UNIVERSIDAD DE GRANADA

/ UGR / decsai

Departamento de Ciencias de la Computación e Inteligencia Artificial

Asignatura: Algorítmica
Grado en Ingeniería Informática
Relación de problemas: Tema 1

1. Introducción

Para elaborar los ejercicios es recomendable utilizar la siguiente bibliografía:

- G. Brassard, P. Bratley, "Fundamentos de Algoritmia", Prentice Hall, 1997
- J.L. Verdegay: Lecciones de Algorítmica. Editorial Técnica AVICAM (2017)

Antes de realizar los ejercicios prácticos, se recomienda al estudiante revisar y comprender las diapositivas estudiadas en clase y complementarlas con la información procedente de la bibliografía básica y recomendada en la guía docente de la asignatura.

En la relación de problemas se presentarán, para cada temática, un conjunto de preguntas sobre conceptos teóricos que el alumno debe dominar antes de abordar los ejercicios prácticos.

2. Órdenes de eficiencia: Conceptos

1. Describa el Principio de Invarianza
2. ¿Qué el "caso de un problema" y en qué consiste el caso peor?
3. ¿Qué significa que un algoritmo es de orden O? ¿Y de orden Ω? ¿Y de orden Θ? ¿Qué relación existe entre el orden O y el Principio de Invarianza? ¿Qué relación existe entre el orden Ω y el caso mejor? ¿Qué quiere decir que un algoritmo tiene un orden Θ?
4. ¿En qué consiste la regla del máximo? ¿Y la de la suma? ¿Y la del producto? Pon ejemplos.
5. ¿Cómo podemos saber si un orden de eficiencia es equivalente a otro, o si es mayor o menor?
6. ¿Cómo calculamos el orden O de un algoritmo si el problema que resuelve depende de varios parámetros? Pon un ejemplo de problema cuyo tamaño dependa de varios parámetros.

3. Órdenes de eficiencia: Ejercicios prácticos

7. Si decimos que dos algoritmos tienen eficiencia O(n), ¿significa esto que ambos son igual de eficientes y que tardan el mismo tiempo en resolver el mismo problema? Si tuvieras que escoger uno de ellos para resolver un problema, ¿cuál sería? Razona tu respuesta dando argumentos basados en el Principio de Invarianza, relacionando la eficiencia de ambos mediante las constantes ocultas y, finalmente, utilizando la definición de orden O y orden Ω de un algoritmo.
8. Dos algoritmos tienen un tiempo de ejecución $T_1(n)= 100n^2$ minutos y $T_2(n)= 5n^3$ minutos. ¿Cuál es el orden de eficiencia de ambos? ¿En qué casos es más eficiente usar un algoritmo u otro? Calcula el tamaño de caso n_0 para el cual, a partir de ese tamaño de problema, uno de los algoritmos es mejor que el otro.

9. Haciendo uso del Principio de Invarianza y la definición de órdenes de eficiencia, demuestre que se cumplen las propiedades reflexiva, simétrica y de la suma para los órdenes de eficiencia estudiados en clase.
10. Justifica la equivalencia o no equivalencia de los siguientes órdenes de eficiencia. Para los que no sean equivalentes, ordénalos de menor a mayor orden de eficiencia: O(n), $O(n^5)$, O(n!), O((n+1)!), $O(n^n)$, $O(\sqrt{n})$, $O(2^n)$, $O(2^{n+1})$, $O(1.5^n)$, O(n*log(n)), O(log(n)).
11. Sea un algoritmo cuyo tiempo de ejecución viene dado por la expresión $T(n)= 1.2n^2+n+2^{n+1}$. Calcular el orden O del algoritmo. Justifica cómo has calculado dicho orden.
12. Dos algoritmos A y B que resuelven el mismo problema son $O(n^2)$. Además, el algoritmo A es $\Theta(n^2)$, pero B no es $\Theta(n^2)$. ¿Cuál de ellos es mejor, en términos asintóticos (cuando $n \to \infty$)? ¿Cuál de ellos implementarías para resolver el problema? Razona tu respuesta usando las definiciones de los órdenes de eficiencia y el Principio de Invarianza, en el caso de ser necesario.
13. Sean dos funciones f(n) y g(n). Demostrar, utilizando las definiciones de los órdenes de eficiencia, las siguientes afirmaciones:

$$a)\ \lim_{n\to\infty}\frac{f(n)}{g(n)}\in\Re^+ \to f(n)\in\Theta(g(n))$$

$$b)\ \lim_{\to\infty}\frac{f(n)}{g(n)}=0 \to f(n)\in O(g(n)),\ pero\ f(n)\notin\Theta(g(n))$$

$$c)\ \lim_{n\to\infty}\frac{f(n)}{g(n)}=+\infty \to f(n)\in\Omega(g(n)),\ pero\ f(n)\notin\Theta(g(n))$$

14. Calcule el orden de eficiencia que tendrían los algoritmos cuyo tiempo de ejecución se expresa en los siguientes apartados:
    a. $T(n)=n^2$
    b. $T(n)=(n+n^2)^2$
    c. $T(n)=\sqrt{n^2+n}$
    d. $T(n)=n!/(n+1)$
    e. $T(n)=(n+n^2)^2$
15. La eficiencia de un algoritmo A viene dada por $T_A(n)= 0.1n^2$, mientras que la de otro algoritmo B viene dada por $T_B(n)= 100n$. ¿Cuál es el primer tamaño de caso para el que el algoritmo B es mejor que el algoritmo A?

4. Análisis de algoritmos: Conceptos

16. ¿Qué es una operación elemental? ¿Cuál es su orden de eficiencia? ¿Qué es lo que hace que una operación sea considerada como elemental? Pon ejemplos de operaciones elementales.
17. ¿Cómo se calcula el orden de eficiencia de una sentencia condicional?
18. ¿Cómo se calcula el orden de eficiencia de un bucle for? Explica el caso particular de los bucles for en C++.
19. ¿Cómo se calcula el orden de eficiencia de un bucle while o do..while? ¿Porqué suele ser más complicado calcular la eficiencia de estos bucles con respecto a un bucle for?
20. ¿Cómo se calcula la eficiencia de una secuencia de sentencias?
21. ¿Cómo se calcula el orden de eficiencia de una función?

22. Cuando dentro de un programa haces una llamada a una función, ¿cuál es el orden de eficiencia de la sentencia que hace la llamada a la función? Expón los casos que puedan darse y justifica en cada uno cuál sería el criterio que usas para decir la eficiencia que tiene dicha llamada.

5. Análisis de algoritmos: Ejercicios prácticos

23. **Ejercicio resuelto:** Calcular la eficiencia del siguiente algoritmo:

```
void selección(double *v, int inicio, int final) {

   int i, j, min;

   for (i= inicial; i<final; i++) {
      min= i;
      for (j= i+1; j<final; j++)
         if (v[j] < v[min])
            min= j;
      Intercambia(v[i], v[min]);
   }
}
void Intercambia(int &a, int &b) {
   int aux= a; a= b; b= aux;
}
```

Solución:

El algoritmo resuelve el problema de ordenación de un subvector, desde una posición "inicial" a una posición "final", (final no inclusive). Se trata del algoritmo de ordenación por selección para ordenar vectores, particularizado para la ordenación de un subvector.

*El tamaño del problema depende de un único parámetro **n**, que es la longitud del subvector a ordenar. Por tanto,*

$$n=final-inicial$$

Comenzamos analizando la función externa Intercambia. Está formada por 3 sentencias simples, consideradas como operaciones elementales. La eficiencia de la secuencia de estas 3 sentencias simples se calcula como el max{O(1), O(1), O(1)}= O(1), por lo que la función Intercambia tiene eficiencia O(1).

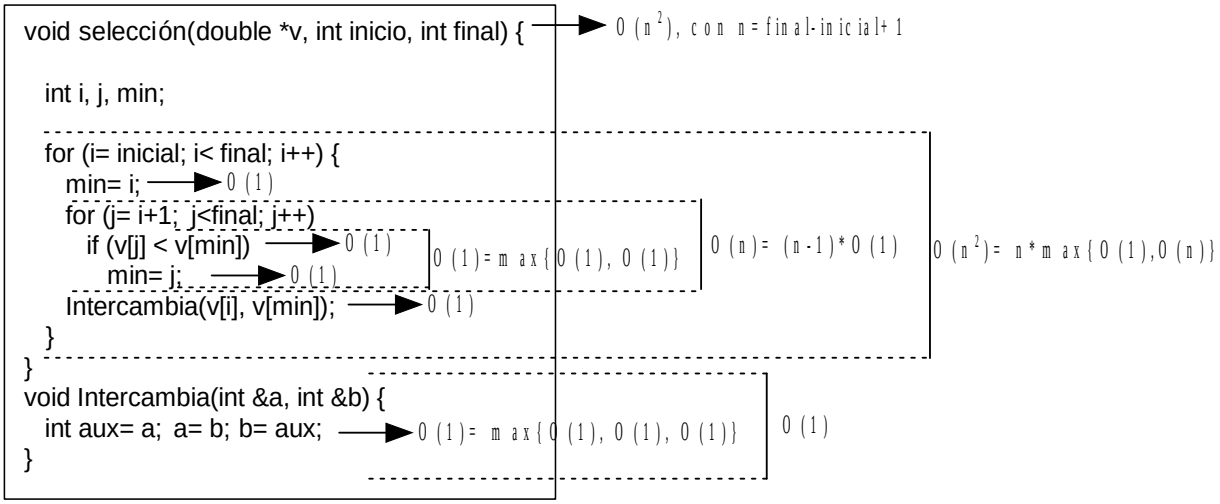
Para analizar el algoritmo selección, comenzaremos desde el código más interno al algoritmo y finalizaremos por las sentencias del primer nivel. Comenzamos por la parte del "if". En esta sentencia condicional, tanto la evaluación de la condición como la sentencia que se ejecuta cuando esta es verdadera son O(1) porque son sentencias simples. Por tanto, la sentencia "if" es de eficiencia max{O(1), O(1)}= O(1). Ahora pasemos a analizar el siguiente nivel, el bucle "for j":

Llamemos "k" al tiempo constante que tarda en ejecutarse la sentencia condicional interna al bucle. El peor caso que pueda darse para que el bucle se ejecute el máximo número de veces posible es cuando j=i+1 e i=inicial. En este caso, el bucle se ejecutará final-inicial-1 veces; es decir, n-1 veces, y el tiempo que tardará en ejecutarse todo el bucle será k(n-1)= k*n-k<=k*n. Por tanto, la eficiencia del bucle interno "For j" es O(n).*

Antes de analizar el bucle externo, tenemos que analizar la secuencia de instrucciones que contiene. La asignación "min=i" y la llamada a "intercambia" son O(1) porque se consideran operaciones elementales. Como el bucle interno "for j" es O(n), la eficiencia de esta secuencia de instrucciones es max{O(1), O(n), O(1)}= O(n).

*Analicemos el bucle externo. Este se ejecutará final-inicial veces; es decir, n veces. Por tanto, la eficiencia del mismo será n*O(n); es decir, $O(n^2)$.*

Para finalizar el análisis, las sentencias que quedan en el algoritmo son definiciones de variables de tipos básicos "int i, j, min;", que son operaciones elementales cuya eficiencia es O(1). Por tanto, la eficiencia de la función "selección" es el max{O(1), $O(n^2)$}= $O(n^2)$.



24. El siguiente algoritmo devuelve la longitud de una cadena de caracteres. ¿Cuál/cuáles son los parámetros que definen el tamaño del problema? Analiza y calcula la eficiencia del algoritmo.

```
int longitud(char v[]) {

 int i= 0;
 while (v[i] != '\0') i++;
 return i;
}
```

25. El siguiente algoritmo tiene como entrada un vector de enteros y devuelve en dos vectores separados los números pares y los impares contenidos en el primer vector. ¿Cuál/cuáles son los parámetros que definen el tamaño del problema? Analiza y calcula la eficiencia del algoritmo.

```
void DesglosaParesImpares(const int original[], int nOriginal,
                                        int pares[], int & nPares,
                                        int impares[], int & nImpares) {

  int i;

  // Inicialmente, pares e impares no tienen componentes utiles
  nPares= 0;
  nImpares= 0;

  for (i= 0; i<nOriginal; i++) {

            if (original[i]%2 == 0) {
                      pares[nPares]= original[i];
                      nPares++;
            } else {
                      impares[nImpares]= original[i];
                      nImpares++;
            }
  }
}
```

26. El siguiente algoritmo comprueba si una cadena de caracteres es palíndromo. ¿Cuál/cuáles son los parámetros que definen el tamaño del problema? Analiza y calcula la eficiencia del algoritmo.

```
bool esPalindromo(char v[]) {
  bool pal= true; // Suponemos que lo es
  int inicio= 0, fin= strlen(v)-1; // Inicio y fin de la cadena

  while ((pal) && (inicio<fin)) {

            if (v[inicio] != v[fin])
                      pal= false;
            inicio++;
            fin--;
  }
  return pal;
}
```

27. El siguiente algoritmo busca un elemento en un vector ya ordenado. ¿Cuál/cuáles son los parámetros que definen el tamaño del problema? Analiza y calcula la eficiencia del algoritmo.

```
int Busqueda (int *v, int n, int elem) {

  int inicio, fin, centro;

  inicio= 0;
  fin= n-1;
```

```
 centro= (inicio+fin)/2;
 while ((inicio<=fin) && (v[centro] != elem)) {

            if (elem<v[centro])
                        fin= centro-1;
            else
                        inicio= centro+1;
             centro= (inicio+fin)/2;
}

 if (inicio>fin)
            return -1;

 return centro;
}
```

28. El siguiente algoritmo busca un elemento en un vector ya ordenado. ¿Cuál/cuáles son los parámetros que definen el tamaño del problema? Analiza y calcula la eficiencia del algoritmo.

```
int Busqueda (int *v, int n, int elem) {

 bool encontrado= false;
 int i;

 for (i= 0; i<n && !encontrado; i++)
            encontrado= (v[i] == elem);

 if (encontrado)
            return i-1;

 return -1;
}
```

29. El siguiente algoritmo multiplica dos matrices A_{nxk} y B_{kxm} para dar como resultado la matriz C_{nxm}. ¿Cuál es el tamaño del caso del problema a resolver? ¿Cuál es la eficiencia del algoritmo propuesto?

```
void Multiplica(double **A, double **B, int n, int k, int m, double **C) {

            for (int i= 0; i<n; i++) {
                        for (int j= 0; j<m; j++) {
                                    C[i][j]= 0;
                                    for (int h= 0; h<k; h++)
                                                C[i][j]+= A[i][k]*B[k][j];
                        }
            }
}
```

30. El siguiente código realiza la ordenación de un vector por el método QuickSort. Calcule la ecuación en recurrencias del algoritmo, para los casos mejor y peor.

```
void QuickSort(int* v, int ini, int fin) {

     int pospivote;
     if(ini < fin){
        pospivote= pivotar(v, ini, fin);
        QuickSort(v, ini, pospivote-1);
        QuickSort(v, pospivote+1, fin);
     }
}
```

```
int pivotar(int *v, int ini, int fin) {
    int i= ini, j= fin, aux, pivote;

    pivote= v[ini];
    do { i++; } while (v[i]<=pivote && i<=fin);
    do { j--; } while (v[j]>pivote);
    while (i<j) {
        aux=v[i]; v[i]=v[j]; v[j]=aux;
        do { i++; } while (v[i]<=pivote && i<=fin);
        do { j--; } while (v[j]>pivote);
    }
    aux=v[ini]; v[ini]=v[j]; v[j]=aux;
    return j;
}
```

31. El siguiente código realiza la ordenación de un vector por el método HeapSort. Calcule la ecuación en recurrencias del algoritmo, para los casos mejor y peor.

```
void HeapSort(int *v, int n){

  double *apo=new double [n];
  int tamapo=0;

  for (int i=0; i<n; i++){
    insertar(apo,tamapo,v[i]);
  }
  for (int i=0; i<n; i++) {
    v[i]=apo[0];
    borrarRaiz(apo,tamapo);
  }
  delete [] apo;

}
```

```
void insertar(double *apo, int &tamapo, double valor){

  apo[tamapo]=valor;
  tamapo++;
  int aux =tamapo-1;
  bool fin =false;
  while (!fin) {
    int padre;
    if (aux==0) {
      fin=true;
    }else{

      if (aux%2==0) {
        padre=(aux-2)/2;
      }else{
        padre=(aux-1)/2;
      }

      if (apo[padre] > apo[aux]) {
        double tmp=apo[aux];
        apo[aux]=apo[padre];
        apo[padre]=tmp;
        aux=padre;

      }else{
        fin=true;
      }
  }
```

```
void borrarRaiz(double *apo, int &tamapo){

  apo[0]=apo[tamapo-1];
  tamapo--;

  int aux=0;
  bool fin = false;

  while (!fin) {
    if (2*aux+1 >= tamapo) fin=true;
    else{
      int minhijo=2*aux+1;
      if ((minhijo+1 < tamapo) && (apo[minhijo]>apo[minhijo+1])) minhijo++;
      if (apo[aux]>apo[minhijo]) {
        double tmp = apo[aux];
        apo[aux]=apo[minhijo];
        apo[minhijo]=tmp;
        aux=minhijo;
      }else fin=true;
    }
  }

}
```

32. El siguiente código calcula el n-ésimo elemento de la sucesión de Fibonacci, mediante un algoritmo iterativo. Calcule la eficiencia del algoritmo. ¿Cuál es más eficiente, si lo comparamos con el algoritmo recursivo estudiado en clase?

```
73 int fibonacciIteracion (int n){
74
75   int f, fn_1=0, fn_2=0;
76
77   if (n<=1) return n;
78   else{
79     fn_1=1;
80     fn_2=0;
81
82     for (int i = 2; i<=n; i++) {
83       f= fn_1+fn_2;
84       fn_2 = fn_1;
85       fn_1 = f;
86     }
87   }
88   return f;
89 }
```

33. Calcule la ecuación en recurrencias del código siguiente:

```
int maximo(int v[], int n) {

 int resul, aux;

 if (n==0)
        resul= v[0];
 else {
        aux= maximo(v, n-1);
        if (aux>v[n])
                resul= aux;
        else
                resul= v[n];
 }
 return resul;
}
```

34. Calcule la ecuación en recurrencias del código siguiente:

```
int minimo(int v[], int posIni, int posFin) {

 int resul1, resul2;

 if (posIni==posFin)
          return v[posIni];
 else {
          resul1= minimo(v, posIni, (posIni+posFin)/2);
          resul2= minimo(v, (posIni+posFin)/2+1, posFin);
          return (resul1 > resul2) ? resul2 :resul1 ;
 }
}
```

35. Calcule la eficiencia del siguiente código:

```
1:  void ejemplo1 (int n)
2:  {
3:    int i, j, k;
4:
5:    for (i = 0; i < n; i++)
6:       for (j = 0; j < n; j++)
7:       {
8:          C[i][j] = 0;
9:          for (k = 0; k < n; k++)
10:             C[i][j] += A[j][k] * B[k][j];
11:      }
12: }
```

36. Calcule la eficiencia del siguiente código:

```
1:  long ejemplo2 (int n)
2:  {
3:    int  i, j, k;
4:    long total = 0;
5:
6:    for (i = 1; i < n; i++)
7:       for (j = i+1; j <= n; j++)
8:          for (k = 1; k <= j; k++)
9:             total += k*i;
10:
11:   return total;
12: }
```

37. Calcule la eficiencia del siguiente código:

```
1:  void ejemplo3 (int n)
2:  {
3:    int i, j, x=0, y=0;
4:
5:    for (i = 1; i <= n; i++)
6:       if (i % 2 == 1)
7:       {
8:          for (j = i; j <= n; j++)
9:             x++;
10:         for (j = 0; j < i; j++)
11:            y++;
12:      }
13: }
```

38. Calcule la eficiencia del siguiente código:

```
1:  int ejemplo4 (int n)
2:  {
3:    if (n <= 1)
4:       return 1;
5:    else
6:       return (ejemplo4(n - 1) + ejemplo4(n - 1));
7:  }
```

39. Calcule la eficiencia del siguiente código:

```
1:  int ejemplo5 (int n)
2:  {
3:    if (n == 1)
4:       return n;
5:    else
6:       return (ejemplo5(n/2) + 1);
7:  }
```

6. Análisis de algoritmos recursivos: Conceptos

40. Los algoritmos recursivos se analizan de una forma diferente a los iterativos, planteando su tiempo de ejecución en función del tiempo que tardaría en ejecutarse el algoritmo para el caso o los casos de tamaño menor. ¿Qué tipos de ecuaciones aprendemos a resolver en la asignatura?
41. Para resolver una ecuación en recurrencias lineal homogénea es necesario, en primer lugar, conseguir la ecuación característica. ¿Qué pasos hay que seguir para conseguir dicha ecuación. Pon un ejemplo, explicando en qué consiste cada paso.
42. ¿Qué relación existe entre el Teorema Fundamental del Álgebra y la resolución de la ecuación característica en una ecuación recurrente lineal homogénea? ¿Qué es el polinomio característico?

¿Cuál es la fórmula que finalmente expresa la solución de la ecuación? Escríbela y describe qué significa cada elemento de la misma. Continúa con el ejemplo desarrollado en el ejercicio anterior para dar la solución final a la ecuación en recurrencias y calcular el orden del algoritmo.

43. ¿En qué se diferencia una ecuación lineal homogénea de otra lineal no homogénea? Escribe la forma general de una ecuación característica de una ecuación lineal no homogénea, y describe cada uno de sus elementos.
44. Pon un ejemplo de ecuación en recurrencias lineal no homogénea. Resuélvela explicando cada paso dado y relacionándolos con los ejercicios anteriores.
45. ¿En qué consiste el cambio de variable? ¿Cuándo debe aplicarse? Pon un ejemplo de ecuación en recurrencias en el que sea necesario aplicar un cambio de variable, y propón el cambio de variable a realizar. Resuelve la ecuación y, finalmente, di cuál es el orden de eficiencia del algoritmo de cuyo código se ha extraído la ecuación en recurrencias.
46. ¿En qué consiste el cambio de rango? ¿Cuándo debe aplicarse? Pon un ejemplo de ecuación en recurrencias que requiera de un cambio de rango y resuelve la ecuación. ¿Cuál es el orden de eficiencia del algoritmo del que se extrajo la ecuación en recurrencias?

7. Análisis de algoritmos recursivos: Ejercicios prácticos

47. **Ejercicio resuelto.** Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: T(n)= 3T(n-1) + 4T(n-2)

Solución:

T(n)= 3T(n-1)+4T(n-2), no necesita cambio de variable ni de rango. Cambiando la notación y despejando, queda:
t_n-3t_{n-1}-4t_{n-2} = 0. Esta es una ecuación lineal y homogénea, que se resuelve como sigue:

Haciendo la sustitución de t_n por x^n, queda:
x^n-3x^{n-1}-4x^{n-2} = 0. Sacando factor común x^{n-2} obtenemos la ecuación característica:
x^2-3x-4 = 0. Resolviendo esta ecuación, obtenemos dos raíces R_1= 4 y R_2= -1, ambas con multiplicidad 1.

Utilizando la fórmula para resolver la recurrencia,

$$t_n = \sum_{i=1}^{r} \sum_{j=0}^{m_i - 1} k_{ij} R_i^n n^j$$

*Tenemos que t_n= k_{10}*4^n + k_{20}*$(-1)^n$*
Por tanto, aplicando la regla de la suma y la del máximo sobre la anterior solución, obtenemos que el
orden de eficiencia del algoritmo es de $O(4^n)$

48. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: T(n)= 4T(n-1)-4T(n-2)

49. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: T(n)= 3T(n-1)-3T(n-2)-T(n-3)

50. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: $T(n)= 2T(n-1) + n^2$

51. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: $T(n)= 2T(n-1) + n + n^2$

52. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: $T(n)= nT^2(n/2)$

53. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: T(n)= T(n-1) + n

54. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: $T(n)= T(n/2)*T^2(n/4)$

55. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: T(n)= 2T(n-1) + 1

56. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: T(n)= 3T(n/2) + n

57. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: $T(n)= 2T(n/2) + \log_2(n)$

58. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: $T(n) = \sqrt{n^2 + n}$

59. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: $T(n) = n!/(n+1)$

60. Calcula la eficiencia del algoritmo cuyo tiempo de ejecución viene dado por la siguiente ecuación en recurrencias: $T(n) = \log(n!)$

61. Calcule la eficiencia de los algoritmos de los ejercicios de la sección 5, para el caso mejor y peor. ¿Cuáles de ellos tienen orden exacto?