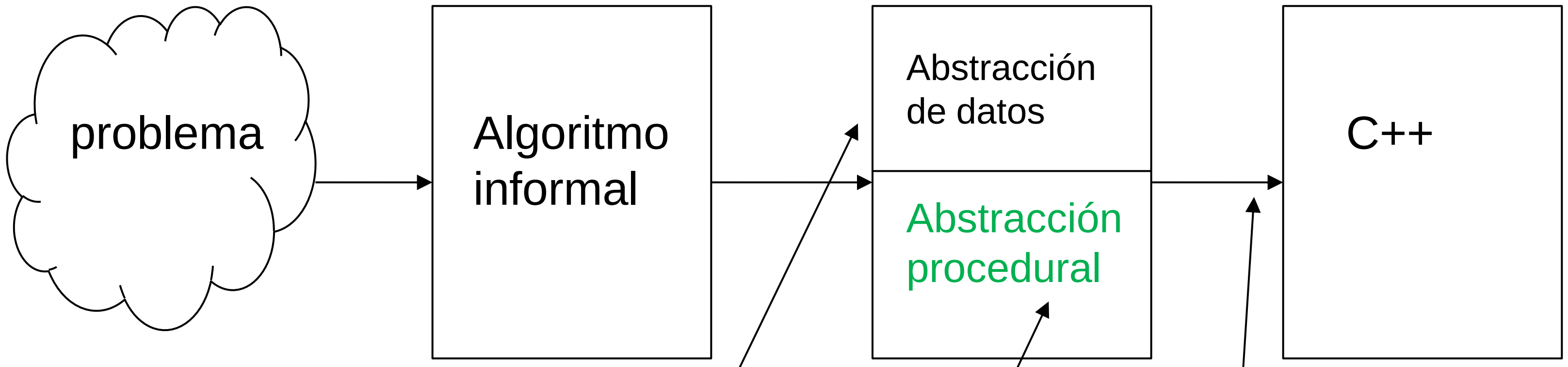
Técnicas de diseño de algoritmos

Liliana Favre

Análisis y diseño de algoritmos 1- 2025

Metodología de construcción de soluciones algorítmicas



Divide y Conquista

Greedy

Programación Dinámica

✓ **TDA (Tipos de datos Abstractos)**

Técnicas de diseño de algoritmos

✓ **Análisis de eficiencia**

Técnicas de diseño de algoritmos
Divide y Conquista

Divide y Conquista

Las bases de la técnica de diseño de algoritmos "**Divide y Conquista**" pueden resumirse en estos tres pasos:

1. Si la instancia del problema a resolver es simple, se encuentra la solución mediante un método directo

2. Si la instancia del problema no es simple, se divide en partes $x_1, x_{2,} x_3, \ldots x_k$ y se resuelven independientemente y recursivamente el problema para cada una de las partes

3. Las soluciones obtenidas para cada parte se combinan para resolver el problema original

Divide y Conquista
Esquema algorítmico D&C

tipoResultado **DivideyConquista** (tipoDato X)
{ tipoDato x_1, x_2,…, x_k;
if (simple(x)) **return** soluciónDirecta(x);
else { x_1 = $Parte_1$(x);
x_2= $Parte_2$ (x);
…
x_k = $Parte_k$ (x);
return Combinar (**DivideyConquista** (x_1), **DivideyConquista** (x_2),…
DivideyConquista (x_k));
}

Si k=1 se denomina esquema de reducción

Divide y Conquista. Consideraciones

La solución de un problema se obtiene combinando la solución de subproblemas idénticos al problema original

- No se resuelve el mismo problema más de una vez
- El tamaño de las entradas de las partes es una fracción del tamaño de la entrada del problema original

Para lograr algoritmos eficientes

- Convienen problemas balanceados
- Las funciones $Parte_1$, $Parte_2$,…. y *Combina* deben ser eficientes!

Divide y Conquista.

Ya hemos diseñado algoritmos basados en estas consideraciones para resolver algunos problemas:

- Problema de las torres de Hanoi (k=2, 2 subproblemas)

- Búsqueda Binaria (k=1, un subproblema, “reducción”)

Divide y Conquista.

Torres de Hanoi
K=2
$T(n) \in O(2^n)$

$$
\text{Hanoi}(n, A, B, C) \begin{cases} AB & n=1 \\ & n > 1 \\ \text{Hanoi } (n-1, A, C, B) : AB : \\ \text{Hanoi}(n-1, C, B, A) \end{cases}
$$

$$
T(n) \begin{cases} c1 & n=1 \\ 2\,T(n-1) + c2 & n>1 \end{cases}
$$

Técnicas de diseño de algoritmos
Divide y conquista

```
 void buscar (const int a[], int primero, int ultimo, int clave, bool & pertenece, int& posicion)
{  int mitad;
    if (primero > ultimo) {pertenece = false;
                                 posición = -1;
                                }
       else {mitad = (primero + ultimo) /2;
                if (clave == a[mitad])
                     { pertenece= true;
                      posicion = mitad;
                      }
                    else { if (clave < a[mitad])
                                buscar ( a, primero, mitad -1, clave, pertenece, posicion);
                                 else
                                  buscar ( a, mitad+1, ultimo, clave, pertenece, posicion);
                          }
              }
}
```

Búsqueda Binaria
K=1 reducción
T(n)∊ $O(\log_2 m)$

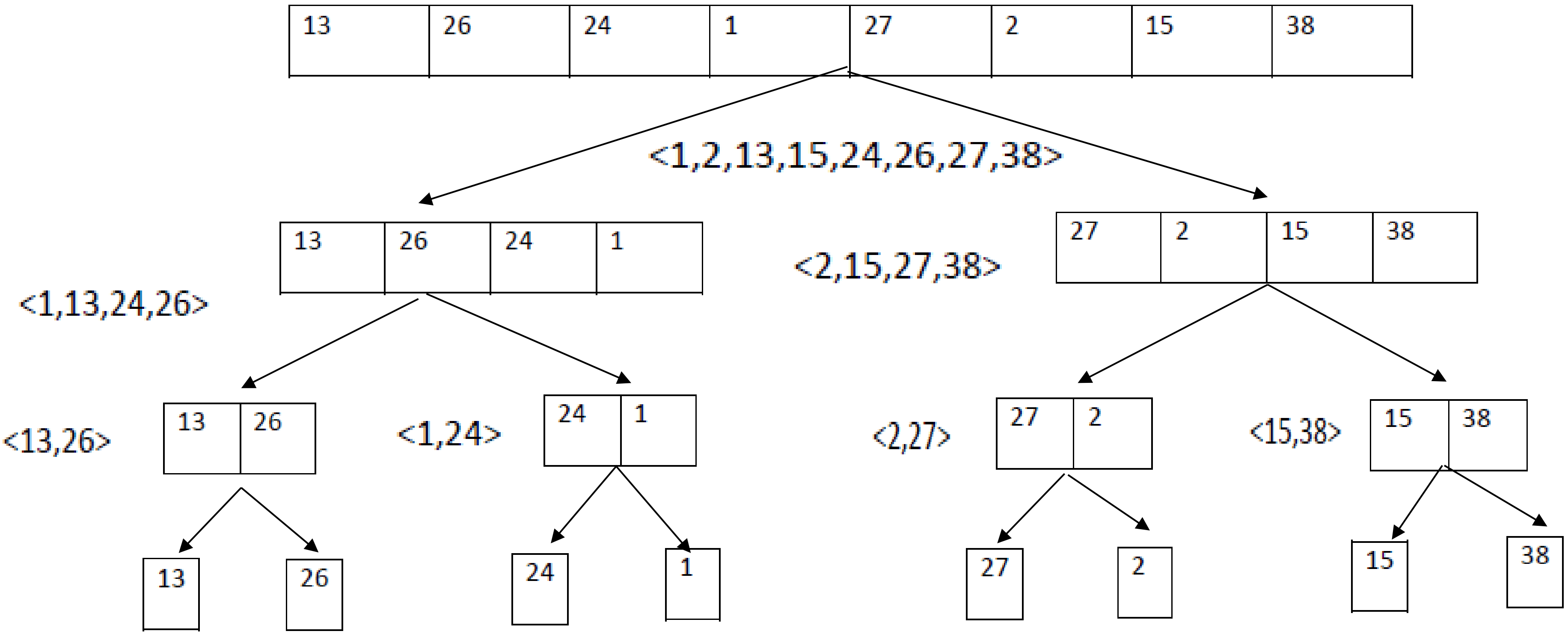
m tamaño del problema

Divide y conquista
Ordenamiento Mergesort

Dada una lista desordenada
- Si la longitud de la lista es 1, entonces ya está ordenada.
- Si la longitud de la lista es mayor que 1, dividir la lista desordenada en dos listas de aproximadamente la mitad del tamaño que contengan a la “primera mitad” de los elementos y la “segunda mitad” respectivamente.
- Ordenar cada “mitad” recursivamente aplicando el ordenamiento **mergesort**
- Intercalar las dos “mitades” ordenadas en una sola lista ordenada.

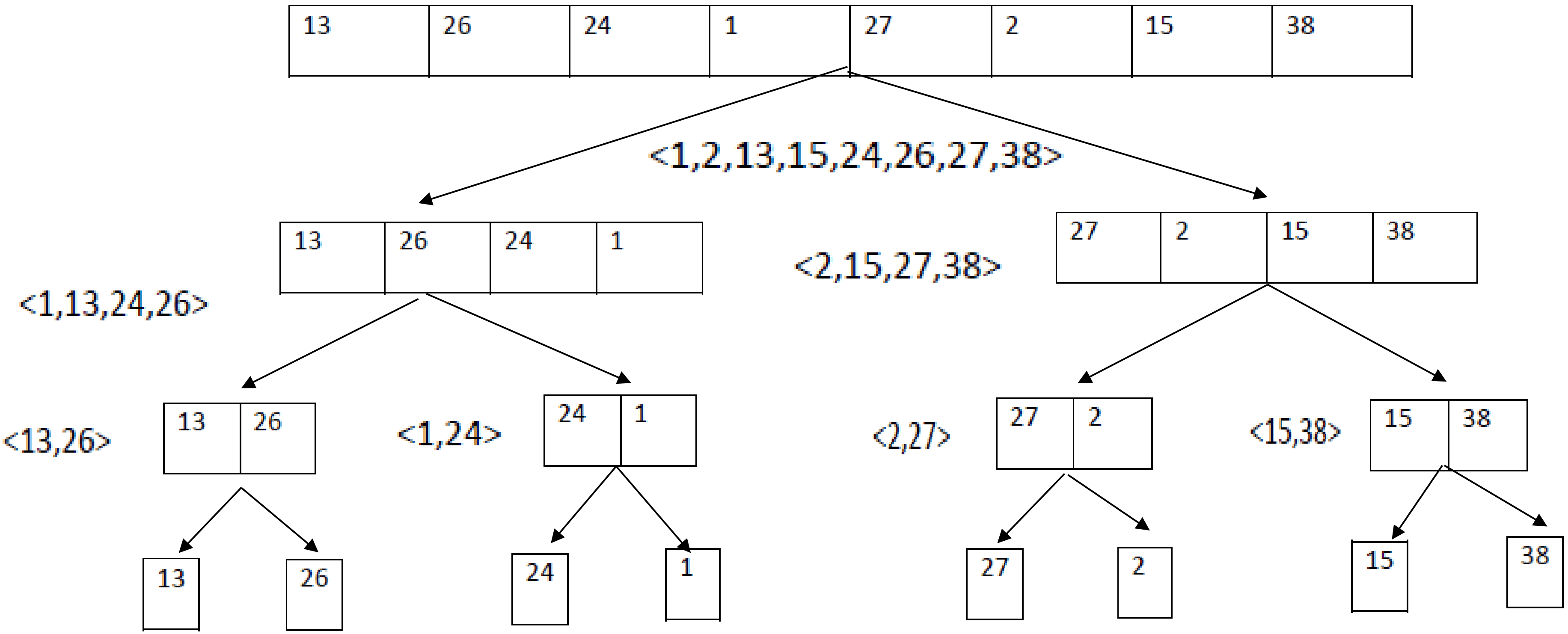
Ideas principales
- Una lista pequeña necesitará menos pasos para ordenarse que una lista grande.
- Se necesitan menos pasos para construir una lista ordenada a partir de dos listas también ordenadas, que a partir de dos listas desordenadas.

Divide y Conquista
Ordenamiento Mergesort



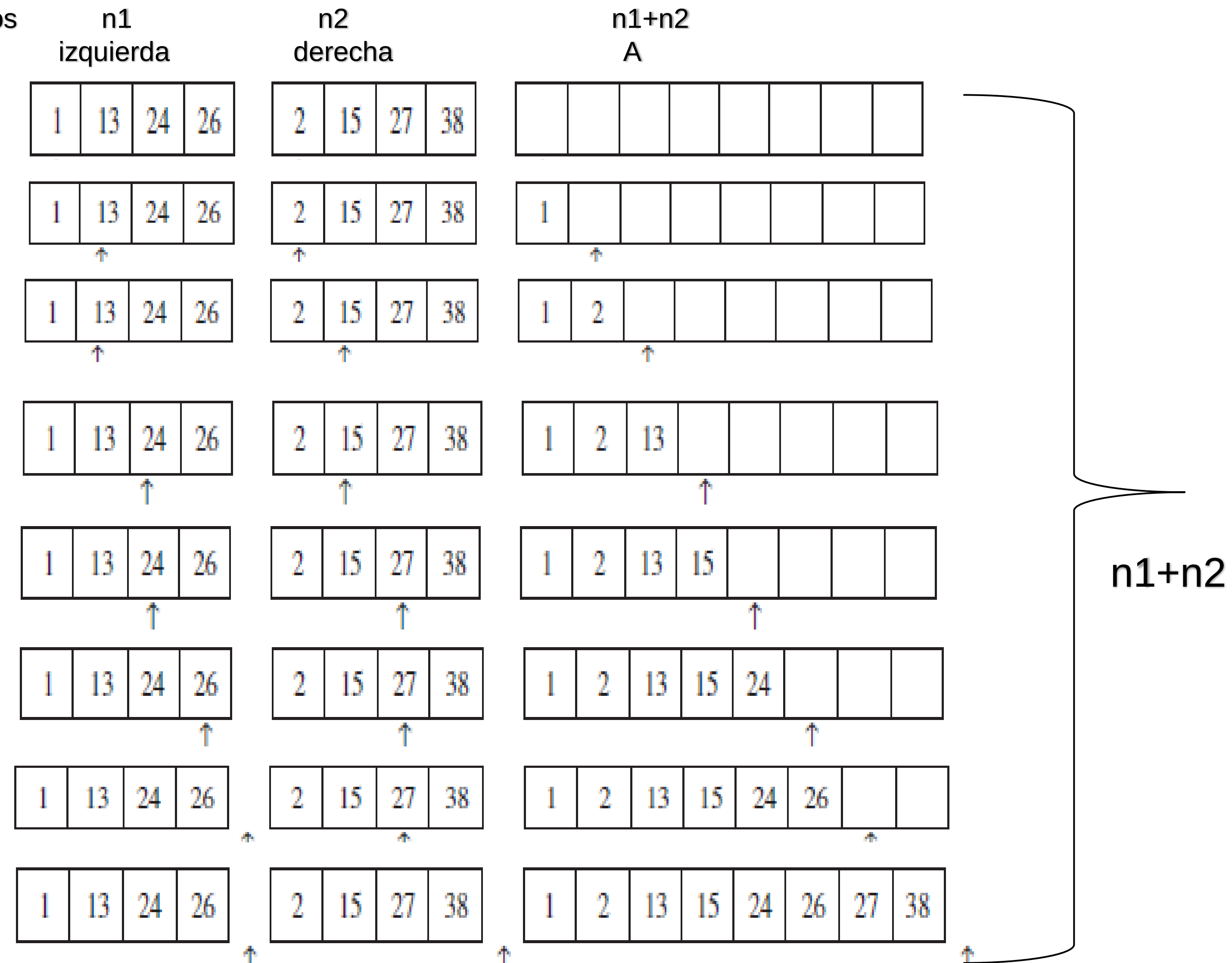
ORDENAMIENTO MERGESORT

Divide y Conquista
Ordenamiento Mergesort



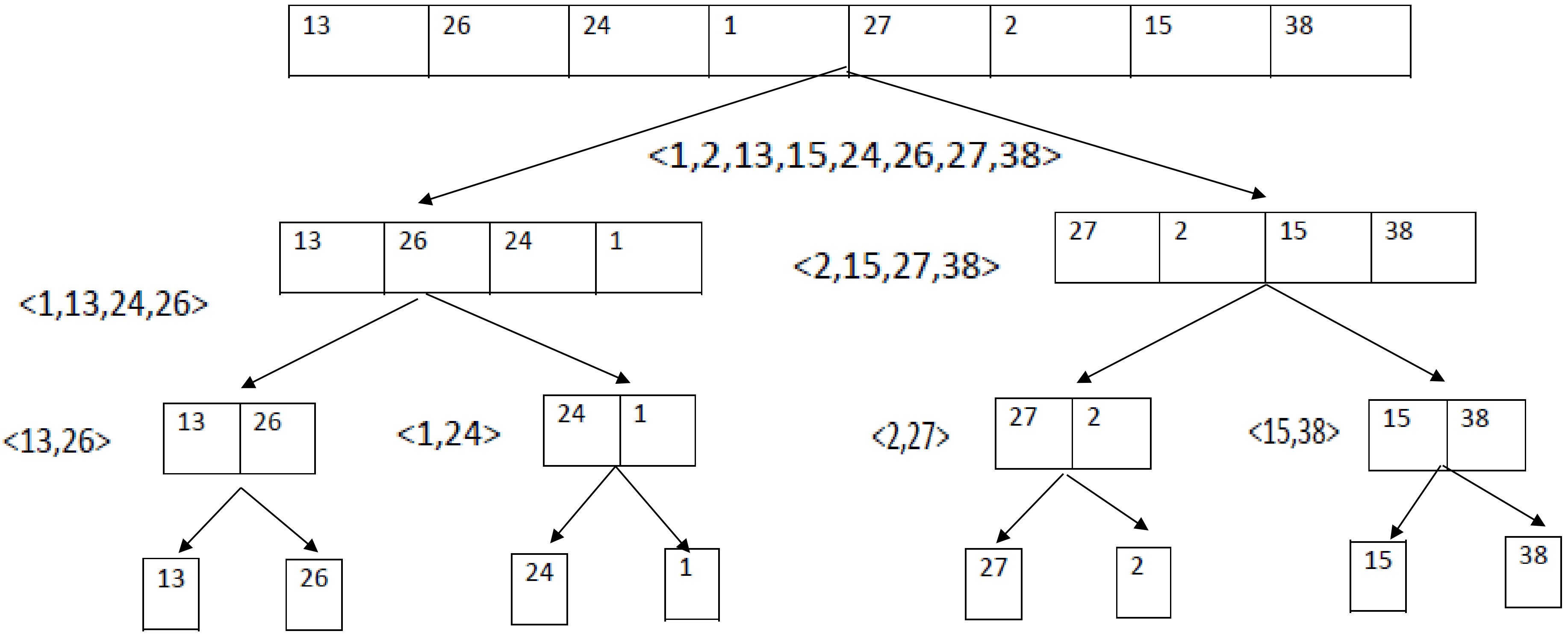
cantidad de elementos

Merge
Intercala dos secuencias ordenadas



m= n1 + n2
O(m)

¿Cómo acotar el tiempo de ejecución de una función recursiva?



ORDENAMIENTO
MERGESORT

¿Cómo acotar el tiempo de ejecución de una función recursiva?


```
void mergesort ( int a[ ], unsigned int inicio, unsigned int fin)
{    if (inicio < fin)
     {
         unsigned int mitad = (inicio+fin) /2;
         mergesort (a, inicio, mitad);
         mergesort (a, mitad + 1, fin);
         merge (a, inicio, mitad, fin);
     }
}
```

```
#include <iostream>
using namespace std;
const int MAX=16;
void merge(int a[ ] , int inicio, int mitad, int fin) {
   int n1 = mitad - inicio + 1;
   int n2 = fin - mitad;
  int izquierda[MAX];
 int derecha[MAX];
   // Copiar datos a los subarreglos temporales izquierda y derecha
   for (int i = 0; i < n1; i++) {
      izquierda[i] = A[inicio + i]; }
   for (int j = 0; j < n2; j++) {
      derecha[j] = A[mitad + 1 + j]; }
```

```
// Mezclar los subarreglos de vuelta en A
    int i = 0, j = 0, k = inicio;
   while (i < n1 && j < n2) {
      if (izquierda[i] <= derecha[j]) {
         A[k] = izquierda[i];
         i++;}
       else  {A[k] = derecha[j];
         j++;}
      k++;   }
```

```
 // Copiar los elementos restantes de izquierda[]
  while (i < n1) {
     A[k] = izquierda[i];
     i++;
     k++;
  }
  // Copiar los elementos restantes de derecha[]
  while (j < n2) {
     A[k] = derecha[j];
     j++;
     k++;
  }
}
```

```
void mergesort (int A[ ], unsigned int inicio, unsigned int fin)
{ if (inicio < fin)
   { unsigned int mitad= (inicio + fin) /2;
      mergesort(A, inicio, mitad);
      mergesort (A, mitad +1, fin);
      merge (A,inicio, mitad, fin);
                  }
}
```

```
int main() {
   int  A [MAX ]  = {13,26,24,1,27,2,15,38};
    int inicio = 0, fin = 7;
     mergesort(A, inicio, fin);
      for (int i=inicio; i<= fin; i++) {
        cout << A[i] << " "<<endl;
    }
   return 0;
}
```

¿Cómo acotar el tiempo de ejecución de una función recursiva?

13	26	24	1	27	2	15	38

inicio mitad fin

```
void mergesort (int a[ ], unsigned int inicio, unsigned int fin)
{    if (inicio < fin)
     {
        unsigned int mitad = (inicio+fin) /2;
        mergesort (a, inicio, mitad);
        mergesort (a, mitad + 1, fin);
        merge (a, inicio, mitad, fin);
     }
}
```

merge intercala dos sub-arreglos de $\frac{n}{2}$

Sea n el tamaño del problema (la cantidad de elementos a ordenar). Teniendo en cuenta que

$\mathbf{2^k < n \leq 2^{k+1}}$

$T(n) \leq T(2^{k+1})$. Podemos suponer que n es potencia de 2

$$T(2^k) \begin{cases} c3 & n=1 \\ 2\,T(\frac{n}{2}) + c1\,n + c2 & n>1 \end{cases}$$

mergesort **merge**

¿Cómo acotar el tiempo de ejecución de una función recursiva?

13	**26**	**24**	**1**	**27**	**2**	**15**	**38**

inicio mitad fin

```
void mergesort ( int a[ ] , unsigned int inicio, unsigned int fin)
{    if (inicio < fin)
     {
         unsigned int mitad = (inicio+fin) /2;
         mergesort (a, inicio, mitad);
         mergesort (a, mitad + 1, fin);
         merge (a, inicio, mitad, fin);
     }
}
```

merge intercala dos sub-arreglos de $\frac{n}{2}$

Sea n el tamaño del problema (la cantidad de elementos a ordenar). Teniendo en cuenta que

$\mathbf{2^k < n \leq 2^{k+1}}$

$T(n) \leq T(2^{k+1})$. Podemos suponer que n es potencia de 2

$$T(2^k) \begin{cases} c3 & n=1 \\ 2\,T(\frac{n}{2}) + c1\,n + c2 & n>1 \end{cases}$$

mergesort **merge**

¿Cómo acotar el tiempo de ejecución de una función recursiva?

13	26	24	1	27	2	15	38

inicio mitad fin

```
void mergesort (int a[ ], unsigned int inicio, unsigned int fin)
{    if (inicio < fin)
     {
         unsigned int mitad = (inicio + fin) /2;
         mergesort (a, inicio, mitad);
         mergesort (a, mitad + 1, fin);
         merge ( a, inicio, mitad, fin);
     }
}
```

$$T(n)\begin{cases} c3 & n=1 \\ 2\,T(\frac{n}{2}) + c1\,n + c2 & n>1 \end{cases}$$

otra definición

$$T(2^k)\begin{cases} c3 & k=0 \\ 2\,T(2^{k-1}) + c1\,2^k + c2 & k>0 \end{cases}$$

¿Cómo acotar el tiempo de ejecución de una función recursiva?

$$T(2^k) = \begin{cases} c3 & k=0 \\ 2\,T(2^{k-1}) + c1\,2^k + c2 & k>0 \end{cases}$$

$T(n) \in O(n \log n)$

$T(2^k) = 2\,T(2^{k-1}) + c\,1\,2^k + c2$

$T(2^k) = 2\,(2\,T(2^{k-2}) + c1\,2^{k-1} + c2) + c1\,2^k + c2$

$T(2^k) = 2^2\,T(2^{k-2}) + c1\,2^k + 2\,c2 + c1\,2^k + c2$

$T(2^k) = 2^2(2\,T(2^{k-3}) + c1\,2^{k-2} + c2) + c1\,2^k + 2\,c2 + c1\,2^k + c2$

$T(2^k) = 2^3\,T(2^{k-3}) + c1\,2^k + 2^2\,c2 + c1\,2^k + 2\,c2 + c1\,2^k + c2$

i-ésimo reemplazo

$T(2^k) = 2^i\,T(2^{k-i}) + i\,2^k\,c1 + c2\sum_{j=0}^{i-1} 2^j$

i=k

$n = 2^K$

$\log_2 n = k \log_2 2$

$T(2^k) = 2^k\,T(1) + k\,2^k\,c1 + \sum_{j=0}^{k-1} 2^j = 2^k\,c3 + k\,2^k\,c1 + c2(2^k - 1)$

$T(n) = n\,c3 + n \log n\,c1 + (n-1)\,c2$

Divide y Conquista.

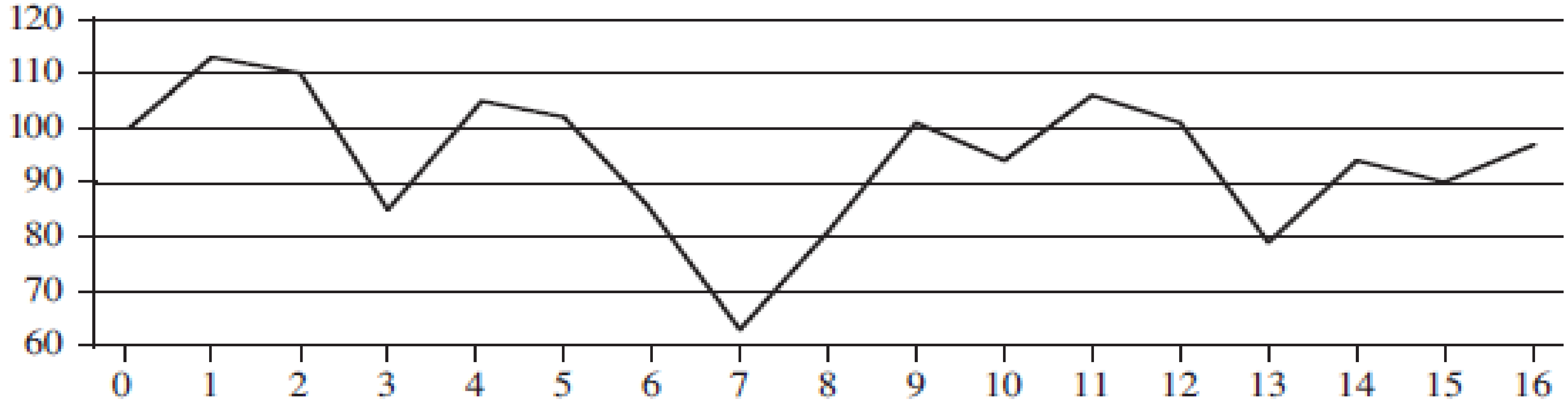
Un problema…

Supongamos que queremos invertir en acciones de una empresa.

Podemos comprar en un determinado día y vender en otro y se conocen las previsiones de los precios de los activos

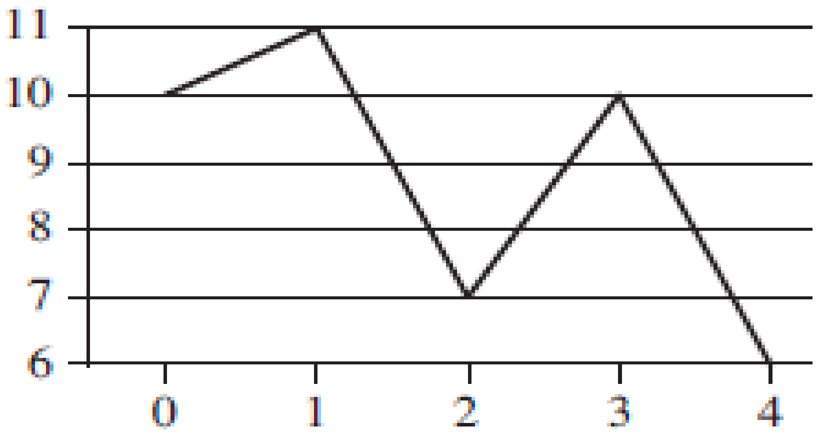
Divide y Conquista.

Día	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Precio	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
diferencia		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7



Divide y Conquista.

Analicemos un caso simple…



	0	1	2	3	4
	10	11	7	10	6
		1	−4	3	−4

Vendo el día

Compro el día

	0	1	2	3	4
0	-	1	-3	0	-4
1	-	-	-4	-1	-5
2	-	-		3	-1
3	-	-	-	-	-4
4	-	-	-	-	-

Conviene comprar el día 2 en 7$ y vender el 3 en 10$

Divide y Conquista.

Comprar el día 7 a 63$ y vender el 11 a $106, el beneficio es 43

Encontrar el sub-arreglo de suma máxima

A[8]..A[11] y la suma de los elementos es (18 + 20 -7 +12)

Día	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Precio	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
diferencia		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Divide y Conquista.

Nuestro problema:

Encontrar el sub-arreglo de suma máxima

Día	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Precio	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
diferencia		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Divide y Conquista.

Problema:
Encontrar el sub-arreglo de un arreglo de enteros de suma máxima

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Divide y Conquista.

Un algoritmo "ingenuo" podría generar todos los pares de compra y venta. En n días hay $\binom{n}{2}$ pares

$$\Downarrow$$

$O(n^2)$

Es posible mejorarlo?

Divide y Conquista
Esquema algorítmico

tipoResultado **DivideyConquista** (tipoDato X)

{ tipoDato x_1, x_2;

  if (simple(x)) **return** solución (x);

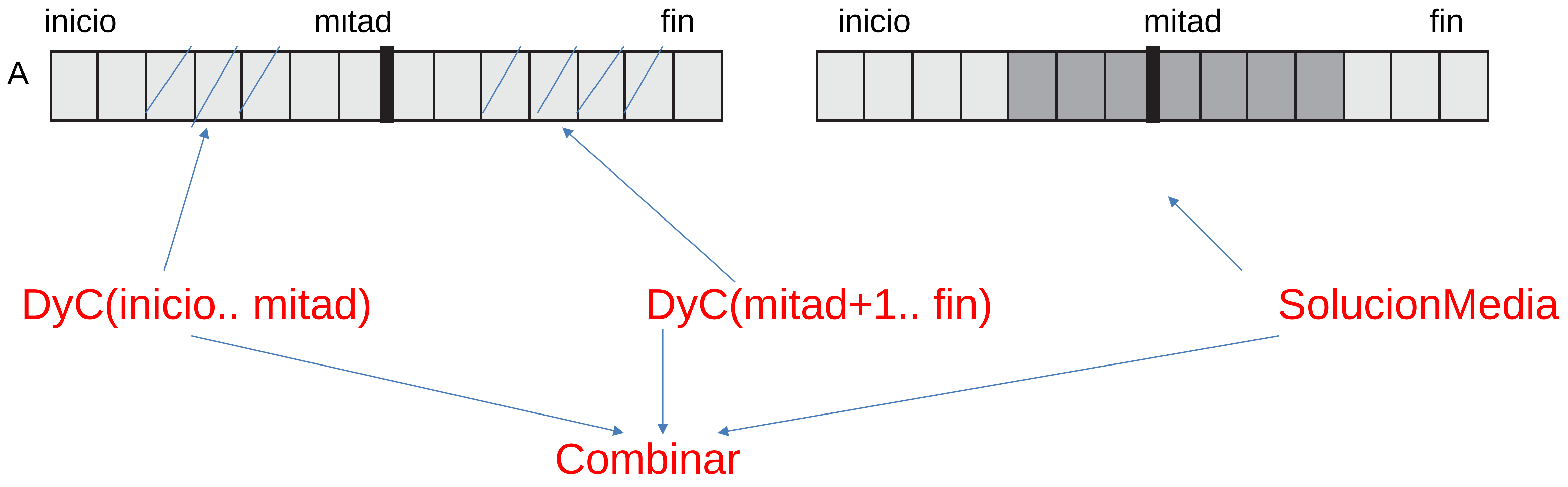
  else { x_1 = Parte$_1$(x);

             x_2= Parte$_2$ (x);

return Combinar (**DivideyConquista** (x_1), **DivideyConquista** (x_2));

}

Divide y Conquista.



Divide y Conquista.

Inicio mitad fin

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

DyC(inicio..mitad) 20

Solucion_Media 43

DyC (mitad+1..fin) 25

Combinar

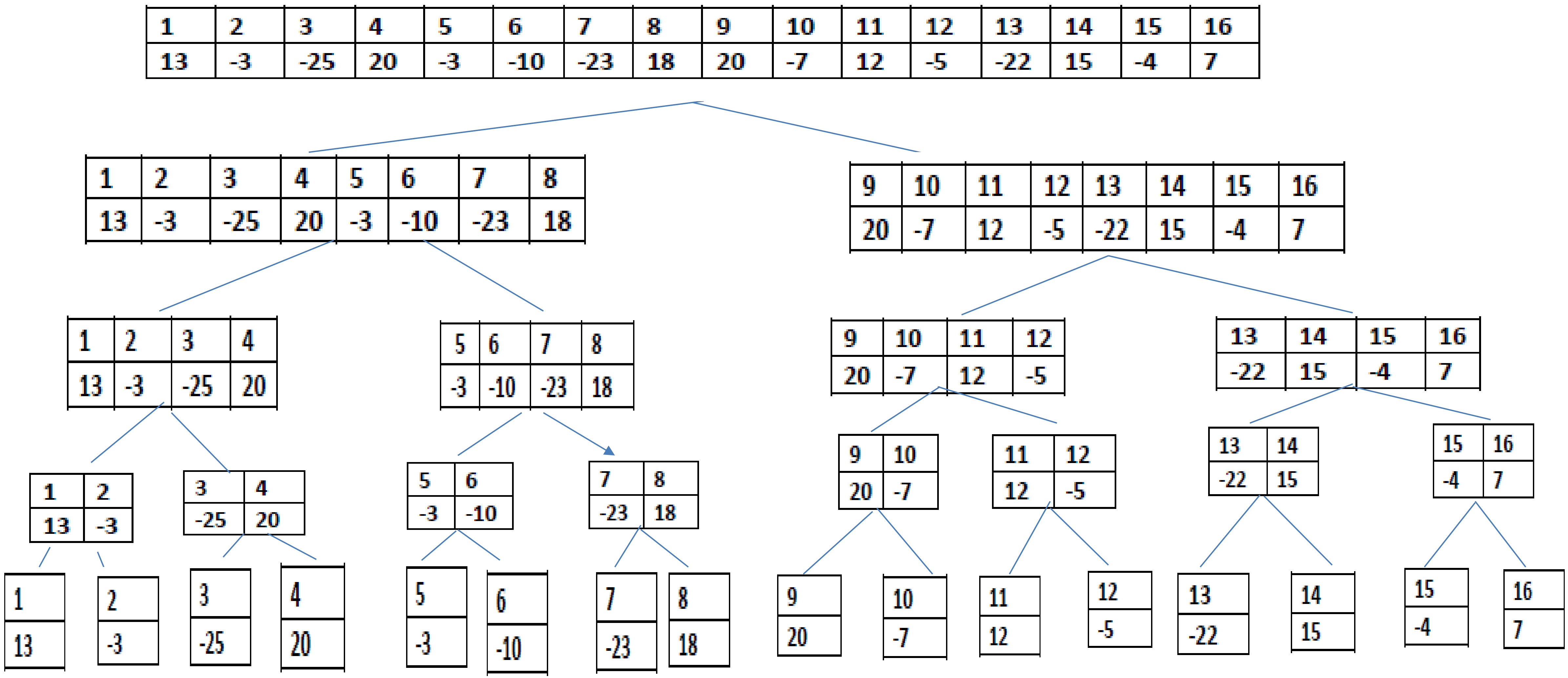
max (20, 25, 43)

Divide y Conquista.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

	sumaIzq	maxIzq
18	18	8
-23	-5	8
-10	-15	8
-3	-18	8
20	2	8
-25	-23	8
-3	-26	8
13	-13	8

	sumaDer	maxDer
20	20	9
-7	13	9
12	25	11
-5	20	11
-22	-2	11
15	13	11
-4	9	11
7	16	11


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13 14
-22 15
15 16
-4 7
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5
13
-22
14
15
15
-4
16
7

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

mitad

1	2	3	4	5	6	7	8
13	-3	-25	20	-3	-10	-23	18

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

mitad

1	2	3	4	5	6	7	8
13	-3	-25	20	-3	-10	-23	18

mitad

1	2	3	4
13	-3	-25	20

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

1	2	3	4	5	6	7	8
13	-3	-25	20	-3	-10	-23	18

1	2	3	4
13	-3	-25	20

1	2
13	-3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

1	2	3	4	5	6	7	8
13	-3	-25	20	-3	-10	-23	18

1	2	3	4
13	-3	-25	20

1	2
13	-3

1..1 13

1
13

i..j desde el i-ésimo elemento hasta el j-ésimo elemento

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

1	2	3	4	5	6	7	8
13	-3	-25	20	-3	-10	-23	18

1	2	3	4
13	-3	-25	20

1	2
13	-3

1..1 13

1
13

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

1	2	3	4	5	6	7	8
13	-3	-25	20	-3	-10	-23	18

1	2	3	4
13	-3	-25	20

1	2
13	-3

1..1 13

2..2 -3

1
13

2
-3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

1	2	3	4	5	6	7	8
13	-3	-25	20	-3	-10	-23	18

1	2	3	4
13	-3	-25	20

1..1 13 =(max(13,-3, 13-3)

1	2
13	-3

1..1 13

2..2 -3

1
13

2
-3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

1	2	3	4	5	6	7	8
13	-3	-25	20	-3	-10	-23	18

1	2	3	4
13	-3	-25	20

1..1 13 =(max(13,-3, 13-3)

1	2
13	-3

1..1 13

2.2 -3

1
13

2
-3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

1	2	3	4	5	6	7	8
13	-3	-25	20	-3	-10	-23	18

1	2	3	4
13	-3	-25	20

1..1 13

1	2
13	-3

3	4
-25	20

1..1 13

2..2 -3

1
13

2
-3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

1	2	3	4	5	6	7	8
13	-3	-25	20	-3	-10	-23	18

1	2	3	4
13	-3	-25	20

1..1 13

1	2
13	-3

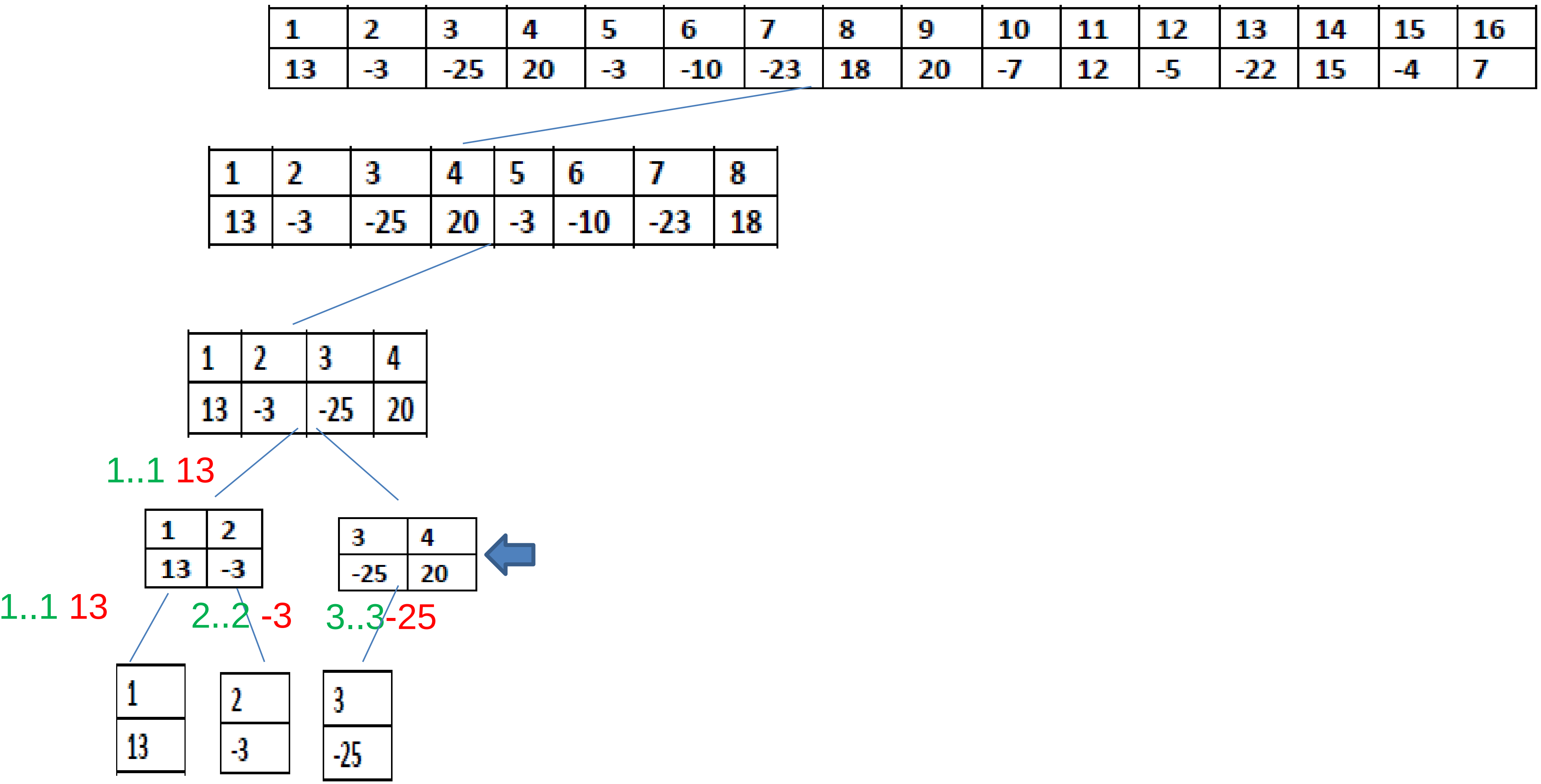
3	4
-25	20

1..1 13 2..2 -3 3..3 -25

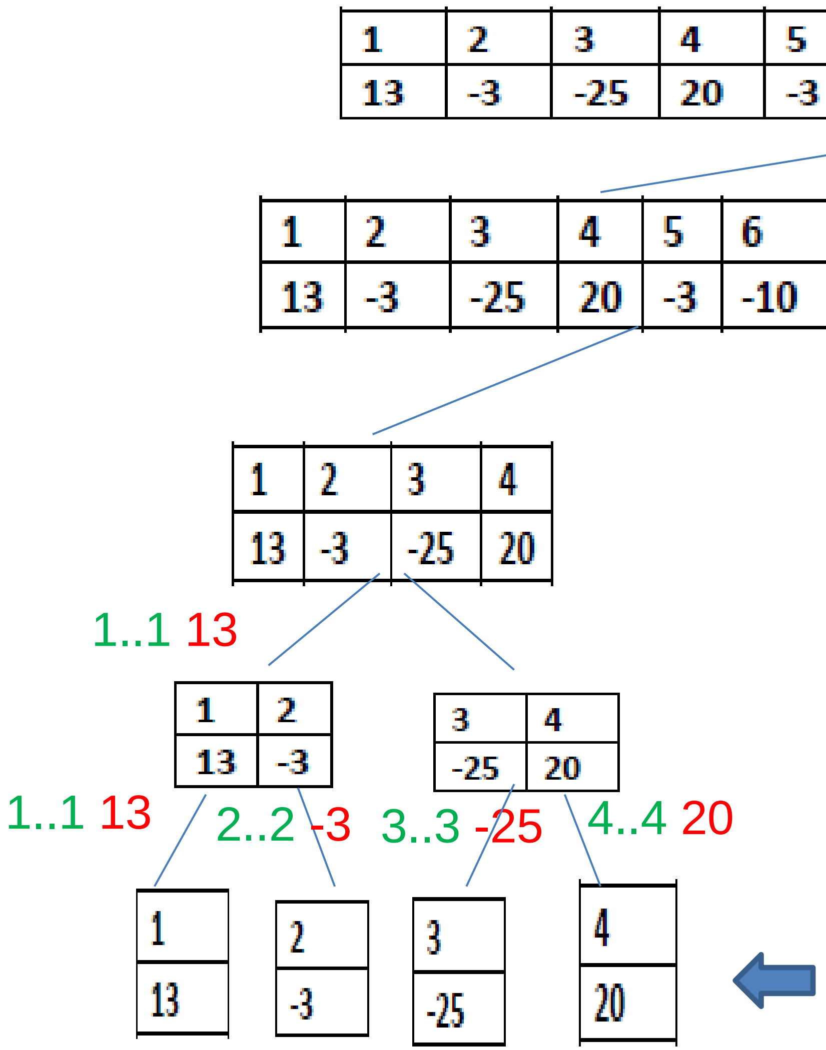
1
13

2
-3

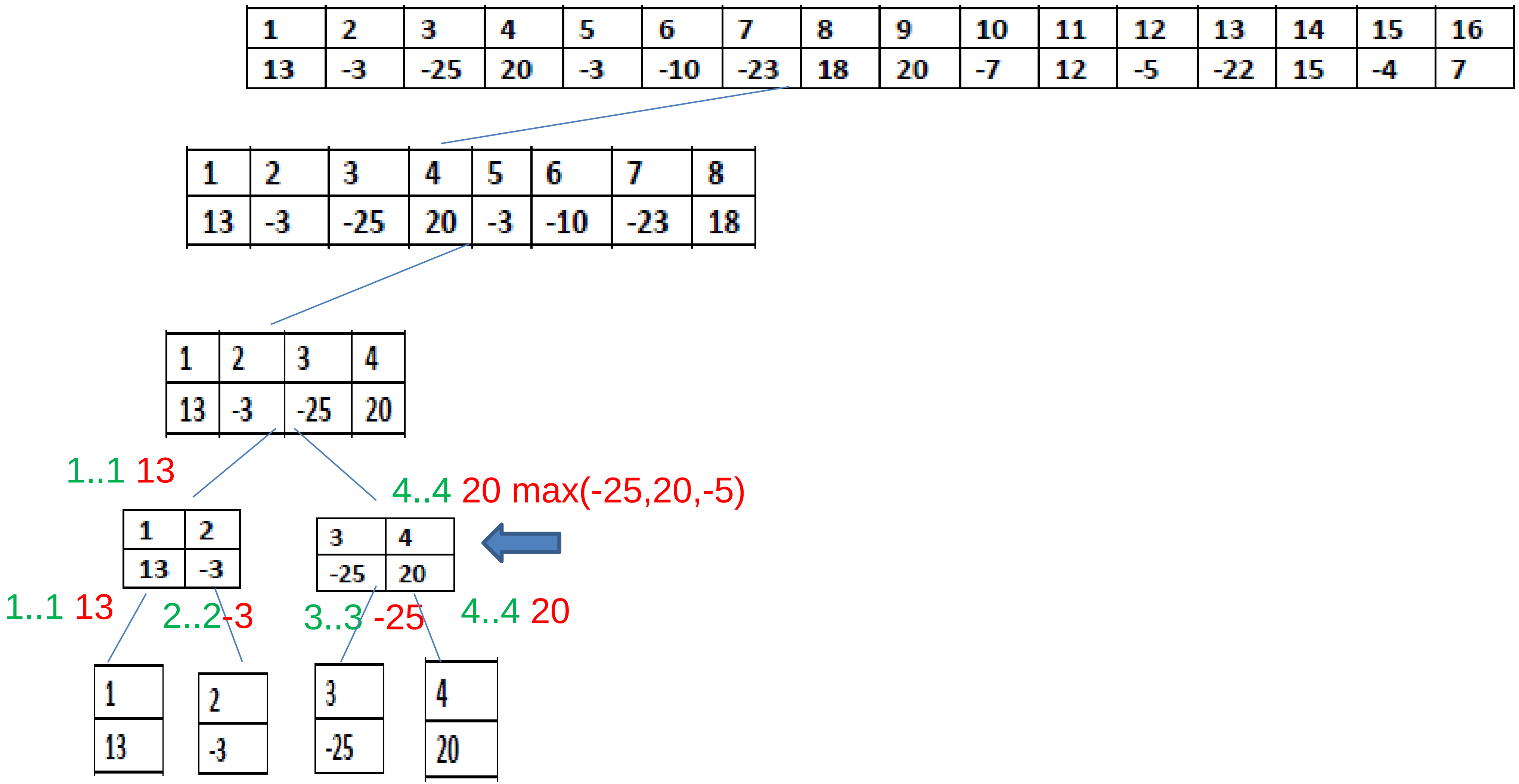
3
-25



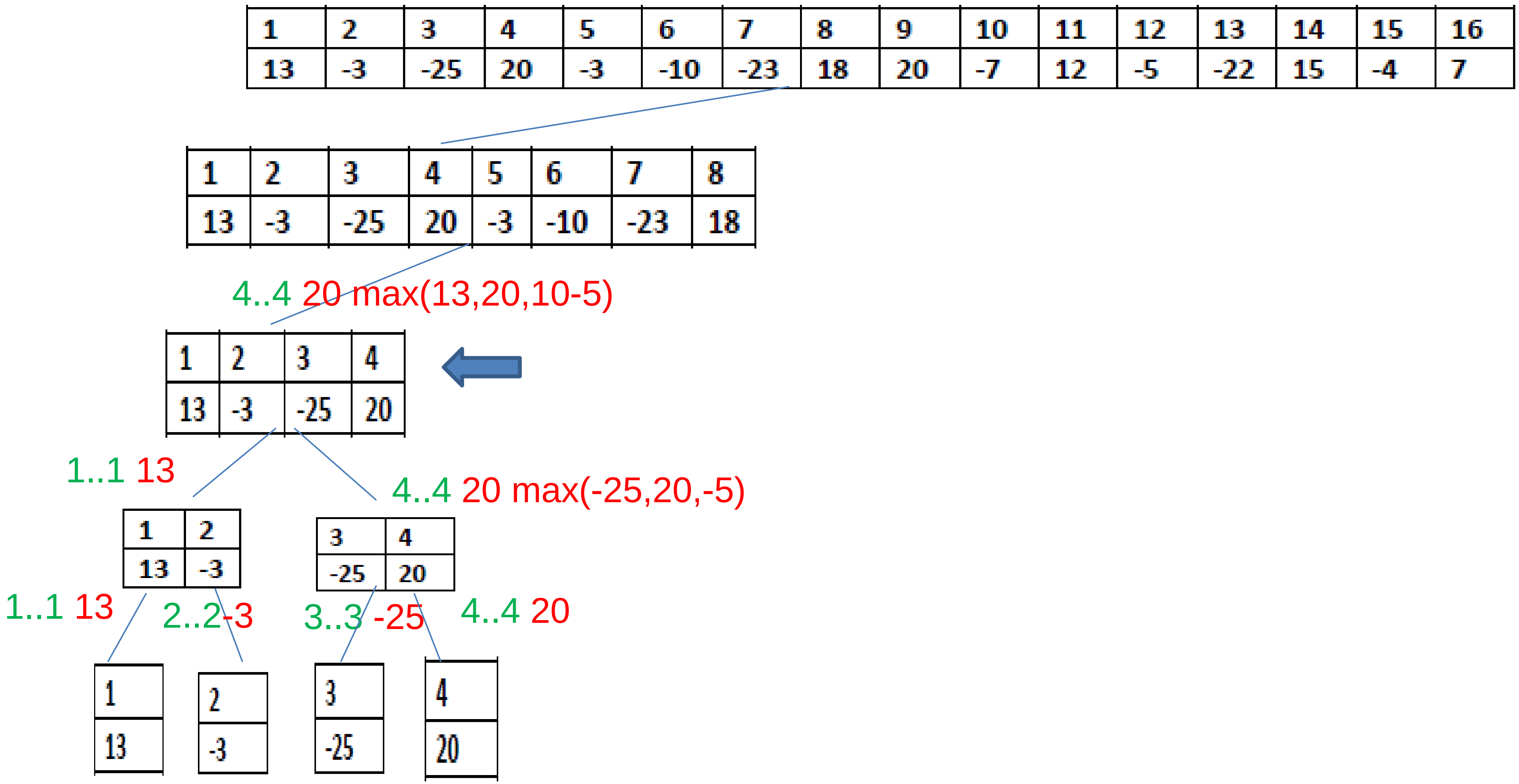
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
1 2 3 4
13 -3 -25 20
1..1 13
1 2
13 -3
3 4
-25 20
1..1 13
2..2 -3
3..3-25
1
13
2
-3
3
-25

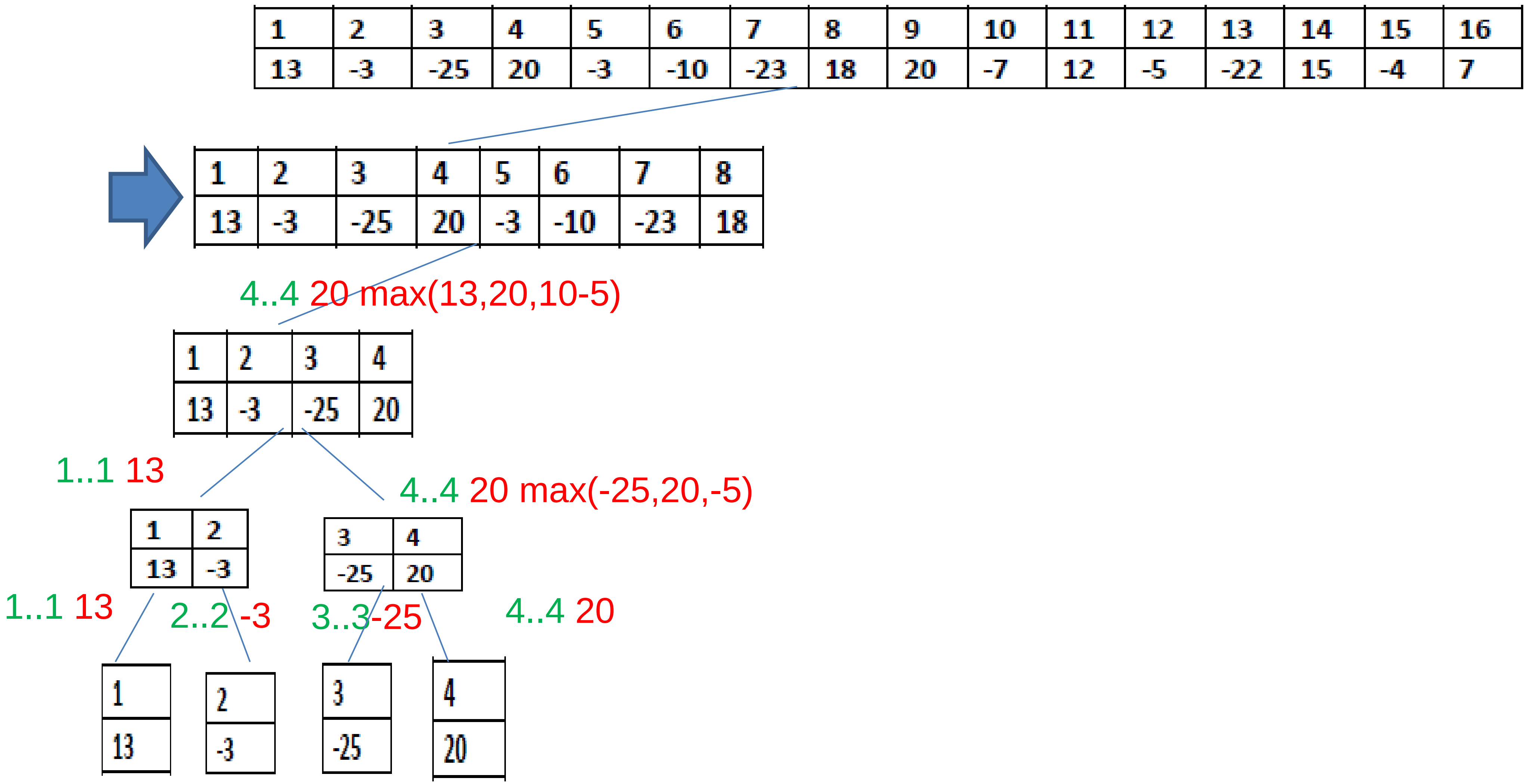


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
1 2 3 4
13 -3 -25 20
1..1 13
1 2
13 -3
3 4
-25 20
1..1 13
2..2 -3
3..3 -25
4..4 20
1
13
2
-3
3
-25
4
20

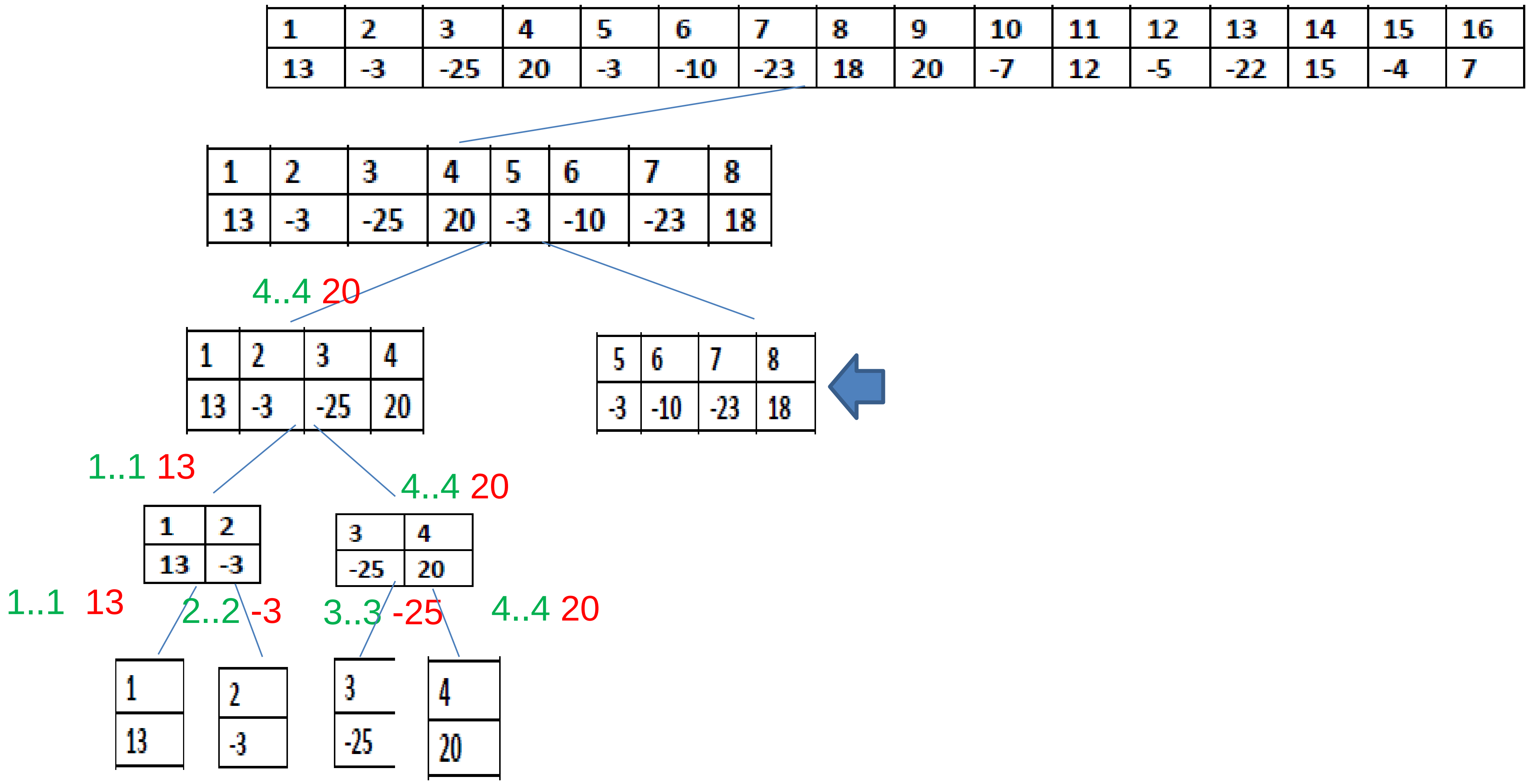


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
1 2 3 4
13 -3 -25 20
1..1 13
4..4 20 max(-25,20,-5)
1 2
13 -3
3 4
-25 20
1..1 13
2..2-3
3..3 -25
4..4 20
1
13
2
-3
3
-25
4
20

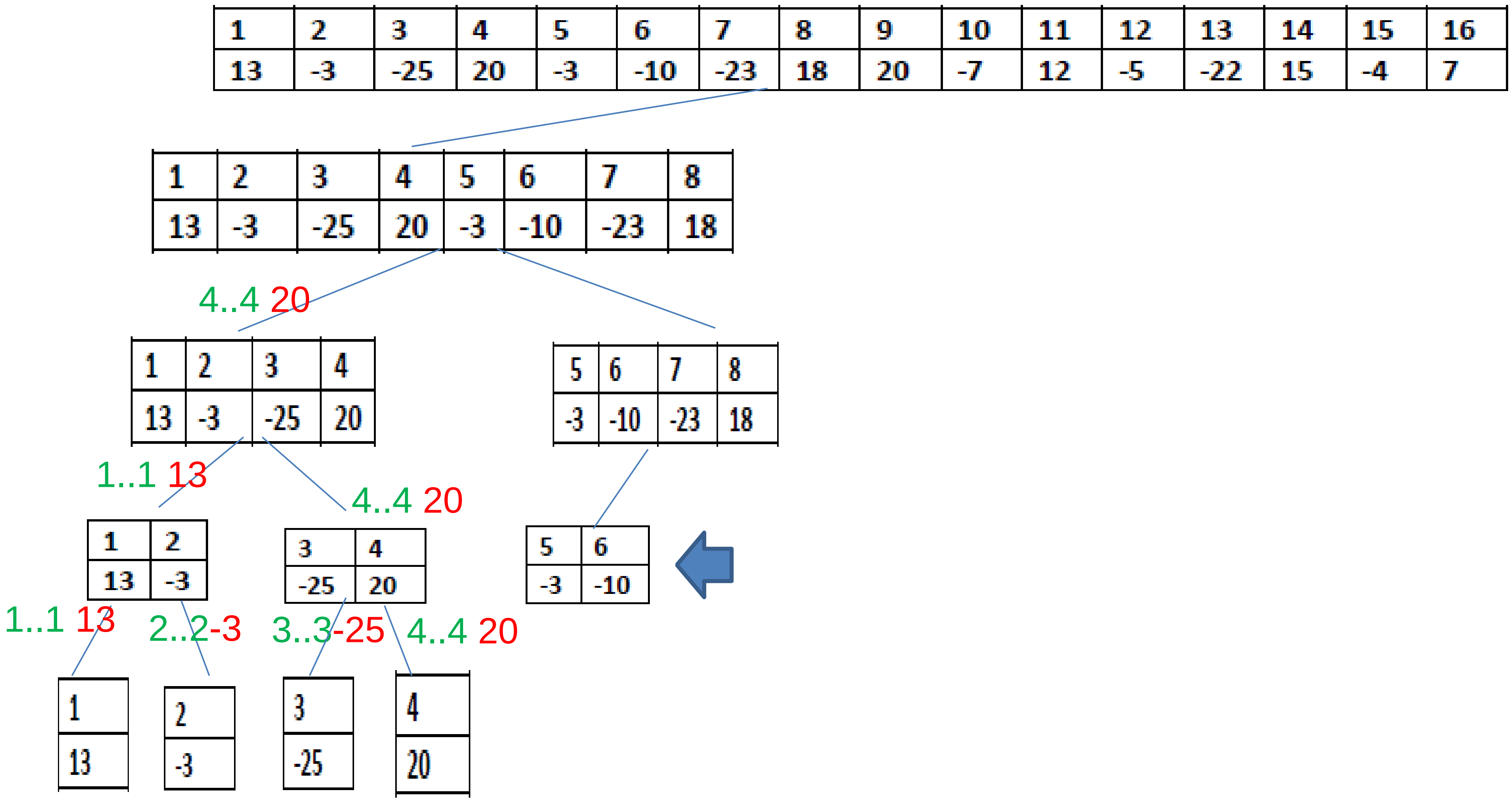

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20 max(13,20,10-5)
1 2 3 4
13 -3 -25 20
1..1 13
4..4 20 max(-25,20,-5)
1 2
13 -3
3 4
-25 20
1..1 13
2..2-3
3..3 -25
4..4 20
1
13
2
-3
3
-25
4
20



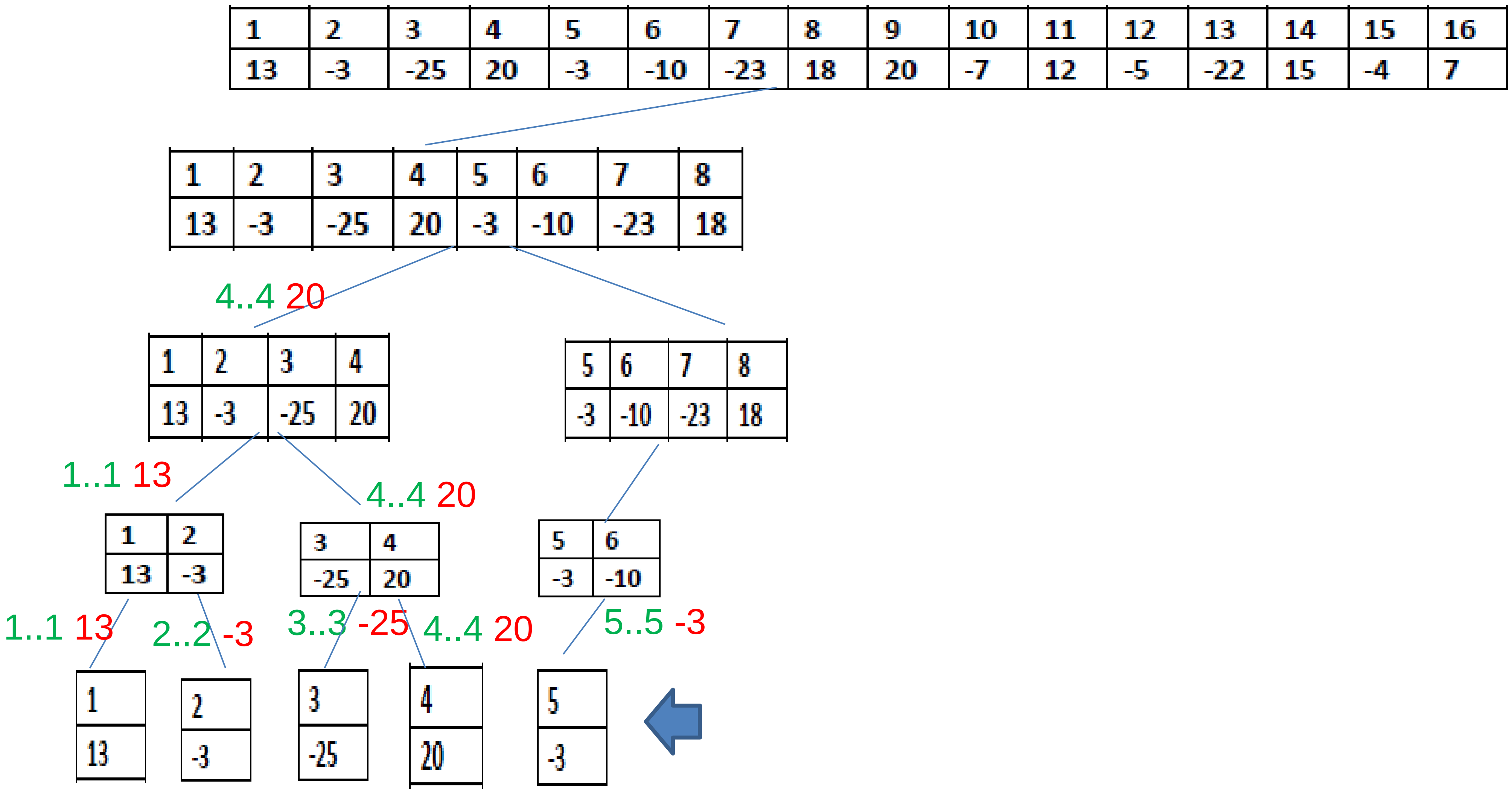
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20 max(13,20,10-5)
1 2 3 4
13 -3 -25 20
1..1 13
4..4 20 max(-25,20,-5)
1 2
13 -3
3 4
-25 20
1..1 13
2..2 -3
3..3-25
4..4 20
1
13
2
-3
3
-25
4
20



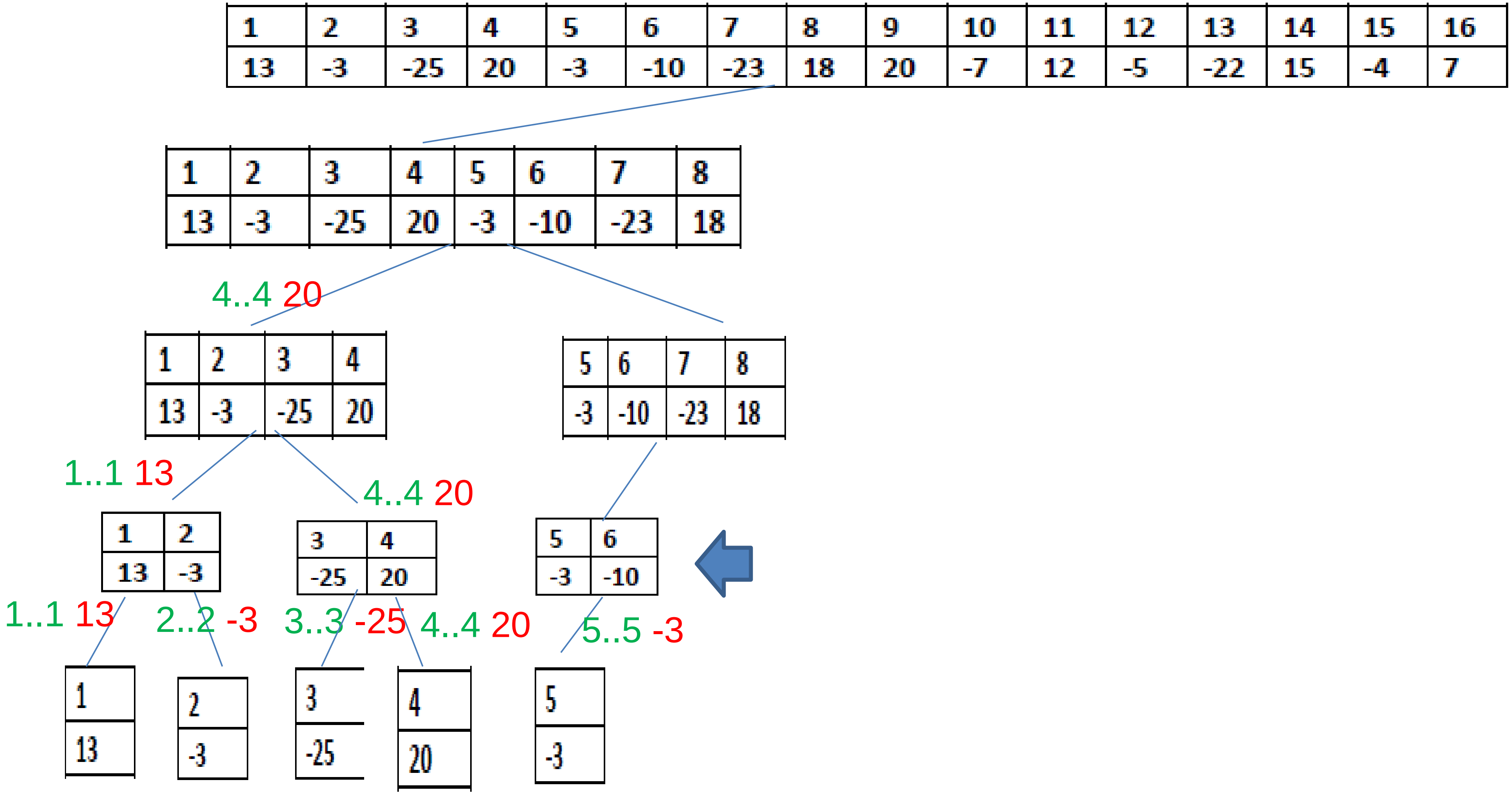
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
1 2
13 -3
3 4
-25 20
1..1 13
2..2 -3
3..3 -25
4..4 20
1
13
2
-3
3
-25
4
20



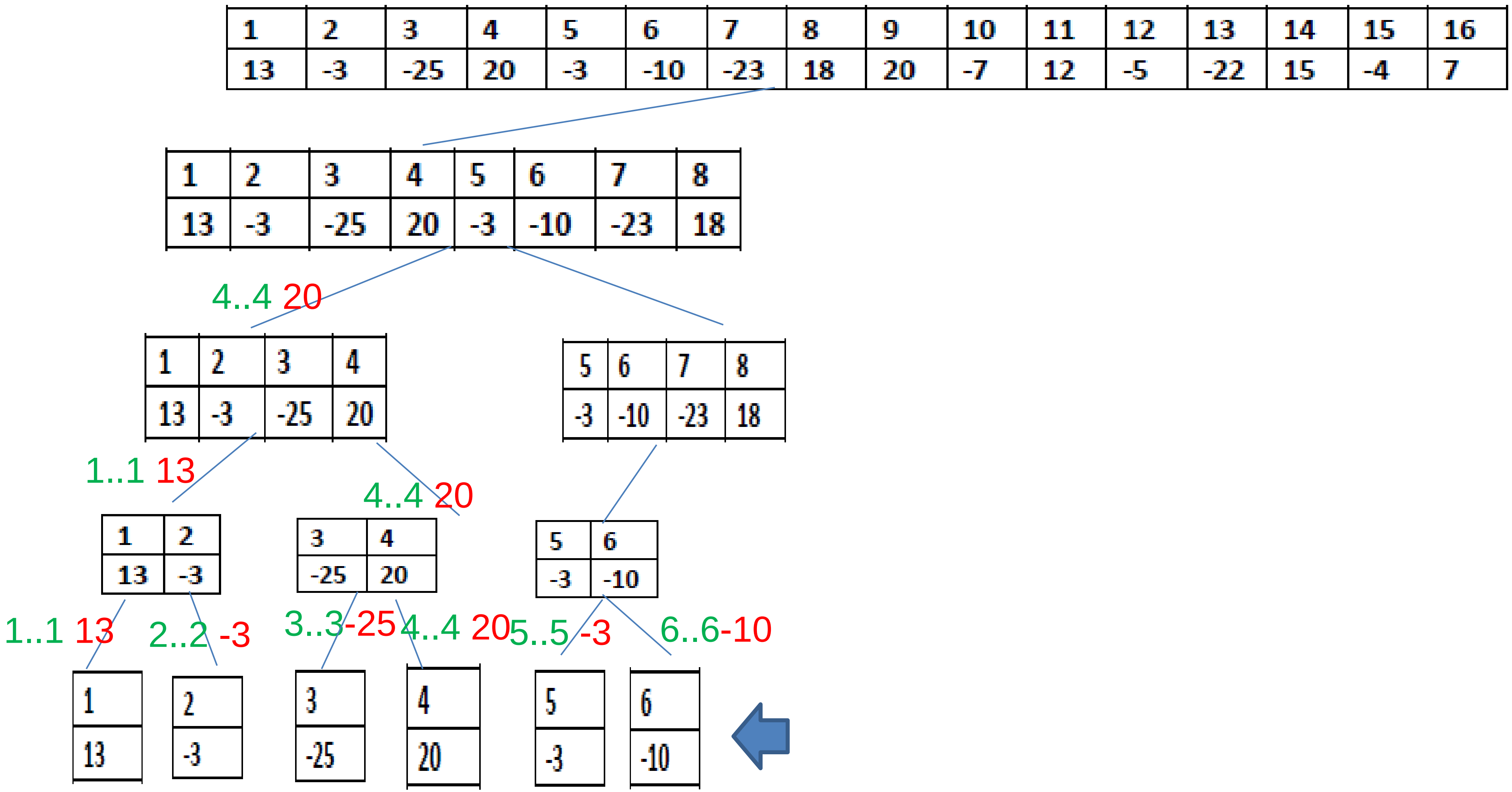
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
1 2
13 -3
3 4
-25 20
5 6
-3 -10
1..1 13
2..2 -3
3..3 -25
4..4 20
1
13
2
-3
3
-25
4
20



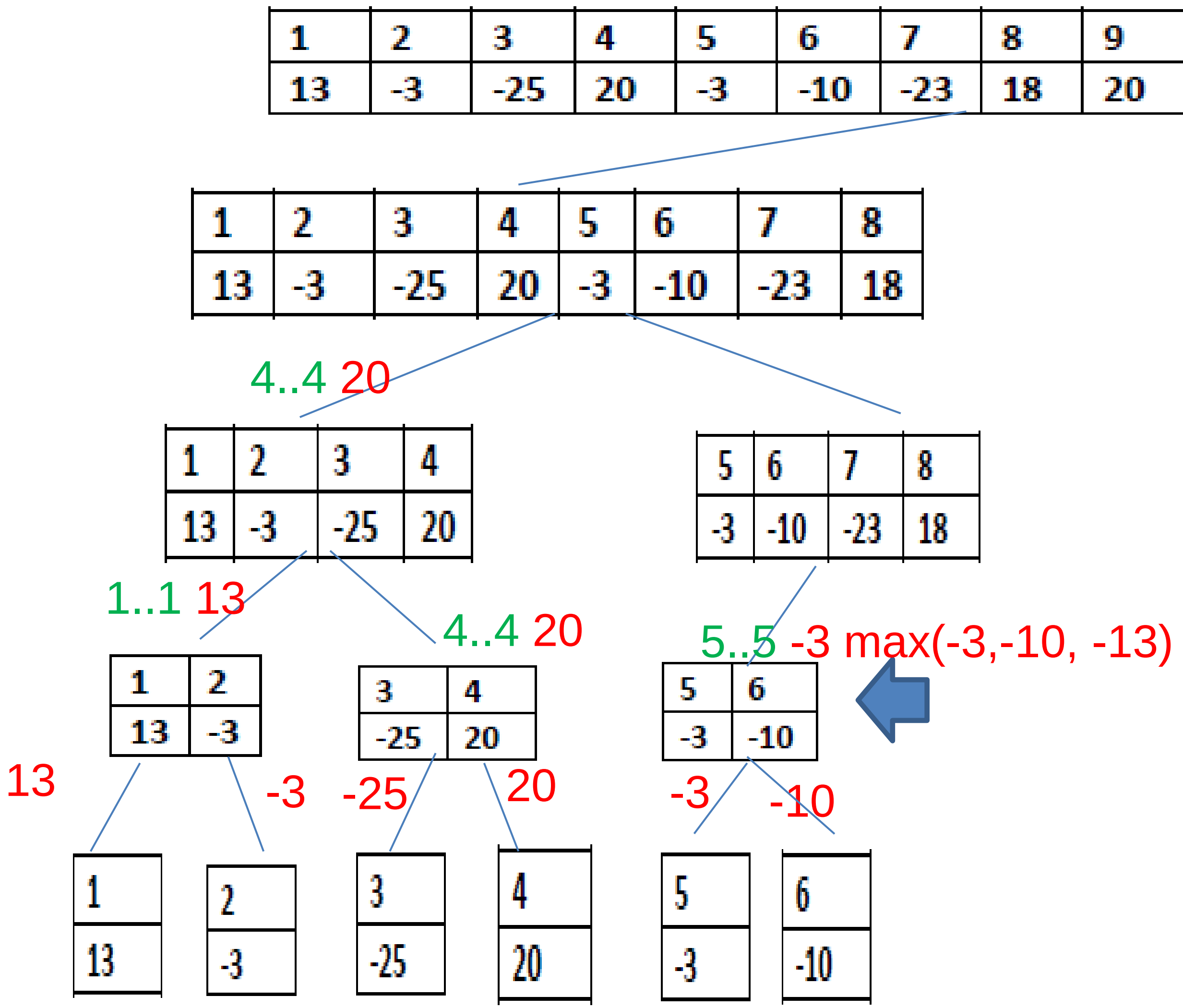
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
1 2
13 -3
3 4
-25 20
5 6
-3 -10
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
1
13
2
-3
3
-25
4
20
5
-3



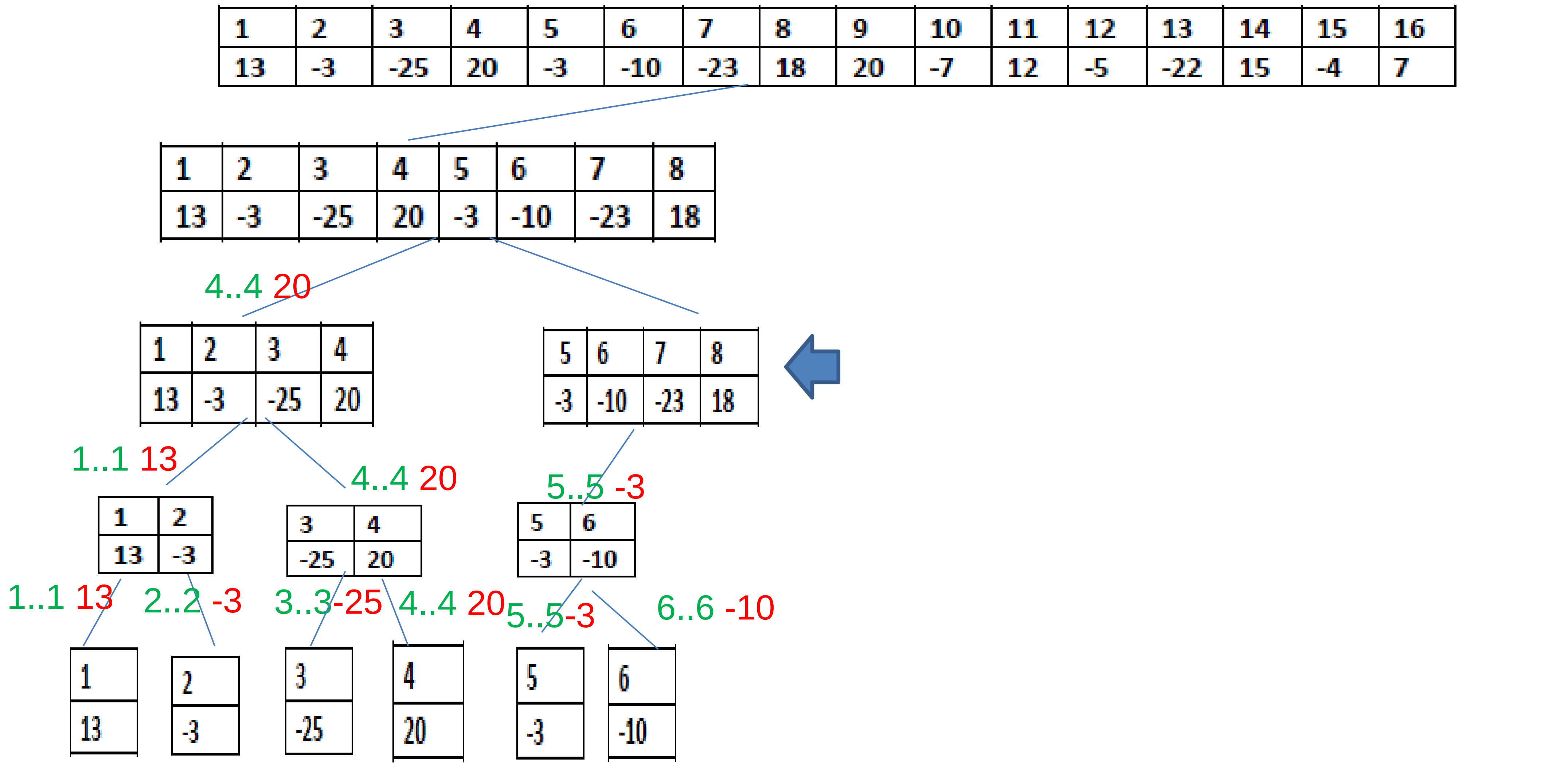
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
1 2
13 -3
3 4
-25 20
5 6
-3 -10
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
1
13
2
-3
3
-25
4
20
5
-3

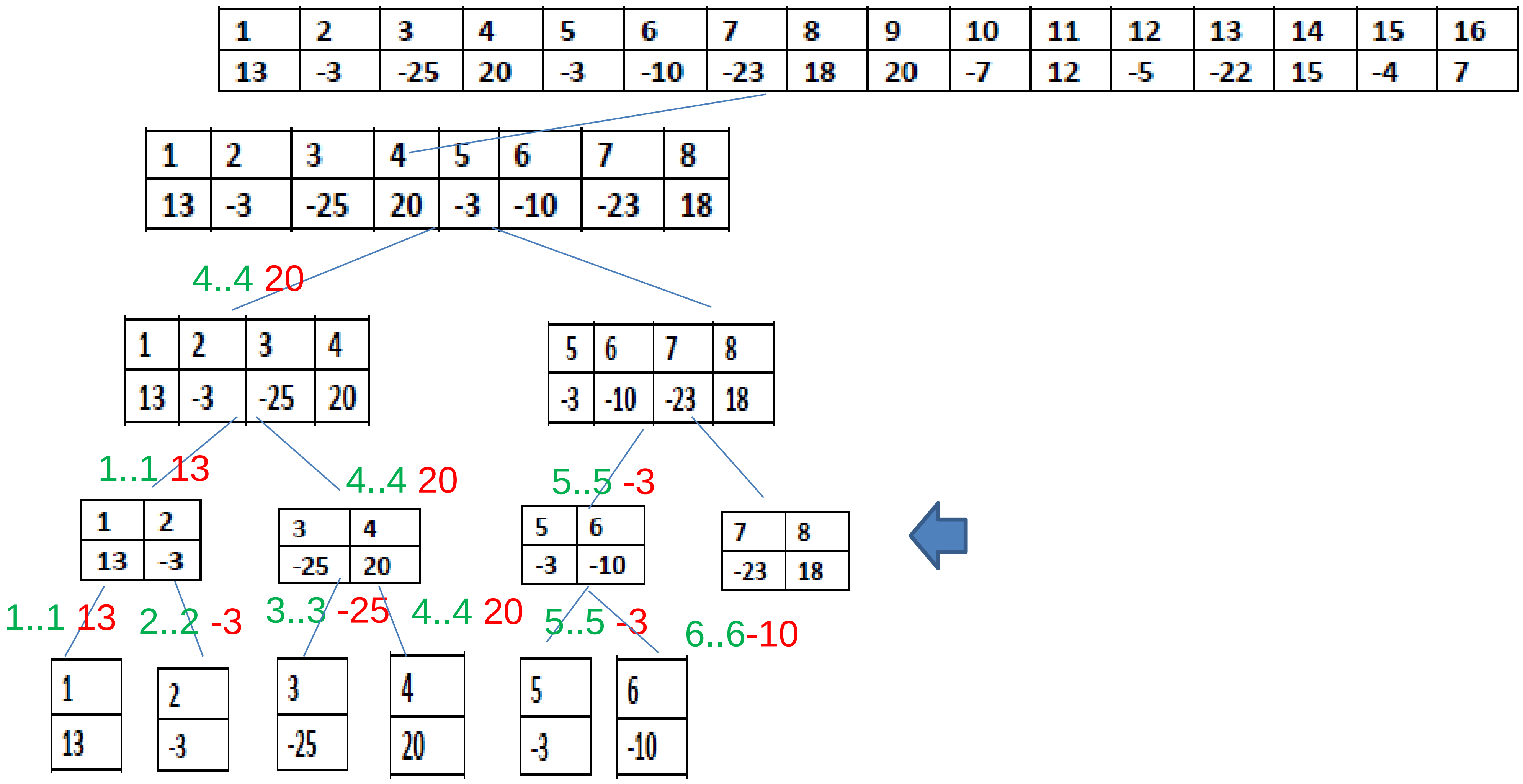


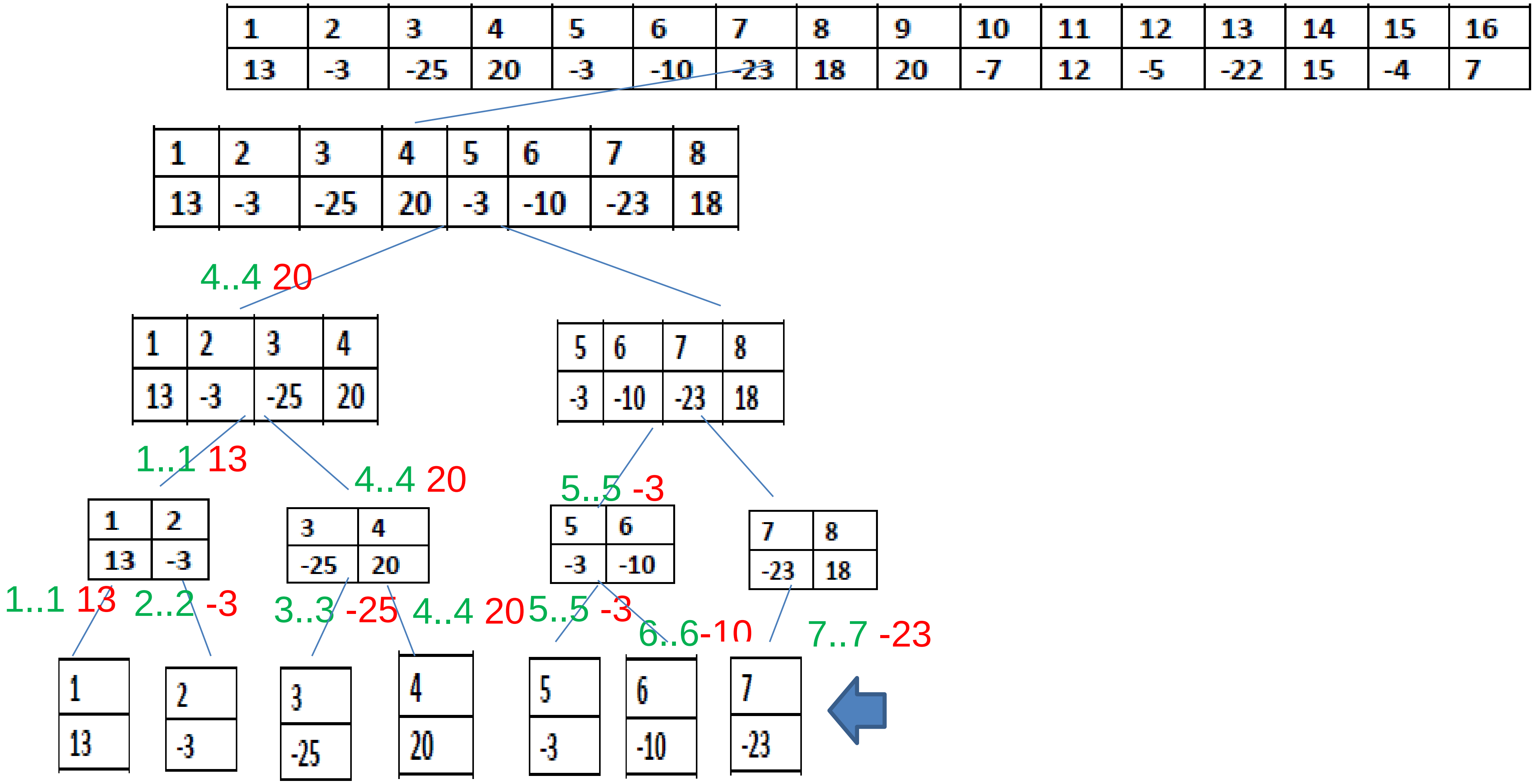
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
1 2
13 -3
3 4
-25 20
5 6
-3 -10
1..1 13
2..2 -3
3..3-25
4..4 20
5..5 -3
6..6-10
1
13
2
-3
3
-25
4
20
5
-3
6
-10



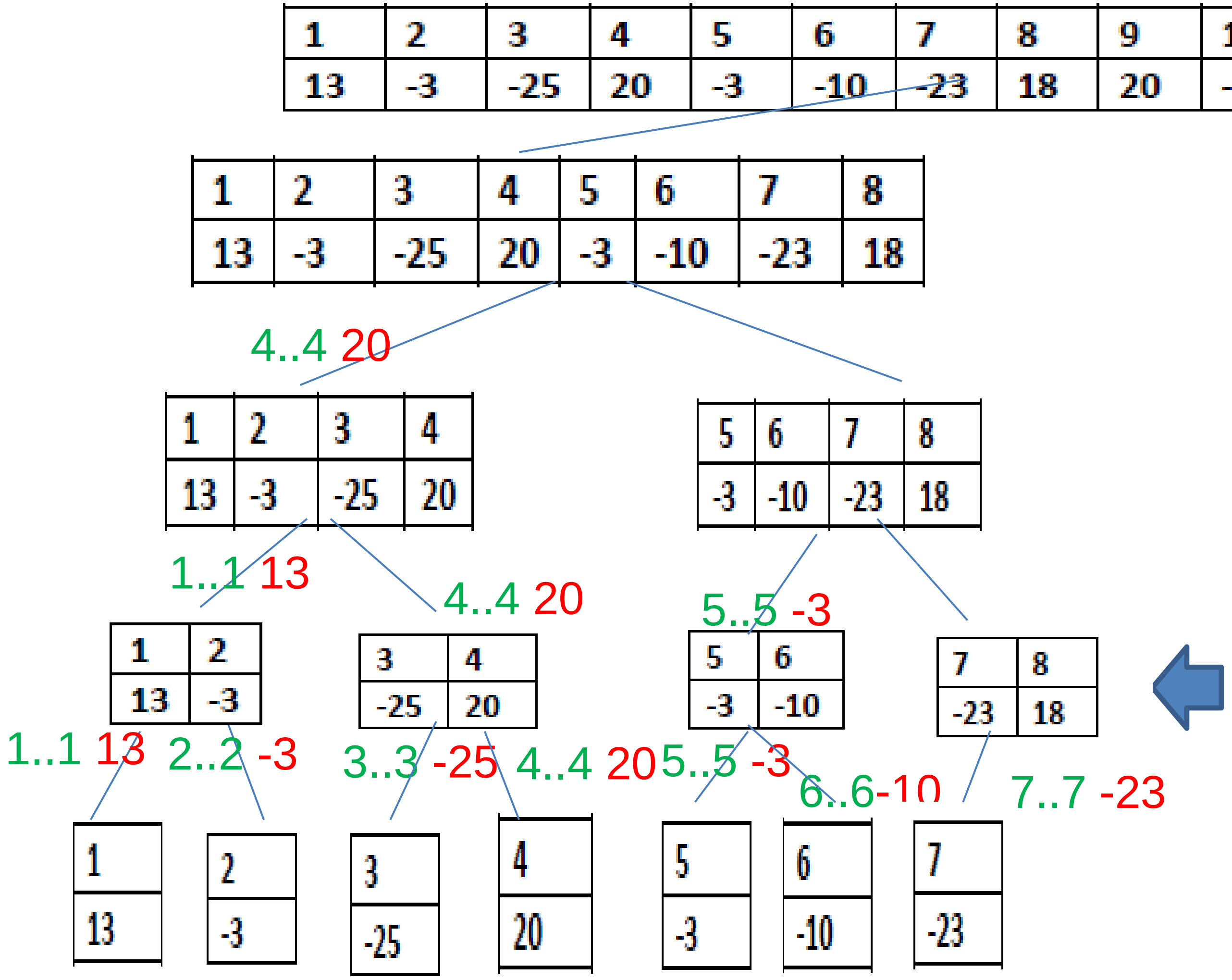
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
5..5 -3 max(-3,-10, -13)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
13
-3
-25
20
-3
-10
1
13
2
-3
3
-25
4
20
5
-3
6
-10

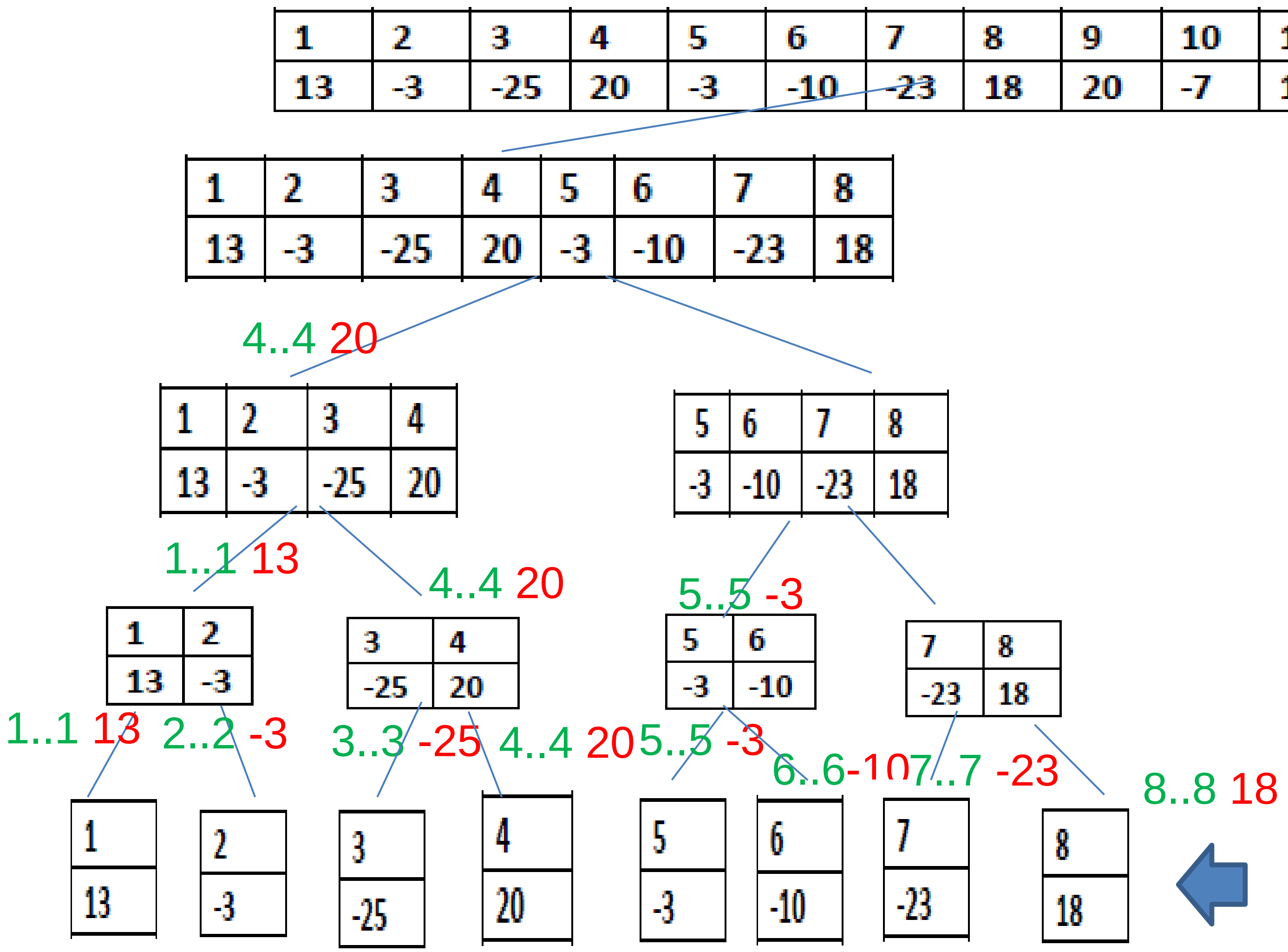

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
5..5 -3
1 2
13 -3
3 4
-25 20
5 6
-3 -10
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
6..6 -10
1
13
2
-3
3
-25
4
20
5
-3
6
-10


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
5..5 -3
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
6..6-10
1
13
2
-3
3
-25
4
20
5
-3
6
-10

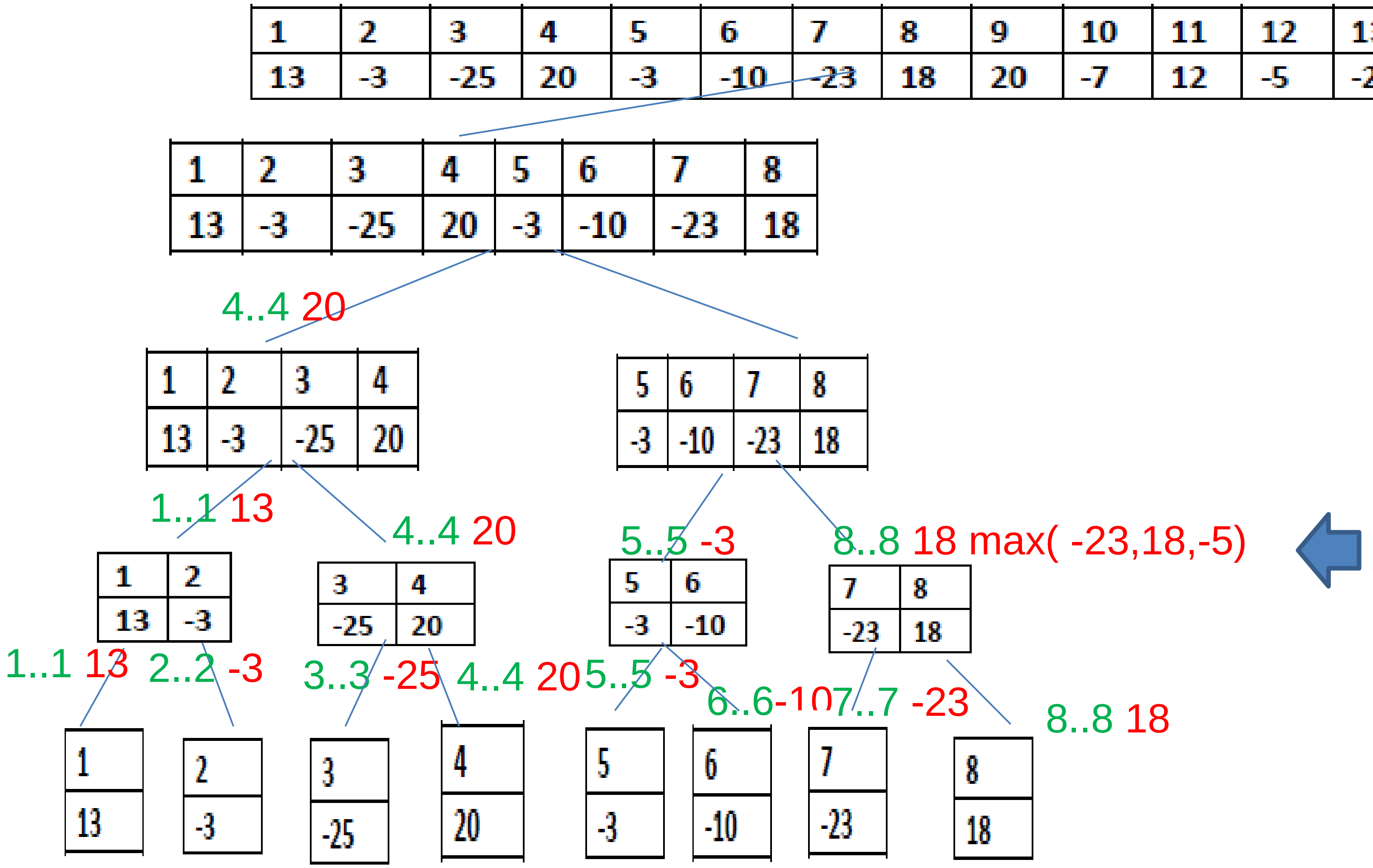


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
5..5 -3
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
6..6 -10
7..7 -23
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23

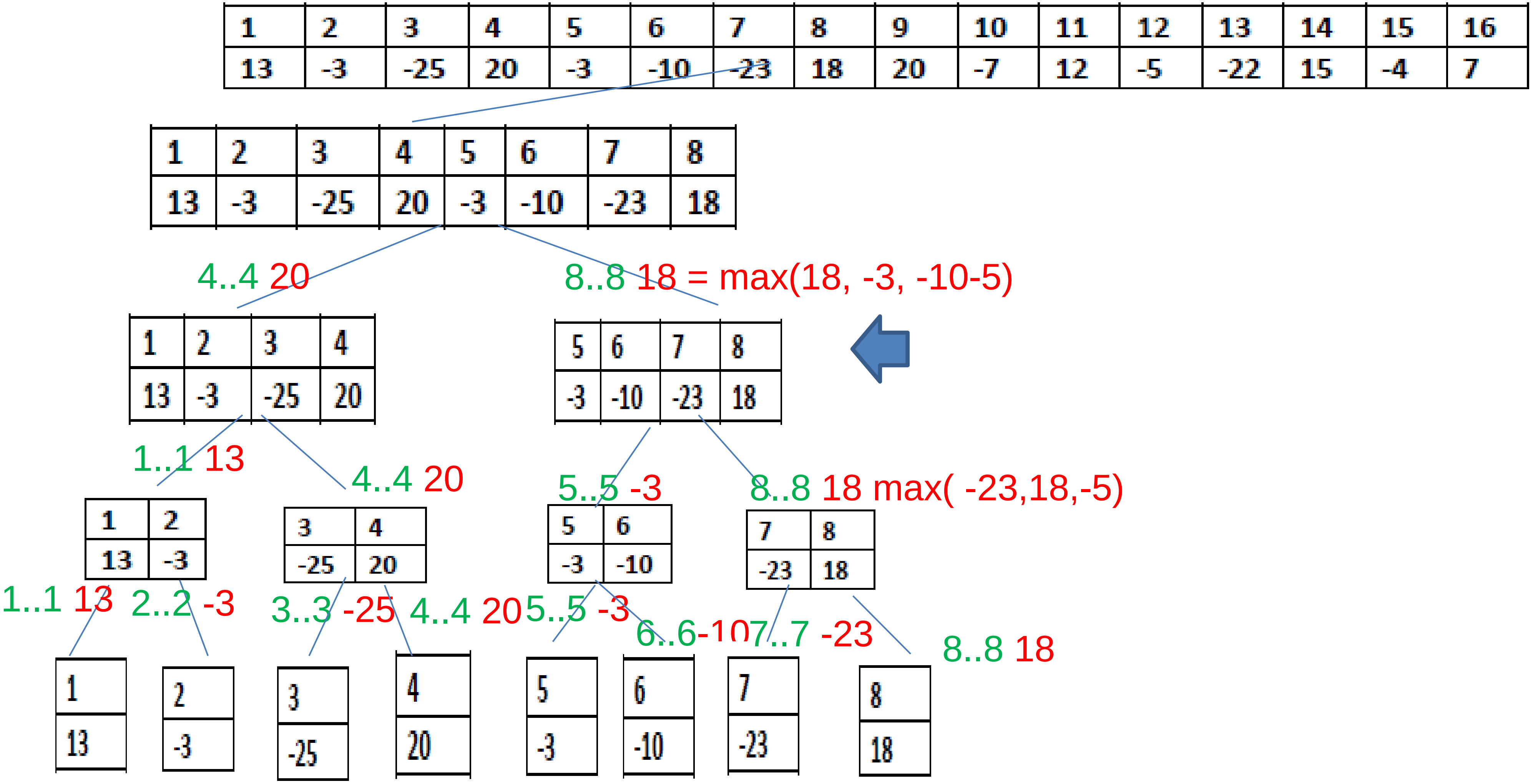

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
5..5 -3
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
6..6 -10
7..7 -23
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23

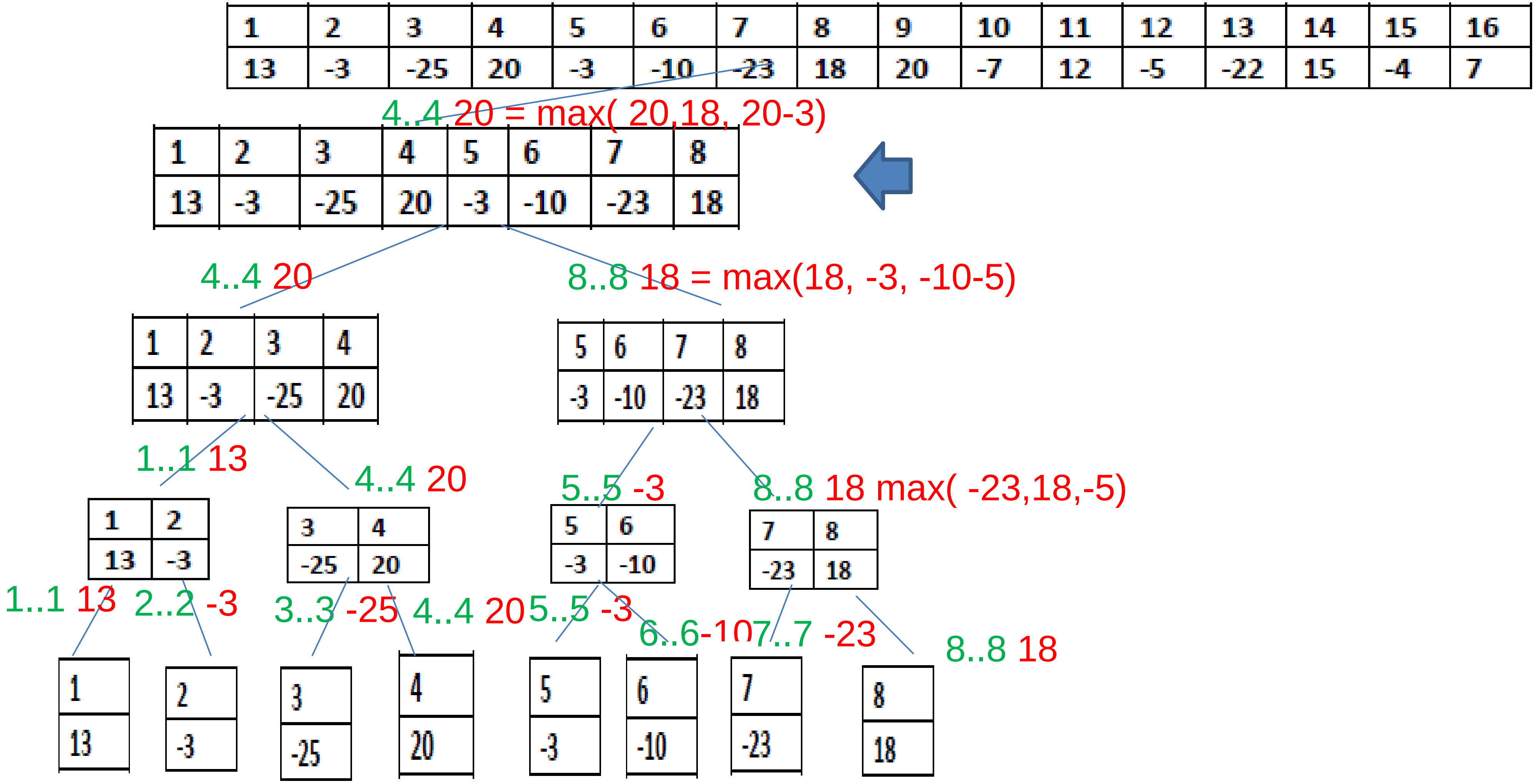


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
5..5 -3
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
6..6 -10
7..7 -23
8..8 18
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18

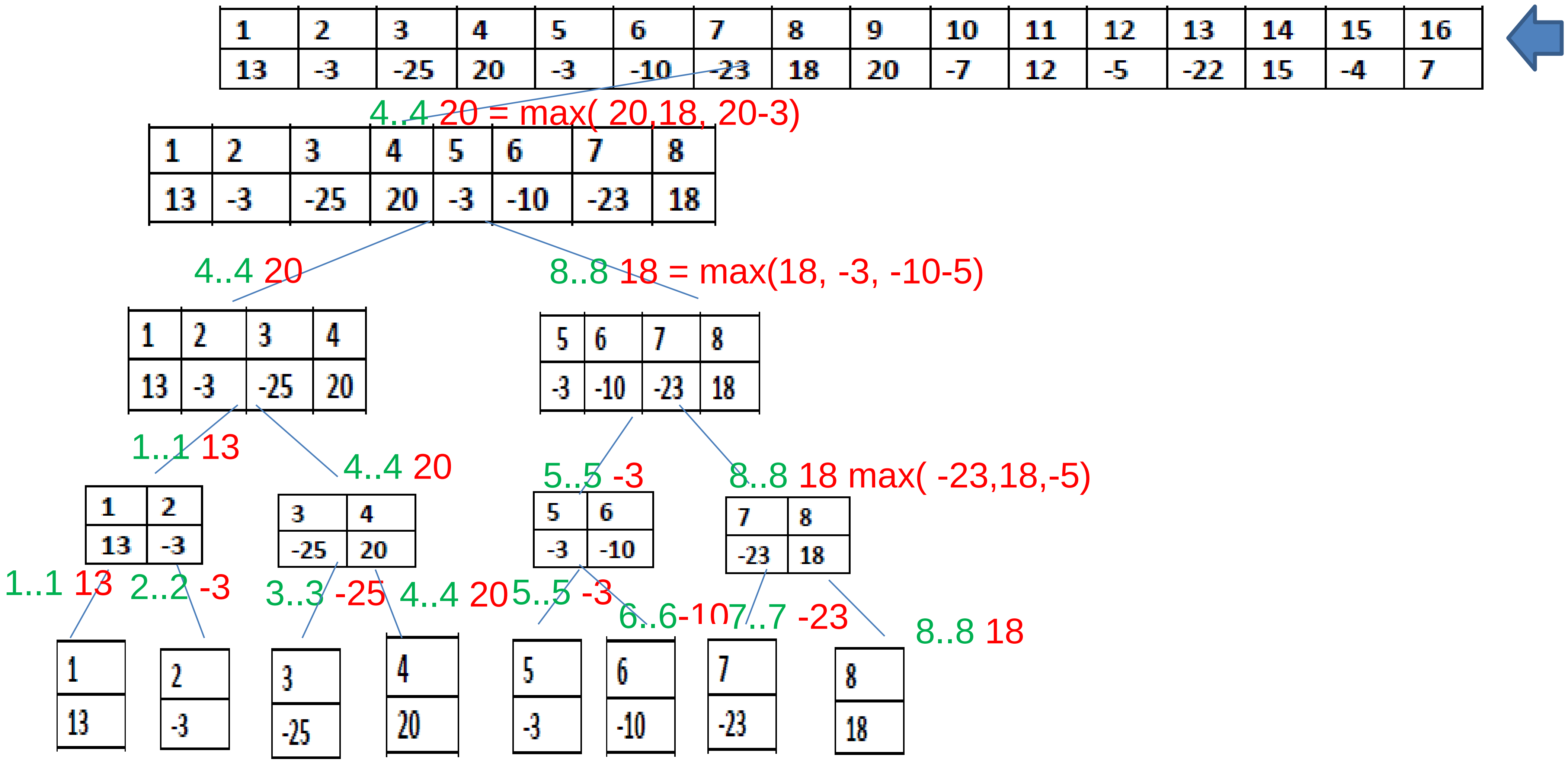


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
5..5 -3
8..8 18 max(-23,18,-5)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
6..6-10
7..7 -23
8..8 18
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18

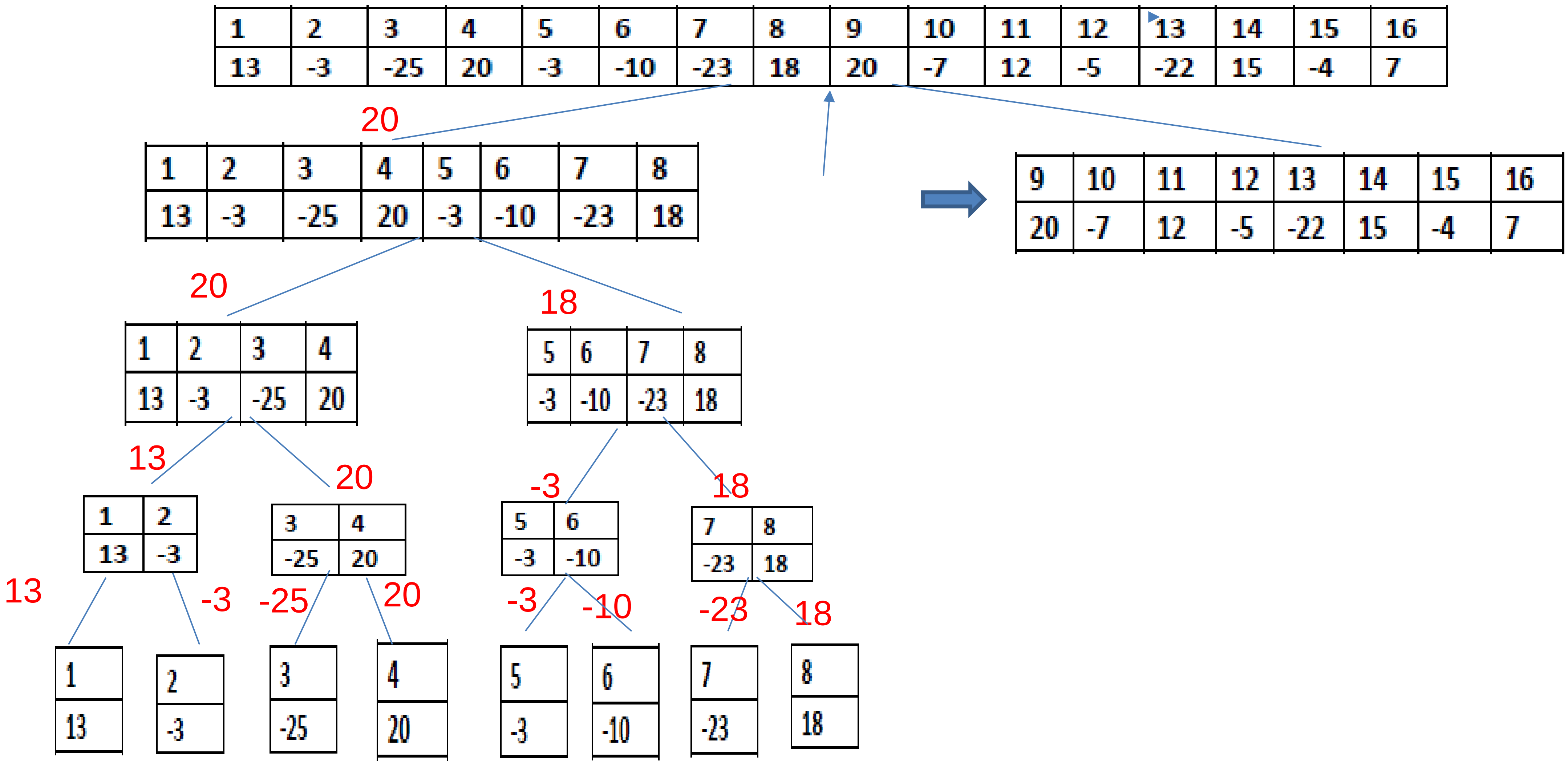

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
8..8 18 = max(18, -3, -10-5)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
5..5 -3
8..8 18 max(-23,18,-5)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
6..6-10
7..7 -23
8..8 18
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18



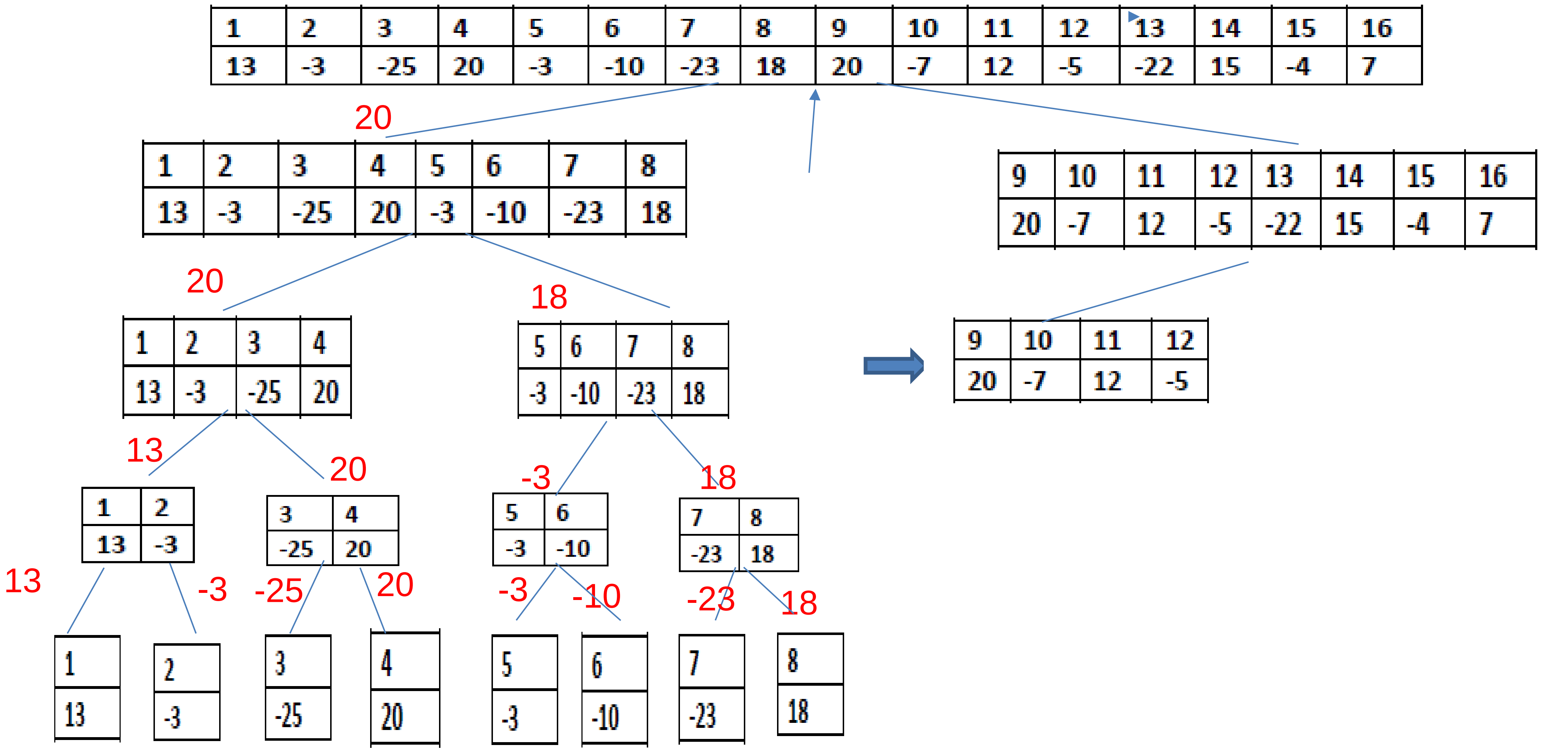
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
4..4 20 = max(20,18, 20-3)
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
8..8 18 = max(18, -3, -10-5)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
5..5 -3
8..8 18 max(-23,18,-5)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
6..6-10
7..7 -23
8..8 18
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18



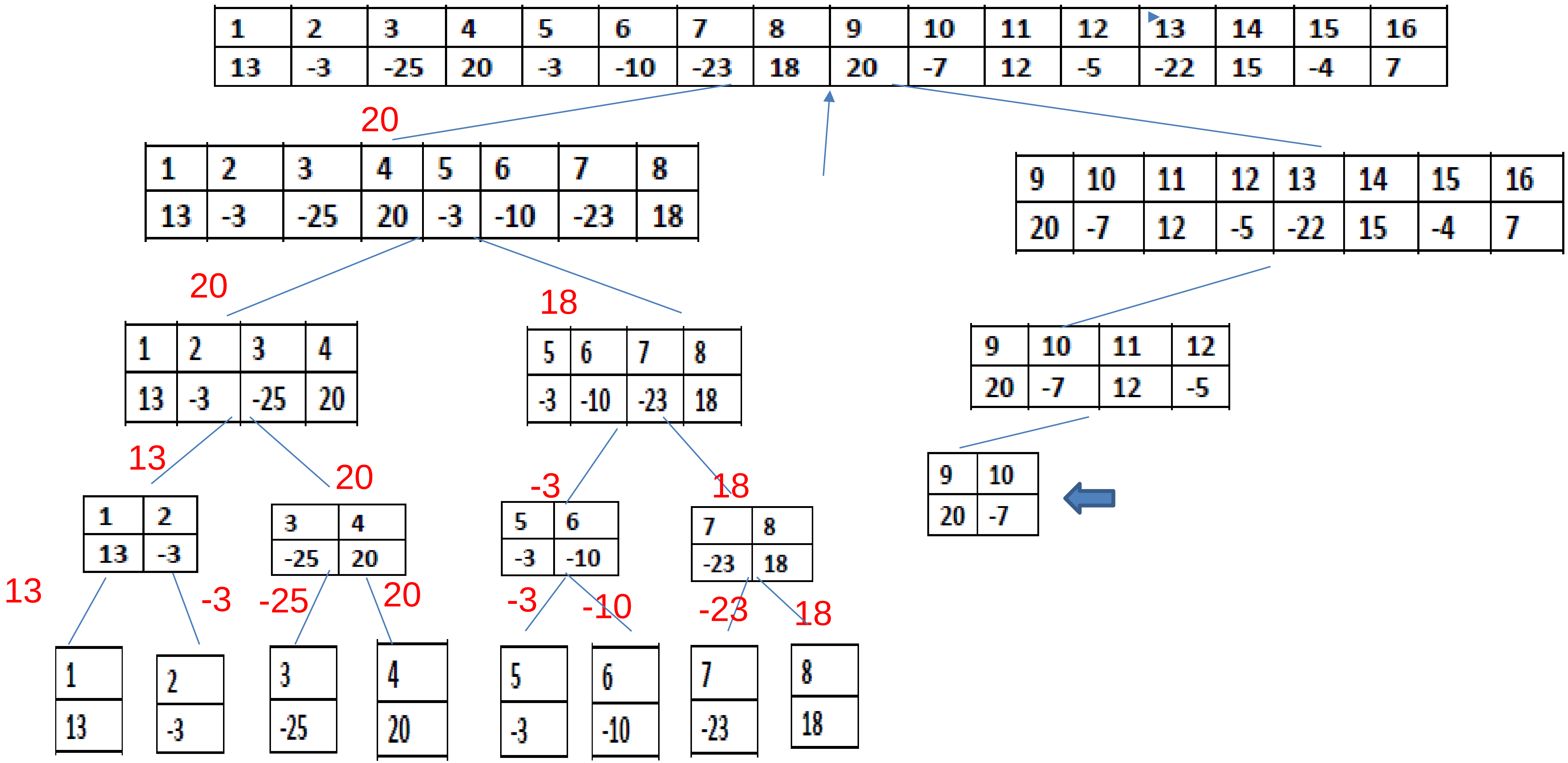
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
4..4 20 = max(20,18, 20-3)
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
4..4 20
8..8 18 = max(18, -3, -10-5)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
1..1 13
4..4 20
5..5 -3
8..8 18 max(-23,18,-5)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
1..1 13
2..2 -3
3..3 -25
4..4 20
5..5 -3
6..6-10
7..7 -23
8..8 18
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18



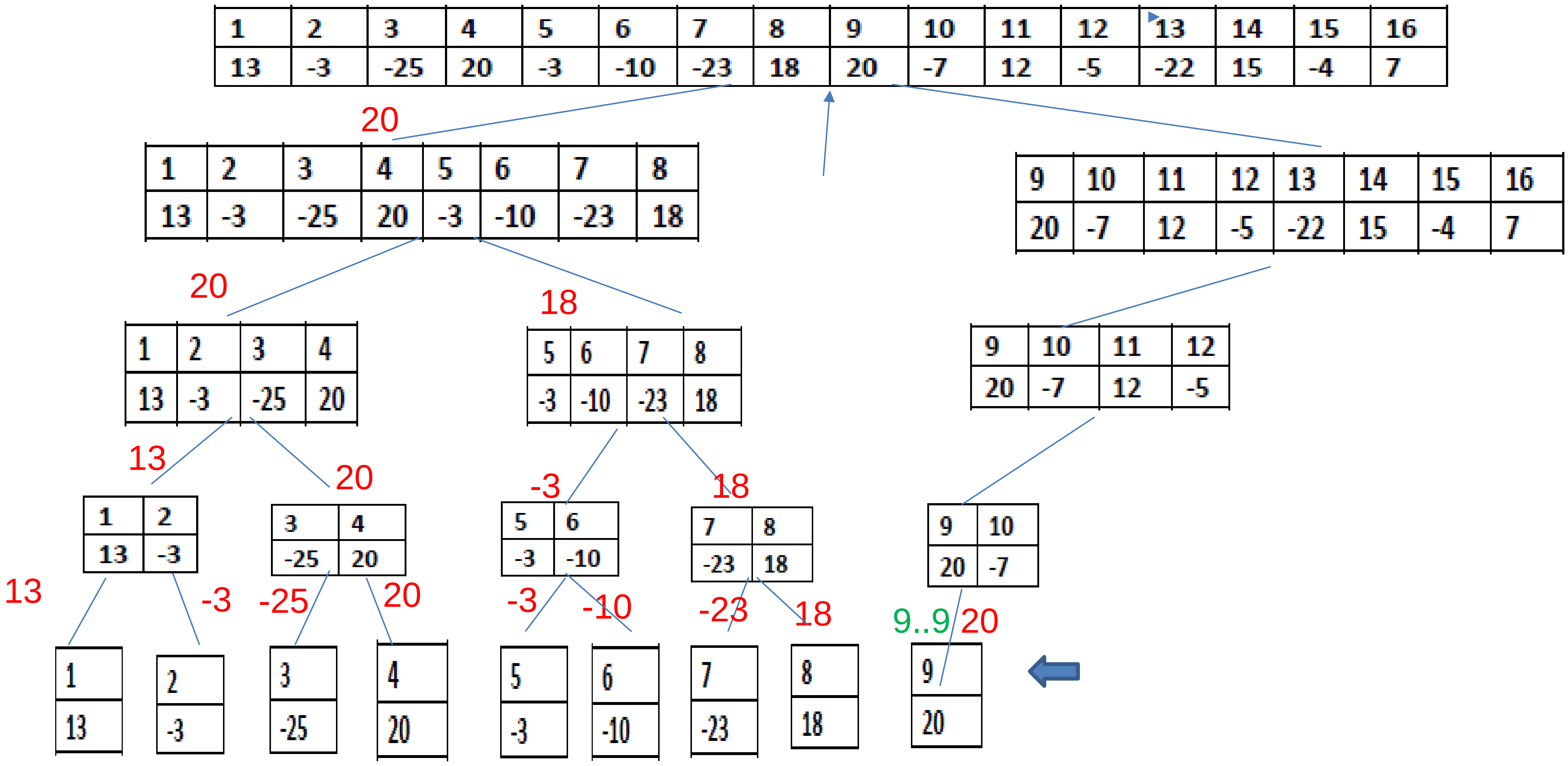
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
13
1 2
13 -3
20
3 4
-25 20
-3
5 6
-3 -10
18
7 8
-23 18
13
-3
-25
20
-3
-10
-23
18
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18



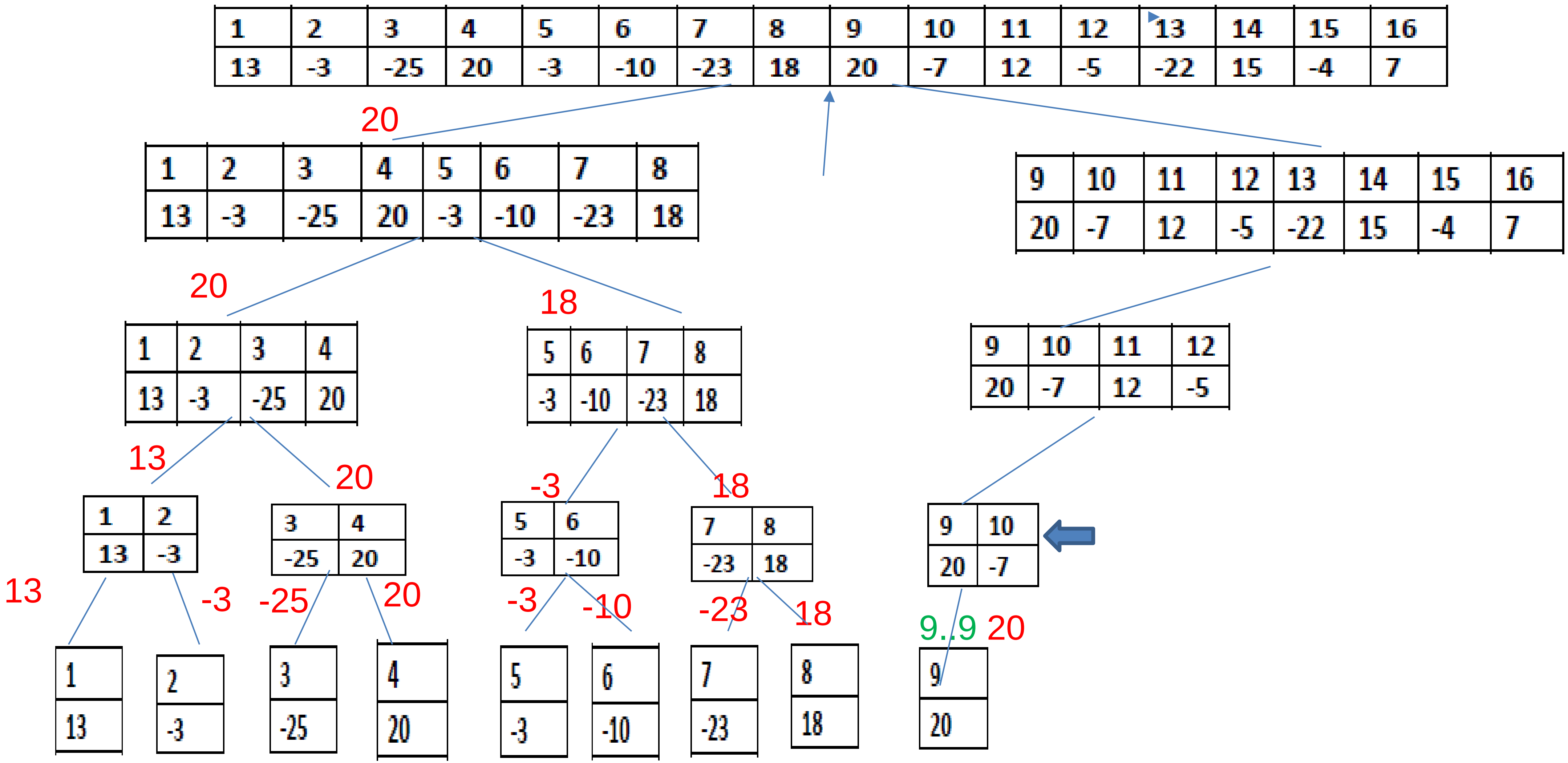
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13
20
1 2
13 -3
3 4
-25 20
-3
18
5 6
-3 -10
7 8
-23 18
13
-3
-25
20
-3
-10
-23
18
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18



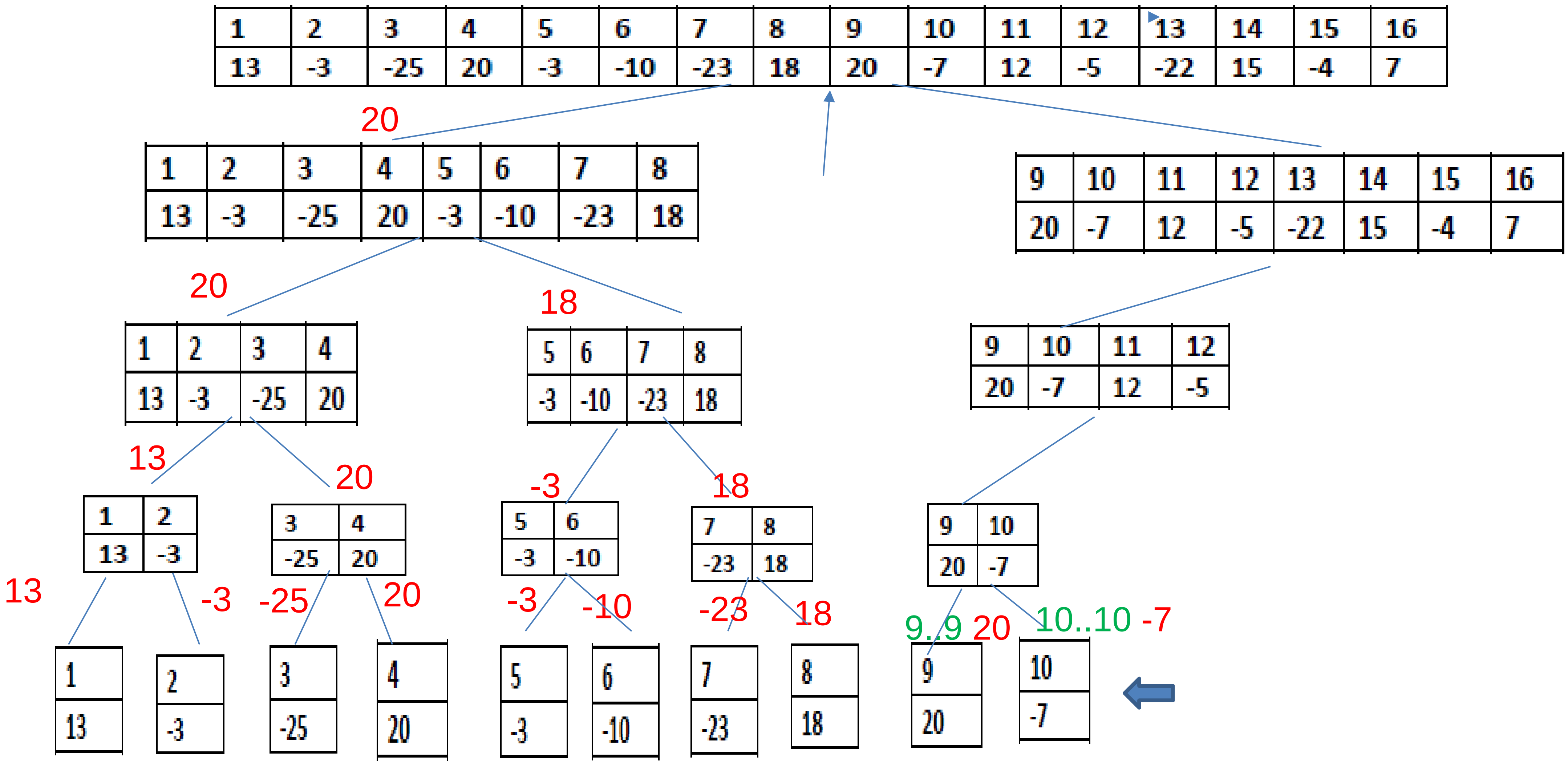
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13
1 2
13 -3
20
3 4
-25 20
-3
5 6
-3 -10
18
7 8
-23 18
9 10
20 -7
13
-3
-25
20
-3
-10
-23
18
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18

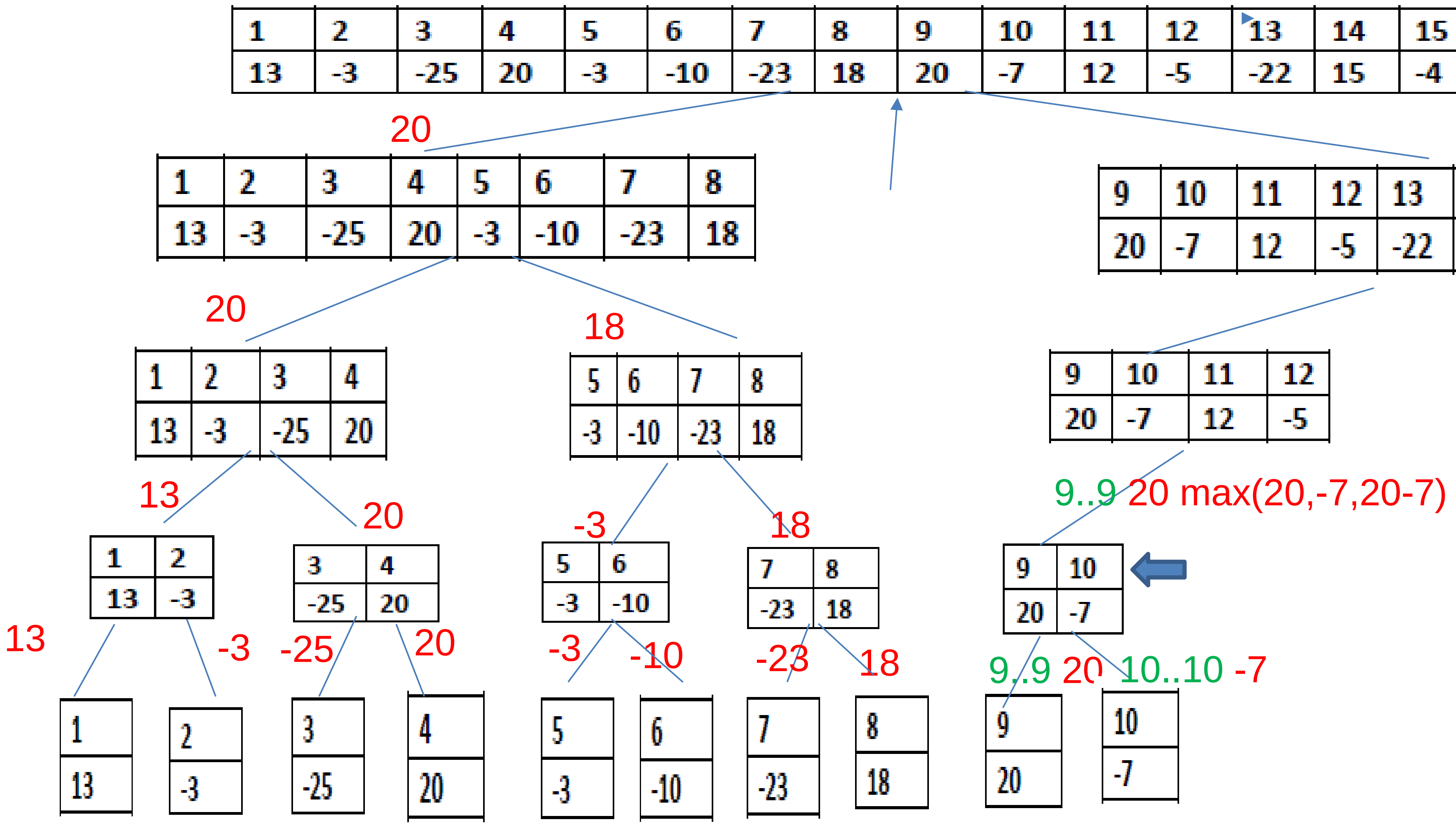


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13
1 2
13 -3
20
3 4
-25 20
-3
5 6
-3 -10
18
7 8
-23 18
9 10
20 -7
13
-3
-25
20
-3
-10
-23
18
9..9
20
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20

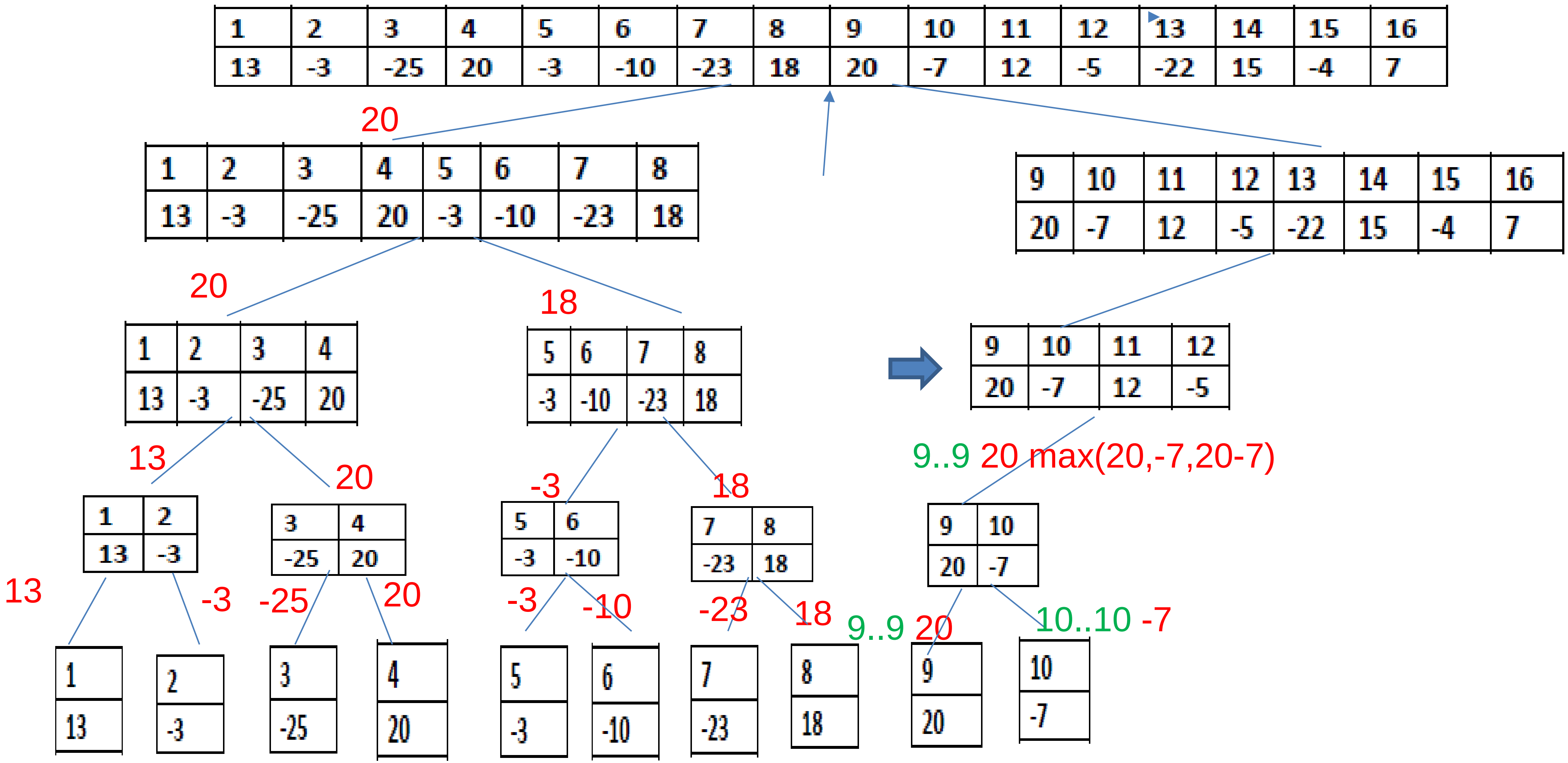


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13
1 2
13 -3
20
3 4
-25 20
-3
5 6
-3 -10
18
7 8
-23 18
9 10
20 -7
13
-3
-25
20
-3
-10
-23
18
9..9 20
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20

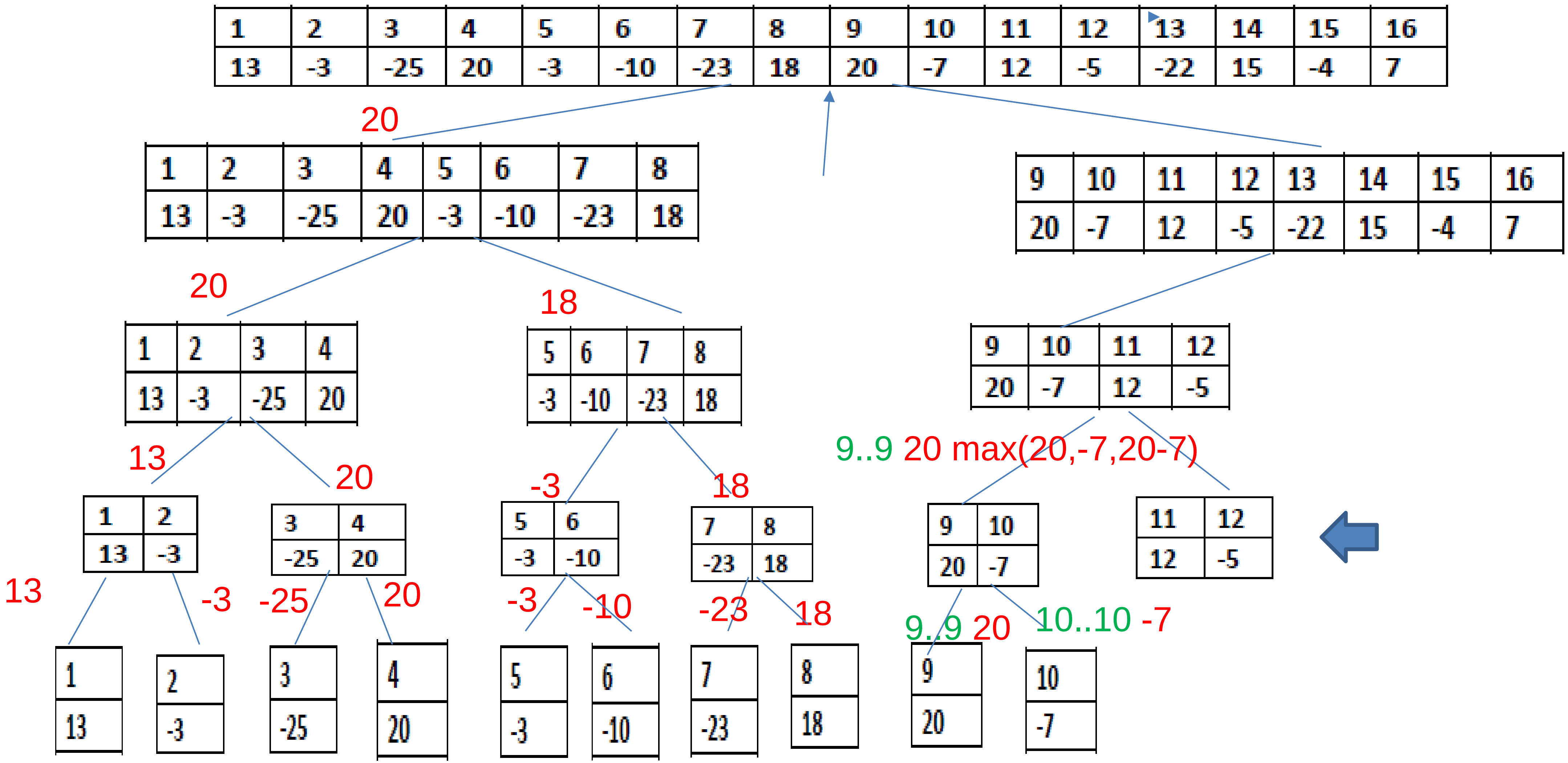

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13
1 2
13 -3
20
3 4
-25 20
-3
5 6
-3 -10
18
7 8
-23 18
9 10
20 -7
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7

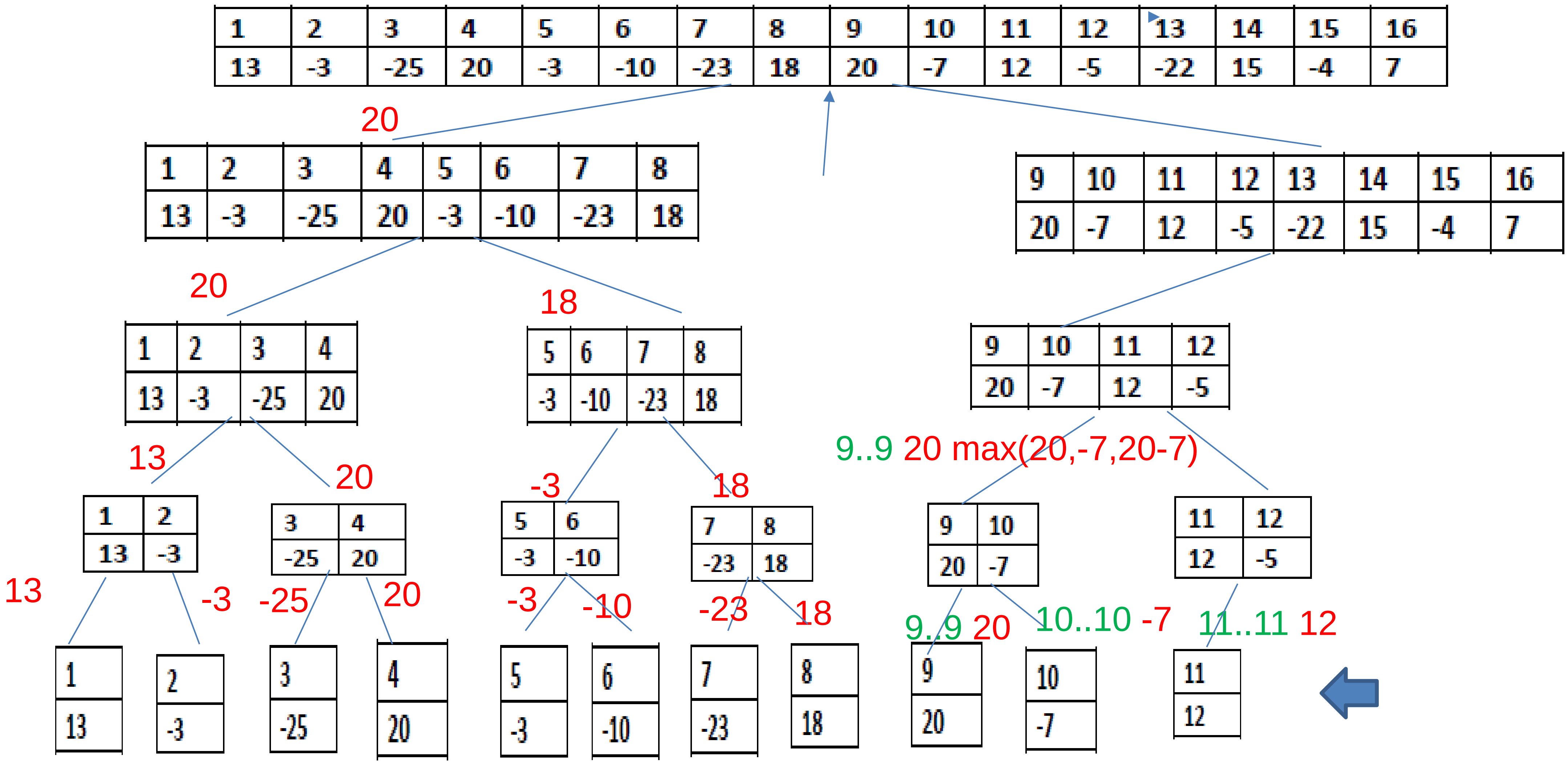


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13
1 2
13 -3
20
3 4
-25 20
-3
5 6
-3 -10
18
7 8
-23 18
9..9 20 max(20,-7,20-7)
9 10
20 -7
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7

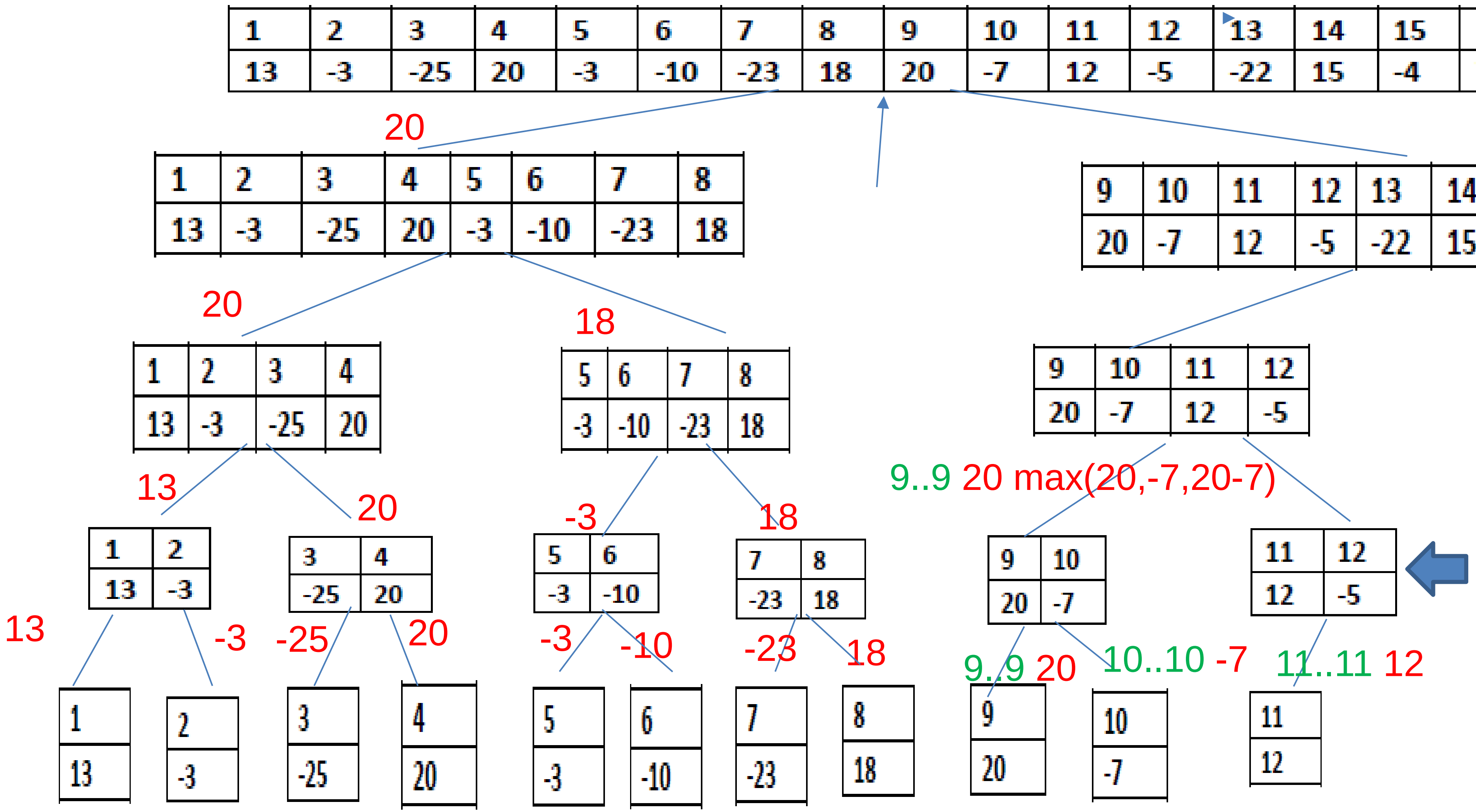


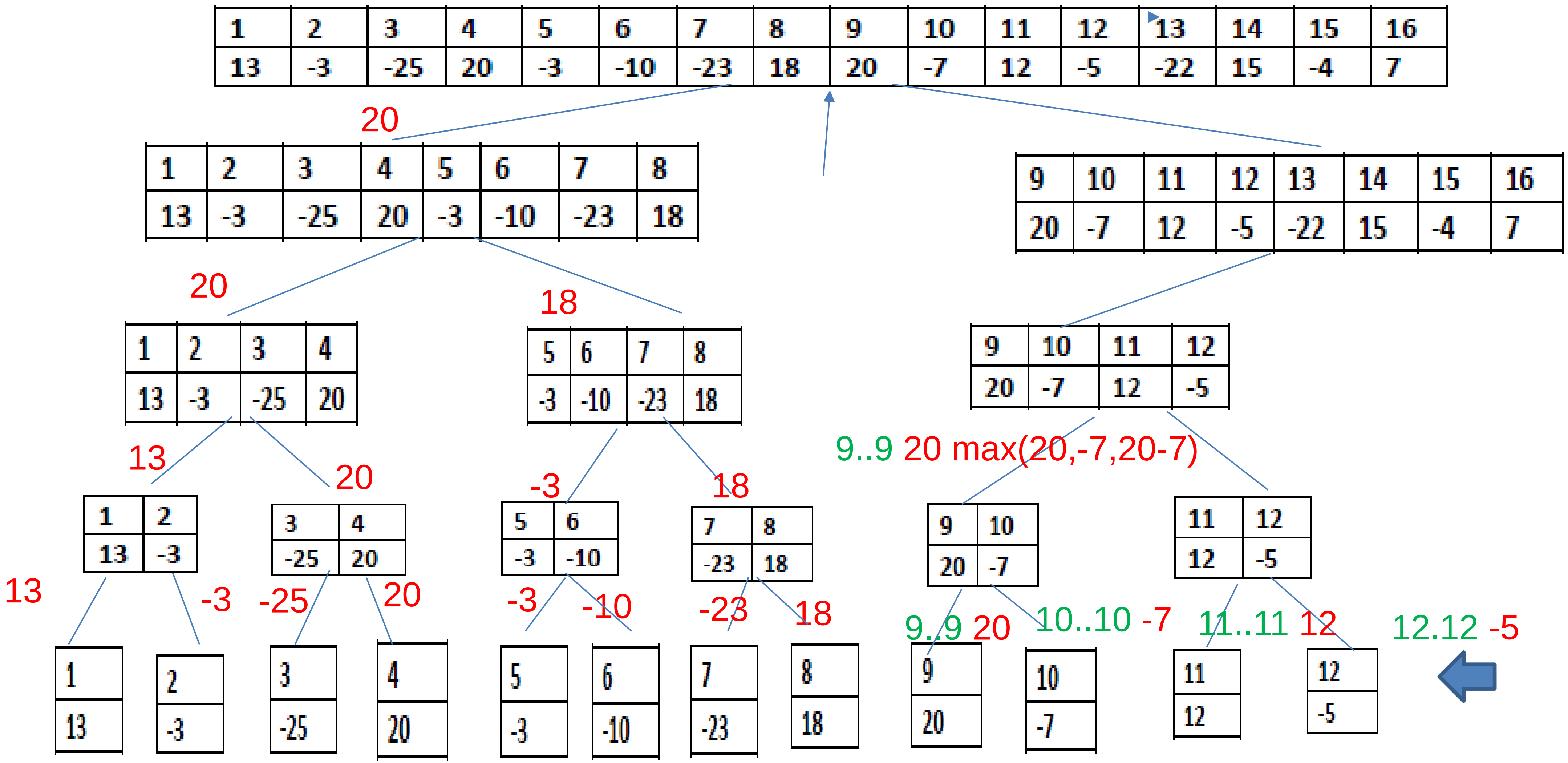
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13
20
9..9 20 max(20,-7,20-7)
-3
18
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7

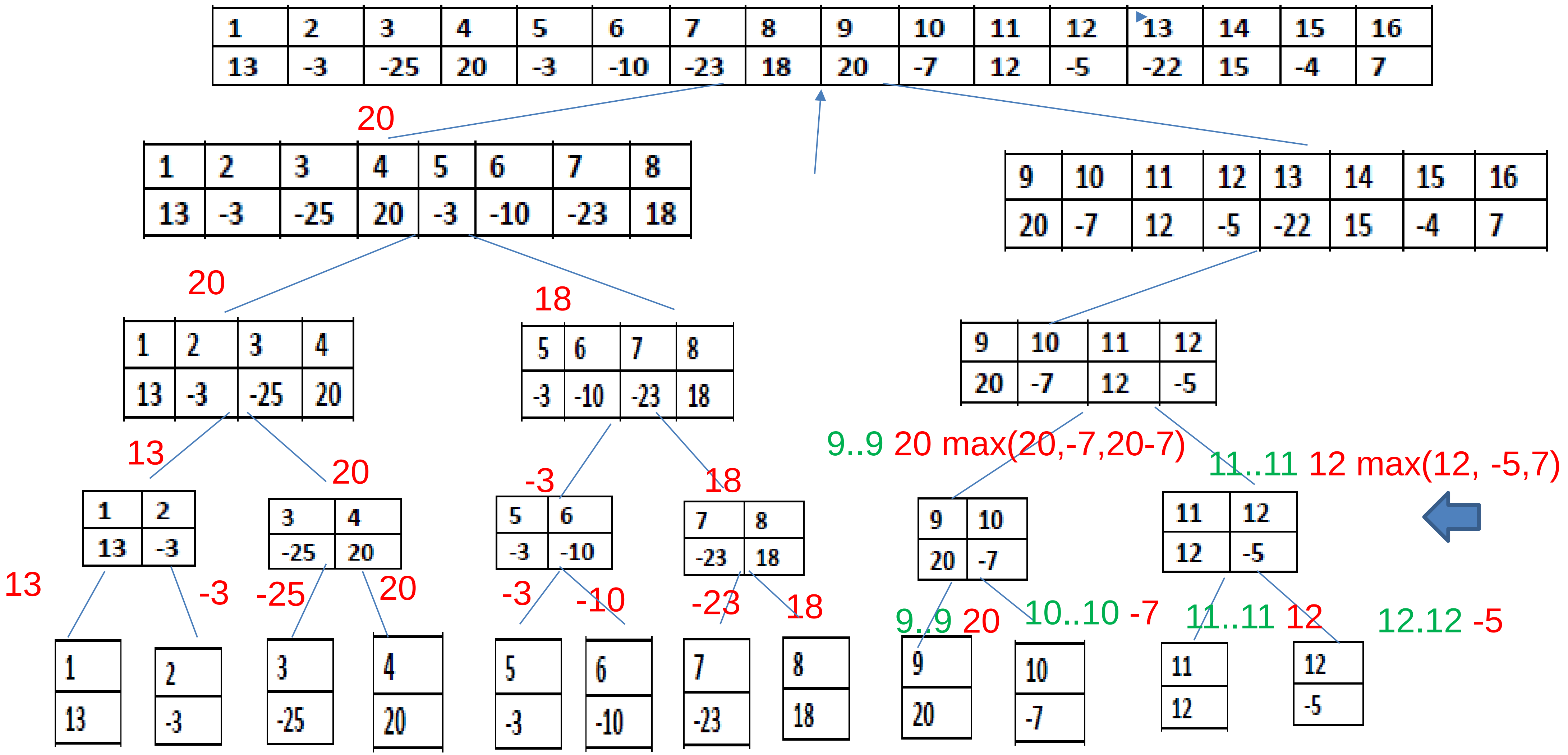

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
9..9 20 max(20,-7,20-7)
13
20
-3
18
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7

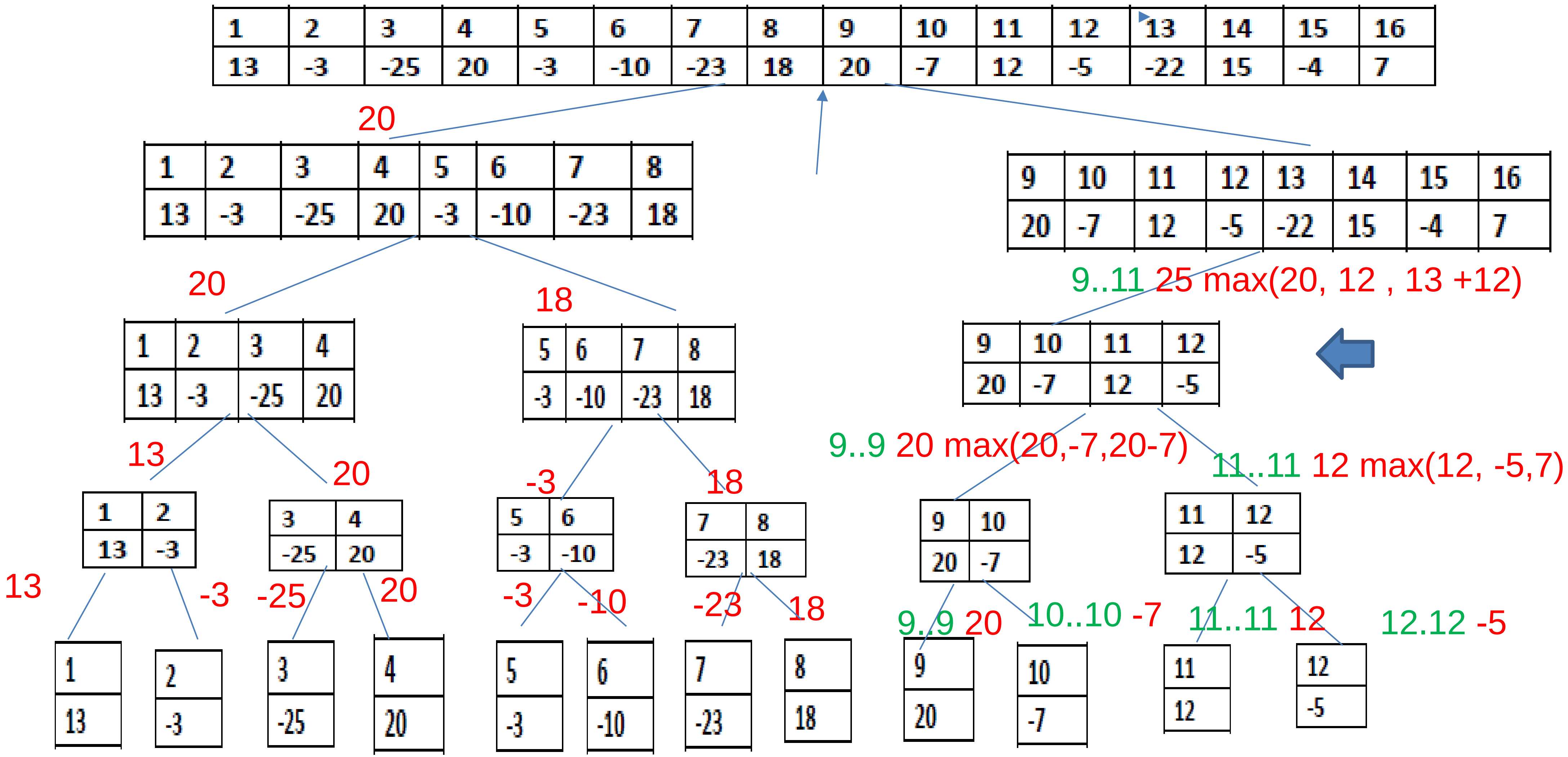


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
9..9 20 max(20,-7,20-7)
13
20
-3
18
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12

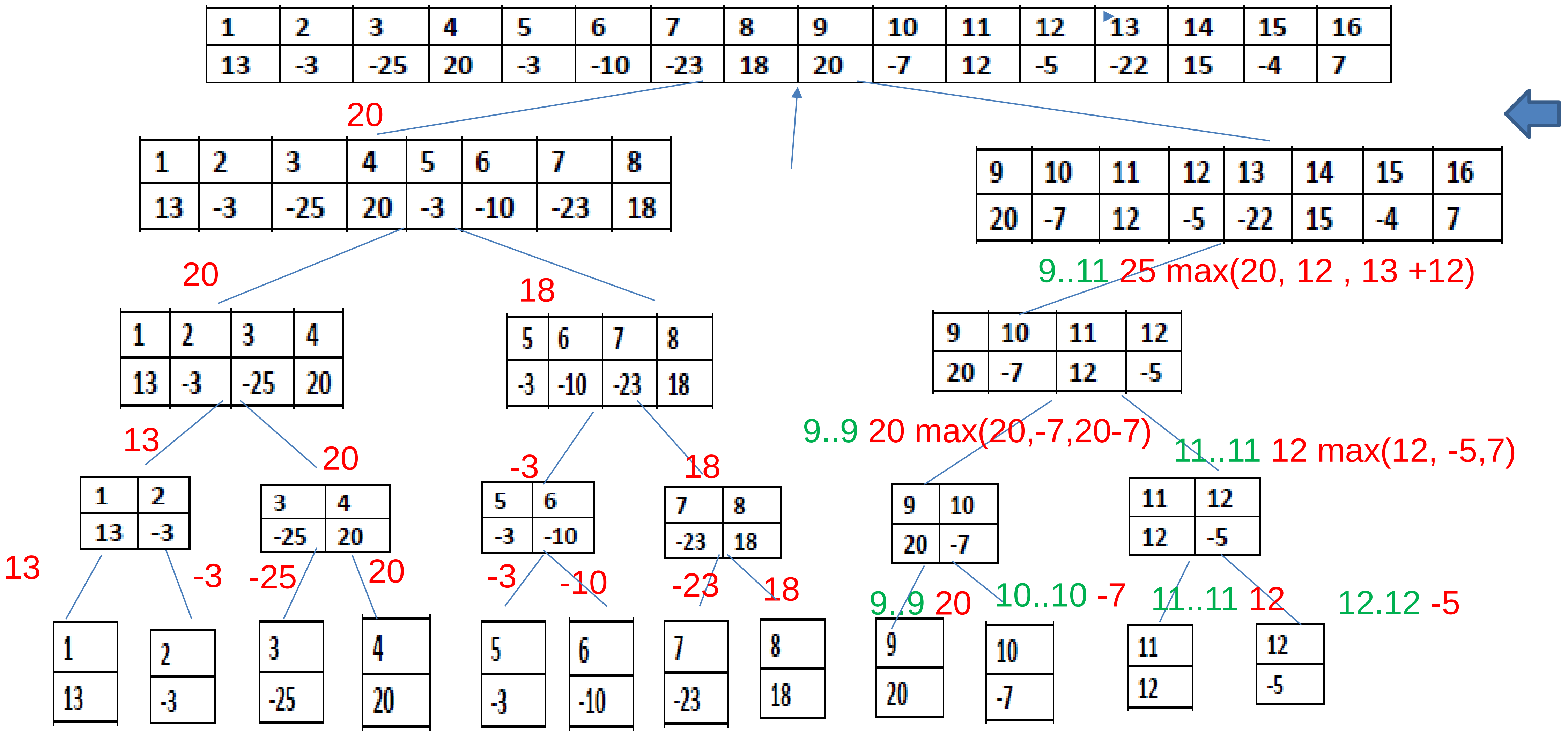

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
18
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
9..9 20 max(20,-7,20-7)
13
20
-3
18
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13
20
-3
18
9..9 20 max(20,-7,20-7)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5

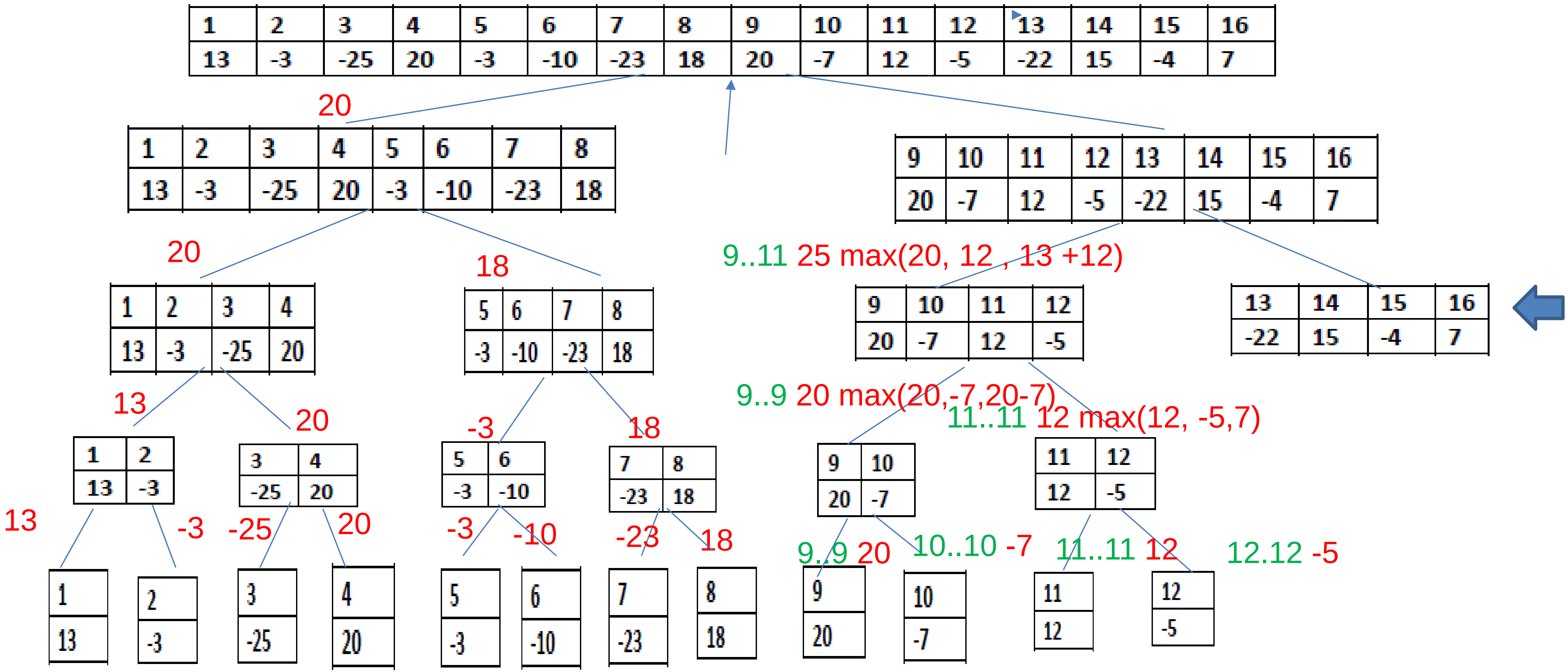

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13
20
-3
18
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5



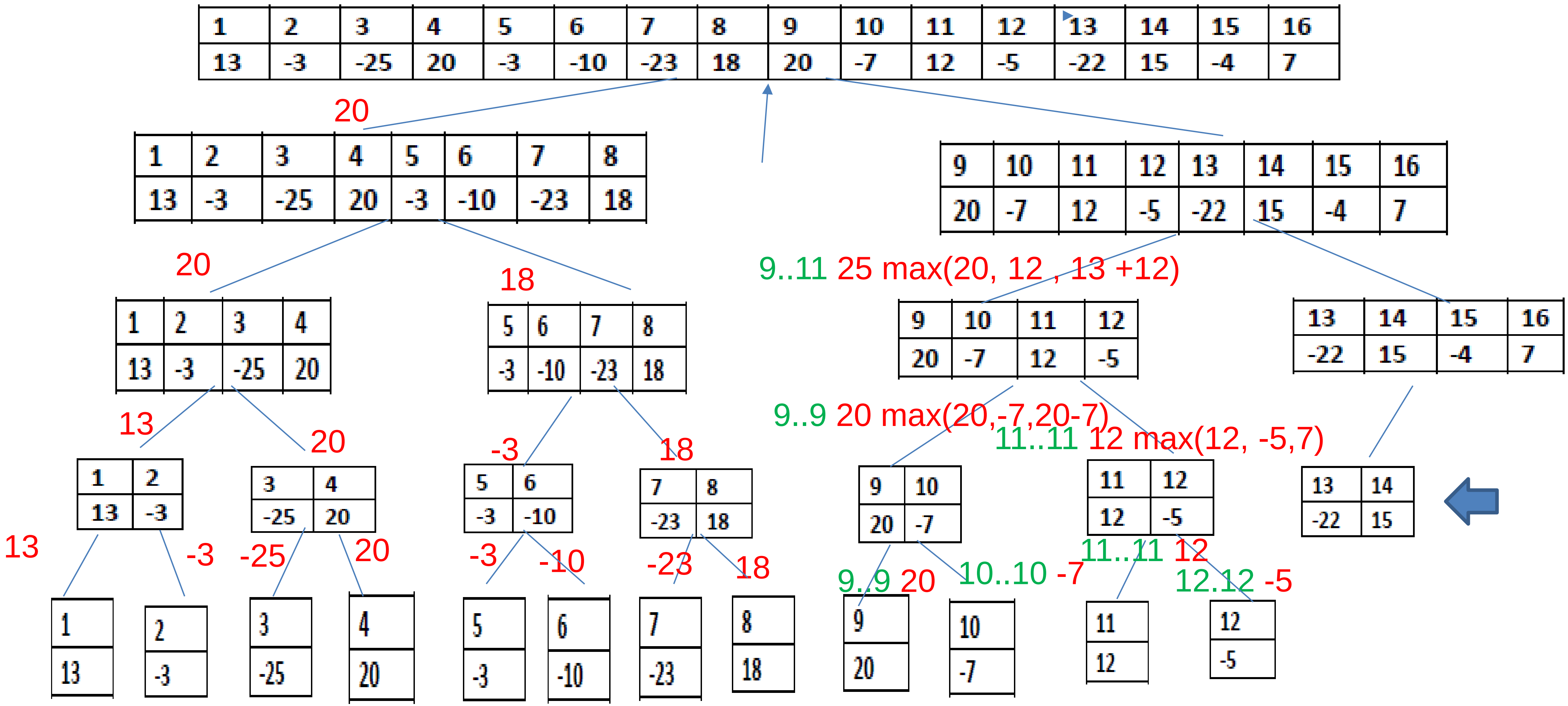
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
9..11 25 max(20, 12 , 13 +12)
20
18
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
13
20
-3
18
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5

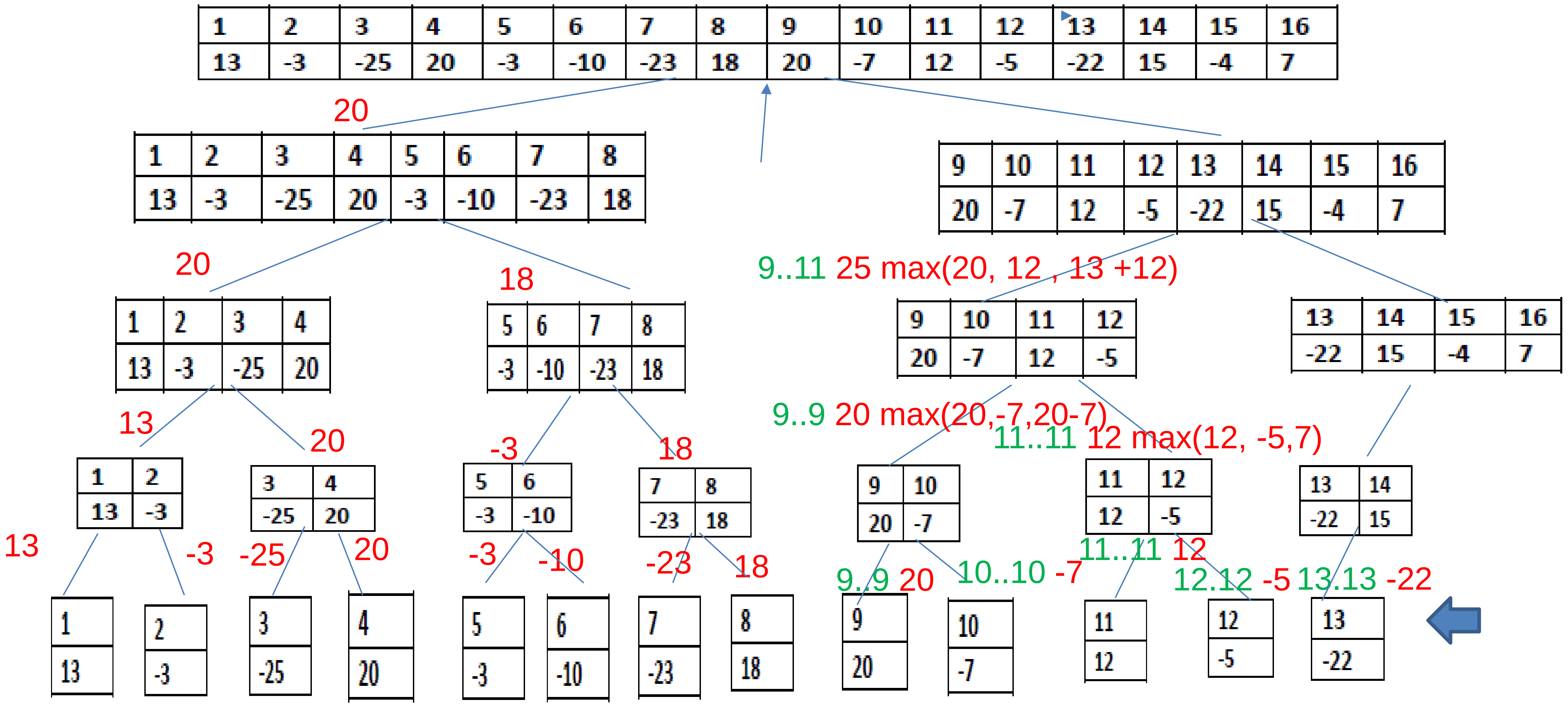


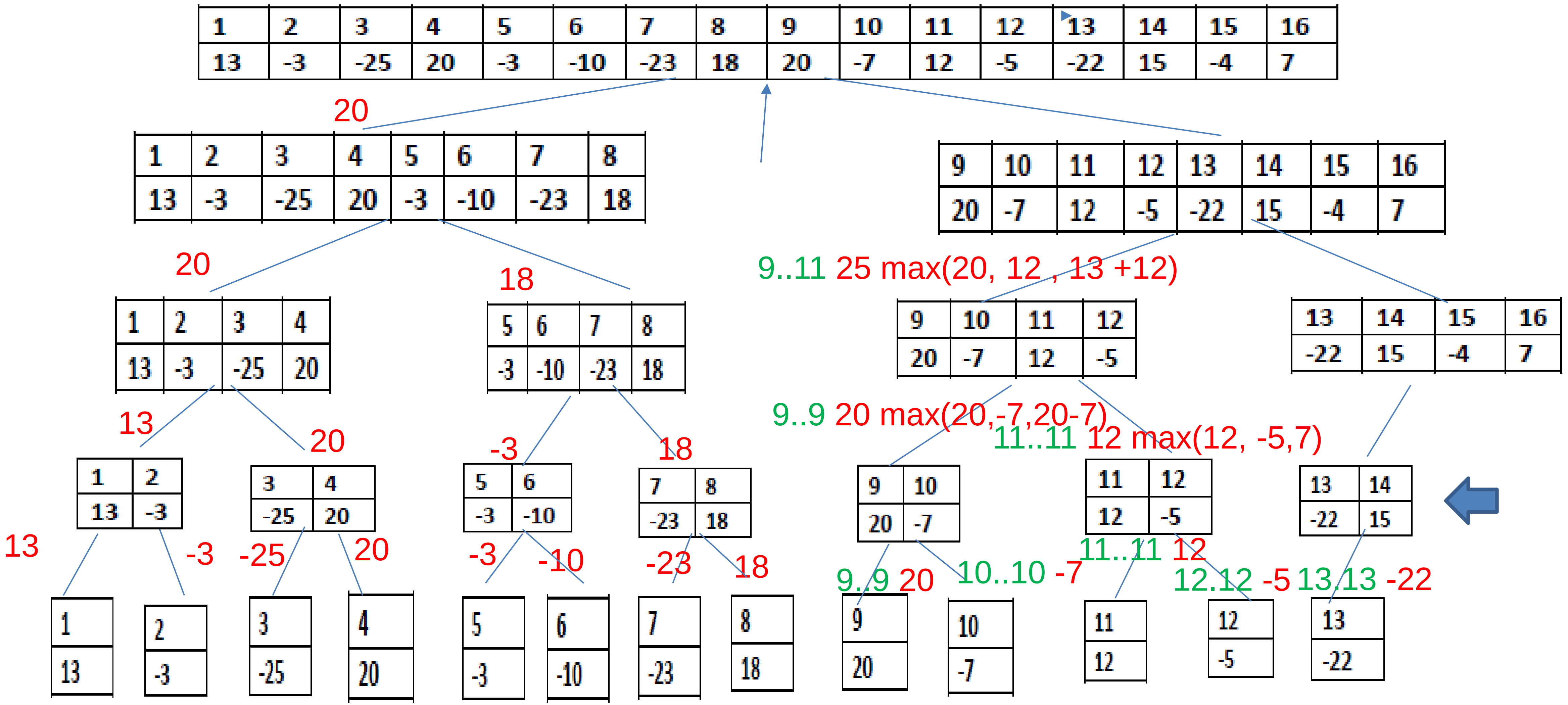
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
9..11 25 max(20, 12 , 13 +12)
20
18
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
13
20
-3
18
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5



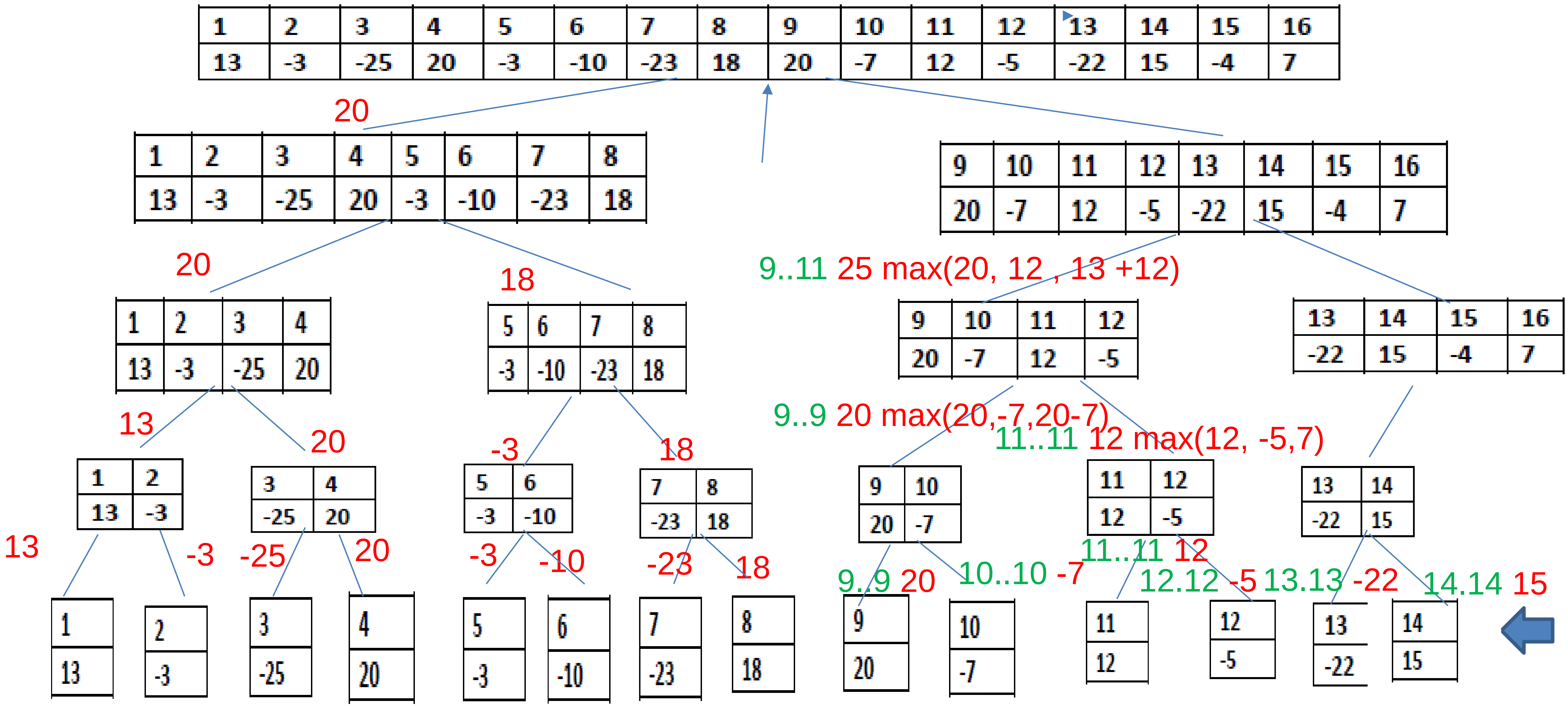
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
18
9..11 25 max(20, 12 , 13 +12)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
13
20
-3
18
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
1 2 3 4
13 -3 -25 20
18
5 6 7 8
-3 -10 -23 18
9..11 25 max(20, 12 , 13 +12)
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
13
20
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
-3
18
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13 14
-22 15
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5

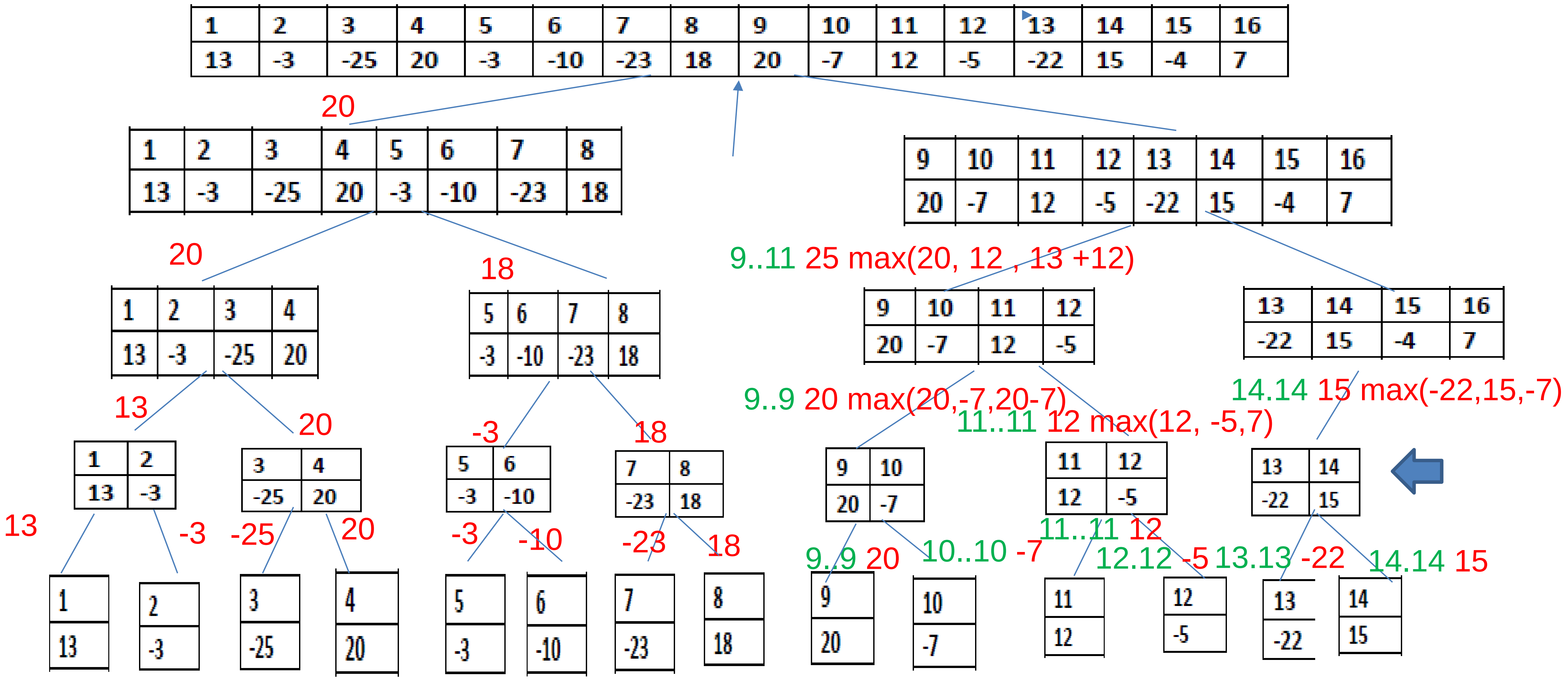

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
20
18
9..11 25 max(20, 12 , 13 +12)
13
20
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
-3
18
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22



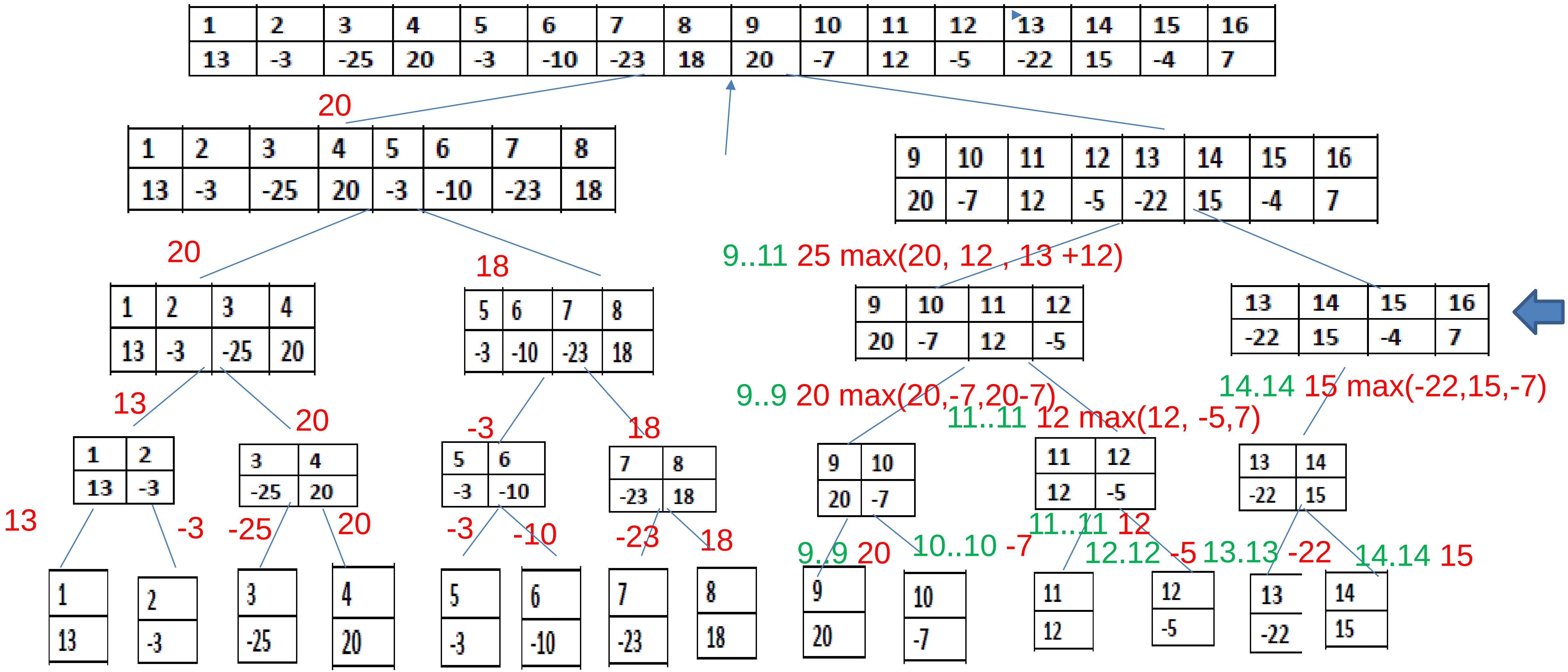
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
18
13
-3
9..11 25 max(20, 12 , 13 +12)
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
-25
-10
-23
9..9 20
10..10 -7
11..11 12
12.12 -5
13,13 -22



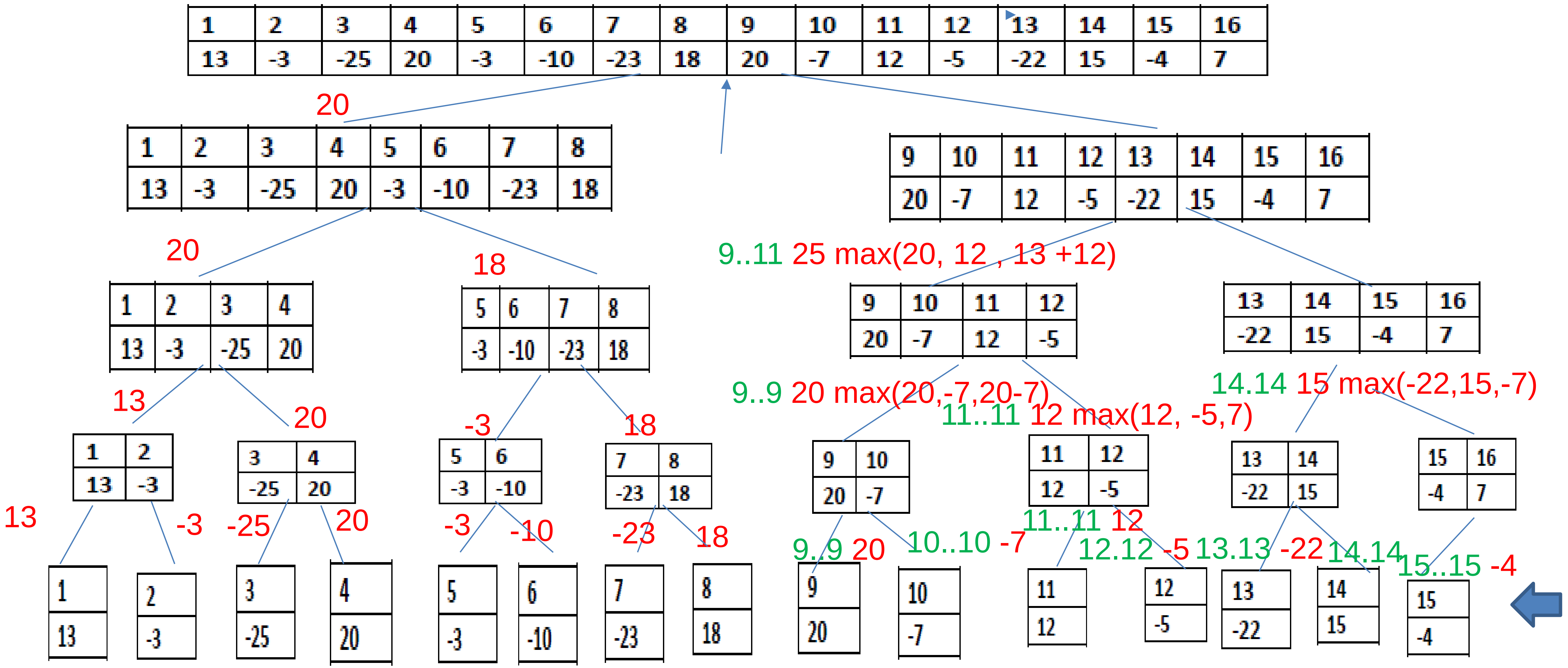
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
18
9..11 25 max(20, 12 , 13 +12)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
13
20
-3
18
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13 14
-22 15
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22
14.14 15
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5
13
-22
14
15



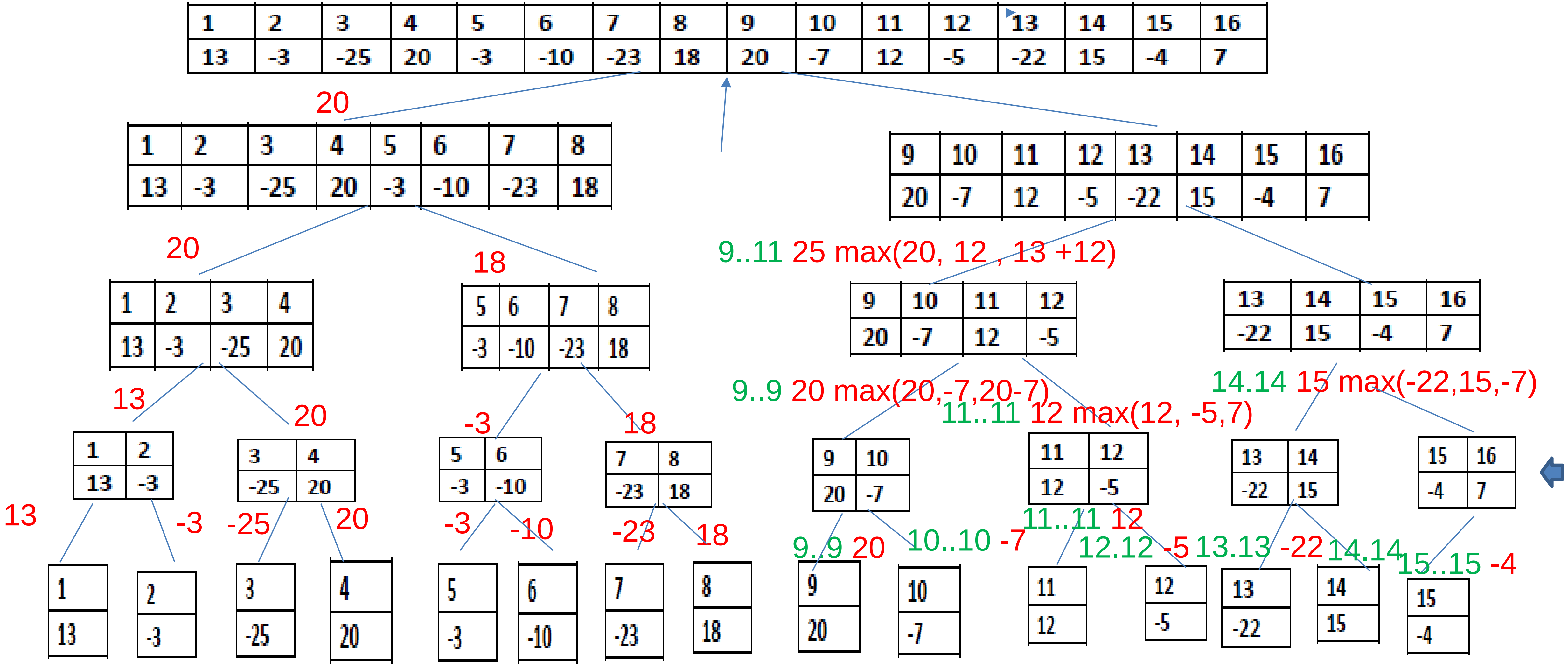
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
18
9..11 25 max(20, 12 , 13 +12)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
13
20
-3
18
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
14.14 15 max(-22,15,-7)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13 14
-22 15
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22
14.14 15
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5
13
-22
14
15



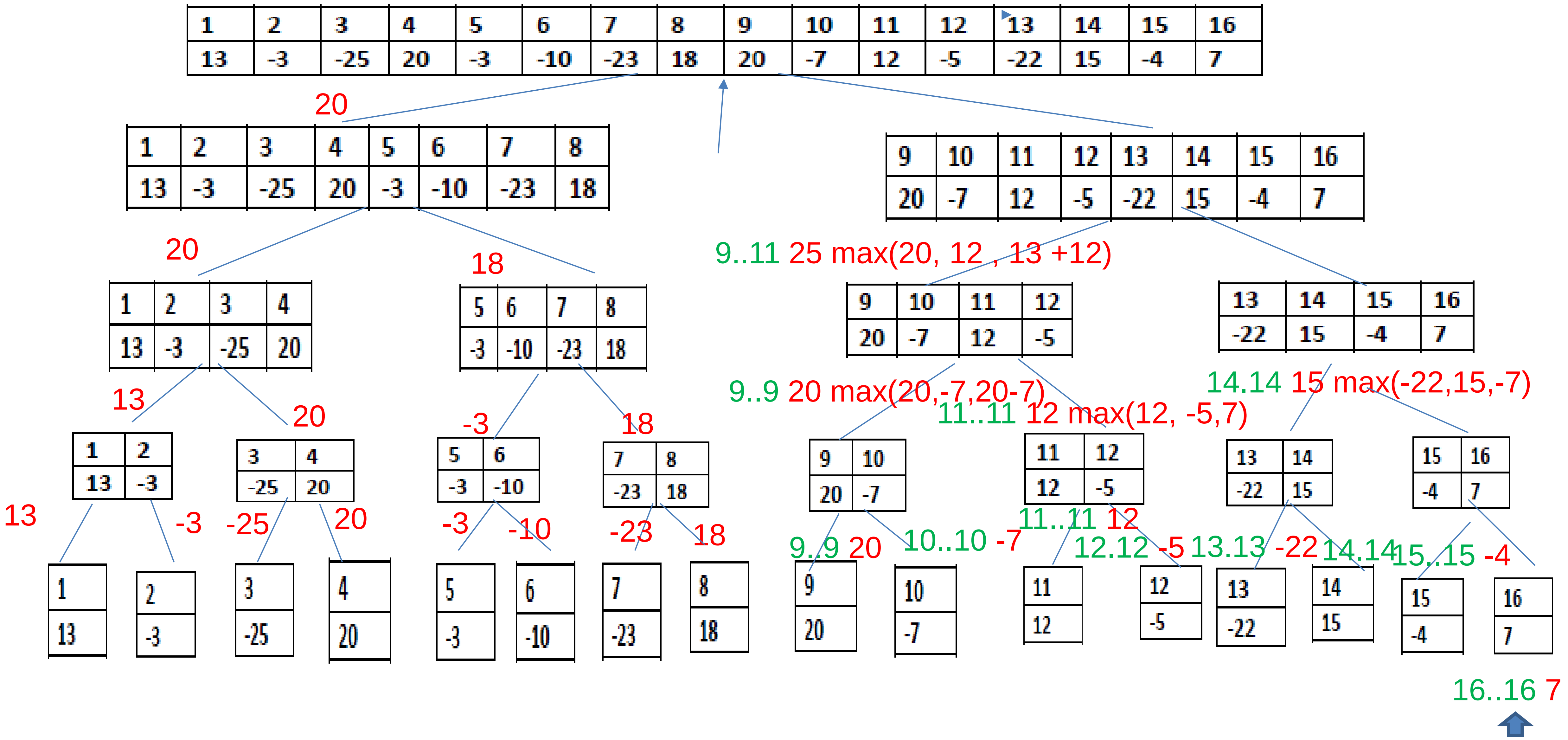
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
18
9..11 25 max(20, 12 , 13 +12)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
13
20
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
14.14 15 max(-22,15,-7)
-3
18
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13 14
-22 15
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22
14.14 15
1 13
2 -3
3 -25
4 20
5 -3
6 -10
7 -23
8 18
9 20
10 -7
11 12
12 -5
13 -22
14 15

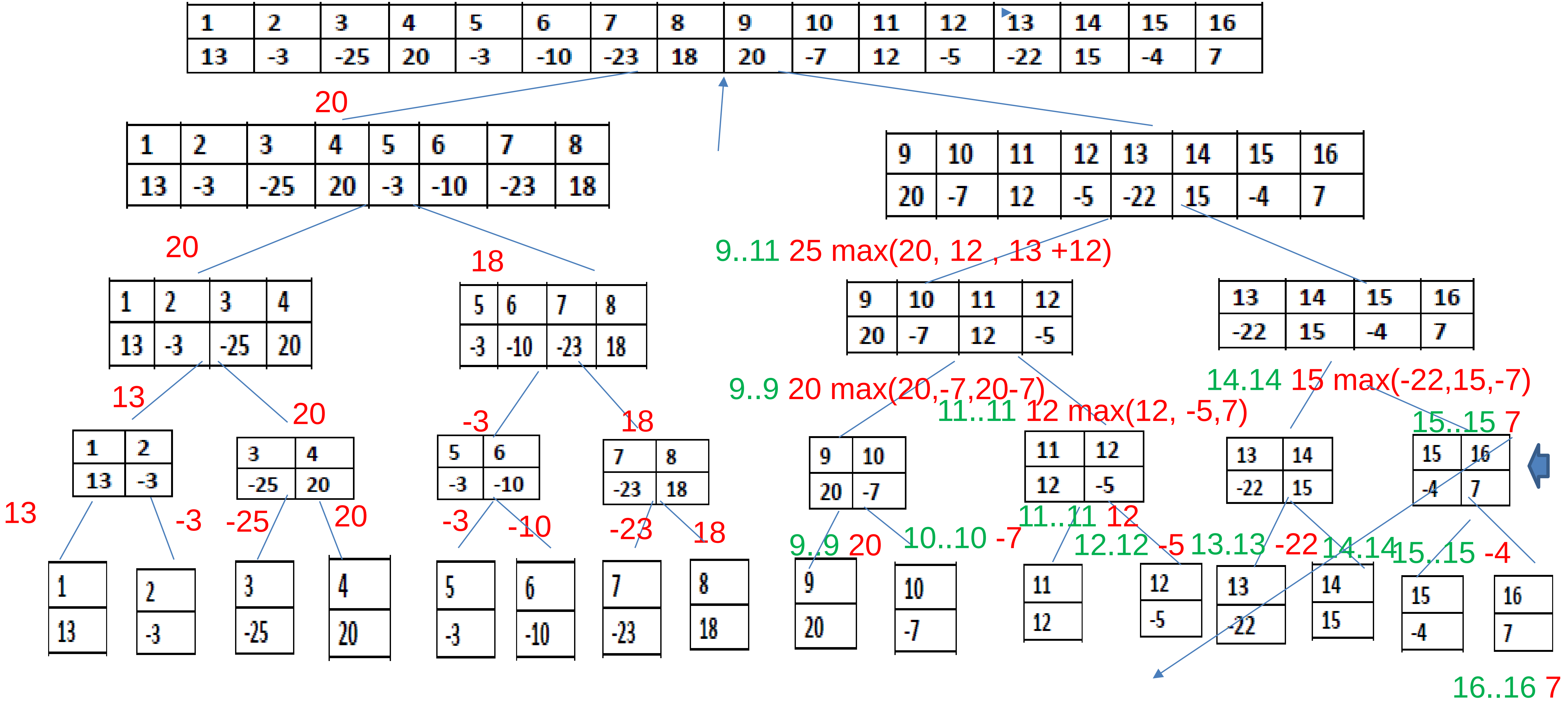


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
18
9..11 25 max(20, 12 , 13 +12)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
13
20
-3
18
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
14.14 15 max(-22,15,-7)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13 14
-22 15
15 16
-4 7
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22
14.14
15..15 -4
1 13
2 -3
3 -25
4 20
5 -3
6 -10
7 -23
8 18
9 20
10 -7
11 12
12 -5
13 -22
14 15
15 -4



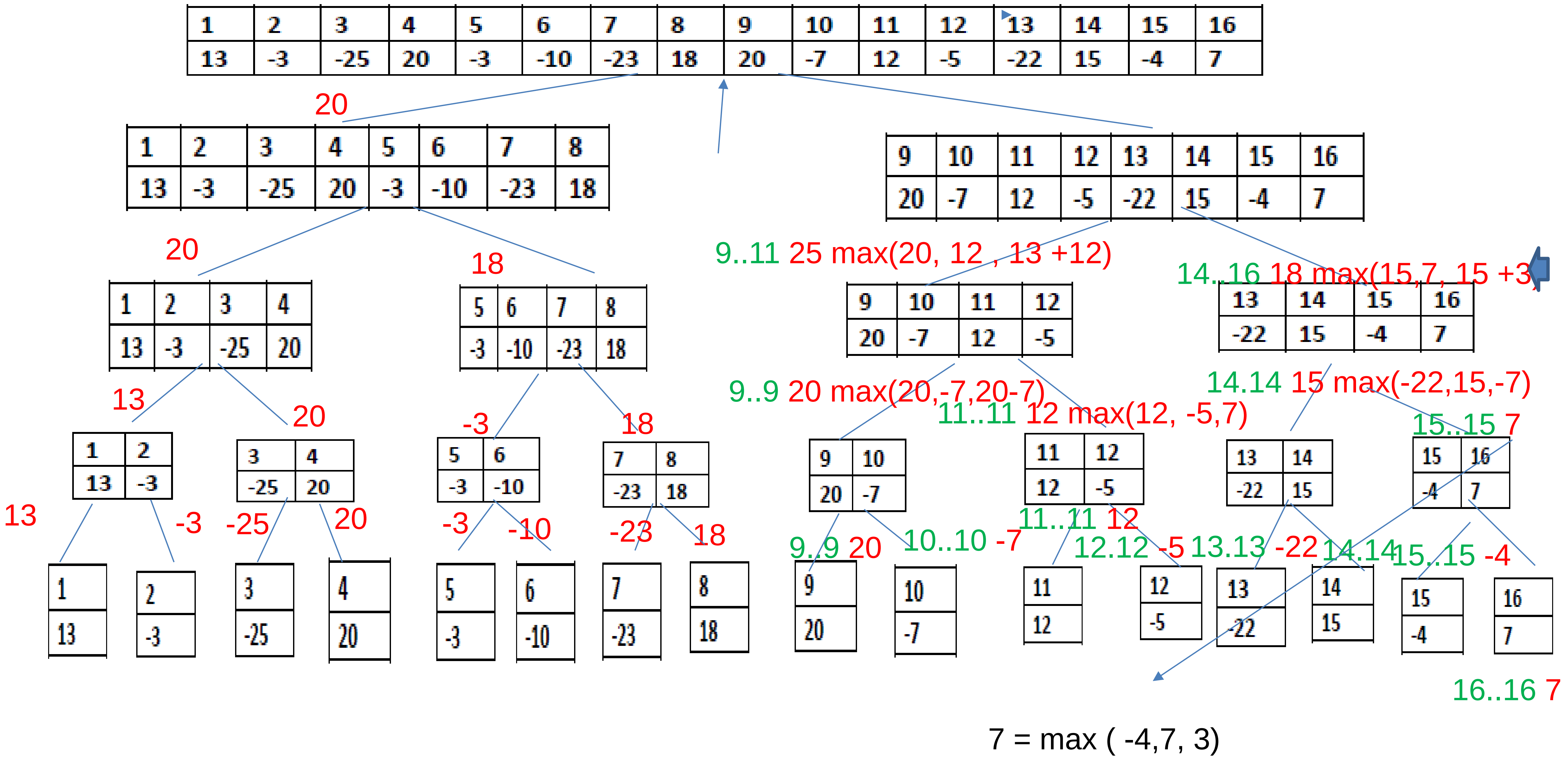
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
18
9..11 25 max(20, 12 , 13 +12)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
13
20
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
14.14 15 max(-22,15,-7)
-3
18
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13 14
-22 15
15 16
-4 7
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22
14.14
15..15 -4
1 13
2 -3
3 -25
4 20
5 -3
6 -10
7 -23
8 18
9 20
10 -7
11 12
12 -5
13 -22
14 15
15 -4


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
18
9..11 25 max(20, 12 , 13 +12)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
13
20
-3
18
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
14.14 15 max(-22,15,-7)
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13 14
-22 15
15 16
-4 7
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22
14.14
15..15 -4
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5
13
-22
14
15
15
-4
16
7
16..16 7

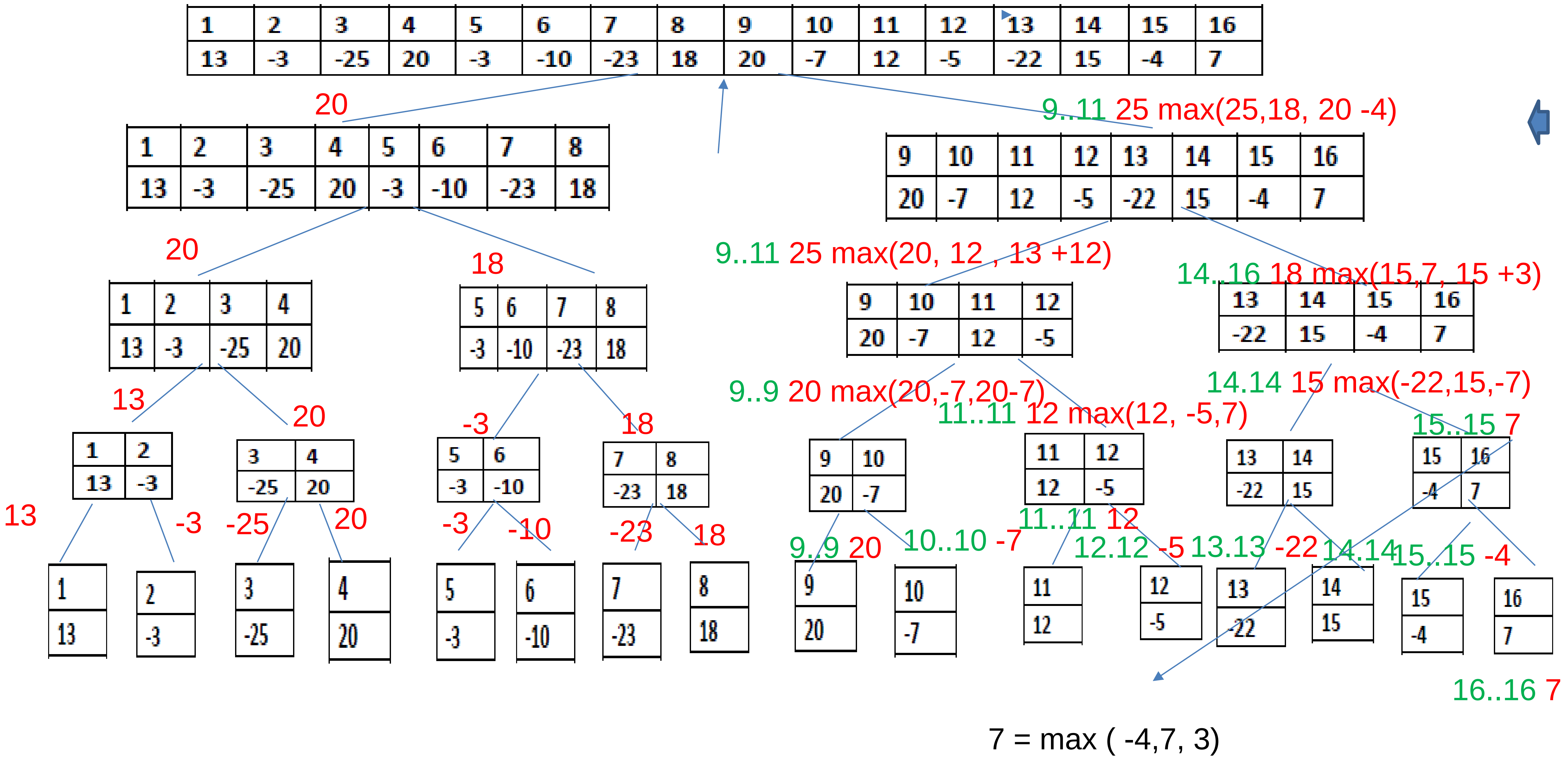


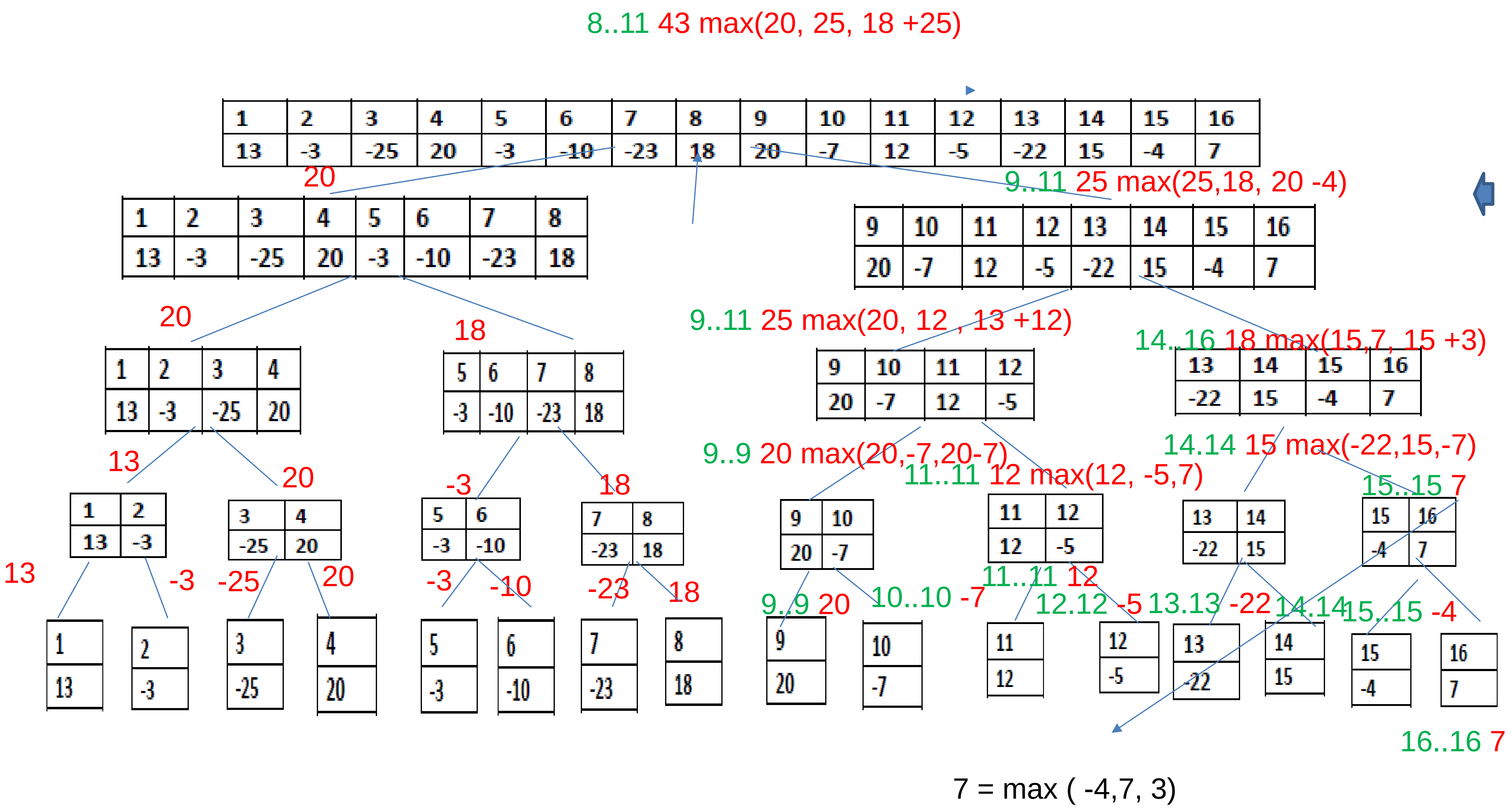
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
9..11 25 max(20, 12 , 13 +12)
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
14.14 15 max(-22,15,-7)
15..15 7
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22
14.14
15..15 -4
16..16 7

7 = max (-4,7, 3)



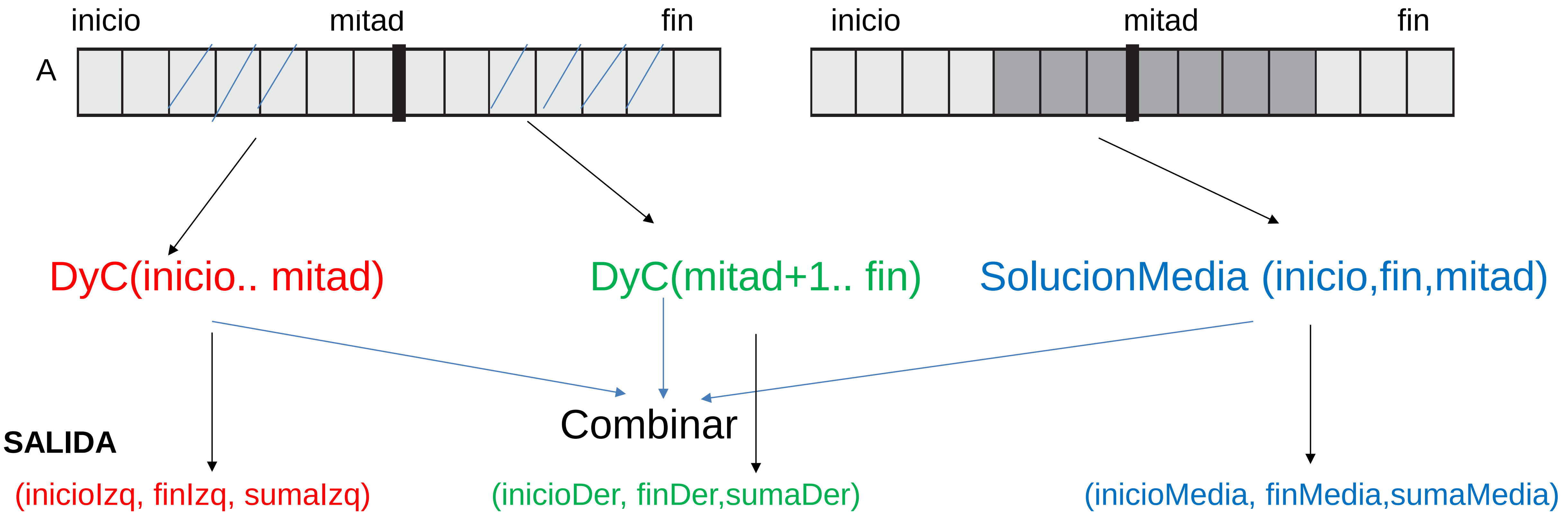
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
18
13
-3
-25
-10
-23
9..11 25 max(20, 12 , 13 +12)
14..16 18 max(15,7, 15 +3)
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
14.14 15 max(-22,15,-7)
15..15 7
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22
14.14
15..15 -4
16..16 7
7 = max (-4,7, 3)


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
9..11 25 max(25,18, 20 -4)
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
18
9..11 25 max(20, 12 , 13 +12)
14..16 18 max(15,7, 15 +3)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
13
20
-3
18
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
14.14 15 max(-22,15,-7)
15..15 7
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13 14
-22 15
15 16
-4 7
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22
14.14
15..15 -4
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5
13
-22
14
15
15
-4
16
7
16..16 7
7 = max (-4,7, 3)



8..11 43 max(20, 25, 18 +25)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
13 -3 -25 20 -3 -10 -23 18 20 -7 12 -5 -22 15 -4 7
20
9..11 25 max(25,18, 20 -4)
1 2 3 4 5 6 7 8
13 -3 -25 20 -3 -10 -23 18
9 10 11 12 13 14 15 16
20 -7 12 -5 -22 15 -4 7
20
18
9..11 25 max(20, 12 , 13 +12)
14..16 18 max(15,7, 15 +3)
1 2 3 4
13 -3 -25 20
5 6 7 8
-3 -10 -23 18
9 10 11 12
20 -7 12 -5
13 14 15 16
-22 15 -4 7
13
20
-3
18
9..9 20 max(20,-7,20-7)
11..11 12 max(12, -5,7)
14.14 15 max(-22,15,-7)
15..15 7
1 2
13 -3
3 4
-25 20
5 6
-3 -10
7 8
-23 18
9 10
20 -7
11 12
12 -5
13 14
-22 15
15 16
-4 7
13
-3
-25
20
-3
-10
-23
18
9..9 20
10..10 -7
11..11 12
12.12 -5
13.13 -22
14.14
15..15 -4
1
13
2
-3
3
-25
4
20
5
-3
6
-10
7
-23
8
18
9
20
10
-7
11
12
12
-5
13
-22
14
15
15
-4
16
7
16..16 7
7 = max (-4,7, 3)

Divide y Conquista.



Divide y conquista

Pseudo-código maximoSubarreglo
ENTRADA (A, inicio, fin)
SALIDA (inicioSol, finSol, sumaSol)
// **CASO BASE**

```
 if (inicio== fin)
    return (inicio, fin, A[inicio]);
```

// CÁLCULO DE SUBPROBLEMAS

```
 else
{ mitad= (inicio + fin) / 2;
```

// Subproblema- Parte1

```
(inicioIzq, finIzq,sumaIzq) =
maximoSubarreglo(A, inicio, mitad);
```

// Subproblema-Parte 2

```
(inicioDer, finDer,sumaDer) =
maximoSubarreglo(A,mitad+1, fin);
```

Divide y Conquista

```
//Solución media
(inicioMedia, finMedia, sumaMedia) =
                                  SolucionMedia( A, inicio, fin, mitad);
//Combinar soluciones
If ((sumaIzq > sumaDer ) and ( sumaIzq > sumaMedia))
                                  return (inicioIzq, finIzq, sumaIzq);
If ((sumaDer >= sumaIzq) and (sumaDer > = sumaMedia))
                                  return (inicioDer, finDer, sumaDer);
else return( inicioMedia, finMedia, sumaMedia);
```

Divide y Conquista.

Pseudo-código SolucionMedia
Entrada (A, inicio, mitad, fin)
Salida = (maxIzq, maxDer, sumaMedia)

```
sumaIzq= -∞;
suma = 0;
for (i=1; i <= mitad; i++)
{suma += A[mitad-i +1];
   if (suma > sumaIzq)
     { sumaIzq = suma;
        maxIzq = mitad –i +1;
     }
}
```

```
sumaDer =-∞;
suma = 0;
for (j = mitad + 1; j <= fin; j++)
{suma += A[j];
  if (suma > sumaDer)
     {sumaDer = suma;
      maxDer = j;
     }
}
sumaMedia = sumaIzq +
sumaDer
```

Divide y Conquista.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	-3	-25	20	-3	-10	-23	18	20	-7	12	-5	-22	15	-4	7

	sumaIzq	maxIzq
18	18	8
-23	-5	8
-10	-15	8
-3	-18	8
20	2	8
-25	-23	8
-3	-26	8
13	-13	8

	sumaDer	maxDer
20	20	9
-7	13	9
12	25	11
-5	20	11
-22	-2	11
15	13	11
-4	9	11
7	16	11

```
void SolucionMedia(int a[], unsigned int inicio,unsigned int mitad,unsigned  int fin,
                   unsigned int & iniciomedia, unsigned int & finmedia, int & cantmedia)
{

          int sumaizq =valor_inicial;
          int suma =0;
          for( unsigned int i =0; i<= mitad; i++)
              {suma += a[mitad-i];
               if ( suma > sumaizq)
                {sumaizq = suma;
                  iniciomedia= mitad -i;
                }
              }
           int sumader = valor_inicial;
           suma = 0;
          for (unsigned int j=mitad +1; j <= fin; j++)
               {suma += a[j];
               if (suma > sumader)
                {sumader = suma;
                finmedia =j;}
                }
               cantmedia = sumaizq + sumader;
}
```

```
void MayorSubarreglo(int a[], unsigned int inicio, unsigned int fin, unsigned int & iniciomax,
             unsigned  int & finmax, int & cant)
{
       unsigned int inicioizq ,finizq, inicioder, finder, iniciomedia,finmedia;
       int cantmedia, cantder,cantizq;

        if (inicio == fin)
            {iniciomax= inicio;
             finmax= fin;
             cant = a[inicio];
            }
        else
        {
        unsigned int mitad = (fin + inicio)/2;
        MayorSubarreglo(a, inicio , mitad, inicioizq, finizq,cantizq);
        MayorSubarreglo(a, mitad +1, fin, inicioder, finder,cantder);
        SolucionMedia(a,inicio,mitad,fin,iniciomedia,finmedia,cantmedia);
```

```
    if ((cantizq >= cantder) and (cantizq >= cantmedia))
    {
        iniciomax=inicioizq;
        finmax=finizq;
        cant = cantizq;
    }
    else
        if ((cantder >= cantizq) and (cantder >= cantmedia))
            {
                iniciomax=inicioder;
                finmax= finder;
                cant = cantder;
            }
         else

            {
               iniciomax=iniciomedia;
               finmax=finmedia;
               cant = cantmedia;
            }
      }
}
```

Divide y conquista

Complejidad temporal

$$T(n) \begin{cases} c3 & n=1 \\ 2\,T(\frac{n}{2}) + c1\,n + c2 & n>1 \end{cases}$$

Algoritmo “ingenuo” $O(n^2)$

Algoritmo por Divide y Conquista

$$T(2^k) \begin{cases} c3 & k=0 \\ 2\,T(2^{k-1}) + c1\,2^k + c2 & k>0 \end{cases}$$

$O(n \log n)$

Divide y conquista

La solución de un problema se obtiene combinando la solución de subproblemas idénticos al problema original

- No se resuelve el mismo problema más de una vez
- El tamaño de las entradas de las partes es una fracción del tamaño de la entrada del problema original
- Los problemas son balanceados
- Las funciones $Parte_1$, $Parte_2$,…. y *Combina* son “baratas”

Dividimos y conquistamos eficiencia en el uso del recurso tiempo

Divide y conquista

Analicemos el siguiente problema

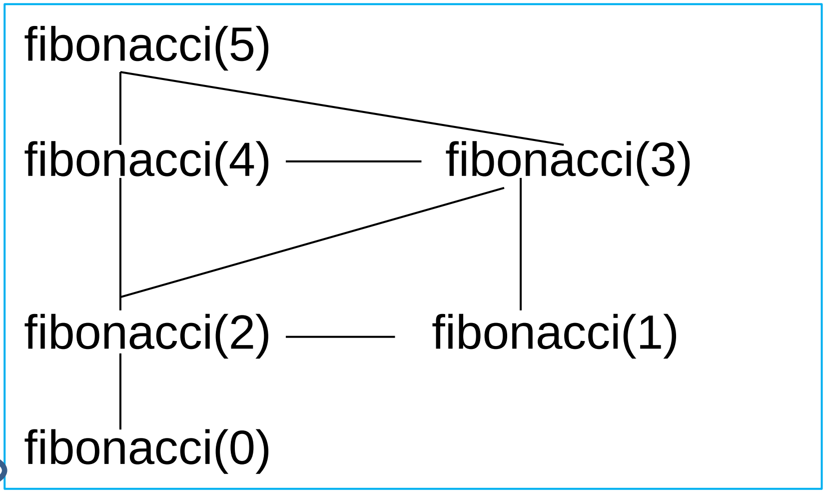
La sucesión de Fibonacci

0,1,1,2,3,5,8,13,21,34,55,89,144,233,377…

Puede definirse recursivamente

$$
\text{fibonacci}(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & n>1 \end{cases}
$$

```
unsigned int fibonacci(unsigned int n)
{
    if (n <=1) return n;
        else return (fibonacci(n-1) + fibonacci(n-2));
}
```



Un mismo subproblema es calculado más de una vez!!

Divide pero … no conquista

Divide y conquista

Analicemos el siguiente problema

La sucesión de Fibonacci

0,1,1,2,3,5,8,13,21,34,55,89,144,233,377…

Puede definirse recursivamente

$$\text{fibonacci}(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & n>1 \end{cases}$$

Un mismo subproblema es calculado más de una vez!!

Divide pero … no conquista nada

Complejidad temporal exponencial (Cormen, pág. 59-60)

```
unsigned int fibonacci(unsigned int n)
{
   if (n <=1) return n;
      else return (fibonacci(n-1) +
                         fibonacci(n-2));
}
```

O(n)

```
int fib ( int n)
{
   int i, f[max_valor];
   f[0] = 0;
   f[1] = 1;
   for (i=2; i<= n; i++)
     f[i]= f[i-1] + f[i-2];
   return f[n];
}
```

Resolver por Divide y Conquista
Dado un arreglo de enteros, encontrar la subsecuencia de mayor longitud de valores negativos.
Determinar la complejidad temporal

Técnicas de diseño de algoritmos

Greedy Algorithms

“Algoritmos voraces”

Técnicas de diseño de algoritmos: Greedy

Problema: "Programación de tareas"

Supongamos un servidor (por ejemplo, un procesador, un cajero de un banco, un expendedor de combustible) que tiene n clientes a quiénes proveerles un servicio.

Se conoce el tiempo requerido por cada cliente.

Sea t_i (1≤ i ≤n). Se requiere encontrar una secuencia de atención a clientes que minimice el tiempo total de espera.

tiempo total de espera =

$$\sum_{i=1}^{n}(\textit{tiempo del cliente i en el sistema})$$

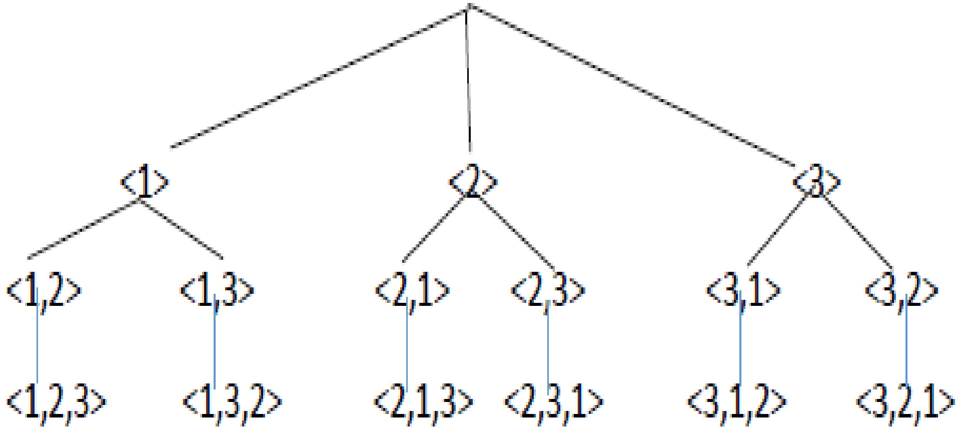
Técnicas de diseño de algoritmos: Greedy

Problema: “Programación de tareas”

Supongamos

cliente	1	2	3
tiempo	5	10	3

Posibles secuencias de atención



Si tenemos N clientes

N! secuencias de atención

3! Secuencias de atención

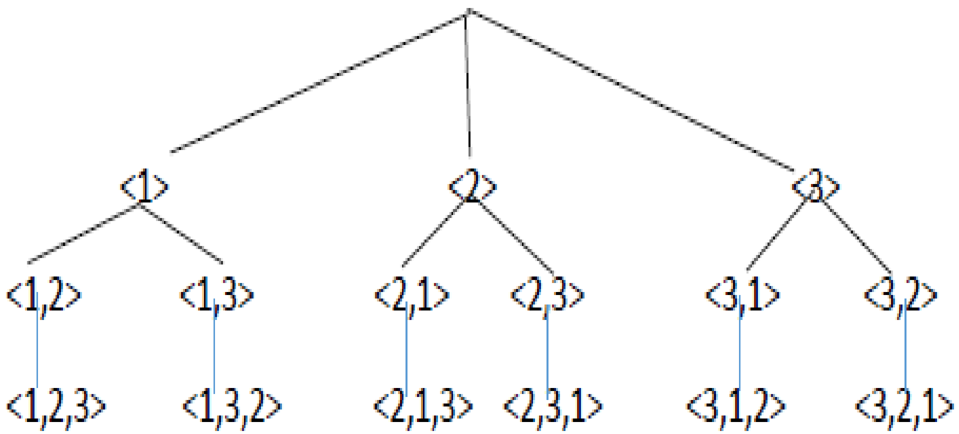
Técnicas de diseño de algoritmos: Greedy

Problema: “Programación de tareas”

Supongamos

cliente	1	2	3
tiempo	5	10	3

Posibles secuencias de atención



<1,2,3> 5 + (5 + 10) + (5 + 10 +3) = 38

<1.3.2> 5 + (5+3)+ (5+3+10) =31

<2,1,3> 10 + (10 + 5) + (10 + 5 +3) = 43

<2,3,1> 10 + (10 + 3) + (10 + 3 + 5) = 41

<3,1,2> 3 + (3 + 5) + (3 + 5 + 10) = 29

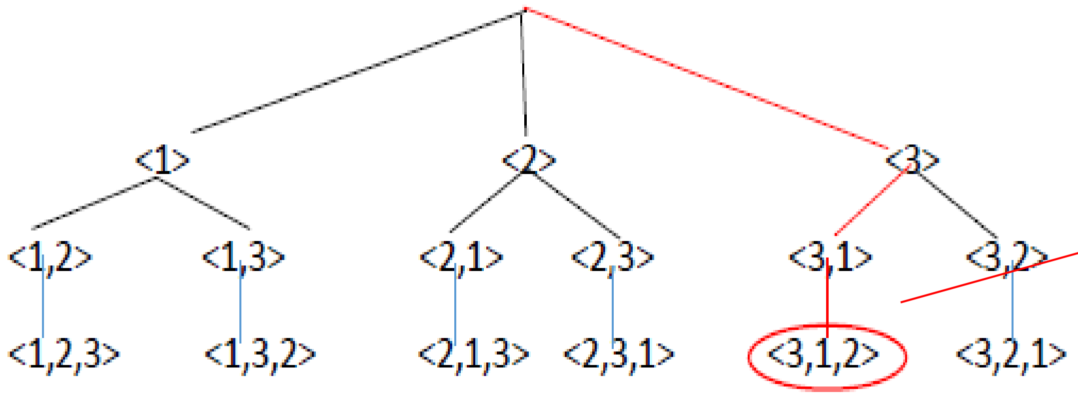
<3,2,1> 3 + (3+ 10) + (3 + 10 + 5) = 34

Técnicas de diseño de algoritmos: Greedy

Problema: “Programación de tareas”

cliente	1	2	3
tiempo	5	10	3

Posibles secuencias de atención



<1,2,3> 5 + (5 + 10) + (5 + 10 +3) = 38

<1,3,2> 5 + (5+3)+ (5+3+10) = 31

<2,1,3> 10 + (10 + 5) + (10 + 5 +3) = 43

<2,3,1> 10 + (10 + 3) + (10 + 3 + 5) = 41

<3,1,2> 3 + (3 + 5) + (3 + 5 + 10) = 29

<3,2,1> 3 + (3+ 10) + (3 + 10 + 5) = 34

La intuición nos dice que nos conviene ordenar ascendentemente a los clientes por su tiempo ti y en este caso funcionó!

. Ordenar a los clientes

. Seleccionar a los clientes en este orden y sumar sus tiempos de espera en el sistema

Técnicas de diseño de algoritmos: Greedy

Generalmente, los problemas que resolvemos por **greedy** se caracterizan por tener ***n entradas*** y se requiere encontrar un subconjunto de las entradas que satisface algunas restricciones.

Los subconjuntos de las entradas que satisfacen restricciones se denominan ***soluciones factibles.***

Se requiere encontrar una solución factible que maximice o minimice una ***función objetivo***.

La solución factible que maximiza o minimiza una función objetivo es la ***solución óptima***.

Usualmente es sencillo encontrar soluciones factibles por greedy y no siempre podemos obtener la solución óptima. Así distinguimos a ***Greedy ´óptimo*** cuando asociamos al algoritmo una prueba que garantiza que obtuvimos la solución óptima.

Generalmente obtener la solución requiere un ***pre-procesamiento de las entradas***

Técnicas de diseño de algoritmos: Greedy

```
tipoSolucion Greedy (Lista[tipoEntrada] S)
// S contiene a las n entradas preprocesadas
{ tipoSolucion solucion;
  solucion.inicializar();
  for (int i=1; i<=  S.longLista(), i++)
    { tipoEntrada x = S.select(i);
      If factible(solucion, x)
       solucion.agregar(x);
     }
  return solucion;
  }
```

Técnicas de diseño de algoritmos: Greedy

Problema de la mochila

Versión 1

Greedy
Problema de “la mochila”

Problema de la mochila

Se tienen n objetos y una mochila de capacidad M.

El objeto i tiene un peso p_i y su incorporación produce un beneficio b_i

Si se coloca una fracción $0 \leq xi \leq 1$ del objeto i se obtiene un beneficio $b_i\, x_i$

Encontrar una asignación de objetos a la mochila que maximice el beneficio total

$$\sum_{i=1}^{n} b_i\, x_i$$

Greedy
Problema de "la mochila"

Definición formal del problema:

Restricción $\sum_{i=1}^{n} x_i\, pi \leq M$ (1)

Función objetivo $\sum_{i=1}^{n} x_i\, bi$ **a maximizar** (2)

con $0 \leq x_i \leq 1$ **y** $1 \leq i \leq n$ (3)

Solución factible → conjunto ($x_1, \ldots, x_n$) que satisface (1) y (3)

Solución óptima es una solución factible para la cual (2) es máximo.

Greedy
Problema de "la mochila"

$n = 3 \quad M = 20$

$(b1,b2,b3) = (25,24,15)$

$(p1,p2,p3) = (18,15,10)$

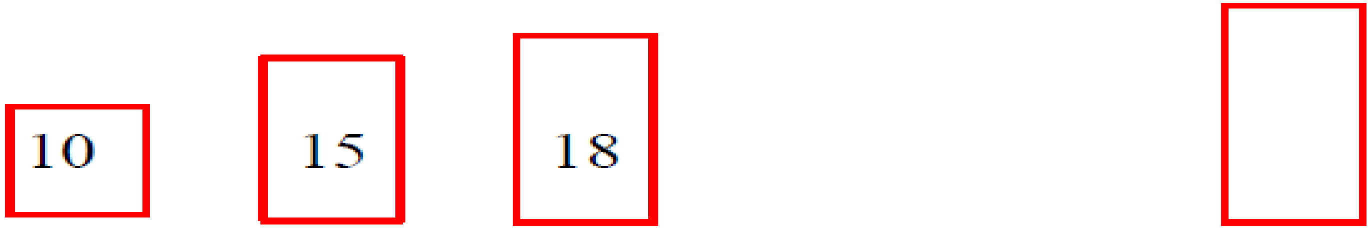
Algunas soluciones factibles

(x_1, x_2, x_3)	$\sum_{i=1}^{3} p_i x_i$	$\sum_{i=1}^{3} b_i x_i$
$(\frac{1}{2}, \frac{1}{3}, \frac{1}{4})$	**16,5** $(\frac{1}{2}\ x\ 18 + \frac{1}{3}\ x\ 15 + \frac{1}{4}\ x\ 10)$	**24,25** $\frac{1}{2}\ x\ 25 + \frac{1}{3}\ x\ 24 + \frac{1}{4}\ x\ 15$
$(1, \frac{2}{15}, 0)$	**20**	**28,2**
$(0, \frac{2}{3}, 1)$	**20**	**31**
$(0, 1, \frac{1}{2})$	**20**	**31,5**

Greedy
Problema de “la mochila”

Ejemplo

Una instancia del problema de la mochila:



$b = 15$ $b = 24$ $b = 25$ $M = 20$

¿Con qué criterio seleccionamos los objetos que colocaremos en la mochila?

Greedy
Problema de "la mochila"

Ejemplo: $M = 20$

$p_1=10$, $p_2=15$, $p_3=18$

$b_1=15$, $b_2=24$, $b_3=25$

1° criterio: Elegimos primero los objetos más valiosos

2 / 15	**2 / 15 * 24 = 3,2**
18	**25**

28,2

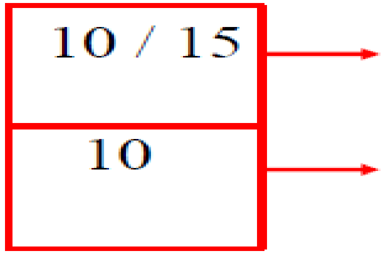
Greedy
Problema de “la mochila”

Ejemplo: $M = 20$

$p_1=10, \quad p_2=15, \quad p_3=18$

$b_1=15, \quad b_2=24, \quad b_3=25$

2° criterio: Elegimos primero los objetos menos pesados para tratar de llenar la mochila lo más tarde posible.



$10/15 * 24 = 16$

15

31

Greedy
Problema de “la mochila”

Ejemplo: $M = 20$

$p_1=10$, $p_2=15$, $p_3=18$
$b_1=15$, $b_2=24$, $b_3=25$

3° criterio: Elegimos primero aquel objeto con mayor ganancia por unidad de peso.

b_i/ p_i	1.5	1.6	1.3

5 / 10	$5/10 * 15 = 7,5$
15	24

= 31,5 ⇒ Solución óptima

Greedy
Problema de “la mochila”

Problema de la mochila

Se tienen n objetos y una mochila de capacidad M.

El objeto i tiene un peso p_i y su incorporación produce un beneficio b_i

Si se coloca una fracción $0 \leq xi \leq 1$ del objeto i se obtiene un beneficio $b_i\ x_i$

Greedy
Problema de “la mochila”

Problema de la mochila

$p_i > 0$, $b_i >$o, $0 \leq x_i \leq 1$ ($\forall\, i: 1 \leq i \leq n$), n>0 (restricciones explícitas)

Función objetivo maximizar $\sum_{i=1}^{n} b_i x_i$

sujeto a

$\sum_{i=1}^{n} p_i x_i \leq$ M (restricciones implícitas)

Greedy
Problema de “la mochila”

Pseudo-código del algoritmo Greedy

Ordenar descendentemente a los elementos por relación beneficio/peso **O(n log n)**

```
for ( i=1; i<=n: i++)
   x[i]=0;
cap= M;
j=1;
while (j <=n && p[j] < cap)
   { x[j] =1;
      cap= cap – p[j];
      j++;
    }
If (j< n)
  x[i] = cap/ p[j];
```

O(n)

Greedy
Problema de “la mochila”

// ordenar descendentemente por relación beneficio/peso

```
for ( i=1; i<=n: i++)
   x[i]=0;
cap= M;
j=1;
while (j <=n && p[j] < cap)
   { x[j] =1;
      cap= cap – p[j];
      j++;
    }
If (j< n)
  x[i] = cap/ p[j];
```

ENTRADA

$(b_1,b_2,b_3) = (25, 24, 15)$

$(p_1,p_2,p_3) = (18, 15, 10)$

$M = 20$

$\frac{b1}{p1} = 1.38 \quad \frac{b2}{p2} = 1.6 \quad \frac{b3}{p3} = 1.5$

ORDENAR

$(b'_1,b'_2,b'_3) = (24, 15, 25)$

$(p'_1,p'_2,p'_3) = (15, 10, 18)$

SALIDA (respecto a la ENTRADA)

$X = (0, 1, ½)$

$\sum_{i=1}^{3} b_i \; x_i = 0 + 1 \times 24 + \frac{1}{2} \times 15 = 31.5$

Bibliografía

Horowitz, E.; Sahni, S. ; Rajasekaran, S. Computer Algorithms C++

Chapter 4. pág. 195 (Knapsack Problem)

Ordenamiento secuencial de tareas con plazos

GREEDY
"Ordenamiento secuencial de tareas con plazos"

Se tienen que realizar n tareas. Cada una debe ser procesada en una máquina en una unidad de tiempo. Por cada tarea se obtiene una ganancia g_i $(1 \leq i \leq n)$ si es completada dentro de su plazo d_i $(1 \leq i \leq n)$. Un subconjunto de tareas J que puede completarse en sus plazos es una solución factible. El valor de la solución factible es la suma de las ganancias de las tareas en J:

$$\sum_{i \in J} g_i$$

Encontrar la solución óptima.

GREEDY
“Ordenamiento secuencial de tareas con plazos”

Ejemplo

$n = 4$; $(g_1,g_2,g_3,g_4) = (100,10,15,27)$; $(d_1,d_2,d_3,d_4) = (2,1,2,1)$

Las soluciones factibles y sus valores son:

Solución	Solución factible	Secuencia de procesamiento	valor
1	(1,2)	<2,1>	110
2	(1,3)	<1,3> o <3,1>	115
3	(1,4)	<4,1>	127
4	(2,3)	<2,3>	25
5	(3,4)	<4,3>	42
6	(1)	<1>	100
7	(2)	<2>	10
8	(3)	<3>	15
9	(4)	<4>	27

Solución óptima

GREEDY
“Ordenamiento secuencial de tareas con plazos”

Función objetivo: $\sum_{i \in J} g_i$

Se requiere maximizar la función objetivo

Estrategia greedy

en cada paso incluir la tarea que incremente la ganancia sujeto a la restricción que J sea factible. Esto requiere considerar a las tareas ordenadas descendentemente por ganancias.

GREEDY
“Ordenamiento secuencial de tareas con plazos”

Aplicación de la estrategia

$n= 4$; $(g_1,g_2,g_3,g_4) =(100,10,15,27)$; $(d_1,d_2,d_3,d_4) = (2,1,2,1)$

Ordenamos tareas por ganancias. Sea i_1,i_2,i_3,i_4 una permutación de las 4 tareas tal que $g_{i1} \geq g_{i2} \geq g_{i3} \geq g_{i4}$. En nuestro ejemplo, $i_1=1$, $i_2=4$, $i_3=3$, $i_4=2$.

Sea J un conjunto de tareas factibles (inicialmente J=Ø)

La tarea i_1 se agrega a J y J={1} es una solución factible. Luego es considerada la tarea i_2 y J={1,4} es solución factible bajo el ordenamiento <4,1>.

Luego es considerada la tarea i_3 y dado que J={1,4,3} no es factible es descartada. Finalmente es considerada la tarea i_4 y dado que J={1,4,2} no es factible es descartada.

La solución óptima es {1,4} bajo el procesamiento <4,1>

¿Cómo determinamos que una solución es factible?

GREEDY
“Ordenamiento secuencial de tareas con plazos”

Ejemplo

$n = 4$; $(g_1,g_2,g_3,g_4) = (100,10,15,27)$; $(d_1,d_2,d_3,d_4) = (2,1,2,1)$

Las soluciones factibles y sus valores son:

Solución	Solución factible	Secuencia de procesamiento	valor
1	(1,2)	<2,1>	110
2	(1,3)	<1,3> o <3,1>	115
3	(1,4)	<4,1>	127
4	(2,3)	<2,3>	25
5	(3,4)	<4,3>	42
6	(1)	<1>	100
7	(2)	<2>	10
8	(3)	<3>	15
9	(4)	<4>	27

Solución óptima

GREEDY
“Ordenamiento secuencial de tareas con plazos”

Aplicación de la estrategia

Supongamos que las ganancias fueron ordenadas previamente en forma decreciente con un costo O(n logn) y que los plazos corresponden a las ganancias ordenadas

n= 5; $(g_1,g_2,g_3,g_4,\ g_5)$ =(20,15,10,,5,1); $(d_1,d_2,d_3,d_4,\ d_5)$ = (2,2,1,3,3)

¿Cómo determinamos que una solución es factible?

J={1}   <1>

J= {1,2} <1,2>

J= {1,2,3} NO ES FACTIBLE

J = {1,2,4} <1,2,4>

J = { 1,2,4,5} NO ES FACTIBLE

SOLUCIÓN ÓPTIMA <1,2,4>

<1   2   4>

$d_1 \geq 1$

$d_2 \geq 2$

$d_4 \geq 3$

GREEDY
“Ordenamiento secuencial de tareas con plazos”

<u>Aplicación de la estrategia</u>

Supongamos que las ganancias fueron ordenadas previamente en forma decreciente con un costo O(n log n) y que los plazos corresponden a las ganancias ordenadas

n= 5; (g_1,g_2,g_3,g_4, g_5) =(20,15,10,,5,1); (d_1,d_2,d_3,d_4, d_5) = (5,4,3,2,1)

¿Cómo determinamos que una solución es factible?

J={1}	**<1>**	
J= {1,2}	**<1,2>**	**<2,1>**
J= {1,2,3}	**<2,1,3>**	**<3,2,1>**
J = {1,2,3, 4}	**<3,2,1,4>**	**< 4,3,2,1>**
J = { 1,2,3, 4,5}	**<4,3,2,1,5>**	**<5,4,3,2,1>**

SOLUCIÓN ÓPTIMA <5,4,3,2,1> ganancia 51

GREEDY
“Ordenamiento secuencial de tareas con plazos”

Sea J un conjunto de k tareas y σ = $i_1, i_2, \ldots, i_k$ una permutación de tareas en J si las tareas que pertenecen a J pueden ser procesadas en el orden σ´sin violar ningún plazo, J es una solución factible. Por lo tanto, sólo tenemos que mostrar que si J es factible, si existe un

σ´=r_1,r_2 , …,r_k tal que dr_q ≥ q, 1≤q≤k.

Prácticamente, esto lo realizamos paso a paso reacomodando el orden de ejecución de las tareas

$<i_1 \quad i_2 \ldots \ i_k>$

$dr_1 \geq 1$

$dr_2 \geq 2$

…

$dr_q \geq q$

GREEDY
“Ordenamiento secuencial de tareas con plazos”

A continuación mostramos una posible implementación en $O(n^2)$ para la función Factible. Luego esta función es más costosa que el preprocesamiento $O(n \log n)$ y el algoritmo para obtener la solución óptima es $O(n^2)$.

Usando otras representaciones basadas en el TDA Union-Find puede lograrse una implementación en $O(n)$ para Factible y una implementación del algoritmo Greedy para obtener la solución óptima de $O(n \log n)$ (lo veremos en Algoritmos 2!)

GREEDY
“Ordenamiento secuencial de tareas con plazos”

```
int Factible  (int d[], int J[], int n)
   {  d[0]=0;
      J[0]=0;
      J[1]=1;
      int k=1;
      for (int i=2; i<=n;i++)
      {  int r=k;
```

Factible retorna la cantidad de elementos (k) de la solución factible y la solución en los elementos J [1..k]

```
while( (d[J[r]] > d[i]) && (d[J[r]] != r))
            r--;
   if ((d[J[r]] <= d[i]) &&(d[i] > r))
    {for (int q = k; q >= (r+1); q--)
         J[q+1] = J[q];
       J[r+1] = i;
       k++;
     }
   }
   return k;
  }
```

n= 5; (g_1,g_2,g_3,g_4, g_5) =(20,15,10,,5,1);

(d_1,d_2,d_3,d_4, d_5) = (5,4,3,2,1)

K=2

<2 1>

K=3

<3 2 1>

k=4

<4 3 2 1>

K=5

<5 4 3 2 1>

GREEDY
“Ordenamiento secuencial de tareas con plazos”

```
int Factible  (int d[], int J[], int n)
//  retorna la cantidad de elementos (k) de la solución //factible
y la solución en los k primeros elementos J
  {  d[0]=0;        J[0]=0;
     J[1]=1;
     int k=1;
     for (int i=2; i<=n;i++)
     {  int r=k;
        while( (d[J[r]] > d[i]) && (d[J[r]] != r))
           r--;
        if ((d[J[r]] <= d[i]) &&(d[i] > r))
        {  for (int q = k; q >= (r+1); q--)
              J[q+1] = J[q];
           J[r+1] = i;
           k++;
        }
     }
     return k;
  }
```

$O(n^2)$

```
int main()
{//test simple de Factible
//ganancias ordenadas descendentemente
int ganancia[8]={0, 13,12,10,8,6,4,1};
//Plazos correspondientes
 int deadline[8]={0,2,1,3,2,1,1,4};
 int  tareas [8];
 int n=7;
 int  cant_elem =  Factible(deadline,tareas, n);
 int ganancia_total= 0;
 for (int m =1; m<=cant_elem; m++)
  {
    ganancia_total = ganancia_total + ganancia[tareas[m]];
    cout << "tarea "<< m << "="<<tareas[m]<<" "<< endl;
   }
    cout<< "la ganancia total  es"<<ganancia_total<<endl;
  return 0;
}
```

¿ Qué imprime? ¿Es la solución óptima?

Bibliografía

Horowitz, E.; Sahni, S. ; Rajasekaran, S. Computer Algorithms C++

Chapter 4. pág. 195 (Job Sequencing with Deadlines)

Técnicas de diseño de algoritmos
Programación Dinámica

Programación dinámica

Programación dinámica

Permite resolver problemas a partir de la combinación de la solución de subproblemas idénticos al original y que, en general, tienen una sub-estructura óptima

se pueden usar soluciones óptimas de subproblemas para encontrar la solución óptima del problema en su conjunto

Programación dinámica

- Se aplica cuando los subproblemas *no son independientes entre sí*, es decir se usa un mismo subproblema para resolver diferentes problemas mayores.
- La técnica "Divide y conquista" no es conveniente para resolver estos problemas dado que los problemas no son independientes y un mismo problema sería calculado más de una vez.
- Se resuelven subproblemas desde las instancias más pequeñas hasta la mayor instancia.
- Los subproblemas se resuelven combinando las soluciones óptimas de las instancias pequeñas ya calculadas.
- Los resultados parciales son *almacenados* para ser reusados sin tener que recalcularlos.

Programación dinámica

Los problemas que tienen una subestructura óptima pueden resolverse a partir de 3 pasos generales:

•Dividir el problema en subproblemas más pequeños.

•Encontrar la solución óptima estos problemas

•Usar estas soluciones óptimas para construir una solución óptima al problema original

<u>Programación Dinámica</u>
Multiplicación encadenada de matrices

Programación dinámica
Multiplicación encadenada de matrices

Sean n matrices M_1, M_2, M_3,…M_n , calcular eficientemente

$$M_1 \times M_2 \times M_3 \times \ldots \times M_n$$

siendo cada M_i de dimensión $d_{i-1} \times d_i$ $(1 \leq i \leq n)$

La multiplicación es asociativa

Ejemplo

Sean

	dimensión
M_1	200 x 2
M_2	2 x 30
M_3	30 x 20
M_4	20 x 20

(((M1 x M2) x M3) x M4)

(M1 x ((M2 x M3) x M4))

((M1 x M2) x (M3 x M4))

Programación dinámica
Multiplicación encadenada de matrices

Sean n matrices M_1, M_2, M_3,…M_n , calcular eficientemente

$$\mathbf{M_1 \times M_2 \times M_3 \times \ldots \times M_n}$$

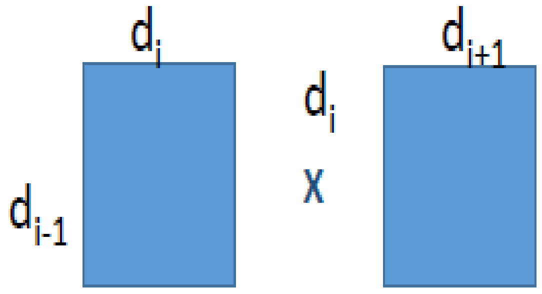
siendo cada M_i de dimensión $d_{i-1} \times d_i$ $(1 \leq i \leq n)$

Ejemplo

Sean

	dimensión
M_1	200 x 2
M_2	2 x 30
M_3	30 x 20
M_4	20 x 20

	dimensión
M_1	$d_0 \times d_1$
M_2	$d_1 \times d_2$
M_3	$d_2 \times d_3$
M_4	$d_3 \times d_4$



productos = $d_{i-1} \times d_i \times d_{i+1}$

Programación dinámica
Multiplicación encadenada de matrices

Ejemplo

$(((M_1 \times M_2) \times M_3) \times M_4)$

Sean

	dimensión
M_1	200 x 2
M_2	2 x 30
M_3	30 x 20
M_4	20 x 20

	#productos
$(M_1 \times M_2)$	200 x 2 x 30
$((M_1 \times M_2) \times M_3)$	200 x 30 x 20
$(((M_1 \times M_2) \times M_3) \times M_4)$	200 x 20 x20

(12000 +120000 + 80000)

212000 productos

Programación dinámica
Multiplicación encadenada de matrices

Ejemplo

$((M_1 \times M_2) \times (M_3 \times M_4))$

Sean

	dimensión
M_1	200×2
M_2	2×30
M_3	30×20
M_4	20×20

	#productos
$(M_1 \times M_2)$	$200 \times 2 \times 30$
$(M_3 \times M_4)$	$30 \times 20 \times 20$
$((M_1 \times M_2) \times (M_3 \times M_4))$	$200 \times 30 \times 20$

$(12000 + 12000 + 120000)$

144000 productos

Programación dinámica
Multiplicación encadenada de matrices

Ejemplo

$(M_1 \times ((M_2 \times M_3) \times M_4))$

Sean

	dimensión
M_1	200 x 2
M_2	2 x 30
M_3	30 x 20
M_4	20 x 20

	#productos
$(M_2 \times M_3)$	2 x 30 x 20
$((M_2 \times M_3) \times M_4)$	2 x 20 x 20
$(M_1 \times ((M_2 \times M_3) \times M_4))$	200 x 2 x 20

(1200 + 800 + 8000)

10000 productos

Programación dinámica
Multiplicación encadenada de matrices

Ejemplo

Sean

	dimensión
M_1	200×2
M_2	2×30
M_3	30×20
M_4	20×20

$(((M_1 \times M_2) \times M_3) \times M_4)$

212000 productos

$(M_1 \times ((M_2 \times M_3) \times M_4))$

10000 productos

$((M_1 \times M_2) \times (M_3 \times M_4))$

144000 productos

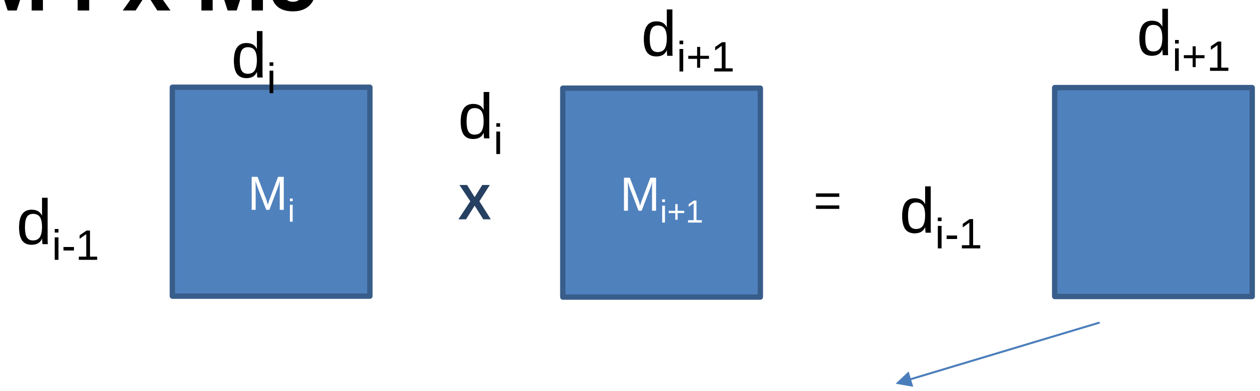
Programación dinámica
Multiplicación encadenada de matrices

¿ Cómo encontrar la secuencia de cálculo de productos matriciales que realice la menor cantidad de productos elementales?

Sea el siguiente producto

M1 x M2 x M3 x M4 x M5

	dimensión
M_1	$d_0 \times d_1$
M_2	$d_1 \times d_2$
M_3	$d_2 \times d_3$
M_4	$d_3 \times d_4$
M_5	$d_4 \times d_5$



 # productos = $d_{i-1} \times d_i \times d_{i+1}$

Programación dinámica
Multiplicación encadenada de matrices

Calculamos la cantidad de productos para todos los subproblemas de tamaño 2. Sean m_{12}, m_{23}, m_{34} y m_{45}.

m_{ij} desde M_i a M_j M_i x..M_j

dimensión

M_1 $d_0 \times d_1$

M_2 $d_1 \times d_2$

M_3 $d_2 \times d_3$

M_4 $d_3 \times d_4$

M_5 $d_4 \times d_5$

Subproblemas	Dimensión	#productos
$M_1 X M_2$	$d_0 \times d_2$	$m_{12} = d_0 \times d_1 \times d_2$
$M_2 X M_3$	$d_1 \times d_3$	$m_{23} = d_1 \times d_2 \times d_3$
$M_3 X M_4$	$d_2 \times d_4$	$m_{34} = d_2 \times d_3 \times d_4$
$M_4 X M_5$	$d_3 \times d_5$	$m_{45} = d_3 \times d_4 \times d_5$

Programación dinámica
Multiplicación encadenada de matrices

Calculamos la cantidad de productos para todos los subproblemas de tamaño 3. Sean m_{13}, m_{24} y m_{35}.

	dimensión
M_1	$d_0 \times d_1$
M_2	$d_1 \times d_2$
M_3	$d_2 \times d_3$
M_4	$d_3 \times d_4$
M_5	$d_4 \times d_5$

Subproblemas	Dimensión	#productos
$M_1 \times M_2 \times M_3$ $-M_1 \times (M_2 \times M_3)$ $-(M_1 \times M_2) \times M_3$	$d_0 \times d_3$	$m_{13} = \min($ $m_{23} + d_0 \times d_1 \times d_3,$ $m_{12} + d_0 \times d_2 \times d_3)$
$M_2 \times M_3 \times M_4$ $-M_2 \times (M_3 \times M_4)$ $-(M_2 \times M_3) \times M_4$	$d_1 \times d_4$	$m_{24} = \min($ $m_{34} + d_1 \times d_2 \times d_4,$ $m_{23} + d_1 \times d_3 \times d_4)$
$M_3 \times M_4 \times M_5$ $-M_3 \times (M_4 \times M_5)$ $-(M_3 \times M_4) \times M_5$	$d_2 \times d_5$	$m_{35} = \min($ $m_{45} + d_2 \times d_3 \times d_5,$ $m_{34} + d_2 \times d_4 \times d_5)$

Programación dinámica
Multiplicación encadenada de matrices

Calculamos la cantidad de productos para todos los subproblemas de tamaño 4. Sean m_{14} y m_{25} .

Subproblemas	Dimensión	#productos
$(M_1 X M_2 X M_3 X M_4)$ - $M_1 X (M_2 X M_3 X M_4)$ - $(M_1 X M_2) X (M_3 X M_4)$ -$(M_1 X M_2 X M_3) X M_4$	$d_0 x d_4$	m_{14}= min ($m_{24} + d_0 x d_1 x d_4$, $m_{12} + m_{34} + d_0 x d_2 x d_4$, $m_{13} + d_0 x d_3 x d_4$)
$(M_2 x M_3 X M_4 X M_5)$ - $M_2 x (M_3 X M_4 X M_5)$. $(M_2 x M_3) X (M_4 X M_5)$. $(M_2 x M_3 X M_4) X M_5$	$d_1 x d_5$	m_{25} = min($m_{35} + d_1 x d_2 x d_5$, $m_{23} + m_{45} + d_1 x d_3 x d_5$, $m_{24} + d_1 x d_4 x d_5$)

Programación dinámica
Multiplicación encadenada de matrices

Calculamos la cantidad de productos para el único subproblemas de tamaño 5. Sea m_{15} .

Subproblemas	Dimensión	#productos
M_1 X M_2 x M_3 x M_4 x M_5 -M_1 X (M_2 X M_3 x M_4 x M_5) -(M_1 X M_2) X (M_3 x M_4 x M_5) -(M_1 x M_2 x M_3) x (M_4 x M_5) - ((M1 x M2 X M3 X M4) X M5)	d_0 x d_5	m_{15}= min(m_{25} + d_0 x d_1 x d_5, m_{12} +m_{35} +d_0 x d_2 x d_5 , m_{13} + m_{45} + d_0 x d_3 x d_5, m_{14} + d_0 x d_4 x d_5)

Programación dinámica
Multiplicación encadenada de matrices

Generalización para n matrices

$$m_{15} = \min(\, m_{11} + m_{25} + d_0 \times d_1 \times d_5,\; m_{12} + m_{35} + d_0 \times d_2 \times d_5,\; m_{13} + m_{45} + d_0 \times d_3 \times d_5,\; m_{14} + m_{55} + d_0 \times d_4 \times d_5)$$

$m_{11} = 0,\ m_{55} = 0$

$m_{15} = \mathbf{min}\ (\, m_{1k} + m_{(k+1)5} + d_0 \times d_k \times d_5)$

$\mathbf{1 \leq k < 5}$

Programación dinámica
Multiplicación encadenada de matrices

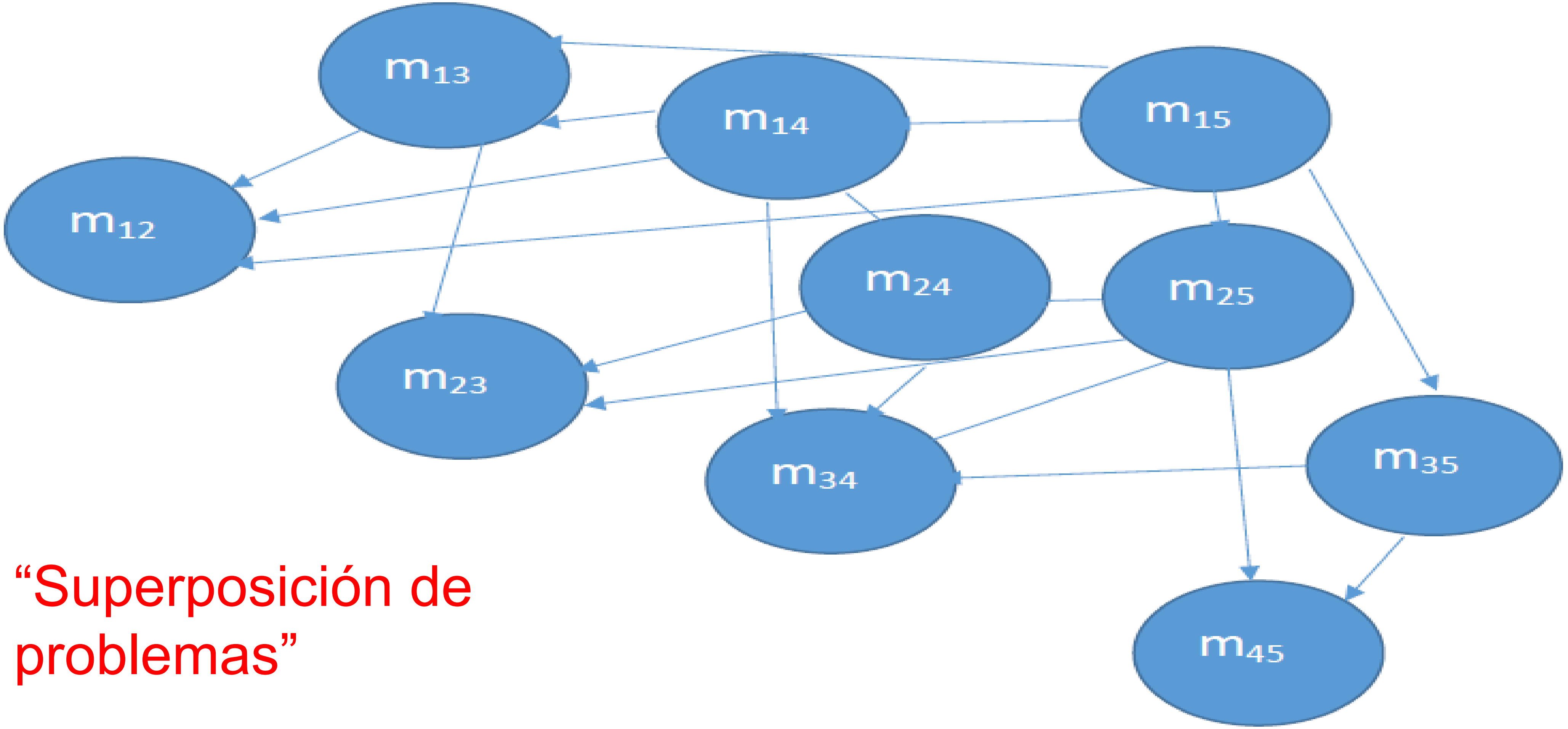
Generalización para n matrices

$m_{ii} = 0$

$$m_{ij} = \min_{i \le k < j} \left(m_{ik} + m_{(k+1)j} + d_{i-1} \times d_k \times d_j \right)$$

$$1 \le i,j \le n$$

Programación dinámica
Multiplicación encadenada de matrices



“Superposición de problemas”

Programación dinámica
Multiplicación encadenada de matrices

Ejemplo

M_1 30 x 35
M_2 35 x 15
M_3 15 X 5
M_4 5 X 10
M_5 10 X 20
M_6 20 X 25

	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

Programación dinámica
Multiplicación encadenada de matrices

Pseudo-código

```
// inicialización
 …
for(i=1; i<=n; i++)
for(j=1;j<=n; j++)
{ Mat [i][j]= ∞;
   Mejor[i][j]=0;
}
for (j=1; j<=n; j++)
Mat[j][j]=0;
```

```
//cálculo
for (l=1; l<n; l++)
for (i=1;i<=n-l; i++)
{j= i+l;
 for (k=i; k<j; k++)
    { t= Mat[i][k] + Mat[k+1][j] + d [i] * d[k] * d[j];
       if (t < Mat[i][j])
          { Mat[i][j] = t;
             Mejor[i][j]= k;
          }
    }
```

Programación dinámica
Multiplicación encadenada de matrices

Ejemplo

M_1 30 x 35
M_2 35 x 15
M_3 15 X 5
M_4 5 X 10
M_5 10 X 20
M_6 20 X 25

	1	2	3	4	5	6
1	0	1	1	3	3	3
2		0	2	3	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						0

MEJOR

	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

MAT

MEJOR[1][6] = 3

$m_{16} = m_{13} + m_{46} + d_0 \times d_3 \times d_6$

$m_{16} = 7875 + 3500 + 30 \times 5 \times 25 = 15125$

Programación dinámica
Multiplicación encadenada de matrices

Ejemplo

M_1 30 x 35

M_2 35 x 15

M_3 15 X 5

M_4 5 X 10

M_5 10 X 20

M_6 20 X 25

	1	2	3	4	5	6
1	0	1	1	3	3	3
2		0	2	3	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						0

MEJOR

	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

MAT

$m_{16} = m_{13} + m_{46} + d_0 \times d_3 \times d_6$

$m_{16} = 7875 + 3500 + 30 \times 5 \times 25 = 15125$

$m_{13} = m_{11} + m_{23} + d_0 \times d_1 \times d_3 = 2625 + 30 \times 35 \times 5 = 7875$

$m_{46} = m_{45} + m_{66} + d_3 \times d_5 \times d_6 = 1000 + 5 \times 20 \times 25 = 3500$

Ejercicio

1. Realizar un seguimiento para el producto M_1 x M_2 x…x M_6

M_1 30 x 35

M_2 35 x 15

M_3 15 X 5

M_4 5 X 10

M_5 10 X 20

M_6 20 X 25

2. Implementar en C++ una función *recuperaSecuencia* de cálculo óptima para un producto desde M_i a M_j

Programación dinámica

- Se aplica cuando los subproblemas *no son independientes entre sí*, es decir se usa un mismo subproblema para resolver diferentes problemas mayores.
- La técnica “Divide y conquista” no es conveniente para resolver estos problemas dado que los problemas no son independientes y un mismo problema sería calculado más de una vez.
- Se resuelven subproblemas desde las instancias más pequeñas hasta la mayor instancia.
- Los subproblemas se resuelven combinando las soluciones óptimas de las instancias pequeñas ya calculadas.
- Los resultados parciales son *almacenados* para ser reusados sin tener que recalcularlos.

Programación dinámica

Los problemas que tienen una subestructura óptima pueden resolverse a partir de 3 pasos generales:

•Dividir el problema en subproblemas más pequeños.

•Encontrar la solución óptima estos problemas

•Usar estas soluciones óptimas para construir una solución óptima al problema original

Programación Dinámica

“Árboles binarios de mínimo costo”

Programación Dinámica
Árboles binarios de mínimo costo

Supongamos un árbol binario de naturales con las siguientes características:

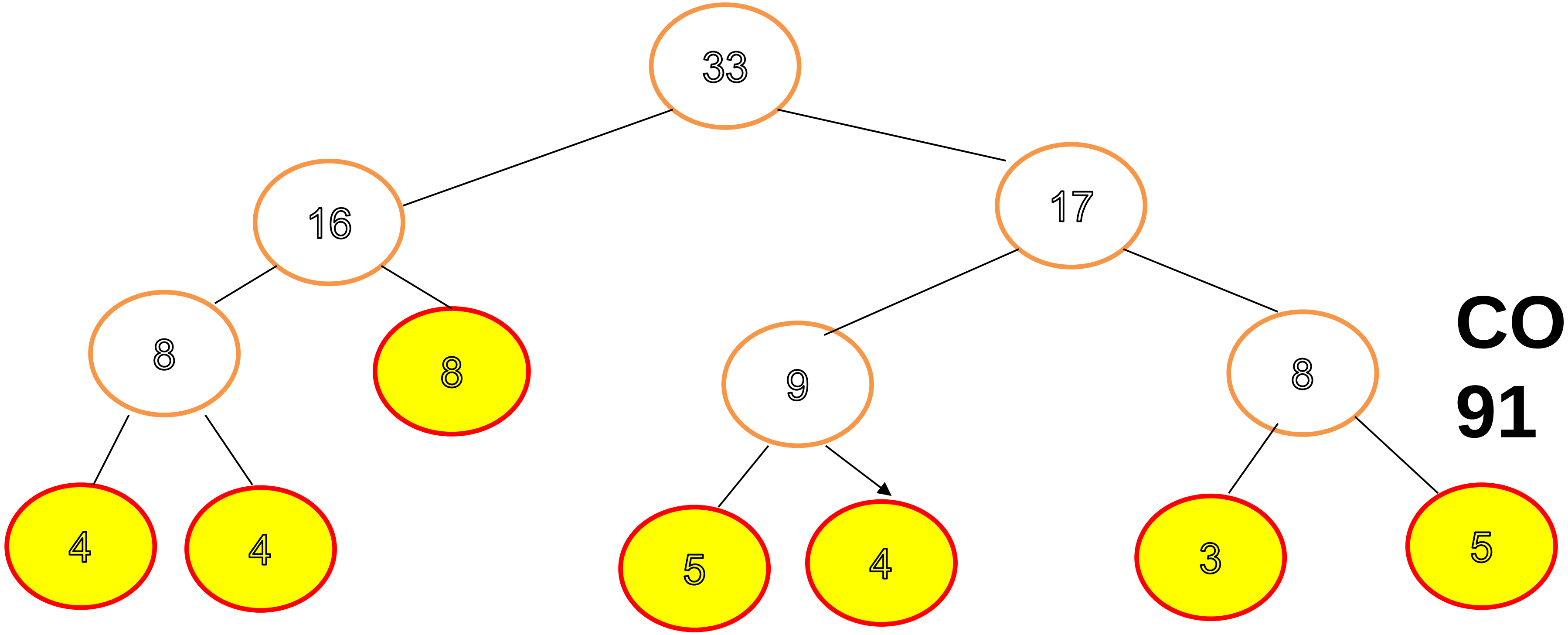
- Las hojas contienen naturales
- Cada nodo interno contiene la suma de los valores de sus hijos

La frontera del árbol es el listado de sus hojas de izquierda a derecha.

Para una misma frontera tengo diferentes árboles binarios

¿ cuál es el árbol binario que minimiza la suma de los valores de los nodos interiores?

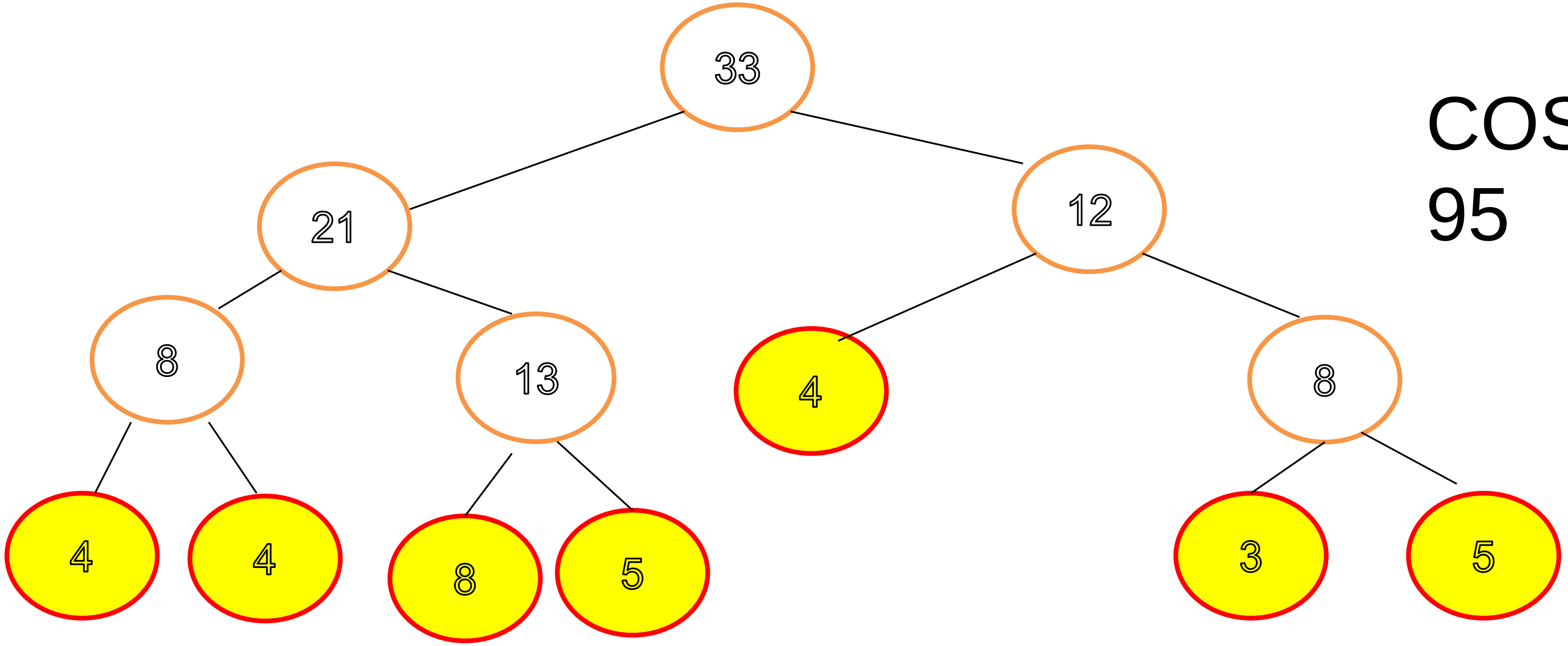
Programación Dinámica
Árboles binarios de mínimo costo



COSTO = 91

FRONTERA = < 4,4,8,5,4,3,5>

Programación Dinámica
Árboles binarios de mínimo costo



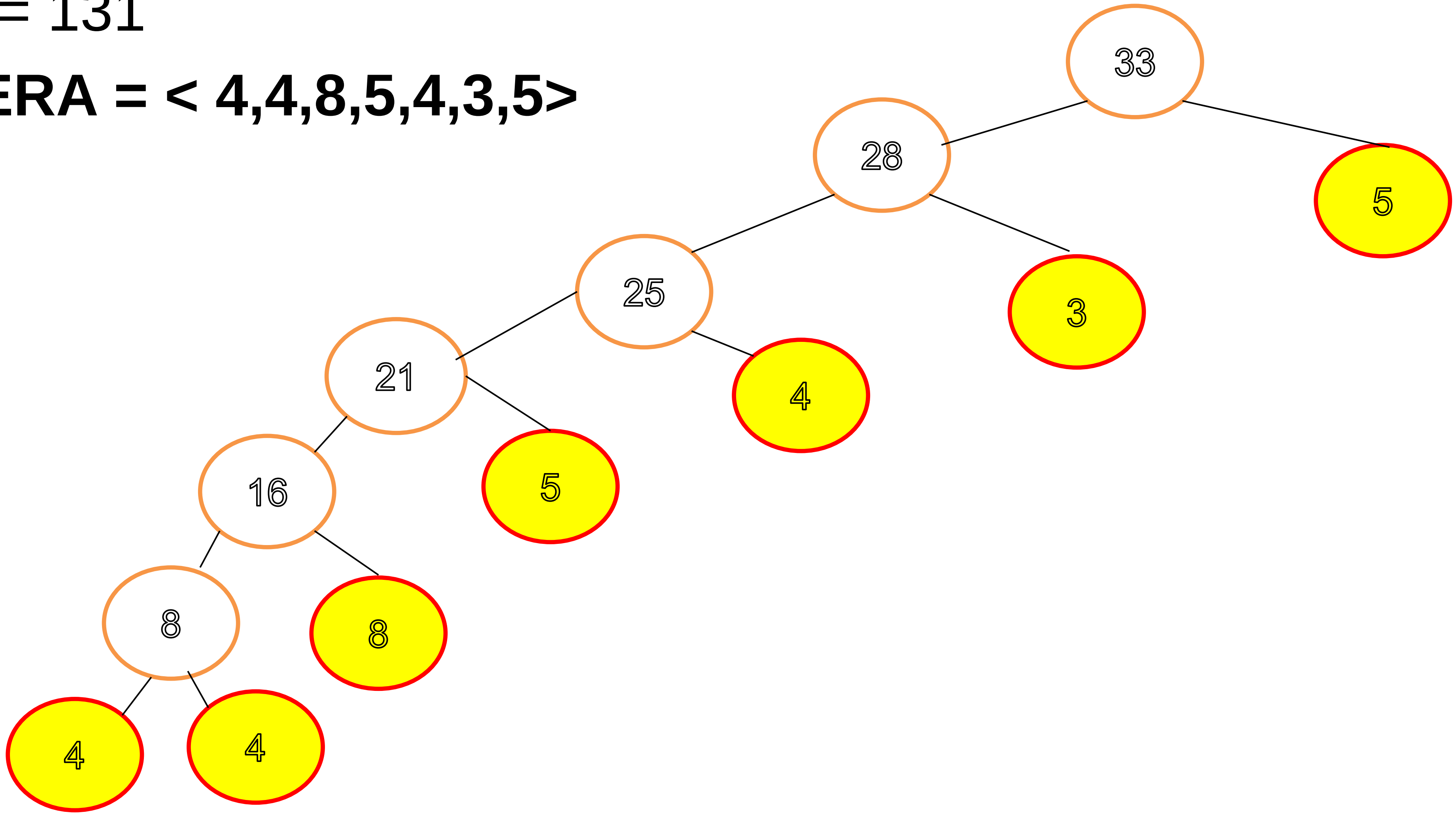
COSTO = 95

FRONTERA = < 4,4,8,5,4,3,5>

Programación Dinámica
Árboles binarios de mínimo costo

COSTO = 131

FRONTERA = < 4,4,8,5,4,3,5>



Programación Dinámica
Árboles binarios de mínimo costo

Ejemplo: < **4, 4, 8, 5, 4, 3, 5** >

C i j
desde i hasta j

tamaño 2:

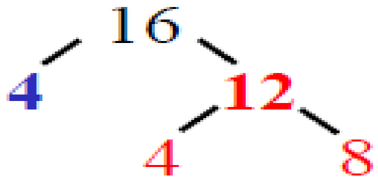
8 4 4	12 4 8	13 8 5
C 12= 8	C 23=12	C 34=13
9 5 4	7 4 3	8 3 5
C 45=9	C 56=7	C 67=8

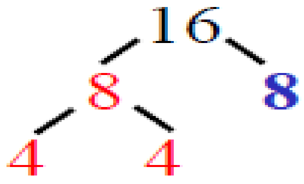
Programación Dinámica
Árboles binarios de mínimo costo

Ejemplo: < **4, 4, 8,** 5, 4, 3, 5 >

tamaño 3:

C 13 =





C 11 + **C23** + 16 = 28

C 12 + **C33** + 16 = 24

C 13 = min [(C11 + C23) , (C12 + C33)] + 16

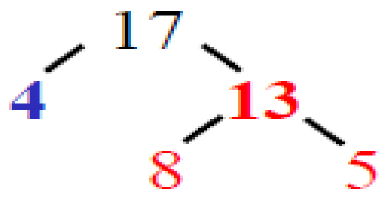
= min [(0 + 12) , (8 + 0)] + 16 = 24

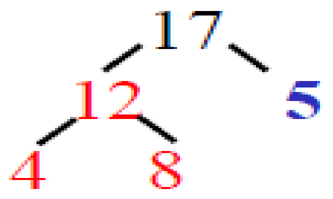
Programación Dinámica
Árboles binarios de mínimo costo

Ejemplo: < 4, **4, 8, 5,** 4, 3, 5 >

tamaño 3:

C 24 =





C 22 + **C34** + 17 = 30

C 23 + **C44** + 17 = 29

C 24 = min [(C22 + C34) , (C23 + C44)] + 17

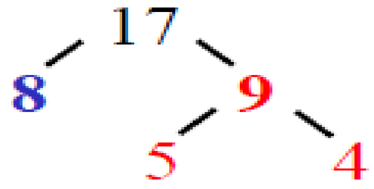
= min [(0 + 13) , (12 + 0)] + 17 = 29

Programación Dinámica
Árboles binarios de mínimo costo

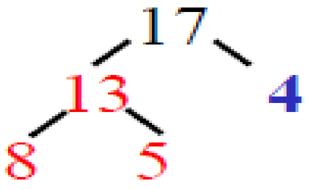
Ejemplo: < 4, 4, **8, 5, 4**, 3, 5 >

tamaño 3:

C 35 =



$\mathbf{C\ 33} + \mathbf{C45} + 17 = 26$



$\mathbf{C\ 34} + \mathbf{C55} + 17 = 30$

$$C\ 35 = \min [(C33 + C45) , (C34 + C55)] + 17$$

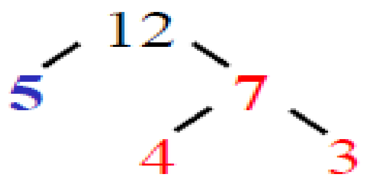
$$= \min [(0 + 9) , (13 + 0)] + 17 = 26$$

Programación Dinámica
Árboles binarios de mínimo costo

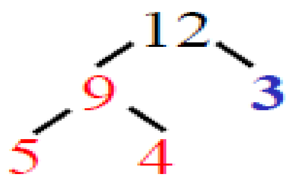
Ejemplo: $< 4, 4, 8, 5, 4, 3, 5 >$

tamaño 3:

$C\ 46 =$

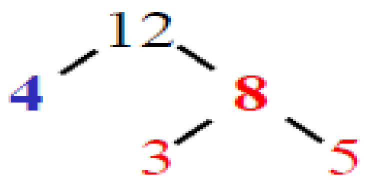


$C\ 44 + C56 + 12 = \underline{\mathbf{19}}$

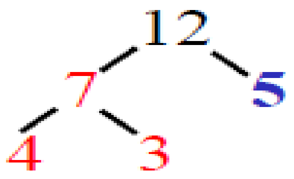


$C\ 45 + C66 + 12 = 25$

$C\ 57 =$



$C\ 55 + C67 + 12 = 20$



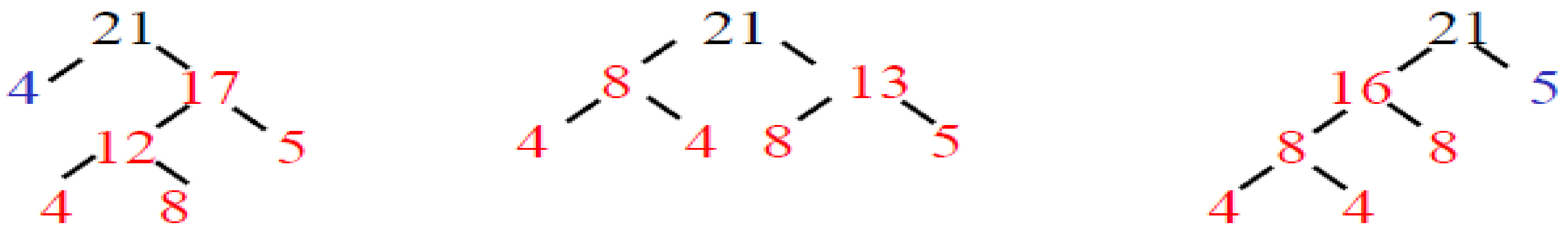
$C\ 56 + C77 + 12 = \underline{\mathbf{19}}$

Programación Dinámica
Árboles binarios de mínimo costo

Ejemplo: < **4, 4, 8, 5,** 4, 3, 5 >

tamaño 4:

C 14 =



C 11 + **C24** + 21 = 50   **C 12** + **C34** + 21 = 42   **C 13** + **C44** + 21 = 45

C 14 = min [(C11 + C24) , (C12 + C34) , (C13 + C44)] + 21
= min [(0 + 29) , (8 + 13), (24 + 0)] + 21 = **42**

Programación Dinámica
Árboles binarios de mínimo costo

Ejemplo: < **4, 4, 8, 5, 4, 3, 5** >

tamaño 4:
C 14 C 25 C 36 C 47

tamaño 5:
C 15 C 26 C 37

tamaño 6:
C 16 C 27

tamaño 7:
C 17 → **Solución**

Programación Dinámica
Árboles binarios de mínimo costo

Suponga la siguiente secuencia de valores V= V1, V2, …, Vn para n elementos

$C_{i,k}$ es el costo del árbol óptimo con la secuencia de nodos que van desde i hasta k

$$W_{i,k} = \sum_{j=i}^{k} V_j$$

$C_{i,i}=0$

$$C_{i,k} = \min_{i \le j < k} \left(C_{i,j} + C_{j+1,k}\right) + W_{i,k}$$

Programación Dinámica
Árboles binarios de mínimo costo

$< 4, 4, 8, 5, 4, 3, 5 >$

C_{ij}

j

i

J=1	J=2	J=3	J=4	J=5	J=6	J=7
0						
	0					
		0				
			0			
				0		
					0	
						0

$C_{kk} = 0$ (las hojas del árbol)

Programación Dinámica
Árboles binarios de mínimo costo

$< 4, 4, 8, 5, 4, 3, 5 >$

C_{ij} j

	J=1	J=2	J=3	J=4	J=5	J=6	J=7
C_{12}	0	8					
C_{23}		0	12				
C_{34}			0	13			
				0	9		
					0	7	
C_{67}						0	8
							0

Subárboles de tamaño 2: costo igual a la suma de sus 2 pesos

Programación Dinámica
Árboles binarios de mínimo costo

< 4, 4, 8, 5, 4, 3, 5 >

C_{ij}

j

J=1	J=2	J=3	J=4	J=5	J=6	J=7
0	8	24				
	0	12				
		0	13			
			0	9		
				0	7	
					0	8
						0

Subárboles de tamaño 3

$$C\,13 = \min[\,(C11 + C23)\,,\,(C12 + C33)] + 16$$
$$= \min[\,(0 + 12)\,,\,(8 + 0)] + 16 = 24$$

Árboles binarios de mínimo costo

$< 4, 4, 8, 5, 4, 3, 5 >$

C_{ij} — j

	J=1	J=2	J=3	J=4	J=5	J=6	J=7
C_{13}	0	8	**24**				
C_{24}		0	12	**29**			
C_{35}			0	13	**26**		
				0	9	**19**	
C_{47}					0	7	**19**
						0	8
							0

Subárboles de tamaño 3

Programación Dinámica
Árboles binarios de mínimo costo

$< 4, 4, 8, 5, 4, 3, 5 >$

C_{ij} j

C_{14}

J=1	J=2	J=3	J=4	J=5	J=6	J=7
0	8	24	42			
	0	12	29			
		0	13	26		
			0	9	19	
				0	7	19
					0	8
						0

Subárboles de tamaño 4

$C\,14 = \min [\,(C11 + C24),\ (C12 + C34),\ (C13 + C44)\,] + 21$

$= \min [\quad (0 + 29),\quad (8 + 13),\quad (24 + 0)\,] \; + 21 = \underline{42}$

Programación Dinámica
Árboles binarios de mínimo costo

< **4, 4, 8, 5, 4, 3, 5** >

C_{ij}

j

J=1	J=2	J=3	J=4	J=5	J=6	J=7
0	8	24	42	58	71	91
	0	12	29	42	55	75
		0	13	26	39	57
			0	9	19	34
				0	7	19
					0	8
						0

→ **Solución**

Programación Dinámica
Árboles binarios de mínimo costo

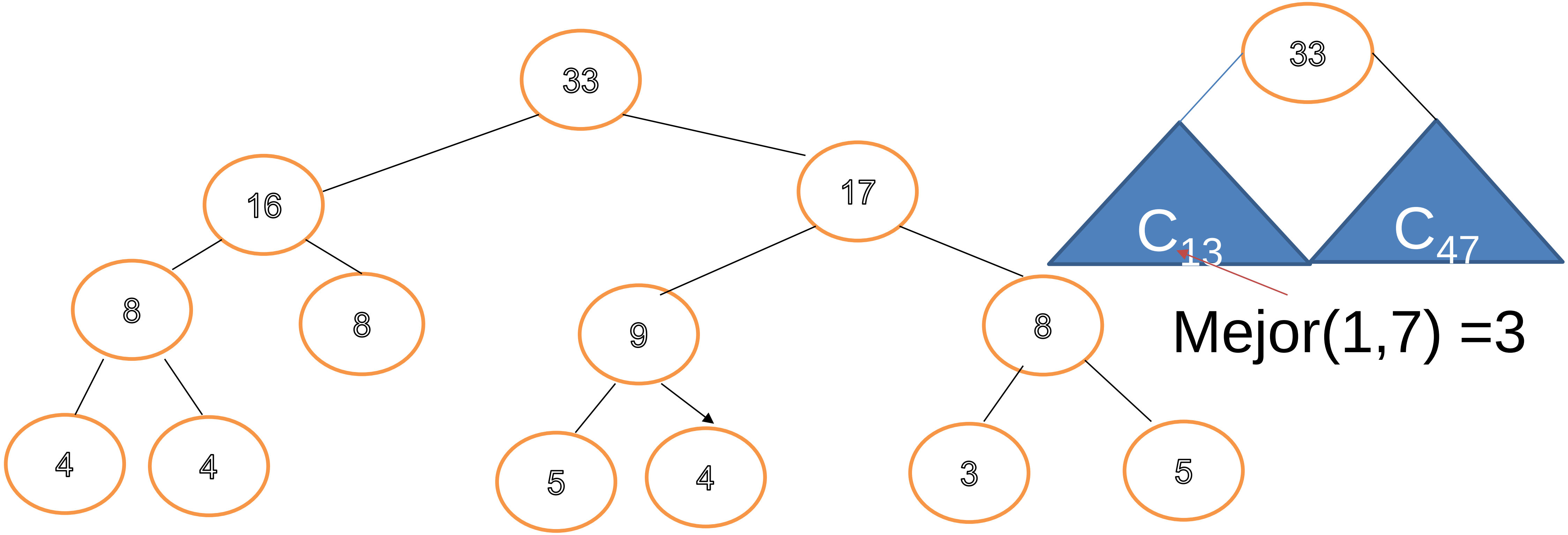
C

J=1	J=2	J=3	J=4	J=5	J=6	J=7
0	8	24	42	58	71	91
	0	12	29	42	55	75
		0	13	26	39	57
			0	9	19	34
				0	7	19
					0	8
						0

MEJOR

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0	2	2	3	3	3
2		0	0	3	3	3	3
3			0	0	3	3	4
4				0	0	4	5
5					0	0	6
6						0	0
7							0

Programación Dinámica
Árboles binarios de mínimo costo



Mejor(1,7) =3

Programación Dinámica
Decibilidad de lenguajes libres del contexto

Programación dinámica
Decibilidad de lenguajes libres del contexto

Problema:

Determinar la pertenencia de una cadena a un lenguaje de tipo 2 cuyas reglas de producción están en forma normal de Chomsky

Ejemplo:

G = (N, T, P, S)

T = {a,b} N= {A,B,C}

Reglas de producción P

S::= AB|BC

A::= BA|a

B::= CC| b

C::= AB| a

Toda gramática de tipo 2 puede expresarse en la forma normal de Chomsky

Programación dinámica
Decibilidad de lenguajes libres del contexto

Dada una cadena de terminales $x= x_1x_2x_3\ldots x_n$ ¿pertenece al lenguaje generado por una gramática G?

Estrategia

Encontrar el conjunto de símbolos no terminales a partir de los cuales puedo generar a las subcadenas de x .Si incluye al símbolo distinguido pertenece al lenguaje.

- Comenzar con las subcadenas de tamaño 1, 2,…, n
- Reutilizar las soluciones de los problemas menores para encontrar la solución de subproblemas de mayor tamaño

Programación dinámica
Decibilidad de lenguajes libres del contexto

M

	b	a	a	b	a
1	{B}	{A, C}	{A, C}	{B}	{A, C}
2	{S ,A }	{B}	{S, C}	{S, A}	
3	Ø	{B}	{B}		
4	Ø	{S, A, C}			
5	{S, A, C}				

x = baaba

S::= AB|BC
A::= BA|a
B::= CC| b
C::= AB| a

S->AB
S->BC
A->BA
A->a
B->CC
B->b
C->AB
C->a

$M_{j\,i}$ es el conjunto de símbolos no terminales a partir de los cuales es posible generar la subcadena de x que empieza en el i-ésimo símbolo y tiene tamaño j

Programación dinámica
Decibilidad de lenguajes libres del contexto

	b	a	a	b	a
1	{B}	{A, C}	{A, C}	{B}	{A, C}
2	{S ,A }	{B}	{S, C}	{S, A}	
3	Ø	{B}	{B}		
4	Ø	{S, A, C}			
5	{S, A, C}				

x = baaba

S::= AB|BC
A::= BA|a
B::= CC| b
C::= AB| a

M_{11} es {B} porque x_1 = b puede generarse desde B-> b

M_{12} es {A,C } porque x_2 = a puede generarse desde A->a y C -> a

M_{21} es {S, A} porque la subcadena ba puede ser generada desde S y desde A

b a

{B} {A,C}

¿Existen reglas de producción cuyo lado derecho sea BA o BC?

S -> BC S-> BC -> b C -> ba

A -> BA A -> BA -> bA ->ba

Programación dinámica
Decibilidad de lenguajes libres del contexto

	b	a	a	b	a
1	{B}	{A, C}	{A, C}	{B}	{A, C}
2	{S ,A }	{B}	{S, C}	{S, A}	
3	Ø	{B}	{B}		
4	Ø	{S, A, C}			
5	{S, A, C}				

x = baaba

S::= AB|BC
A::= BA|a
B::= CC| b
C::= AB| a

M_{21}
b| a
{B} | {A,C}
BA
BC
{A,S}

M_{22}
a | a
{A,C} | {A,C}
~~AA~~
~~AC~~
~~CA~~
CC
{B}

M_{23}
a|b
{A,C}| {B}
AB
~~CB~~
{S,C}

M_{24}
b| a
{B} | {A,C}
BA
BC
{A,S}

Programación dinámica
Decibilidad de lenguajes libres del contexto

	b	a	a	b	a
1	{B}	{A, C}	{A, C}	{B}	{A, C}
2	{S ,A }	{B}	{S, C}	{S, A}	
3	Ø	{B}	{B}		
4	Ø	{S, A, C}			
5	{S, A, C}				

M_{31}

b|aa — **ba|a**

{B} |{B} — **{S,A} |{A,C}**

~~BB~~ — **~~SA~~**, **~~SC~~**, **~~AA~~**, **~~AC~~**

Ø U Ø

x = baaba

S::= AB|BC
A::= BA|a
B::= CC| b
C::= AB| a

M_{32}

a| ab — **aa|b**

{A,C} | {C,S} — **{B} {B}**

~~AC~~
~~AS~~
CC
~~CS~~

{B} U Ø

M_{33}

a|ba — **ab | a**

{A,C}| {S,A} — **{C,S}| {A,C}**

~~AA~~, **~~CA~~** — **~~CA~~**, **CC**, **~~SA~~**, **~~SC~~**

Ø U {B}

Programación dinámica
Decibilidad de lenguajes libres del contexto

	b	a	a	b	a
1	{B}	{A, C}	{A, C}	{B}	{A, C}
2	{S ,A }	{B}	{S, C}	{S, A}	
3	Ø	{B}	{B}		
4	Ø	{S, A, C}			
5	{S, A, C}				

x = baaba

S::= AB|BC
A::= BA|a
B::= CC| b
C::= AB| a

M_{41}

b|aab ba|ab baa|b
{B}|{B} {S,A}| {C,S} Ø {B}

~~BB~~ ~~SC~~
~~SS~~
~~AC~~
~~AS~~

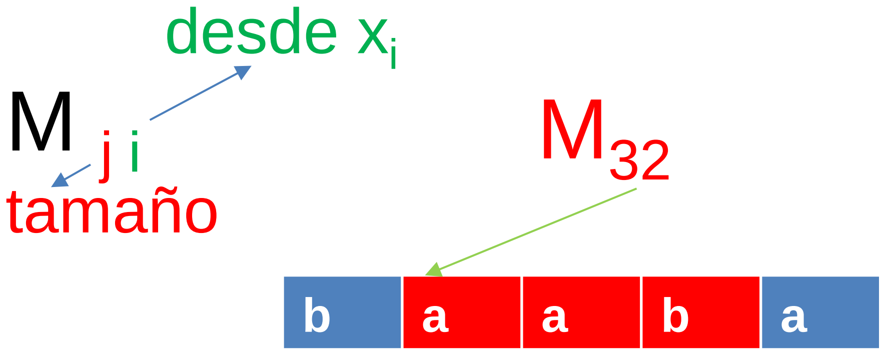
Ø U Ø U Ø

M_{42}

a|aba aa|ba aab|a
{A,C}|{B} {B}|{S,A} {B}|{A,C}
AB ~~BS~~ BA
~~CB~~ ~~BA~~ BC
{S,C} U Ø U {S,A}

Programación dinámica
Decibilidad de lenguajes libres del contexto

Sea M_{ji} el conjunto de símbolos no terminales a partir de los cuales puedo generar la subcadena de x de tamaño j a partir del i-ésimo símbolo (x_i)



M_{1i}= {A/ A->a ∊ P y x_i = a}

$$M_{ji} = \bigcup_{1 \le k < j} \{ A / A \to BC \in P \text{ y } B \in M_{ki} \text{ y } C \in M_{(j-k)(i+k)} \}$$

Programación dinámica
Decibilidad de lenguajes libres del contexto

	b	a	a	b	a
1	{B}	{A, C}	{A, C}	{B}	{A, C}
2	{S ,A }	{B}	{S, C}	{S, A}	
3	Ø	{B}	{B}		
4	Ø	{S, A, C}			
5	{S, A, C}				

M_{51}

b|aaba	ba|aba	baa|ba	baab|a
{B}|{S,A,C}	{S,A}|{B}	Ø	Ø

~~BS~~ ~~SBU~~

BA AB

BC

{S,A} U {S,C} U Ø U Ø

x = baaba

J=5
i=1

S::= AB|BC
A::= BA|a
B::= CC| b
C::= AB| a

$$M_{ji} = \bigcup_{1\leq k<j} \{ A / A\text{->}BC \in P \text{ y } B \in M_{ki} \text{ y } C \in M_{(j-k)(i+k)} \}$$

Programación dinámica
Decibilidad de lenguajes libres del contexto

Pseudo-código

```
for (i=1; i<=n; i++)
  M1i= {A/ A->a ∊ P y xi = a};
for (j=2; j<=n; j++)
 {for(i=1;i<=n-j+1;i++)
    {Mji = Ø;
      for (k=1; k<j; k++)
          Mji =   Mji U { A / A->BC ∊P y B ∊ Mki y C ∊ M(j-k)(i+k)};
    }
  }
```

$M_{1i} = \{A/\ A \to a \in P \text{ y } x_i = a\}$

$M_{ji} = M_{ji} \cup \{ A / A \to BC \in P \text{ y } B \in M_{ki} \text{ y } C \in M_{(j-k)(i+k)}\}$

Programación Dinámica

Triangulación de un polígono convexo

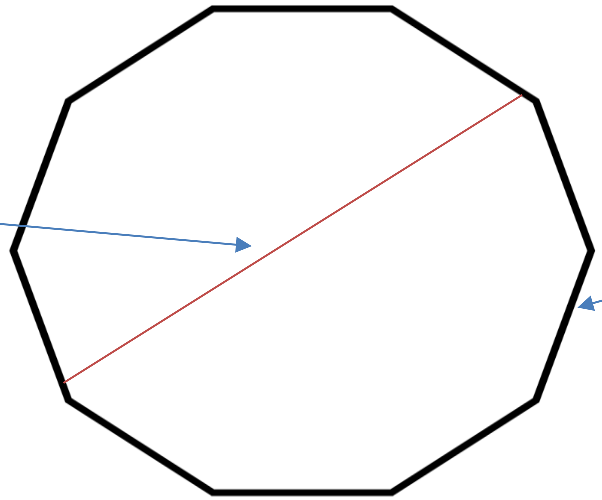
Programación dinámica
Triangulación de un polígono convexo

Polígono convexo

. sus ángulos interiores son menores que 180 °

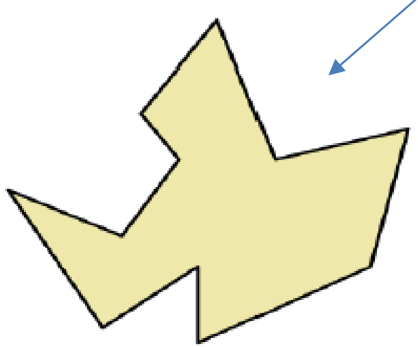
. sus diagonales ,segmentos entre dos vértices no consecutivos, son interiores.

diagonal



convexo

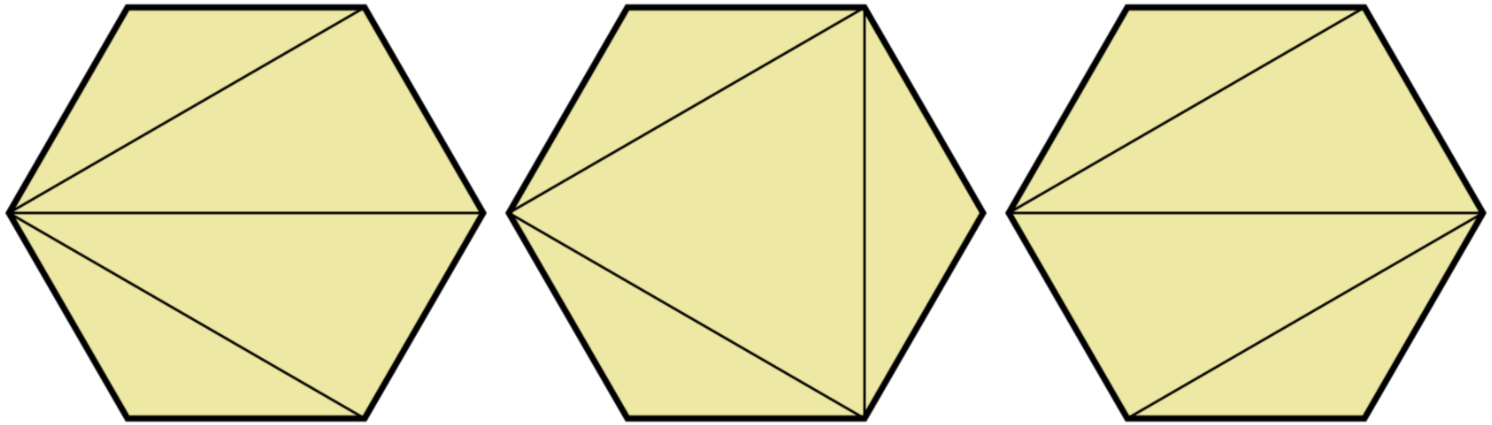
cóncavo



Programación dinámica
Triangulación de un polígono convexo

Una triangulación de un polígono convexo es una división de su área en un conjunto de triángulos que cumplen las siguientes condiciones:

- La unión de todos los triángulos es el polígono original
- Los vértices de los triángulos son vértices del polígono original
- Todo par de triángulos es disjunto o comparte un vértice o una cuerda (segmentos entre vértices no adyacentes)



Programación dinámica
Triangulación de un polígono convexo

Aplicaciones

Geometría computacional

Conjunción de la Geometría Clásica y la informática

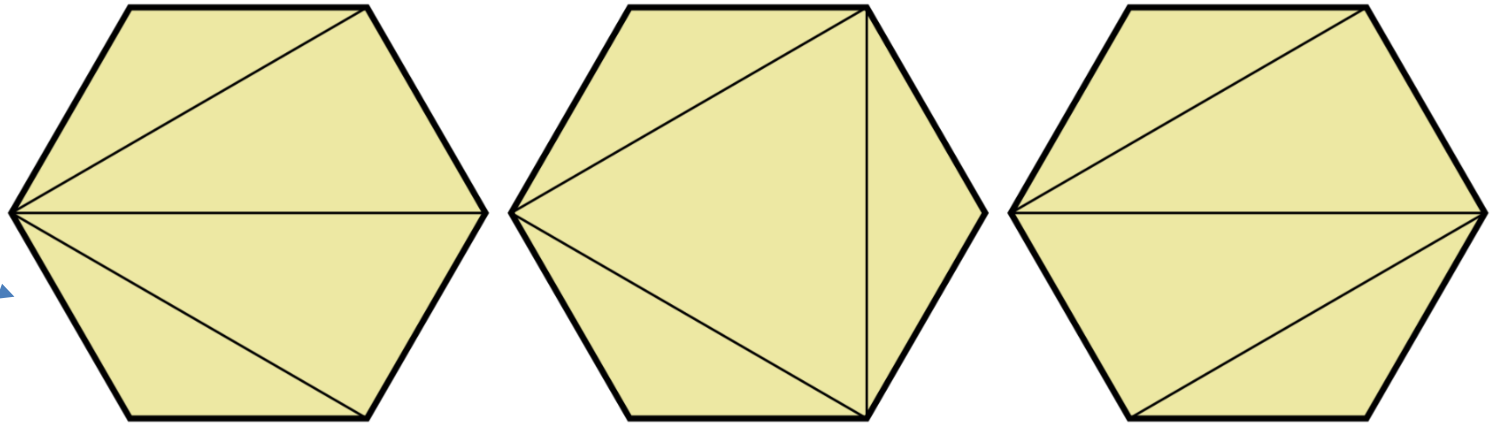
¿Cómo resolver eficientemente problemas de naturaleza geométrica computacionalmente?

Programación dinámica
Triangulación de un polígono convexo

Una triangulación de un polígono convexo es una división de su área en un conjunto de triángulos que cumplen las siguientes condiciones:

- La unión de todos los triángulos es el polígono original
- Los vértices de los triángulos son vértices del polígono original
- Todo par de triángulos es disjunto o comparte un vértice o una cuerda (segmentos entre vértices no adyacentes)

Algunas posibles triangulaciones



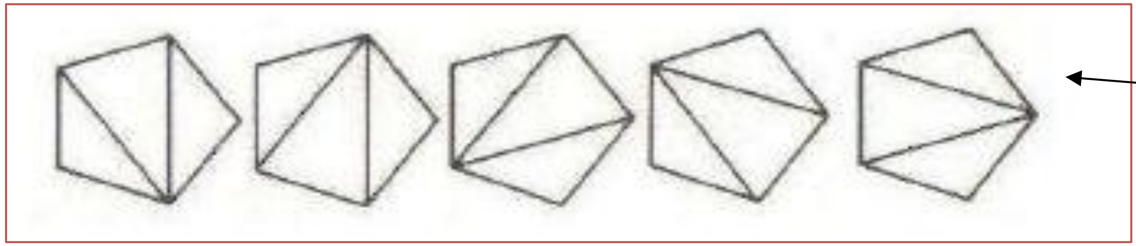
Programación dinámica
Triangulación de un polígono convexo

Euler demostró que la cantidad total de triangulaciones de un polígono convexo es el número Catalan (n-2)

$t_n = C_{n-2}$

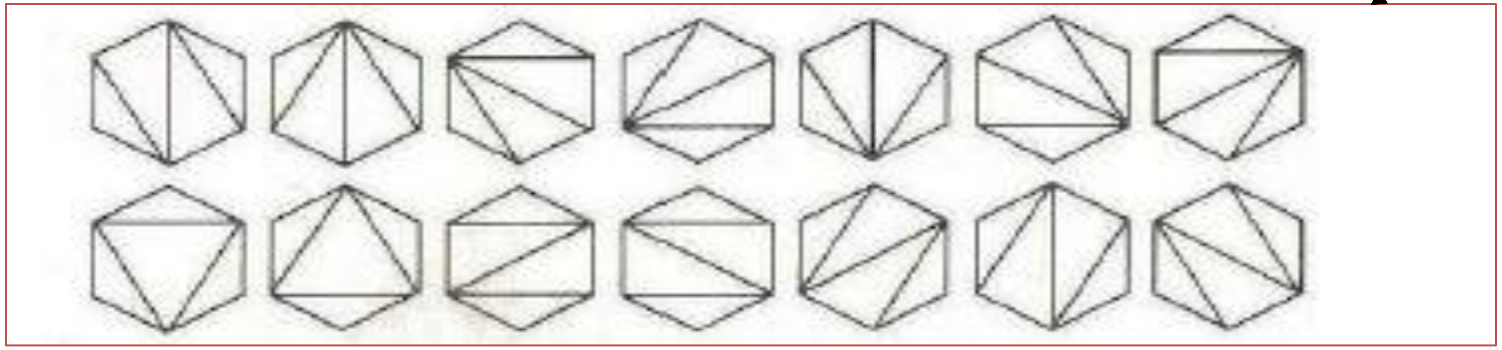
$C_n = (2n)! / (n! (n+1)!)$

n	Cn
1	1
2	2
3	5
4	14
5	42
6	132…



n = 5

n = 7



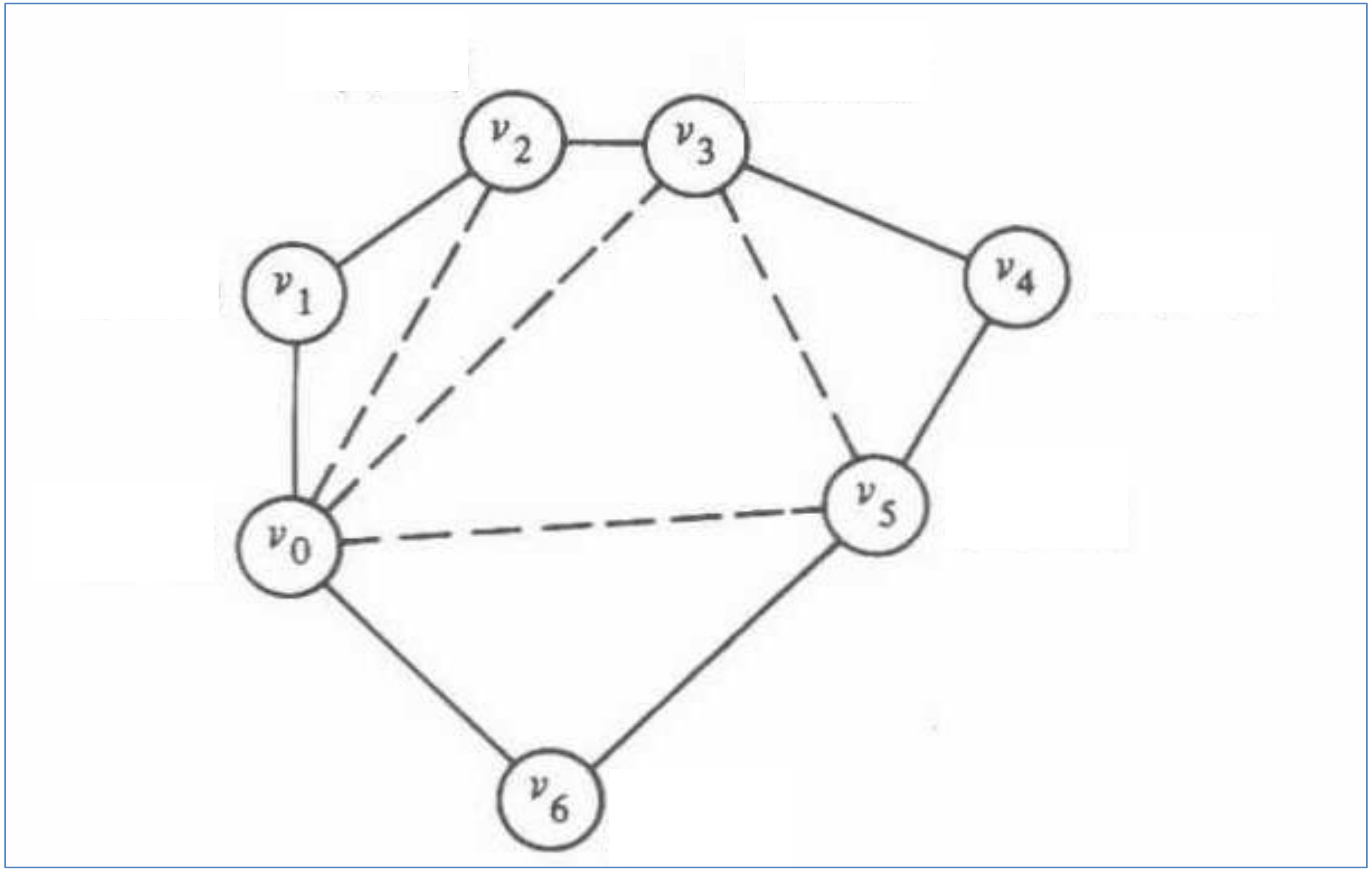
Programación dinámica
Triangulación de un polígono convexo

Problema de la triangulación de un polígono convexo

Seleccionar un conjunto de diagonales que no se corten y partan en triángulos. El costo de la triangulación es la suma de los costos de las diagonales.

Supongamos n vértices ordenados en sentido horario a partir de un vértice v_o del polígono

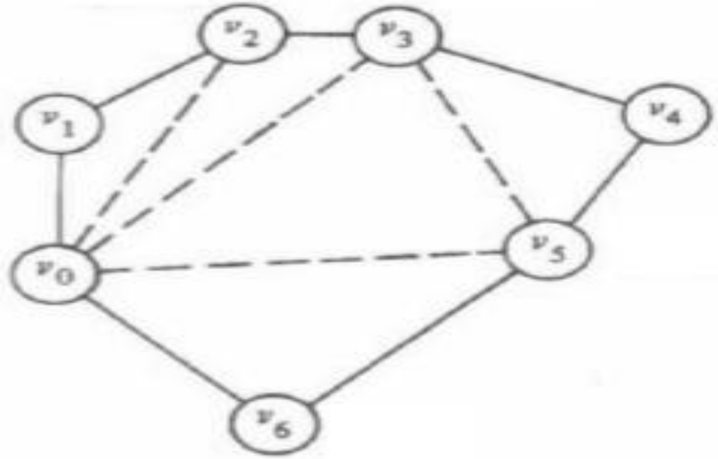
Programación dinámica
Triangulación de un polígono convexo



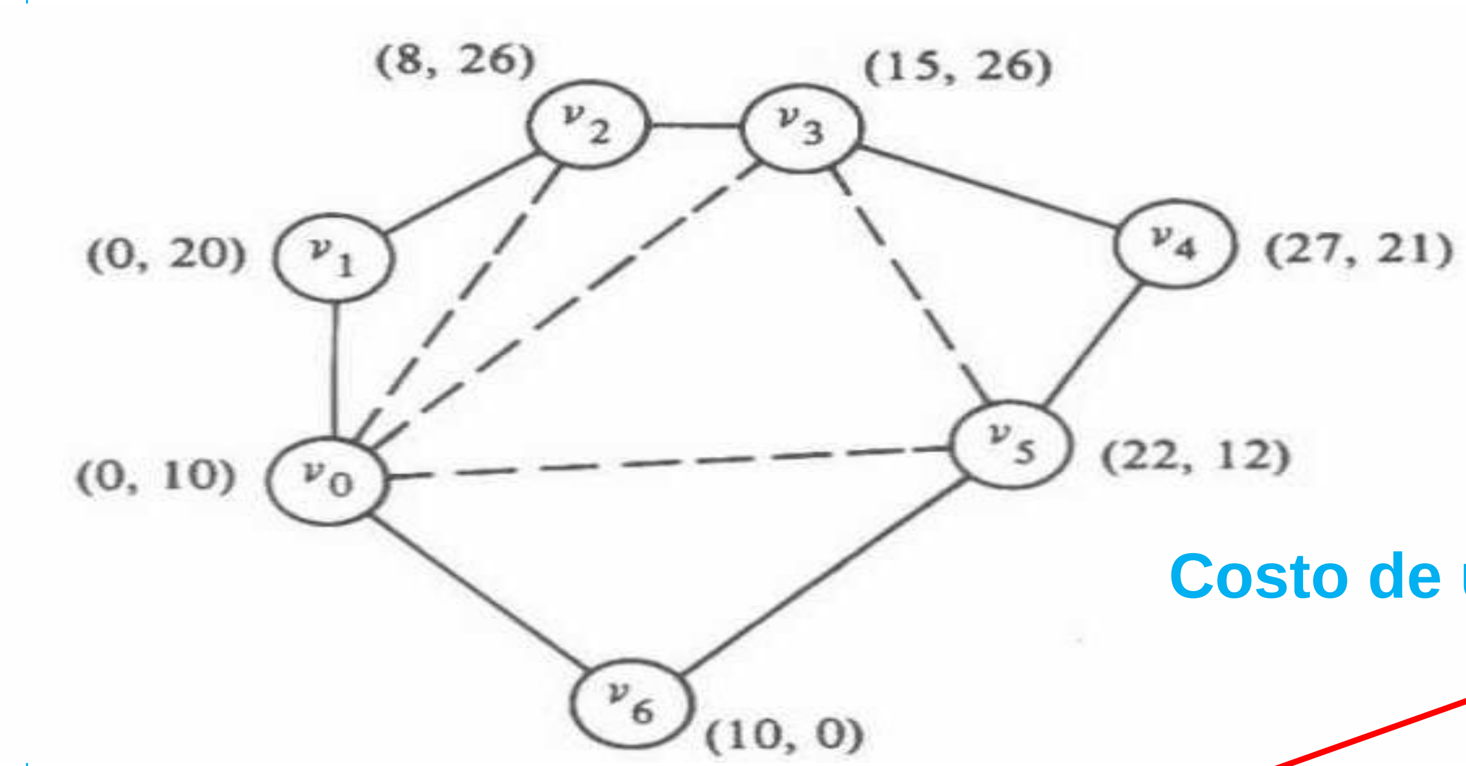
Programación dinámica
Triangulación de un polígono convexo

Consideraciones:

- En toda triangulación de un polígono de más de 3 vértices, cada par de vértices adyacentes es tocado al menos por una diagonal
- Si v_i y v_j definen una diagonal (v_i, v_j) debe existir un v_k / (v_i,v_k) y (v_k,v_j) son lados o diagonal



Programación dinámica
Triangulación de un polígono convexo



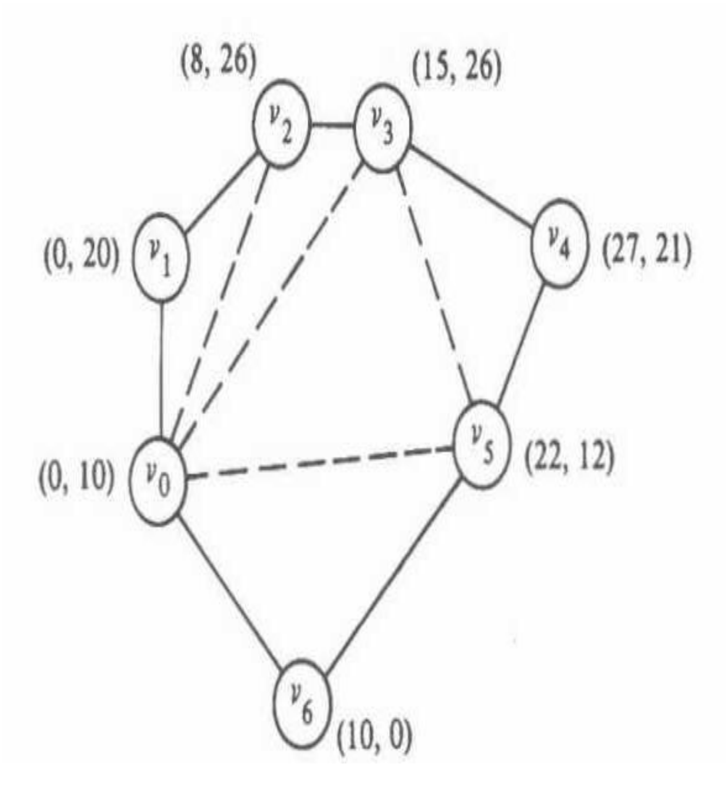
Costo de una triangulación

Costo= $D(v_o, v_2) + D(v_0, v_3) + D(v_3, v_5) + D(v_5, v_0)$

$D(v_i, v_j)$ distancia entre los puntos v_i y v_j

Programación dinámica
Triangulación de un polígono convexo

MATRIZ DE DISTANCIAS ENTRE VÉRTICES



	V_0	V_1	V_2	V_3	V_4	V_5	V_6
V_0	0	10	17,88	21,931	29,154	22,09	14,142
V_1		0	10	16,155	27,018	23,409	22,360
V_2			0	7	19,646	19,798	26,076
V_3				0	13	15,652	26,476
V_4					0	10,295	27,018
V_5						0	16,970
V_6							0

$$\sqrt[2]{(\mathbf{8-0})^2+(\mathbf{26-20})^2}$$

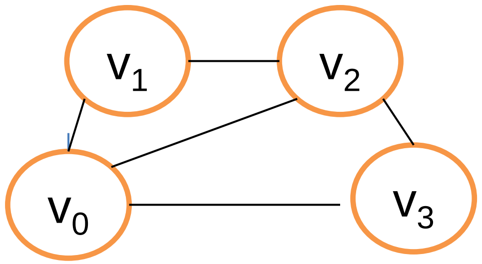
Programación dinámica
Triangulación de un polígono convexo

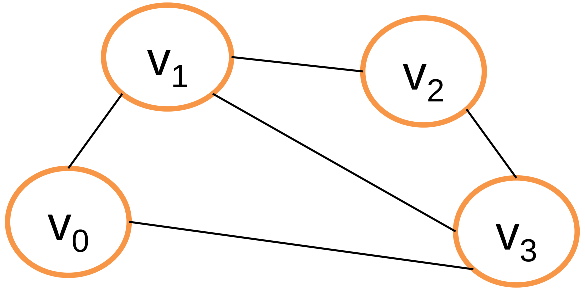
Sea C_{ij} el costo del problema S_{ij}

S_{ij} es el subproblema que incluye j vértices a partir de v_i tomados en sentido horario

$D(v_i, v_j)$ distancia de v_i a v_j (si $\overline{v_i v_j}$ es lado del polígono $D(v_i,v_j)$es cero)

S_{04}





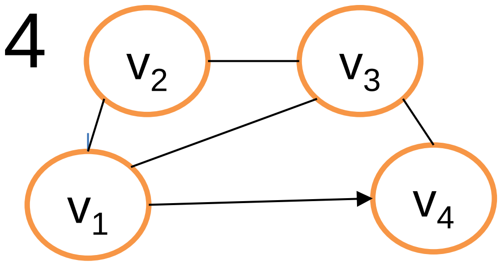
C_{04} = min ($D(v_0,v_2)$, $D(v_1,v_3)$)= min (17,88 ;16,155)

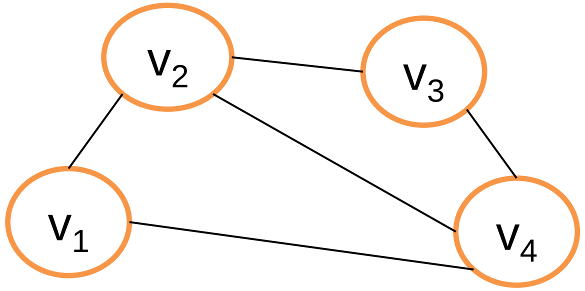
Programación dinámica
Triangulación de un polígono convexo

Sea C_{ij} el costo del problema S_{ij}

S_{ij} es el subproblema que incluye j vértices a partir de v_i tomados en sentido horario

S_{14}

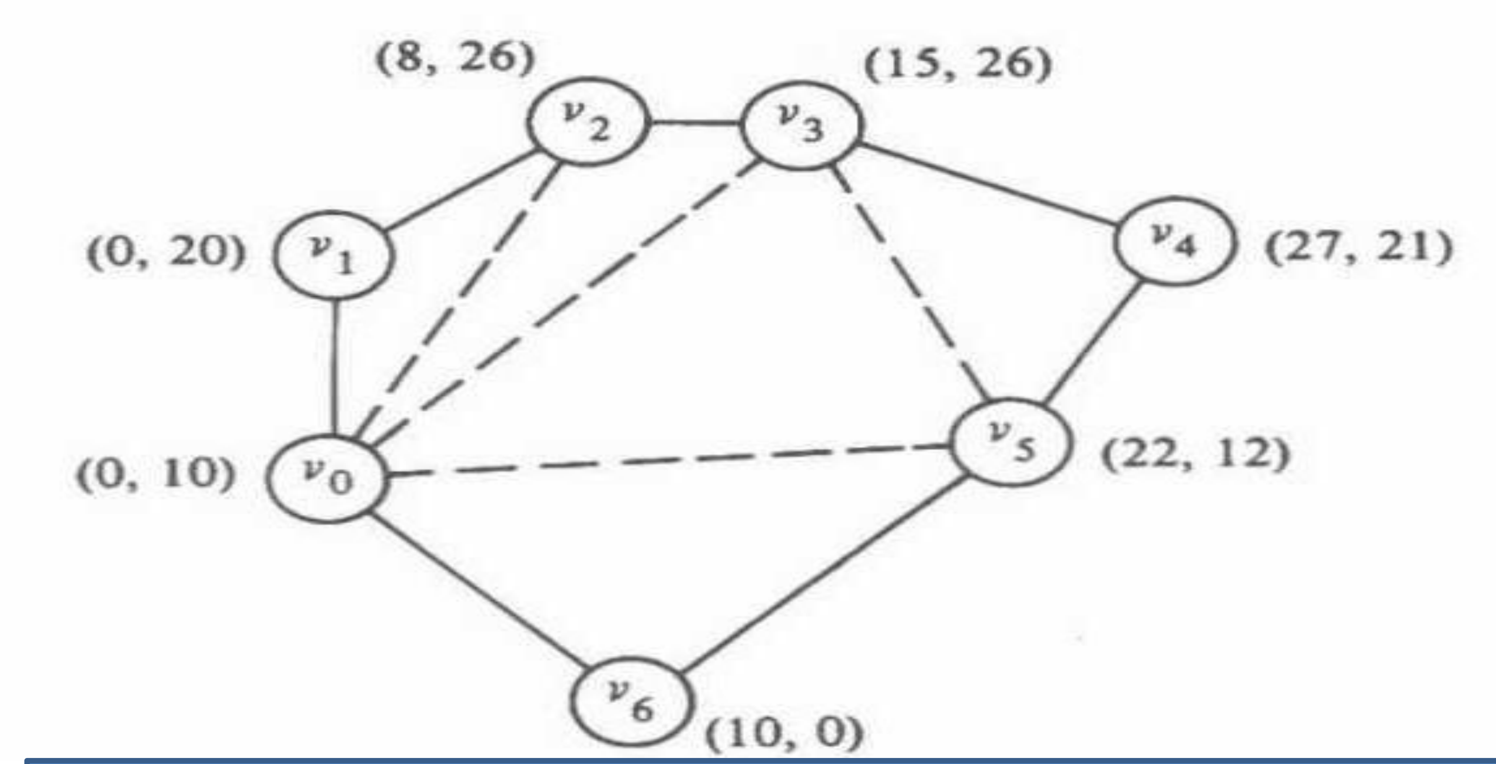




C_{14} = min ($D(v_1,v_3)$, $D(v_2,v_4)$) = min (16,155; 19,646)

¿CUÁNTOS SUBRPOBLEMAS DE TAMAÑO 4 HAY? S_{04} S_{14} S_{24} S_{34} S_{44} S_{54} S_{64}

Programación dinámica
Triangulación de un polígono convexo

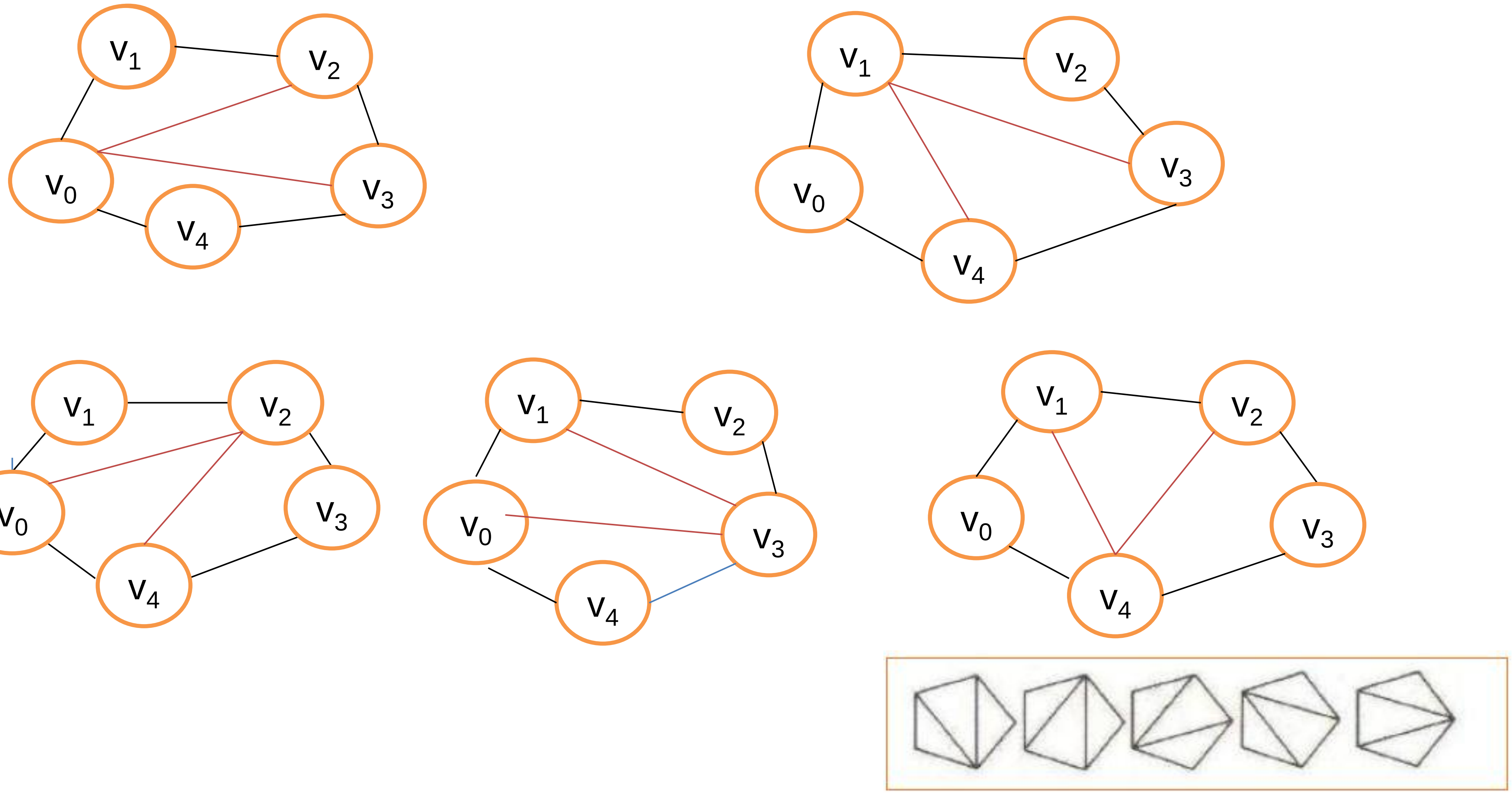


- ¿CUÁNTOS SUBRPOBLEMAS DE TAMAÑO 4 HAY?

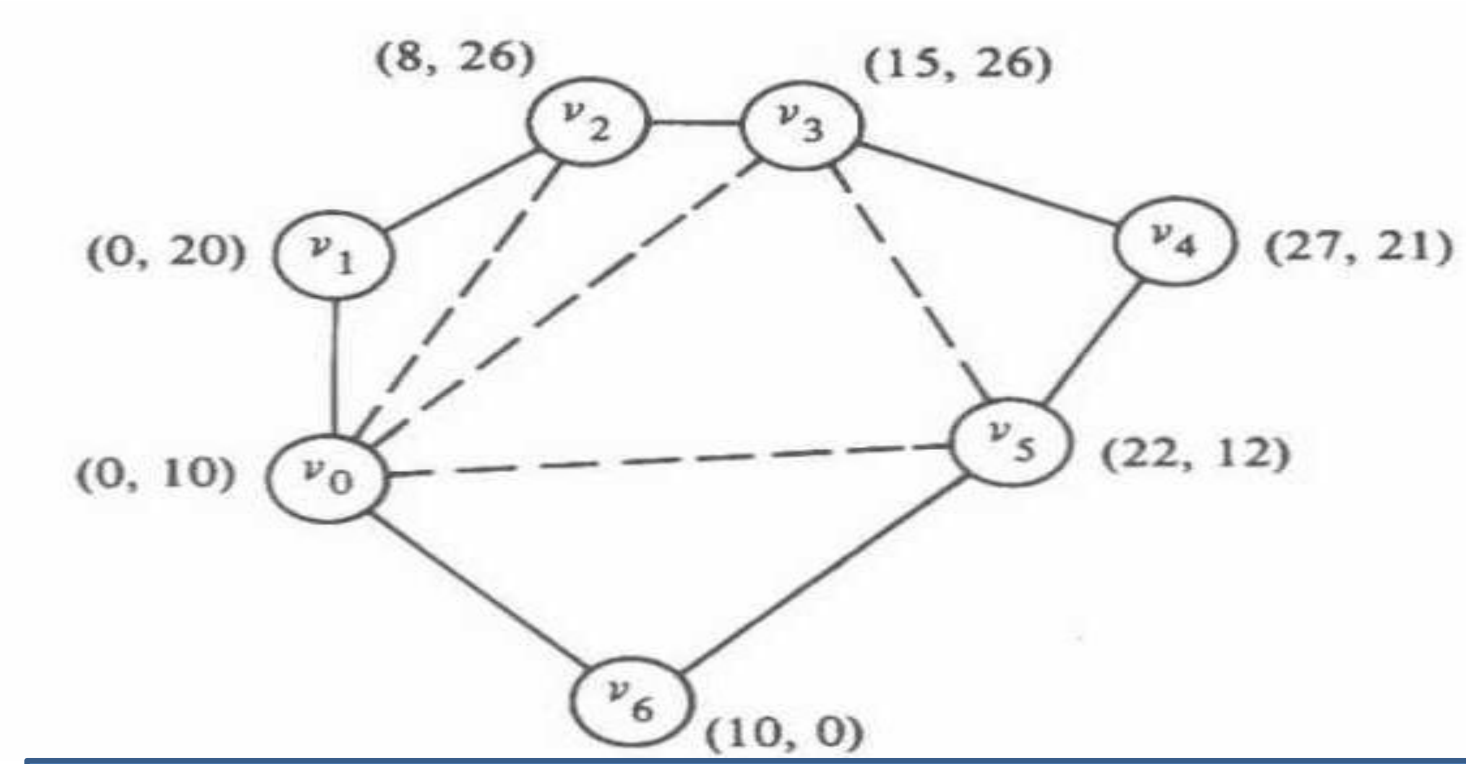
S_{04} S_{14} S_{24} S_{34} S_{44} S_{54} S_{64}

Programación dinámica
Triangulación de un polígono convexo

S05



Programación dinámica
Triangulación de un polígono convexo

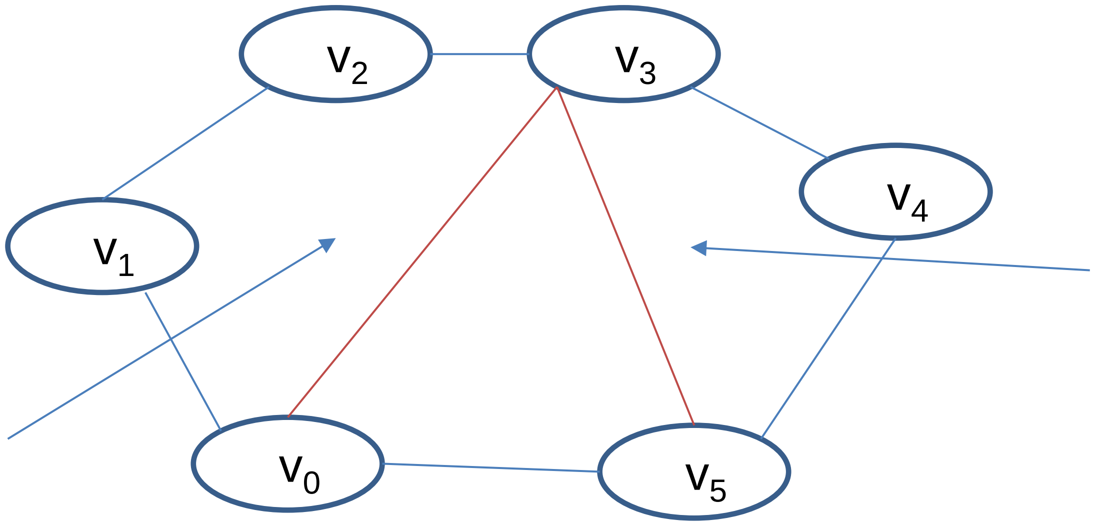


- ¿CUÁNTOS SUBRPOBLEMAS DE TAMAÑO 5 HAY?

S_{05} S_{15} S_{25} S_{35} S_{45} S_{55} S_{65}

Programación dinámica
Triangulación de un polígono convexo

S_{06}



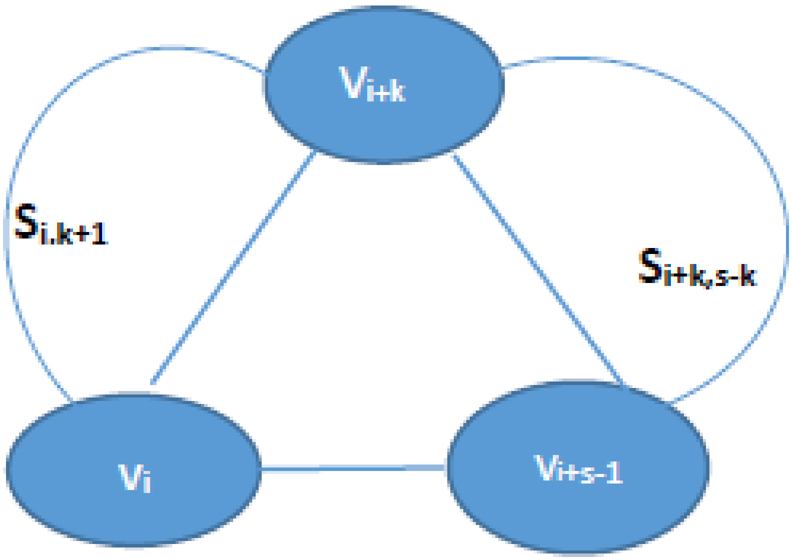
S_{33} = 0

S_{04}
está calculado

Programación dinámica
Triangulación de un polígono convexo

GENERALIZACIÓN Sea C_{ij} el costo del problema S_{ij}

S_{is}

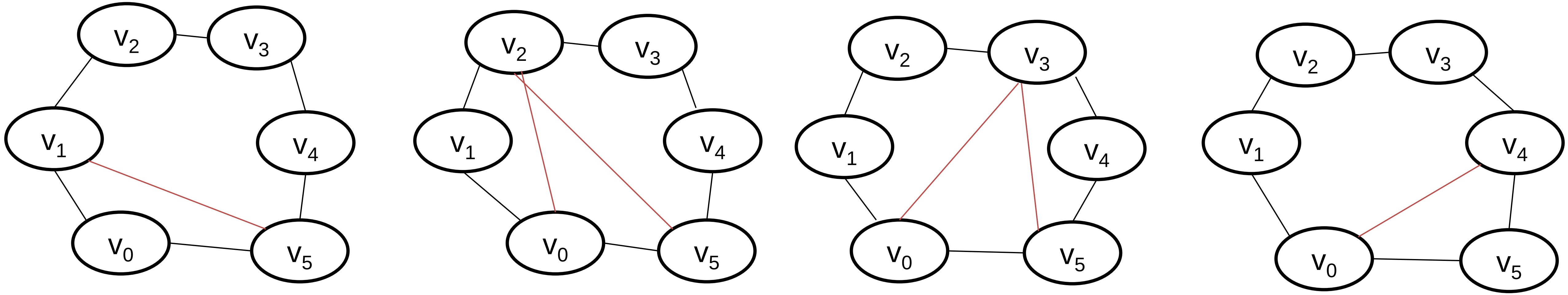


$$C_{is} = \min_{1 \le k \le s-2}\left(C_{i(k+1)} + C_{(i+k)(s-k)} + D\left(v_i, v_{(i+k)}\right) + D\left(v_{(i+k)}, v_{(i+s-1)}\right)\right)$$

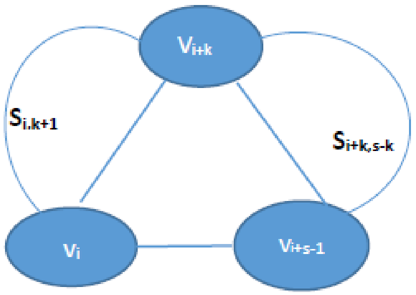
Programación dinámica
Triangulación de un polígono convexo

S_{06} S_{15}

$$C_{is} = \min_{1 \le k \le s-2} \left(C_{i(k+1)} + C_{(i+k)(s-k)} + D(v_i, v_{(i+k)}) + D(v_{(i+k)}, v_{(i+s-1)}) \right)$$



K=1



K=2

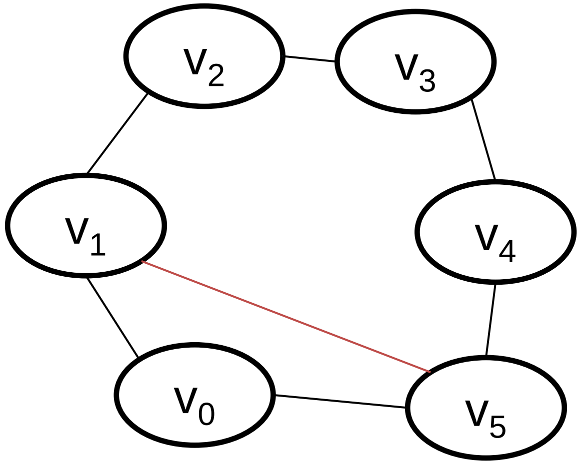
k=3

k=4

Programación dinámica
Triangulación de un polígono convexo

S_{06} S_{15}

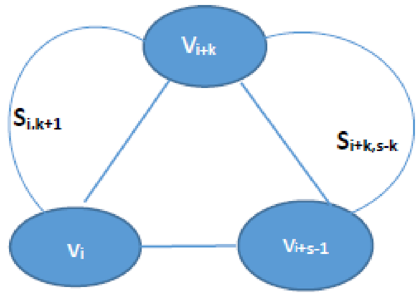
$$C_{is} = \min_{1 \le k \le s-2}\left(C_{i(k+1)} + C_{(i+k)(s-k)} + D(v_i, v_{(i+k)}) + D(v_{(i+k)}, v_{(i+s-1)})\right)$$



i=0 s=6

K=1 $C_{02} + C_{15} + D(v_0, v_1) + D(v_1, v_5)$

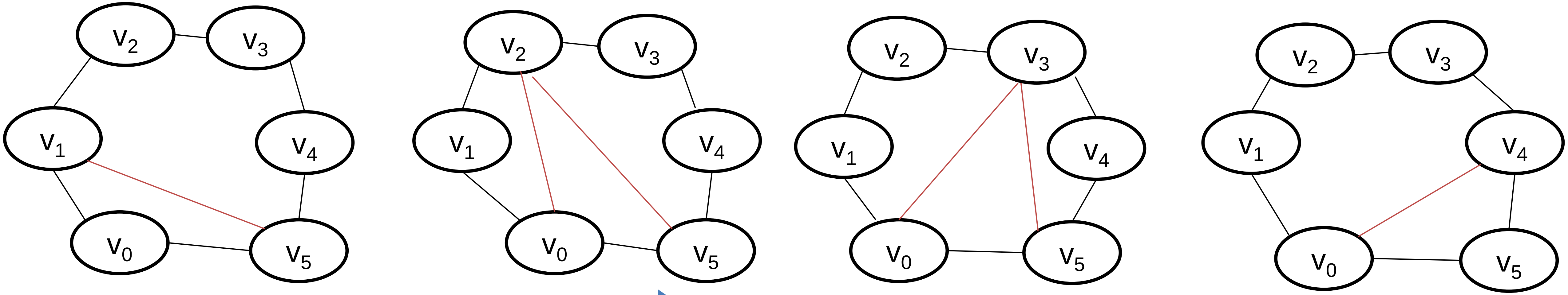
K=1



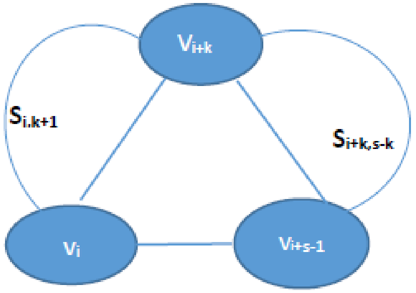
Programación dinámica
Triangulación de un polígono convexo

S_{06} S_{15}

$$C_{is} = \min_{1\le k\le s-2}\left(C_{i(k+1)} + C_{(i+k)(s-k)} + D(v_i, v_{(i+k)}) + D(v_{(i+k)}, v_{(i+s-1)})\right)$$



K=1



i=0 s=6

K=2 $C_{03} + C_{24} + D(v_0, v_2) + D(v_2, v_5)$

Programación dinámica
Triangulación de un polígono convexo

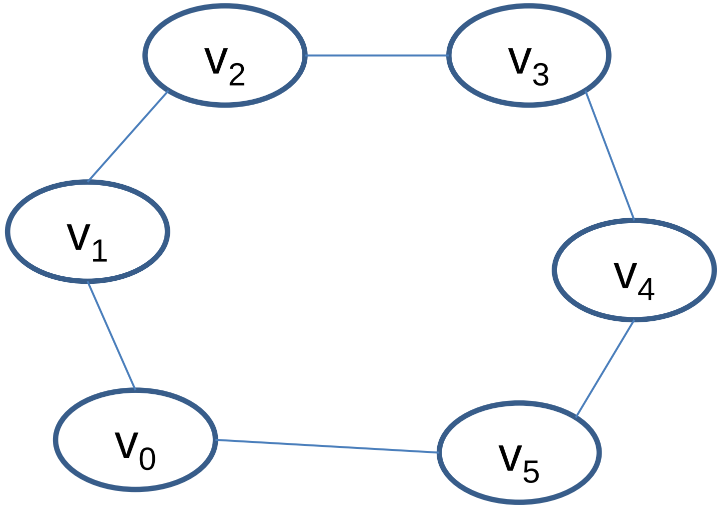
7	$C_{07}=$ 75.43						
6	$C_{06}=$ 53.54	$C_{16}=$ 55.22	$C_{26}=$ 57.58	$C_{36}=$ 64.69	$C_{46}=$ 59.78	$C_{56}=$ 59.78	$C_{66}=$ 63.62
5	$C_{05}=$ 37.54	$C_{15}=$ 31.81	$C_{25}=$ 35.49	$C_{35}=$ 37.74	$C_{45}=$ 45.50	$C_{55}=$ 39.98	$C_{65}=$ 38.09
4	$C_{04}=$ 16.16	$C_{14}=$ 16.16	$C_{24}=$ 15.65	$C_{34}=$ 15.65	$C_{44}=$ 22.69	$C_{54}=$ 22.69	$C_{64}=$ 17.89
s	$i=0$	1	2	3	4	5	6

Programación dinámica
Triangulación de un polígono convexo

7	$C_{07}=$ 75.43						
6	$C_{06}=$ 53.54	$C_{16}=$ 55.22	$C_{26}=$ 57.58	$C_{36}=$ 64.69	$C_{46}=$ 59.78	$C_{56}=$ 59.78	$C_{66}=$ 63.62
5	$C_{05}=$ 37.54	$C_{15}=$ 31.81	$C_{25}=$ 35.49	$C_{35}=$ 37.74	$C_{45}=$ 45.50	$C_{55}=$ 39.98	$C_{65}=$ 38.09
4	$C_{04}=$ 16.16	$C_{14}=$ 16.16	$C_{24}=$ 15.65	$C_{34}=$ 15.65	$C_{44}=$ 22.69	$C_{54}=$ 22.69	$C_{64}=$ 17.89
s	$i=0$	1	2	3	4	5	6

Sugerencia:
Implementarlo!
Encontrar el conjunto de diagonales de la triangulación mínima y su costo

Ejercicio



	V_0	V_1	V_2	V_3	V_4	V_5
V_0	0	10	17,88	21,931	29,154	22,09
V_1		0	10	16,155	27,018	23,409
V_2			0	7	19,646	19,798
V_3				0	13	15,652
V_4					0	10,295
V_5						0

$$\sqrt[2]{(8-0)^2-(26-20)^2}$$

Realice un seguimiento del algoritmo y verifique que genera las 14 triangulaciones!