：$O((nd+e)\log_d(n))$, 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
3. 右側鏈：終點為全堆中最淺節點。
 若 npl(rc)=d，則至少有 2^d-1 個內部節點，d⩽O(log(n))
4. 合併：沿右藤合併，確保左子堆 npl 不小，右边值不小。
5. 插入：相當於左式堆與單節點組成的左式堆合併。
6. 刪除：合併堆頂刪除後剩餘的兩個堆即可。
7. 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))時間內合併：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]]
2. 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
3. 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。
2. 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。
3. 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。
4. 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。

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。
3. 性能：平均划分 O(nlog(n))，取最值為軸點 $O(n^2)$。
4. 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。
5. DUP 版：大量重複元素，勤交換懶擴展。（左右向中心）
6. 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⌉)
3. 第 k 小元素：選取第 k 元素為軸點，遞歸查找。
   ◊$T(n)=(n-1)+1/2\Sigma_0^{n-1}\max\{T(k),T(n-k-1)\}$
   $\le(n-1)+2/n*\Sigma_{n/2}^{n-1}T(k) \le(n-1)+2/n*\Sigma_{n/2}^{n-1}(4k)<4n$
4. 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 $\{2^k\}$: $\Omega(n^2/4)$
   PS $\{2^k-1\}$：外循環 O(log(n))，排序 $O(n^{(3/2)})$
   Pratt $\{2^p3^q\}$: $O(n\log^2(n))$
   Sedqewick $\{9*4^k-9*2^k+1\}\cup\{4^k-3*2^k+1\}$: $O(n^{(7/6)})$, $O(n^{(4/3)})$

題目

1. TM，RAM 的加法操作都是常數級別的。[F]

□二叉樹

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

□排序

6. 插入排序後，逆序對不致增多，循環節不致減少。[F]
7. 選擇排序後，逆序對不致增多，循環節不致減少。[T]
8. 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]
9. 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

10. 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。
11. 存在 CBA 式算法可以在 O(n)時間內找到前 10%
12. 對有序向量做 Fibonacci 查找，最壞情況下成功與失敗的次數相等。[T]
13. 無論有序向量或有序列表，最壞情況下都可在 O(log(n))完成一次查找。[F]
14. 對於同一有序向量，每次折半查找絕不會慢於順序查找。[F，查找第一個]
15. 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況下仍然是 $\Theta(n^2)$。[F]
16. 如果有序向量中元素的分佈滿足獨立均勻分佈，插值查找的平均時間複雜度為：O(loglog(n))。
17. V={2, 3, 5, 7, 11, 13, 17}。V.search(16, 0, 7)需要進行多少次比較？5 次。
18. 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}
19. {2, 3, 5, 7, 11, 13, 17, 19, 23}中用插值查找搜索 7，則 mi=1
20. 長度為 4 的序列共有多少個不同的棧混洗？14

□棧

21. 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。
22. RPN 中各操作數的相對次序與中綴表達式一致。[T]
23. 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

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

□BST

25. 若調用 BSTremove 刪除 x，則需時為 x 的深度。[F]
26. 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。[F]
27. 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]
28. 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。
29. 由 5 個互異節點構成的不同的 BST 共有（42）種。
30. 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。
31. 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

32. 0-2018 插入 splay 樹，樹高 2018，則詞條必單調順序插入。[F]
33. 伸展樹單次查找操作的最壞時間複雜度比 AVL 樹大。[T]
34. 最底層葉節點一旦被訪問並做過 splay 調整後，伸展樹高度必然下降。[F]
35. 訪問序列具有較強的局部性時，splay 才能保證分攤 O(log(n))。[F]
36. 在任何情況下，伸展樹總能保持每次操作 O(log(n))的平均複雜度。[F]
37. 伸展樹每次訪問過某節點後都會把該節點：移動到根。
38. 伸展樹採用雙層伸展策略，可避免最壞情況發生。[F]
39. 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。
40. 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]
41. 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

42. 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。
43. AVL 樹插入元素過程中發生旋轉操作則樹高不變。[T]
44. 規模 n 的 AVL 做一次插入，最壞情況下引發 Θ(log(n))次局部重構。[F, 1 次]
45. 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]
46. 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。
47. AVL 樹中插入一個節點後失衡節點個數最多為 O(logn)。
48. AVL 樹中刪除一個節點後失衡節點個數最多為 O(1)。
49. 高度為 3 的 AVL 樹至少包含幾個節點？7 個。
50. 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。
51. 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。
52. 經過 3+4 重構後的 AVL 樹[中序遍歷序列]不變。

□RB-Tree

53. 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]
54. 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]
55. 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)。
56. 所有 AVL 樹可以染成紅黑樹。[T]
57. 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

58. 4 階 B 樹中每個節點的分支數為：2~4。
59. B 樹查找算法若最終失敗，返回值為：NULL。
60. 若 B 樹的階 m=128，則它的高度是對應 BBST 的 1/6。
61. B 樹高度的增加一定伴隨著：分裂到根。
62. B 樹高度的減少只會發生於：根節點的兩個孩子合併。
63. 將 N 個關鍵碼按隨機次序插入B 樹，則期望的分裂次數為 $O(\log^2 N)$。[T]
64. 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]
65. 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]
66. 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

67. 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]
68. 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

69. （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。
70. 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
71. 對於任何一顆二叉樹 T，其右、左子樹的規模之比"λ=T.rc().size()/T.lc().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）。
72. 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

73. 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]
74. 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？$O(n\log^2 n)$
75. 我們知道了利用 BBST 來進行 2D Range Search 時，查詢時間複雜度是 $O(r+\log^2 n)$。如果將這個數據結構推廣到 3D Range Search 的話，時間複雜度是？$O(r+\log^3 n)$
76. 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

77. 圖 dfs 算法 default 分支，將 dTime(v)<dTime(u)改為 dTime(v)<fTime(u)同樣可行。[T]
78. 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]
79. 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)。
80. 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]
81. 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]
82. 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]
83. 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。
84. 左右都無路可走的節點是後序遍歷第一個被訪問的節點
85. 一筆劃問題即要找出：歐拉路徑。
86. 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
87. 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]
88. 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]
89. 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

91. 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]
92. 完全二叉堆實現 pfs，則各頂點出堆之前深度只增不減。[F]
93. 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)。
94. 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]
95. 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]
96. 勝者樹根節點是冠軍，敗者樹根節點是亞軍。[T]
97. 相對於二叉堆，儘管多叉堆的高度更低，但無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]
98. 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

99. 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]
100. 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]
101. 相對於完全二叉堆，左式堆存在的意義是：高效的合併
102. 左傾的意義是：任何節點右孩子的 NPL 不超過左孩子。
103. 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。
104. 有 2015 個節點的左式堆，左子堆最小規模為（1 個）。

□散列

105. 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]
106. 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]
107. 在存在 n 個詞條的跳轉表中，塔高期望值為 2。[T]
108. 在 n 個節點的跳轉表中，塔高的平均值為 O(logn)。[F]
109. 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]
110. 用哪種數據結構可以解決多槽位法的不足：列表
111. 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%。
112. 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。
113. 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勻性。[T]
114. 相對於除餘法，MAD 法在（高階均勻性、不動點）方面有所改進。

□排序

115. 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]
116. 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]
117. 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)。
118. shellSort 最後的 1-sorting 都只需要 O(n)時間。[F, Shell 否]
119. 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]
120. 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]
121. 只要底層排序算法正確且穩定，則 radixSort 必然正確且穩定。[T]
122. 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：$O(n^2)$
123. 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)。
124. 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。
125. 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
126. 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

127. 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]
128. 相較 KMP 算法，BM 更適合大字符集的應用場合。[T]
129. 若 KMP 算法中使用改進版的 next 表，最壞情況下時間複雜度可能達到 O(mn)。[F]
130. 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。
131. 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
132. 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)。
133. 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。
134. 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)
135. 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)。