Tecnológico de Monterrey.

Campus Querétaro.

TC2038. Análisis y diseño de Algoritmos A

M.C. Ramona Fuentes Valdéz

rfuentes@tec.mx

1

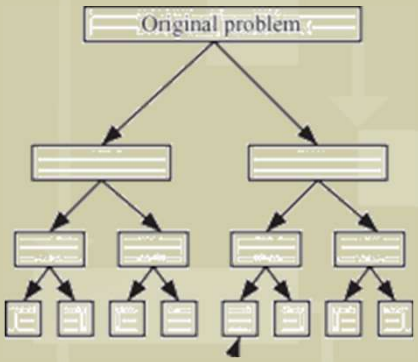
Técnicas de diseño de algoritmos

Introducción

- ¿Cómo diseñar algoritmos?

  Un buen inicio para aprender a diseñar nuevos algoritmos o inclusive, nuevos enfoques para el diseño de algoritmos, es estudiar y entender los enfoques que se han desarrollado y que han sido muy exitosos.

- ¿Con cuáles enfoques iniciamos?
  - Divide y vencerás/conquistarás
    - Programación dinámica
    - Algoritmos avaros
    - Backtracking
    - Ramificación y poda

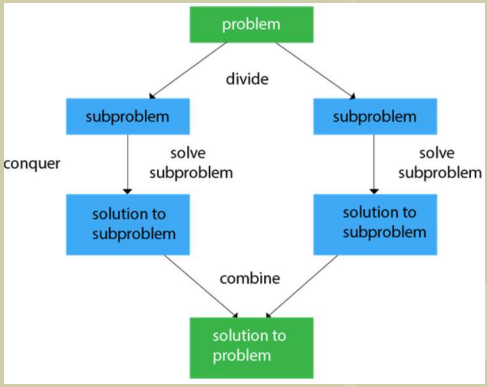


2

Técnicas de diseño de algoritmos

Divide y vencerás (*Divide and conquer*)

1. Divide el problema original en un conjunto de subproblemas más pequeños (dividir), y del mismo tipo del problema original.
2. Resuelve cada subproblema individualmente, de forma recursiva hasta llegar a problemas de solución trivial (conquistar).
3. Combina la solución de los subproblemas (*nivel superior*) en una solución de todo el problema original.

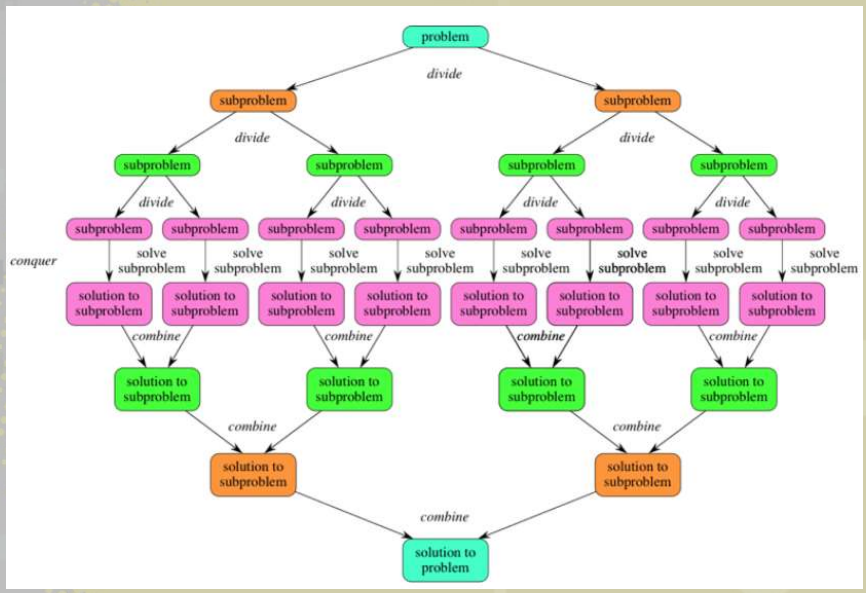


JavaTpoint

3

Técnicas de diseño de algoritmos

Divide y vencerás (*Divide and conquer*)



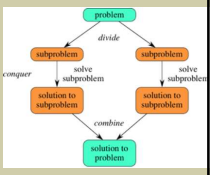
JavaTpoint Khan Academy

4

Técnicas de diseño de algoritmos

Divide y vencerás *(Divide and conquer)*

1. Divide el problema original en un conjunto de subproblemas más pequeños (dividir), y del mismo tipo del problema original.
2. Resuelve cada subproblema individualmente, de forma recursiva hasta llegar a problemas de solución trivial (conquistar).
3. Combina la solución de los subproblemas (*nivel superior*) en una solución de todo el problema original.



➢ Procedimiento

Procedure 1 DIVIDE_AND_CONQUER

Input: X
  if X is simple or known **then**
    return $SOLUTION(X)$
  else
    Decompose X into smaller problems $x_1, x_2, \ldots, x_n$
    for $i \leftarrow 1$ **to** n **do**
      $y_i \leftarrow DIVIDE\_AND\_CONQUER(X_i)$
    end for
    Combine the y_i to get the Y that is solution of X
    return Y
  end if

➢ Complejidad

Se determina de la siguiente manera:

$$T(n) = \begin{cases} 2T(n/2) + S(n) + M(n) & n \geq c \\ b & n < c \end{cases}$$

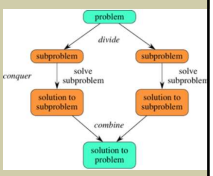
Donde:
- ***S(n)*** : pasos temporales necesarios para dividir el problema en dos subproblemas
- ***M(n)*** : pasos temporales necesarios para fusionar las dos subsoluciones
- ***b*** : constante

5

Técnicas de diseño de algoritmos

Divide y vencerás *(Divide and conquer)*

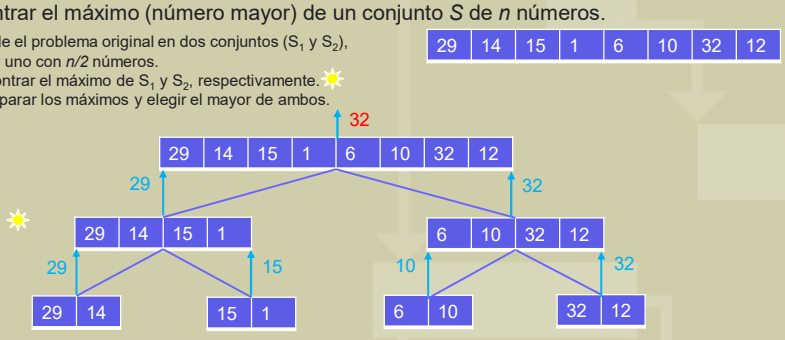
1. Divide el problema original en un conjunto de subproblemas más pequeños (dividir), y del mismo tipo del problema original.
2. Resuelve cada subproblema individualmente, de forma recursiva hasta llegar a problemas de solución trivial (conquistar).
3. Combina la solución de los subproblemas (*nivel superior*) en una solución de todo el problema original.



Ejercicio:

❖ Encontrar el máximo (número mayor) de un conjunto *S* de *n* números.

1. Divide el problema original en dos conjuntos (S_1 y S_2), cada uno con *n/2* números.
2. Encontrar el máximo de S_1 y S_2, respectivamente.
3. Comparar los máximos y elegir el mayor de ambos.

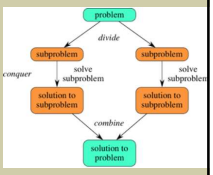


6

Técnicas de diseño de algoritmos

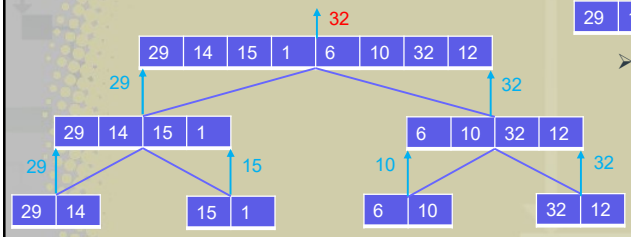
Divide y vencerás (*Divide and conquer*)

1. Divide el problema original en un conjunto de subproblemas más pequeños (dividir), y del mismo tipo del problema original.
2. Resuelve cada subproblema individualmente, de forma recursiva hasta llegar a problemas de solución trivial (conquistar).
3. Combina la solución de los subproblemas (*nivel superior*) en una solución de todo el problema original.



Ejercicio:

- Encontrar el máximo (número mayor) de un conjunto *S* de *n* números.



29 | 14 | 15 | 1 | 6 | 10 | 32 | 12

- Complejidad

$$T(n) = \begin{cases} 2T(n/2) + 1 & n > 2 \\ 1 & n = 2 \end{cases}$$

Si se supone que $n = 2^k$, se tiene:

$$\begin{aligned} T(n) &= 2T(n/2) + 1 \\ &= 2[2T(n/4) + 1] + 1 \\ &\quad \cdots \\ &= 2^{k-1}T(2) + \sum_{j=0}^{k-2} 2^j \\ &= 2^{k-1} + 2^{k-1} - 1 \\ &= 2^k - 1 \\ &= n - 1 \end{aligned}$$

- Complejidad

$$T(n) = \begin{cases} 2T(n/2) + S(n) + M(n) & n \geq c \\ b & n < c \end{cases}$$

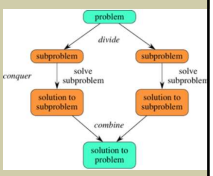
Para el problema de encontrar el máximo, el separar y fusionar requiere un número constante de pasos.

7

Técnicas de diseño de algoritmos

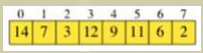
Divide y vencerás (*Divide and conquer*)

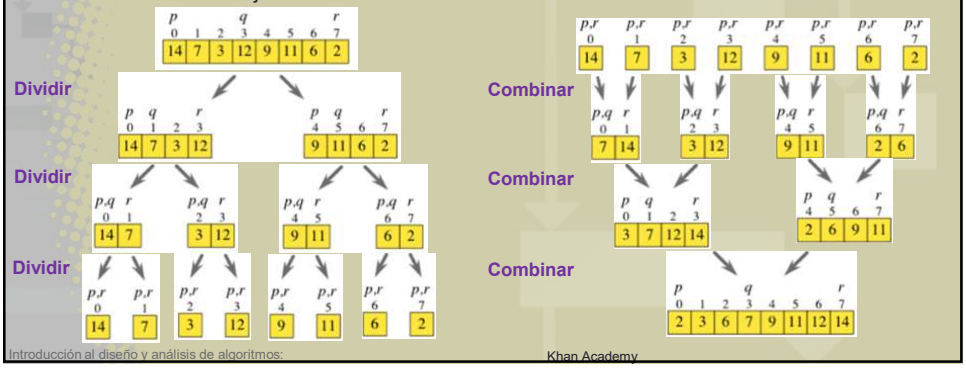
1. Divide el problema original en un conjunto de subproblemas más pequeños (dividir), y del mismo tipo del problema original.
2. Resuelve cada subproblema individualmente, de forma recursiva hasta llegar a problemas de solución trivial (conquistar).
3. Combina la solución de los subproblemas (*nivel superior*) en una solución de todo el problema original.



Ejercicio:

- Ordenar el conjunto *S* de *n* números.



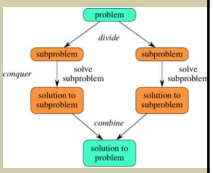


 Khan Academy

8

Técnicas de diseño de algoritmos

Divide y vencerás (*Divide and conquer*)



1. Divide el problema original en un conjunto de subproblemas más pequeños (dividir), y del mismo tipo del problema original.
2. Resuelve cada subproblema individualmente, de forma recursiva hasta llegar a problemas de solución trivial (conquistar).
3. Combina la solución de los subproblemas (*nivel superior*) en una solución de todo el problema original.

Algoritmo:

```
DAC(a, i, j)
{
    if(small(a, i, j))
      return(Solution(a, i, j))
    else
      m = divide(a, i, j)
      b = DAC(a, i, mid)
      c = DAC(a, mid+1, j)
      d = combine(b, c)
   return(d)
}
```

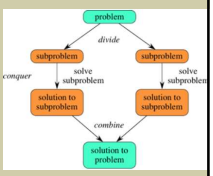
Algunas aplicaciones:

1. Mínimo y máximo
2. Búsqueda binaria O(log(n))
3. Ordenamiento por mezcla (Merge Sort) O(nlog(n))
4. Ordenamiento rápido (Quick Sort) O(nlog(n))
5. Multiplicación de matrices (Algoritmo de Strassen) $O(n^{2.8974})$
6. Algoritmo de multiplicación (Karatsuba) $n^{1.585}$
7. Transformada rápida de Fourier FFT (Cooley–Tukey) O(nlog(n))
8. Encontrar puntos máximos en un espacio bidimensiona
9. Par de puntos más cercanos (Closest pair of points) O(nlog(n))
10. Polígonos convexos (convex hull)
11. Diagramas de Voronoi
12. Torres de Hanoi

9

Técnicas de diseño de algoritmos

Divide y vencerás (*Divide and conquer*)



1. Divide el problema original en un conjunto de subproblemas más pequeños (dividir), y del mismo tipo del problema original.
2. Resuelve cada subproblema individualmente, de forma recursiva hasta llegar a problemas de solución trivial (conquistar).
3. Combina la solución de los subproblemas (*nivel superior*) en una solución de todo el problema original.

Ejercicio:

- Exponenciación.
  Dados dos números, *x* y *n*, calcular el resultado de x^n, haciendo uso de la técnica de divide y vencerás

Procedure 2 FAST_POW

Input: x : *Real*, n : *Integer*
  if $n < 0$ **then**
    return $FAST\_POW(1/x, -n)$
  else if $n == 0$ **then**
    return 1
  else if $n == 1$ **then**
    return x
  else if n mód $2 = 0$ **then**
    return $FAST\_POW(x * x, n/2)$
  else if n mód $2 = 1$ **then**
    return $x * FAST\_POW(x * x, (n-1)/2)$
  end if

```
#include <iostream>

double fastPow(double x, int n) {
  if (n < 0) {
    return fastPow(1/x, -n);
  } else if (n == 0) {
      return 1;
      } else if (n == 1) {
          return x;
          } else if (n % 2 == 0) {
              return fastPow(x * x, n / 2);
              } else {
                  return x * fastPow(x * x, (n - 1) / 2);
                  }
}
int main(int argc, char* argv[]) {
  double x;
  int n;
  std::cin >> x >> n;
  std::cout << x << "^" << n << " = " << fastPow(x,n) << "\n";
}
```


10