Análisis de Algoritmos
Grado en Ingeniería Informática, UAM

Problemas

- **(E)**: ejercicio; **(E+)**: ejercicio avanzado.
- **(P)**: problema; **(P+)**: problema avanzado.
- **(R)**: recomendado; se resolverá en clase.

1 TAEs de algoritmos básicos

Ejercicios

1. **(E; R)** Denotando como `MMul` el algoritmo tradicional de multiplicación de matrices cuadradas, definir una OB adecuada para el mismo, y estimar para la misma $n_{\texttt{MMul}}(A, B)$, para cualquier par de matrices de dimensión N.

2. **(E; R)** El siguiente pseudocódigo corresponde a un algoritmo para encontrar el mínimo de una tabla

```
indice Min(tabla T, indice P, indice U)
  Min=T[P];
  iMin=P;
  para i de P+1 a U:
    si T[i] < Min:
       Min = T[i];
       iMin=i;
  devolver iMin;
```

   Analizar su tiempo de ejecución del algoritmo usando como operación básica la comparación de claves.

3. **(E)** Analizar el tiempo de ejecución del algoritmo trivial de potenciación de pseudocódigo

```
pot(int x, int N)
  p = x;
  para i de 2 a N:
    p = p*x;
  devolver p;
```

4. **(E; R)** Para los siguientes fragmentos de código, establecer una operación básica adecuada y estimar sus tiempos de ejecución en función de n:

   - ```
     sum = 0;
     for (i=1; i<=n; i++)
       sum++;
     ```
     ```

- ```
  sum = 0;
  for (i=1; i<=n; i++)
    for (j=0; j<n; j++)
      sum++;
  ```
- ```
  sum = 0;
  for (i=1; i<=n; i++)
    for (j=0; j<n*n; j++)
      sum++;
  ```

Problemas

5. **(P; R)** Para los siguientes fragmentos de código, establecer una operación básica adecuada y estimar sus tiempos de ejecución en función de n:

   - ```
     sum = 0;
     for (i=1; i<=n; i++)
       for (j=0; j<i; j++)
         sum++;
     ```
   - ```
     sum = 0;
     for (i=1; i<=n; i++)
       for (j=0; j<i*i; j++)
         for (k=0; k<j; k++)
           sum++;
     ```
   - ```
     sum = 0;
     for (i=1; i<=n; i++)
       for (j=0; j<i*i; j++)
         if (j%i == 0)
           for (k=0; k<j; k++)
             sum++;
     ```

6. **(P)** Justificar las desigualdades $(A + B)/2 \leq \max(A, B) \leq A + B$. ¿Qué suponen para el cálculo del número de OBs sobre selecciones?

2 Crecimiento de funciones

Ejercicios

7. **(E; R)** Para un cierto problema se disponen de cinco algoritmos cuyos tiempos de ejecución en microsegundos para una entrada de tamaño N son respectivamente $10.000N$, $1.000N \log_{10} N$, $100N^2$, $10N^3$ y $10^{-3}10^{N/10}$. Dar una tabla con sus tiempos de ejecución para los valores de N 10, 100, 1.000, 10.000, 100.000 y 1.000.000.

   ¿Cuál es para cada algoritmo el tamaño máximo de una entrada para que ésta se ejecute en 1 minuto? ¿Y en un día?
     ```

8. **(E; R)** Ordenar las siguientes funciones según su crecimiento: n, $\sqrt{n}$, $n^{1.5}$, n^2, $n \log n$, $n \log \log n$, $n \log^2 n$, $2/n$, 2^n, $2^{n/2}$, 37, $n^2 \log n$, n^3.

9. **(E)** Los tiempos de ejecución de tres algoritmos para la resolución de un mismo problema, para entradas de tamaño N, son N^2, $10N(\log_2 N)^2$ y $\frac{1}{8}N^2 \log_2 N$, respectivamente. Dar los rangos de valores de N en los que cada uno de estos algoritmos es más conveniente. (Considerar potencias de 2 para los límites de dichos rangos.)

10. **(E; R)** El análisis de dos algoritmos, A y B, muestra que $W_A(N) = \mathrm{O}(1000N \log_{10}(N))$ y $W_B(N) = \mathrm{O}(N^2)$. ¿Cuál es mejor para problemas "pequeños"? ¿Cuál es mejor para problemas "grandes"? Especificar qué sería pequeño y grande en este contexto.

11. **(E; R)** Probar que para todo $a > 0$, $\log n = \mathrm{o}(n^a)$.

12. **(E+)** Probar que $\lfloor N \rfloor = \Theta(N)$ y que también $\lceil N \rceil = \Theta(N)$ (¿puede hacerse mediante límites?).

13. **(E+)** ¿Cuál es el orden de crecimiento teórico de la función $f(N) = N \log N + 100N$? ¿Cuál es su orden de crecimiento "real"?

14. **(E; R)** Se sabe que $T_1 \approx f$ y $T_2 \approx f$; decidir razonadamente la verdad o falsedad de las siguientes afirmaciones (dar un breve argumento en las ciertas y un contraejemplo o contraargumento en las falsas):

    (a) $T_1 + T_2 \approx f$;
    (b) $T_1 - T_2 = \mathrm{o}(f)$;
    (c) $T_1 \cdot T_2 \approx f^2$.

15. **(E; R)** Se sabe que $T_1 \approx f$ y $T_2 \approx f$; decidir razonadamente la verdad o falsedad de las siguientes afirmaciones (dar un breve argumento en las ciertas y un contraejemplo o contraargumento en las falsas):

    (a) $T_1 + T_2 \approx 2f$;
    (b) $T_1^2 - T_2^2 = \mathrm{o}(f)$;
    (c) $T_1/T_2 \approx 1$.

Problemas

16. **(P; R)** Se sabe que $T_1(N) = \mathrm{O}(f(N))$ y que $T_2(N) = \mathrm{O}(f(N))$. ¿Cuál de las siguientes afirmaciones es cierta y cuál falsa?

    (a) $T_1(N) + T_2(N) = \mathrm{O}(f(N))$;
    (b) $T_1(N) - T_2(N) = \mathrm{o}(f(N))$;
    (c) $\frac{T_1(N)}{T_2(N)} = \mathrm{O}(1)$;
    (d) $T_1(N) = \mathrm{O}(T_2(N))$.

    (dar un breve argumento para las afirmaciones ciertas y un contraejemplo para las falsas).

17. **(P; R)** Se sabe que $T_1(N) = \Theta(f(N))$ y que $T_2(N) = \Theta(f(N))$. ¿Cuál de las siguientes afirmaciones es cierta y cuál falsa?

    (a) $T_1(N) + T_2(N) = \Theta(f(N))$;

    (b) $T_1(N) - T_2(N) = \mathrm{o}(f(N))$;

    (c) $\frac{T_1(N)}{T_2(N)} = \Theta(1)$;

    (d) $T_1(N) = \Theta(T_2(N))$.

    (dar un breve argumento para las afirmaciones ciertas y un contraejemplo para las falsas).

18. **(P)** Diremos que $F = f + \mathrm{O}(g)$ si $g = \mathrm{o}(f)$ y $|F - f| = \mathrm{O}(g)$. Comprobar que $\sum_1^N i^k = N^{k+1}/(K+1) + O(N^K)$.

19. **(P+)** Probar que para todo N, $\log N! = \Theta(N \log N)$ y $N! = \mathrm{o}(N^N)$.

20. **(P; R)** ¿Es cierto que para cualquier f y cualquier constante C, $f(Cn) = \mathrm{O}(f(n))$? ¿Es cierto que $f(n + C) = \mathrm{O}(f(n))$?

3 Casos peor, mejor y medio de algoritmos básicos

Ejercicios

21. **(E; R)** Estimar razonadamente el crecimiento de la siguiente función:

$$S(N) = \sum_1^N \frac{1}{i^{1/3}}.$$

22. **(E)** De una cierta tabla de tamaño N se sabe que $P(T[i]) = \frac{i^2}{C_N}$, donde C_N es una constante normalizadora. ¿Cuál será el número medio de comparaciones de clave que realizará sobre la misma el algoritmo de búsqueda lineal en búsquedas con éxito?

23. **(E; R)** Se quiere usar el siguiente pseudocódigo para búsquedas lineales en tablas ordenadas

```
int BLin(tabla T, dim N, clave K)
  i=1;
  mientras i <= N y T[i] < K:
    i++;
  si i > N:
    devolver error;
  else si T[i] != K:
    devolver error;
  else:
    devolver i;
```

    Suponiendo que $P(K = i) = P(i < K < i + 1) = P(i < 1) = P(i > N) = \frac{1}{2N+1}$, calcular $A_{BLin}(N)$ considerando tanto búsquedas con acierto como con error.

24. **(E; R)** Dada una tabla T, se sabe que

$$P(K = T[i]) = \frac{1}{C_N}\frac{\log i}{i},$$

donde $C_N = \sum_1^N \frac{\log i}{i}$.
Calcular razonadamente $A_{BLin}(N)$.

25. **(E)** Dada una tabla T, se sabe que

$$P(K = T[i]) = \frac{1}{C_N} i \log i,$$

donde $C_N = \sum_1^N i \log i$. Calcular razonadamente $A^e_{BLin}(N)$.

Problemas

26. **(E)** Comprobar por inducción la veracidad de la identidad $\sum_1^N a + ib = N(a + \frac{N+1}{2}b)$.

27. **(P; R)** Demostrar por inducción las siguientes fórmulas:
$\sum_1^N n^2 = n(n+1)(2n+1)/6$,
$\sum_1^N n^3 = (n(n+1)/2)^2$.

28. **(P+)** Dar una fórmula para la suma $\sum_1^N nx^n$, y también para $\sum_1^N n^2x^n$. (Observése que la primera se parece a una derivada y la segunda no está demasiado lejos.)

4 Evolución de algoritmos locales

Ejercicios

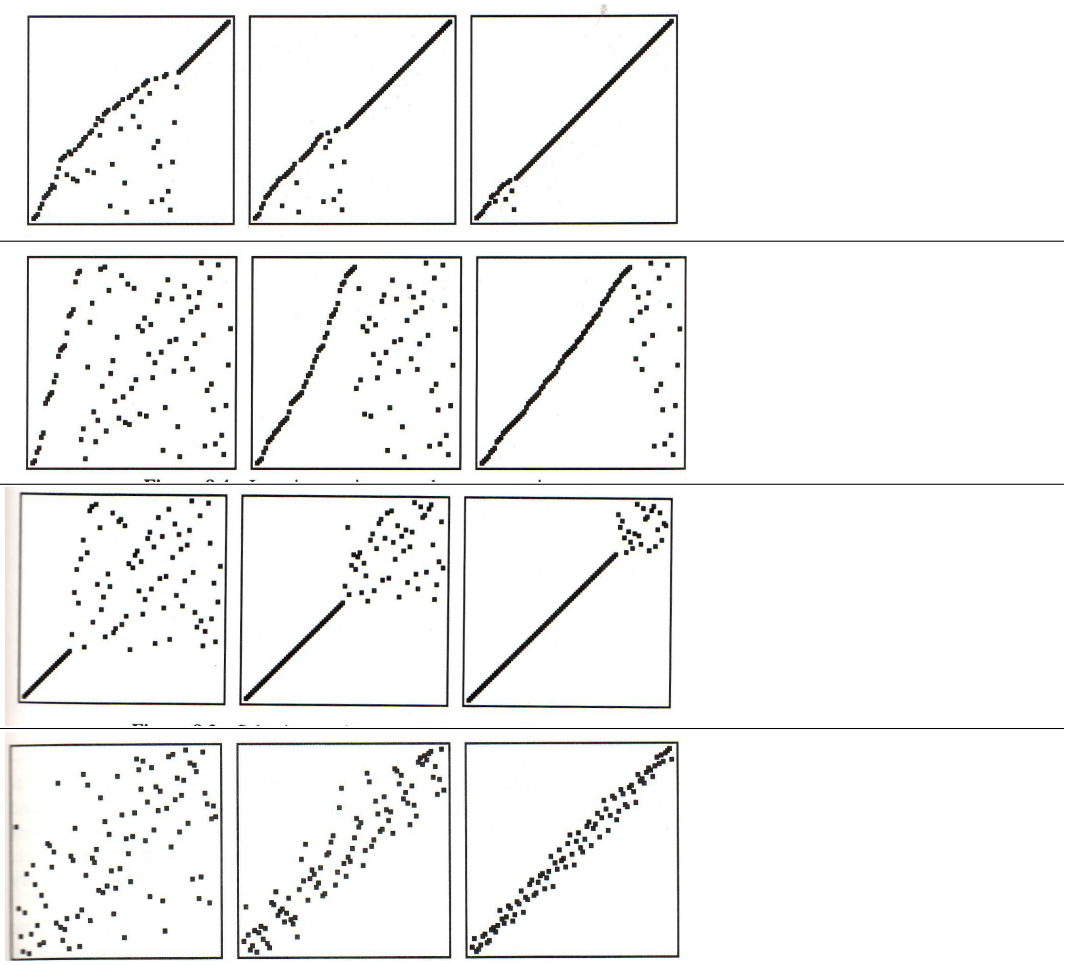
29. **(E)** Dada la lista `[20, 3, 10, 6, 8, 2, 13, 17]`, indicar el estado de la misma tras cada una de las iteraciones del método de la burbuja y del de inserción.

30. **(E)** Sobre la lista `16 6 3 4 9 31 1 8 2 7`, dar razonadamente la evolución del método de inserción, indicando el estado de la tabla tras la ubicación de cada elemento, y el número de comparaciones de elementos realizadas en dicha ubicación.

31. **(E)** Sobre la lista `8 18 5 6 10 3 4 11 31 7`, dar razonadamente la evolución del método de la burbuja, indicando para la primera iteración qué elementos se usan como burbuja y detallando los estados parciales de la tabla, y para las demás iteraciones el estado final de la tabla tras las mismas. ¿Cuántas iteraciones se harán?

Problemas

32. **(P; R)** En los cuadros de la figura inferior se representan tres fases consecutivas de la evolución de una lista de 50 números entre 1 y 50 a la que se aplican ciertos métodos de ordenación. Las coordenadas (i, j) de un punto del gráfico indican que el número de la posición i–ésima de la lista es precisamente j (por ejemplo, un número i está en su posición correcta si $j = i$, esto es, si el par (i, j) está en la diagonal). ¿Cuáles de los métodos de ordenación estudiados en el curso podráa generar estos gráficos?



5 Eficacia de algoritmos locales

Ejercicios

33. **(E; R)** Dadas las permutaciones de S_6 (6 1 5 2 4 3), (6 5 4 1 2 3) y (2 3 6 4 5 1), calcular sus traspuestas, encontrar las inversiones de ambas y comprobar el resultado del problema anterior.

    Si se les aplica un algoritmo de ordenación local, ¿cuántas cdcs efectuará como mínimo sobre ellas?

34. **(E; R)** ¿Cuántas comparaciones de clave realizará como mínimo un algoritmo de ordenación local sobre la permutación de $N = 2K$ elementos `(2K 2K-1 2K-2 ... K+1 1 2 3 ... K)`?

35. **(E; R)** Expresar en función de N cuantas comparaciones de clave realizará como mínimo un algoritmo local si se aplica a la permutación de $N = 3K$ elementos `(2K+1, 2K+2, ..., 3K, 2K, 2K-1, ... K+2, K+1, 1, 2, ..., K)`

36. **(E)** Expresar en función de N cuántas comparaciones de clave realizará como mínimo un algoritmo local si se aplica a la permutación de $N = 4K$ elementos

    `(3K+1, 3K+2, ..., 4K, 3K, 3K-1, ..., 2K+2, 2K+1, 2K, 2K-1, ... K+2, K+1, 1, 2, ..., K)`

37. **(E; R)** Decidir razonadamente el número de comparaciones de clave que un algoritmo de ordenación local hará como mínimo sobre la permutación de $N = 3K$ elementos

$$(3K, 3K-1, , ..., 2K+1, K+1, K+2, ..., 2K, K, K-1, ..., 2, 1).$$

Problemas

38. **(P)** Comprobar mediante la definición de permutación traspuesta que dada una permutación σ, se cumple que $(\sigma^\tau)^\tau$ coincide con σ.

39. **(E+; R)** Se aplica un algoritmo local A de ordenación a la permutación de $N = 2K$ elementos `(1 2K 2 2K-1 3 2K-2 ... K K+1)`. ¿Cuántas comparaciones de clave efectuará A sobre dicha permutación?

40. **(E+; R)** Decidir razonadamente el número de comparaciones de clave que un algoritmo de ordenación local hará como mínimo sobre la permutación de $N = 2K$ elementos `(2K, 1, 2K-1, 2, ..., 2K-i, i+1, ..., K+1, K)`.

41. **(P)** Calcular el valor de $W_{Burbuja}(N)$, $A_{Burbuja}(N)$, $W_{Seleccion}(N)$ y $A_{Seleccion}(N)$.

42. **(E)** Se define $B_{\mathcal{A}}(N)$ como $B_{\mathcal{A}}(N) = \min\{n_{\mathcal{A}}(I) : I \in E_N^{\mathcal{A}}\}$. Analizar $B_{Insert}(N)$, $B_{Selec}(N)$, $B_{Burbuja}(N)$.

43. **(E+)** Modificar el psc del método de Inserción para que pueda trabajar con índices iniciales P y finales U cualesquiera.

44. **(P; R)** Analizar $B_{Insert}(N)$, $B_{Selec}(N)$, $B_{Burbuja}(N)$ usando el intercambio de elementos como operación básica. Analizar también $W_{Select}(N)$, y $W_{Burbuja}(N)$ bajo este punto de vista.

45. **(P+; R)** Dar una estimación aproximada de $A_{Burbuja}(N)$ suponiendo que en la misma se utiliza el siguiente psc:

```
Burbuja(tabla T, dim N)
  switch = 1;
  long   = N;
  mientras switch == 1 y long > 1:
    switch = 0;
    para i de 1 a long-1:
      si T[i] > T[i+1]:
        intercambiar T[i] y T[i+1];
        switch = 1;
    long--;
```

46. **(P; R)** El siguiente pseudocódigo es una ligera variante del habitualmente utilizado en la ordenación por Inserción:

```
InsertSort(tabla T, dim N)
  para i de 2 a N:
    j=i;
```

```
    mientras j > 1 y T[j-1] > T[j]:
      intercambiar T[j-1] y T[j];
      j--;
  volver;
```

Utilizando el intercambio de elementos como operación básica, calcular razonadamente $W_{InserSort}(N)$ y $A_{InserSort}(N)$.

47. **(P)** Un algoritmo de ordenación se dice **estable** si al aplicarlo a una tabla con elementos tal vez repetidos, dos elementos con la misma clave (y por tanto relativamente ya bien ordenados) mantienen sus posiciones relativas al finalizar el algoritmo. ¿Cuáles de los métodos de ordenación estudiados son estables? (Modificar su psc si fuera necesario.)

6 MergeSort y QuickSort

Ejercicios

48. **(E)** Estudiar la evolución de las permutaciones

    (a) (5 9 8 2 3 10 1 7 4 6),

    (b) (3 10 1 7 5 9 8 2 4 6),

    (c) (10 1 7 3 9 8 2 5 4 6),

    cuando se le aplica la rutina `Partir` suponiendo que su subrutina `Medio` devuelve el índice `P`.

49. **(E)** Dar la evolución del algoritmo `Partir` sobre la tabla `(7 8 2 4 5 3 10 1 6)`.

50. **(E+)** Dada la lista `[18, 7, 4, 6, 12, 23, 2]`,
    a. Indicar mediante un árbol las distintas sublistas que se generan en la fase de partir al aplicársele el método MergeSort (situar en un nodo dado una sublista y a su izquierda y derecha las otras dos que se generan de ella). Numerar cada nodo según el orden en que el método MergeSort genera su correspondiente lista.
    b. Tomado como base las sublistas elementales del último nivel del árbol anterior, indicar mediante un árbol invertido las distintas sublistas que se generan en la fase de combinación del método MergeSort (por ejemplo, el último nodo debiera contener la lista inicial ordenada). Numerar también aquí cada nodo según el orden en que el método MergeSort genera su correspondiente lista.

51. **(E)** Dadas las listas `[45 29 47 32 19 12 26 51]` y `[12 9 22 37 25 32 6 14]`, estudiar su evolución sobre un AB cuando se les aplica el método de ordenación de MergeSort y QuickSort.

52. **(E)** Dadas las listas
    `[Q B K C F A G P] , [34 43 12 58 17 92 21]`
    indicar su evolución en un árbol binario al aplicarseles los métodos de MergeSort.

Problemas

53. **(P+)** Supongamos que en una tabla los elementos `T[i]` y `T[i+K]` están en inversión. Probar que si los intercambiamos, se elimina como mínimo una inversión y como máximo $2K-1$.

54. **(P+)** Estudiar el tiempo de ejecución del QuickSort en los casos mejor y peor tomando como operación básica el intercambio de elementos.

55. **(P)** Calcular $A_{Partir}(N)$, donde $Partir$ es la rutina de división que utiliza el método de QuickSort para dividir una tabla en dos subtablas utilizando como operación básica el intercambio de elementos de la tabla.

56. **(P+)** Estudiar el tiempo medio de ejecución del QuickSort tomando como operación básica el intercambio de elementos.

57. **(P)** Estudiar $B_{Combinar}$ y $B_{MS}(N)$.

58. **(P+)** Estudiar $A_{Combinar}$ cuando se aplica a dos tablas de tamaño N.

59. **(P+)** Estimar de manera adecuada el posible valor de $A_{MS}(N)$.

60. **(P+)** Una operación básica alternativa en el Merge Sort podría ser la "copia" de elementos que se realiza en la rutina `Combinar`. Estimar los posibles valores de $W_{MS}(N)$ y $A_{MS}(N)$ respecto a esta operación.

61. **(P+)** Analizar $B_{QS}(N)$.

62. **(P)** Se quiere aplicar el algoritmo QuickSort tomando siempre P como pivote sobre un conjunto de tablas T de tamaño $N = 2M+1$ de la forma

$$[M+1, 2, \ldots M, 1, T[M+2], \ldots, T[N]],$$

donde los valores T[j], $M+2 < \mathrm{j} \leq N$ están entre $M+2$ y $N = 2M+1$; esto es, la primera parte de las tablas es siempre la misma, de la forma T[1] = M+1, T[i] = i si $1 < \mathrm{i} < M+1$, T[M+1]=1 y la segunda parte varía, estando formada por los enteros de $M+2$ a N desordenados. Estimar razonadamente en función de N cuál será el coste medio de tales ordenaciones.

7 Desigualdades recurrentes

Ejercicios

63. **(E; R)** Expresar en función de N los valores de las siguientes sumas

1) $\sum_0^{K-1} 2^j,\ N = 4^K$; 2) $\sum_0^{K-j} 4^j,\ N = 2^K$; 3) $\sum_0^{K-1} 2^j,\ N = 3^K$;
4) $\sum_0^{K-1} 2^{j+2},\ N = 8^K$; 5) $\sum_0^{K-1} \frac{3^{j-2}}{4^{j+2}},\ N = 12^K$; 6) $\sum_0^{K-1} 2^{j/2},\ N = 4^K$;
7) $\sum_0^{K-1} \left(\frac{2}{3}\right)^j,\ N = \left(\frac{3}{2}\right)^K$; 8) $\sum_0^{K-1} 4^j,\ N = 2^K$; 9) $\sum_0^{K-1} 2^j,\ \lg N = 3^K$

64. **(E; R)** De una cierta funcion T se sabe que $T(1) = 0$ y que $T(N) \leq \lg N + T(\lfloor N/2 \rfloor)$. Estimar el orden de $T(N)$ para N de la forma $N = 2^K$ (Sugerencia: reiterar la igualdad anterior).

65. **(E; R)** Estimar el orden de la función $T(N)$ en las siguientes recurrencias suponiendo que $T(1) = 0$

    (a) $T(N) \leq 2T(\lfloor N/2 \rfloor) + N^3$;

    (b) $T(N) \leq 2T(N-1) + 1$;

    (c) $T(N) \leq T(N-1) + N$;

    (d) $T(N) \leq 2T(\lfloor N/4 \rfloor) + \sqrt{N}$;

66. **(E; R)** Estimar razonadamente el crecimiento de la función T que satisface $T(1) = 0$, así como la desigualdad recurrente
$$T(N) \leq \sqrt{N} + 4T(\lfloor N/4 \rfloor).$$

67. **(E; R)** Estimar el crecimiento de la función $T(N)$ de argumento entero que verifica la desigualdad recurrente $T(N) \leq N + 2T(\lfloor N/2 \rfloor)$, donde $T(1) = 0$.

Problemas

68. **(P; R)** Estimar razonadamente el crecimiento de la función T que satisface $T(1) = 0$, así como la desigualdad recurrente
$$T(N) \leq T(\lfloor N/2 \rfloor) + N \lg(N).$$

69. **(P; R)** Estimar el orden de la función $T(N)$ tal que $T(1) = 0$ y $T(N) \leq T(\left\lfloor \sqrt{N} \right\rfloor) + 1$.

8 Análisis de algoritmos recurrentes

Ejercicios

70. **(E; R)** El siguiente algoritmo resuelve el problema de las torres de Hanoi para mover N discos de un poste A a otro B usando un tercero C como almacenamiento intermedio

```
Hanoi(discos N, poste A, poste B, poste C)
  si N == 1:
    mover un disco de A a B;
  else
    Hanoi(N-1, A, C, B);
    mover un disco de A a B;
    Hanoi(N-1, C, B, A);
```

Usando el movimiento de discos como OB, ¿cuántos movimientos se harán para trasladar N discos de A a B? ¿Cuántas llamadas recursivas hace el algoritmo para N discos?

Problemas

71. **(P; R)** El siguiente algoritmo proporciona la posición del elemento máximo de una tabla T entre las posiciones P y U:

```
ind MaxRec(tabla T, ind P, ind U)
   si P == U: devolver P ;
   M = (P+U)/2 ;
   m1 = MaxRec(T, P, M) ;
   m2 = MaxRec(T, M+1, U) ;
   si T[m1] > T[m2] : devolver m1 ;
   else :             devolver m2 ;
```

Usando la cdc como operación básica, estimar razonadamente el trabajo en el caso peor del algoritmo sobre tablas de N elementos. Para ello establecer una desigualdad recurrente para dicho trabajo, resolverla en algún caso particular adecuado y, finalmente, dar la solución general.

72. **(P; R)** El siguiente pseudocódigo corresponde a una versión recursiva del cálculo de la media de una tabla

```
float media(tabla T, pos P, pos U)
   si P == U :
      devolver T[P] ;
   else :
      M = suelo ((P+U)/2) ;
      medI = media(T, P, M) ;
      medD = media(T, M+1, U) ;
      nI = M-P+1 ;      // num elems de subtabla izquierda
      nD = U-M ;        // num elems de subtabla derecha
      nT = U-P+1 ;      // num total de elementos
      med = ( nI*medI + nD*medD ) / nT ;
      devolver med ;
```

Suponiendo que se usa la suma como operación básica, estimar el número de sumas $n(N)$ que hace el algoritmo sobre una tabla de N elementos efectuando para ello los siguientes pasos:
i. escribir una ecuación recurrente para $n(N)$;
ii. resolver la misma en algún caso particular conveniente;
iii. resolver o acotar la misma en el caso general.

73. **(P)** Suponiendo $N = 2^K$, un número A de N cifras puede descomponerse como $A = A_1 * D_K + A_2$, donde A_1, A_2 tienen $N/2$ cifras y $D_K = 10^{2^{K-1}}$. Si así se descomponen dos números A, B de N cifras, la fórmula

$$AB = A_1B_1 * D_K^2 + (A_1B_2 + A_2B_1) * D_K + A_2B_2$$

sugiere el siguiente algoritmo recursivo para multiplicar A y B:

```
int MMR(int A, int B, int N)
  si N==1:
    devolver A*B;
  else:
    calcular A1, A2, B1, B2;
    O = MMR(A1, B1);
    P = MMR(A1, B2) + MMR(A2, B1);
    Q = MMR(A2, B2);
    devolver (O * DK + P) * DK + Q;
```

donde `DK` denota $10^{2^{K-1}}$. Tomando como OB la multiplicación, ¿cuál es la complejidad $T_{MMR}(N)$ de aplicar MMR a dos números de $N = 2^K$ cifras?

74. **(P+)** En las condiciones del problema anterior, la fórmula

$$AB = A_1B_1 * D_K^2 + ((A_1 + A_2)(B_1 + B_2) - A_1B_1 - A_2B_2) * D_K + A_2B_2$$

reduce la multiplicación de dos números de N cifras a esencialmente tres multiplicaciones de dos números de $N/2$ cifras. Dar el pseudocódigo de un algoritmo recursivo $MMRmR$ de multiplicación de números que aproveche dicha fórmula y estimar su complejidad $T_{MMRmR}(N)$ al aplicarlo a dos números de $N = 2^K$ cifras.

75. **(P+; R)** Se quiere estimar el número de llamadas recursivas del algoritmo inferior con una versión sin recursión de cola de las torres de Hanoi. Escribir la correspondiente ecuación recurrente y resolverla.

```
hanoi2(int N, int A, int B, int C)
   mientras N > 1 :
      hanoi(N-1, A, C, B) ;
      mover de A a B ;
      N-- ; T = A ; A = C ; C = T ;
   mover de A a B ;
```

76. **(P+; R)** El pseudocódigo inferior corresponde a un algoritmo para el cálculo del número de hojas de un árbol binario:

```
int numHojas(ab T)
   si T vacio :
      devolver 0 ;
   else si izq(T) vacio y der(T) vacio :
      devolver 1 ;
   else :
      devolver numHojas(izq(T)) + numHojas(der(T)) ;
```

i. Escribir razonadamente la ecuación general en recurrencias para el número de sumas que realiza dicho algoritmo en función de su número de nodos `N`.

ii. Resolver dicha ecuación para árboles binarios completos (si un tal árbol tiene profundidad K, su número de nodos es $2^K - 1$).

iii. ¿Cuál es el rendimiento del algoritmo en el caso peor para árboles binarios generales? Sugerencia: intentar establecer qué situación podría suponer un trabajo máximo del algoritmo.

77. **(P+; R)** Se quiere estimar el número de llamadas recursivas del algoritmo inferior de PreOrden sobre árboles binarios completos. Escribir la correspondiente ecuación recurrente y resolverla.

```
PO(AB T)
   si T != NULL :
      visitar(T) ;
      PO(izq(T)) ;
      PO(der(T)) ;
```

78. **(P+)** En la versión inferior se ha eliminado la segunda recursión de cola del algoritmo de PreOrden. Estimar para árboles binarios completos el nuevo número de llamadas recursivas escribiendo y resolviendo la correspondiente ecuación recurrente.

```
PO2(AB T)
   mientras T != NULL :
      visitar(T) ;
      PO2(izq(T)) ;
      T = der(T) ;
```

79. **(P+)** El algoritmo recursivo inferior calcula los números de Fibonacci:

```
long Fib(int N)
   si N == 0 o N == 1 :
      devolver 1 ;
   else :
      devolver Fib(N-1) + Fib(N-2) ;
```

Si $T(N)$ es el número de sumas que `Fib` ejecuta sobre un entero N, ¿Qué ecuación recurrente verifica $T(N)$?

¿Cuál sería su solución?

80. **(P+; R)** ¿Qué se puede decir de $G(N) = \sum_0^N F_i$, donde F_i es el iésimo número de Fibonacci?

9 HeapSort

Ejercicios

81. **(E)** Dadas las listas
`Q B K C F A G P E D H R`

`34 43 12 58 17 92 21 67 56 72 80`
1. construir sus correspondientes Max heaps indicando la evolución de estos en cada uno de los pasos a dar;
2. usar estos heaps para ordenarlas, indicando la evolución de los mismos en cada uno de los pasos a dar.

82. **(E)** Aplicar el método HeapSort a la ordenación de la lista `8 6 2 10 9 1 3 5 7 4`.

83. **(E)** Dada la lista `[14, 4, 8, 10, 19, 17, 18, 15]`, aplicar a la misma el método HeapSort utilizando un max–heap. Durante la fase de creación del heap, indicar sus estados parciales utilizando su representación sobre un árbol binario. Durante de ordenación, representar el estado del heap y de la lista parcialmente ordenada usando como estructura de datos una tabla.

84. **(E; R)** Dada la lista `[15, 1, 10, 5, 8, 3, 11, 14]`, dar la evolución sobre la misma del método de ordenación HeapSort, indicando en la construcción del max–heap asociado y en la extracción desde el mismo cada uno de sus sucesivos estados.

85. **(E)** Dar la evolución del método HeapSort sobre la tabla `9 8 10 2 5 6 7 12 11`.

Problemas

86. **(P)** Tomando como OB el acceso a un nodo, ¿cuál sería la complejidad en el caso peor para la construcción de un max heap? ¿Cuál sería la complejidad en el caso peor para esta ordenación desde un heap?

    Combinando ambos pasos se consigue un algoritmo de ordenación por comparación de claves. ¿Cuál es su complejidad en el caso peor?

87. **(P+)** Estimar razonadamente $B_{HeapSort}(N)$ y $A_{HeapSort}(N)$

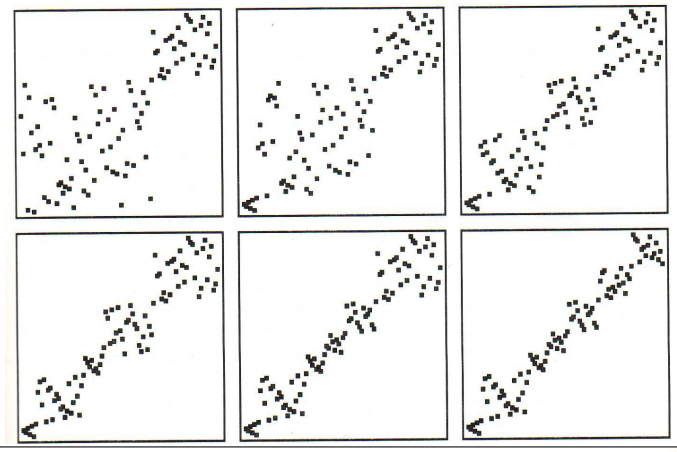
88. **(P)** Los elementos de un heap pueden almacenarse de manera muy sencilla en una tabla haciendo que el elemento que ocupe el lugar i–ésimo del heap cuando éste se recorre por niveles ocupe la posición i–ésima de la tabla. Almacenándolo así, ¿qué relación hay entre el índice i de un nodo y los de sus hijos? ¿Cuál hay entre el índice i de un nodo y el de su padre?

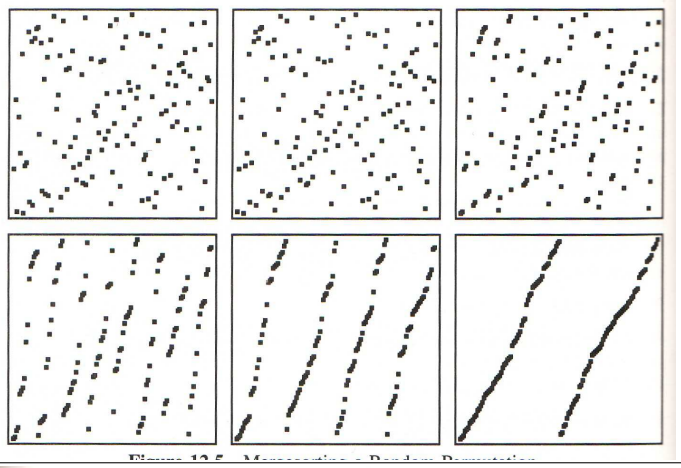
89. **(P) Implementación de HeapSort sobre tablas** Si de acuerdo con el problema anterior se contempla una tabla con índices de 1 a N como un heap, se observa que los únicos elementos que pueden no cumplir la condición de MaxHeap son los situados en los índices $\lfloor N/2 \rfloor, \lfloor N/2 \rfloor - 1, \lfloor N/2 \rfloor - 2, \ldots, 3, 2, 1$.
    Comprobar que reubicando dichos elementos en ese orden se puede conseguir un MaxHeap.

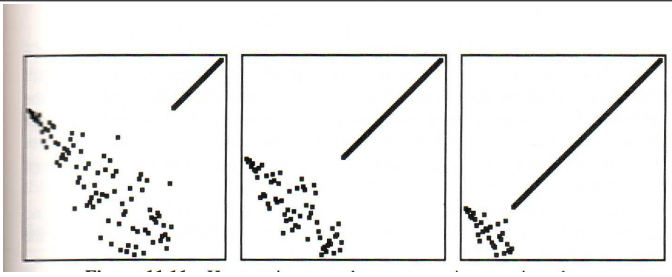
90. **(P)** Una vez creado un MaxHeap sobre una tabla, comprobar que se puede obtener una ordenación de la misma intercambiando repetidamente los elementos en posiciones $N, N-1, \ldots$ con el elemento raíz y reubicando el elemento así "subido".

91. **(P; R)** En los cuadros inferiores se representan tres fases consecutivas de la evolución de una lista de 50 números entre 1 y 50 a la que se aplican ciertos métodos de ordenación. Las coordenadas (i, j) de un punto del gráfico indican que el número de la posición i–ésima de la

lista es precisamente j (por ejemplo, un número i está en su posición correcta si $j = i$, esto es, si el par (i, j) está en la diagonal). ¿Cuáles de los métodos de ordenación estudiados en el curso podría generar estos gráficos?







10 Árboles de decisión

Ejercicios

92. **(E)** Escribir el árbol de decisión de los algoritmos de Burbuja, InsertSort y Selección $N = 3$.

93. **(E; R)** Escribir el árbol de decisión de los algoritmos de QuickSort y MergeSort para $N = 3$.

94. **(E)** Un cierto algoritmo de ordenación tiene el siguiente pseudocódigo:

```
RaroSort(tabla T, ind P, ind U)
   si T[P] < T[P+1]:
      InsertSort(T, P, U) ;
   else:
      MergeSort(T, P, U) ;
```

Dar su árbol de decisión para tablas con 3 elementos.

95. **(E)** Dar razonadamente el árbol de decisión para N=3 del algoritmo inferior de ordenación. Usar la notación `i:j` con $i < j$ para indicar la comparación de los elementos i y j de la tabla inicial `T`.

```
RaroSort(tabla T, dim N)
   si T[1] < T[2]:
      QuickSort(T, N) ; // usar primer elemento como pivote
   else :
      InsertSort(T, N) ;
```

96. **(E+)** Construir razonadamante el árbol de decisión para $N = 3$ para la siguiente variante del algoritmo `SelectSort`.

```
SelectSortMax(tabla T, dim N)
  para i de N a 2:
    iMax=Max(T, 1, i);
    intercambiar T[i] con T[iMax];

indice Max(tabla T, indice P, indice U)
  Max=T[P];
  iMax=P;
  para i de P+1 a U:
    si T[i] > Max:
       Max = T[i];
       iMax=i;
  devolver iMax;
```

97. **(E+; R)** El algoritmo de ordenación por inserción puede ejecutarse sobre el siguiente pseudocódigo:

```
InsertSortMax(tabla T, dim N)
  para i de N-1 a 1:
    A = T[i];
    j = i+1;
    mientras j <= N y T[j] < A:
      T[j-1] = T[j];
      j++;
    T[j-1] = A;
```

Dar razonadamente su árbol de decisión para $N = 3$, señalando tras cada cdc el estado de la evolución de la tabla.

98. **(E+; R)** Construir razonadamente el árbol de decisión para tablas de 3 elementos de la siguiente versión del algoritmo de ordenación de la burbuja:

```
PlomoSort(tabla T, dim N)
  camb = 1;
  ini = 1;
  mientras fin < N y camb == 1:
    camb = 0;
    para i de N a ini+1:
      si T[i] < T[i-1]:
        intercambiar T[i] y T[i-1];
        camb = 1;
    ini++;
```

Dar también el árbol de decisión si en el código se eliminan las sentencias con el indicador de intercambios `camb`.

99. **(E)** a. Dada una tabla T, el algoritmo InsertSort efectúa en primer lugar la comparación entre $T[1]$ y $T[2]$. Suponiendo que $T[1] > T[2]$, ¿contra qué elemento $T[j]$ se compara entonces $T[1]$?

    b. Suponiendo que $T[1]$ gana dicha comparación, dar razonadamente el subárbol de decisión que sigue a la misma para tablas de 4 elementos (esto es, el subárbol que sigue por la derecha a la raíz es $1:2$, y a su hijo derecho de la forma $1:j$). Tras cada cdc, indicar el estado de la tabla y marcar el elemento a insertar.

Problemas

100. **(E+; R)** Sobre una tabla de cuatro elementos el algoritmo MergeSort realiza en primer lugar la comparación `1:2` y si el primer elemento resulta mayor, se realiza la comparación `3:4`. Dar razonadamente el resto del árbol de decisión del MergeSort para tablas de 4 elementos suponiendo que el primer elemento es mayor que el segundo y que el tercero es menor que el cuarto. Indicar tras cada iteración el estado de las tablas implicadas.

101. **(E+; R)** El siguiente pseudocódigo da una versión "descendente" del algoritmo de la Burbuja en la que la lista comienza a ordenarse desde su inicio.

```
PS(tabla T, dim N)
  k = 1;
  mientras k < N :
    para i de N a k+1:
      si T[i] < T[i-1]:
        intercambiar T[i] y T[i-1];
    k++;
```

En una tabla de 4 elementos `PS` realiza 3 : 4 como primera cdc. Si $T[3] > T[4]$, ¿Cuál es la siguiente cdc? Suponiendo que en la siguiente cdc es menor el elemento de mayor índice, construir razonadamente el subárbol de decisión de T^4_{PS} resultante.

102. **(E+; R)** El algoritmo inferior es una variante "descendente" de InsertSort.

```
ISD(tabla T, dim N)
  para i de N-1 a 1 :
    j=i ;
    mientras j < N y T[j] > T[j+1] :
      swap(T[j], T[j+1]) ;
      j++ ;
  volver;
```

En una tabla de 4 elementos `ISD` realiza 3 : 4 como primera cdc. Si $T[3] > T[4]$, ¿Cuál es la siguiente cdc? Suponiendo que en la siguiente cdc es menor el elemento de mayor índice, construir razonadamente el subárbol de decisión de T^4_{ISD} resultante.

103. **(E+; R)** Dar razonadamente el árbol de decisión del algoritmo HeapSort para tablas de 3 elementos.

104. **(P; R)** La construcción de un ABdB puede verse como una forma particular de ordenación de listas. ¿Cuál sería su árbol de decisión para $N = 3$?

105. **(P; R)** Suponiedo que la construcción de un ABdB se aplica a tablas de $N = 4$ elementos, la primera cdc que se efectúa es `1:2` y la segunda 1:3. Dar razonadamente el subárbol del árbol de decisión $T^4_{CrearABdB}$ suponiendo que `T[1] > T[2]` y que `T[1] < T[3]` representando tras cada cdc el estado de los ABdB parciales resultantes.

106. **(P; R)** Probar que si T es un 2-árbol con N nodos entonces T tiene $N + 1$ hojas (sugerencia: inducción).

107. **(P)** Establecer de manera precisa la relación entre la profundidad de un árbol binario y su número de hojas.

108. **(P)** Si un árbol binario completo contiene N elementos. ¿cuál es su profundidad? ¿cuál es su lce?

109. **(P+; R)** La longitud de caminos internos (lci) de un árbol T se define como

$$lci(T) = \sum_{\{N:\text{nodo interno de } T\}} prof(N).$$

Probar que si T es un 2-árbol con N nodos internos, entonces $lce(T) = lci(T) + 2N$ (sugerencia: inducción).

11 Algoritmos básicos de búsqueda

Ejercicios

110. **(E)** Dada la tabla T

     `1 15 2 14 3 13 4 12 5 11`

     escribir su árbol binario de búsqueda B.
     Dar los árboles de búsqueda B que resultan de extraer del anterior los elementos `15, 13, 3` en ese orden.

111. **(E+)** Calcular explícitamente $A_{BBin}(N)$ para $N = 3$ y $N = 7$ (suponer equiprobabilidad).

Problemas

112. **(P)** Estimar el coste medio de la primitiva `Borrar` en un diccionario que usa como edd una tabla ordenada.

113. **(P; R)** Probar que si un nodo de un ABdB tiene 2 hijos, su sucesor sólo tiene a lo sumo el hijo derecho.

114. **(P)** Si `N` es un nodo de un ABdB, su predecesor es el nodo `M` tal que `info(M)` es la clave inmediatamente anterior a `info(N)`. Probar que si un nodo de un ABdB tiene 2 hijos, su predecesor sólo tiene a lo sumo el hijo izquierdo.

115. **(P+; R)** Dar el pseudocódigo de una función `Sucesor` que reciba un puntero a un nodo de un árbol binario y devuelve un puntero a su sucesor.

116. **(P+)** Dar el pseudocódigo de una función `Predecesor` que reciba un puntero a un nodo de un árbol binario y devuelve un puntero a su predecesor.

12 Árboles AVL

Ejercicios

117. **(E)** Construir el AVL asociado a la lista `1 2 3 4 5 6 7 15 14 13 12 11 10 9 8`

118. **(E)** Construir el AVL asociado a la lista inversa de la anterior (de 15 a 8 y de 1 a 7), indicando en cada paso con detalle las rotaciones necesarias.

119. **(E)** Construir el árbol AVL asociado a la lista `1 15 2 14 3 13 4 12 5 11 6 10 7 9 8`.

120. **(E; R)** Construir razonadamente el árbol AVL para la lista `9 8 10 2 5 6 7 12 11`.

121. **(E; R)** Construir todos los árboles de Fibonacci de profundidades 2, 3 y 4 (¡no hay tantos!).

Problemas

122. **(P)** Dar el esquema general de una rotación a derechas y de una rotación doble izquierda-derecha.

123. **(P+)** Probar que en el proceso de creación de un AVL no pueden darse desequilibrios de la forma (2, 0) o (0, 2).

124. **(P+)** Intentar probar que en la construcción de un AVL basta arreglar el desequilibrio más profundo para que los superiores se arreglen automáticamente.

125. **(P+)** Un **árbol de Fibonacci** de profundidad P es un árbol AVL con dicha profundidad y un número mínimo de nodos. Demostrar que si T es un árbol de Fibonacci de profundidad P, entonces T_i es un árbol de Fibonacci de profundidad $P-1$ y T_d es un árbol de Fibonacci de profundidad $P-2$, o al revés, T_i es un árbol de Fibonacci de profundidad $P-2$ y T_d es un árbol de Fibonacci de profundidad $P-1$.

126. **(P+)** Supongamos que los conejitos se hacen adultos en un mes y que una pareja de conejos adultos, sea cual sea su sexo, producen una pareja de conejitos al mes. Si dos conejos jóvenes náufragos llegan a una isla desierta, ¿cuántas parejas de conejos habrá en la isla tras N meses? (por ejemplo, tras dos meses habría dos parejas, una de conejitos y otra de conejos adultos).

127. **(P+; R)** Probar por inducción que el n–ésimo número de Fibonacci F_N cumple

$$F_N = \frac{1}{\sqrt{5}}(\phi^{N+1} - \psi^{N+1})$$

con $\phi = (1+\sqrt{5})/2$ y $\psi = (1-\sqrt{5})/2$ y deducir de aquí el orden de la función $f(N) = F_N$.

128. **(P+; R)** Calcular el orden de $G(N) = \sum_0^N F_i$, donde F_i es el iésimo número de Fibonacci.

129. **(P+; R)** ¿Cuántos árboles de Fibonacci de profundidad P hay?

13 Hashing

Ejercicios

130. **(E)** Construir una función hash adecuada para una tabla hash con encadenamiento y 9 punteros. ¿Cómo se insertarán en la misma los valores 5, 28, 19, 15, 20, 33, 12, 17 y 10?

131. **(E; R)** Se quiere construir una tabla hash con encadenamiento para almacenar 1000 datos. Si se considera aceptable un promedio de 4 colisiones tanto en búsquedas con éxito como sin éxito, ¿cuál puede ser un tamaño de tabla adecuado?

132. **(E; R)** Se quiere construir una tabla hash con direccionamiento abierto y sondeo lineal para almacenar 1000 datos. Si se considera aceptable un promedio de 5 colisiones en búsquedas sin éxito, ¿cuál puede ser un tamaño de tabla adecuado?

    Para el tamaño resultante de la tabla anterior, ¿cuál será el número medio de colisiones en búsquedas con éxito?

133. **(E+; R)** Se puede demostrar que usando sondeos cuadráticos en una tabla de dimensión m con N datos, el número medio de sondeos necesarios en una búsqueda fallida es

$$A^f_{SC}(N,m) \approx \frac{1}{1-\lambda} - \lambda + \log \frac{1}{1-\lambda},$$

donde λ es el factor de carga. ¿Cuál sería número medio $A^e_{SC}(N,m)$ de sondeos necesarios en una búsqueda con éxito?

134. **(E+; R)** De un cierto método hash por **encadenamiento** HX se sabe que $A^f_{HX}(N,m) = \lambda^2$. Hallar de manera suficientemente razonada el valor de $A^e_{HX}(N,m)$.

135. **(E+)** De un cierto método hash por direccionmiento abierto SX se sabe que $A^f_{SX}(N,m) = 1 + 1/(1-\lambda)^2$. Hallar razonadamente el valor de $A^e_{SX}(N,m)$.

Problemas

136. **(P; R)** En general y sobre tablas hash, ¿qué será más alto, el coste medio de búsquedas fallidas o el de búsquedas con éxito? Razonar la respuesta.

137. **(P+; R)** Se quieren construir tablas hash dinámicas con direccionamiento abierto y sondeo lineal según el siguiente esquema denominado **rehashing**: cuando en una tabla de tamaño N el factor de carga llega a 0.5, se construye una nueva tabla de tamaño $2N$ donde se reinsertan los elementos de la tabla previa.

    En promedio, ¿'cuántos sondeos se precisan para construir así una tabla de N elementos? ¿Cuál es el coste extra de este procedimiento frente al de la construcción directa de una tabla con $2N$ posiciones para N datos?

138. **(P; R)** Si en una tabla hash con direccionamiento abierto se utilizan sondeos cuadráticos para resolver colisiones puede haber claves imposibles de insertar aunque la tabla posea espacios libres. Comprobar este efecto cuando la tabla tiene tamaño 16 y se usa como función hash $h(K) = K\%16$.

139. **(P+; R)** Probar que si en una tabla hash de tamaño M, con M primo, con direccionamiento abierto y factor de carga $\lambda < 1/2$, se utilizan sondeos cuadráticos para resolver colisiones, siempre es posible insertar una nueva clave K.

    Sugerencia: suponer ya insertados P elementos, con $P < \lfloor M/2 \rfloor$ y comprobar que dada una clave K, cualquiera que sea la función de hash h, todos los valores del conjunto

$$\Big\{ h(K) + i^2 \ : \ 0 \le i \le \lfloor M/2 \rfloor \Big\}$$

son distintos; observar que en dicho conjunto hay más de P elementos.

140. **(P)** Dar el pseudocódigo de unas rutinas que busquen, inserten y borren elementos de una tabla hash en la que se utilice direccionamiento abierto y resolución de colisiones mediante sondeos lineales.

141. **(P)** En hashing con encadenamiento se supone construida una rutina `BusqH(clave K, tablaH T)` que recibe una clave `K` y devuelve un puntero a un nodo de la lista apuntada por `T[h(K)]` que contiene dicha clave o `NULL` si ningún nodo la contiene.

Construir mediante ella las rutinas `BorrarH(clave K, tablaH T)` y `InsH(clave K, tablaH T)` que borren o inserten la clave `K` del o en el nodo apropiado de la tabla de hash `T`. Construir mediante ésta última otra rutina `CrearH(tabla R, tablaH T)` que a partir de una tabla `R` proporcione un puntero a una tabla de hash que contenga los elementos de `R`.