#include <stdio.h>

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int max\_it(int \* a, int n){ // n = tamanho do array expresso pelo próprio parâmetro n do algoritmo.

// custo #rep #obs

// ----- ---- ----

int max = a[0]; // c1 1

int i = 1; // c2 1

while(i < n){ // c3 n i = 1 até n

if(a[i] > max){ // c4 (n - 1) i = 1 até (n - 1)

max = a[i]; // c5 x número de repetições depende também o estado do array

}

i++; // c6 (n - 1)

}

return max; // c7 1

}

// Análise [ determinar T(n) ]:

//

// T(n) = c1 + c2 + c3n + c4(n - 1) + c5x + c6(n - 1) + c7

//

// T(n) = c1 + c2 + c3n + c4n - c4 + c5x + c6n - c6 + c7

//

// T(n) = (c3 + c4 + c6)n + c5x + (c1 + c2 - c4 - c6 + c7)

//

// 2 cenários: melhor caso (menor valor possível para x) e pior caso (maior valor possível para x)

//

// -----------------------------------------------------------------------------------------------------------

//

// Melhor caso: x = 0 Tm(n) = (c3 + c4 + c6)n + (c1 + c2 - c4 - c6 + c7)

//

// Tm(n) = k1n + k2

//

// Tm(n) ~ k1n

//

// -----------------------------------------------------------------------------------------------------------

//

// Pior caso: x = (n - 1) Tp(n) = (c3 + c4 + c6)n + c5(n-1) + (c1 + c2 - c4 - c6 + c7)

//

// Tp(n) = (c3 + c4 + c5 + c6)n + (c1 + c2 - c4 - c5 - c6 + c7)

//

// Tp(n) = k3n + k4

//

// Tp(n) ~ k3n

//

// -----------------------------------------------------------------------------------------------------------

//

// Caso geral:

//

// k1n <= T(n) <= k3n

//

// T(n) ~ kn

//

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//

// Ideia:

//

// max( { 10, 7, 4, 2, 15, 9, 3 } ) = maior valor entre: 10 e max( { 7, 4, 2, 15, 9, 3 } )

//

//

// max( { 3, 7 } ) = maior entre o 3 e max ( { 7 } )

//

// max ( { 7 } ) = 7

//

int max1(int \* a, int ini, int tam){ // n da função T = tamanho do (sub)array dado por (tam - ini)

// custo #rep (base) #rep (geral) obs

// ----- ----------- ------------ ---

if(ini == tam - 1){ // c1 1 1

return a[ini]; // c2 1 0

}

int b = a[ini]; // c3 0 1

int c = max1(a, ini + 1, tam); // c7 + T(n - 1) 0 1 custo da chamada recursiva?

if(b > c){ // c4 0 1

return b; // c5 0 0/1

}

return c; // c6 0 1/0

}

// Caso base (n = 1): T(n) = c1 + c2 ---> T(n) = k1

//

//

// Caso geral (n > 1): T(n) = c1 + c3 + ??? + c4 + {c5,c6}

//

// T(n) = c1 + c3 + ??? + c4 + cret

//

// T(n) = c1 + c3 + (c7 + T(n - 1)) + c4 + cret

//

// T(n) = T(n - 1) + (c1 + c3 + c7 + c4 + cret)

//

// T(n) = T(n - 1) + k2

//

// ...

//

// T(n) = (n - 1)k2 + k1

// = nk2 - k2 + k1

// = k2n - (k1 - k2)

// = ~k2n

//

// n = 4 T(4) = T(3) + k2 = k1 + 2k2 + k2 = k1 + 3k2

// n = 3 T(3) = T(2) + k2 = k1 + k2 + k2 = k1 + 2k2

// n = 2 T(2) = T(1) + k2 = k1 + k2 = k1 + 1k2

// n = 1 T(1) = k1 = k1 + 0k2

//

// ...

//

// T(n) = ?????

//

// T(n) tempo que leva para resolver um problema de tamanho n.

// T(n - 1) tempo que leve para resolver um subproblema de tamanho (n - 1)

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Ideia:

//

// max( { 10, 7, 4, 2, 15, 9, 3, 11 } ) = maior valor entre: max({ 10, 7, 4, 2 }) e max({ 15, 9, 3, 11 })

//

int max2(int \* a, int ini, int fim){ // n = fim - ini + 1

int i;

// DEBUG:

/\*

for(i = ini; i <= fim; i++){

printf("%d ", a[i]);

}

printf("\n");

\*/

// custo #rep (base) #rep (geral)

// ----- ----------- ------------

if(ini == fim){ // c1 1 1

return a[ini]; // c2 1 0

}

int med = (ini + fim) / 2; // c3 0 1

int b = max2(a, ini, med); // c4 + T(n/2) 0 1

int c = max2(a, med + 1, fim); // c5 + T(n/2) 0 1

if(b > c){ // c6 0 1

return b; // c7 0 0/1

}

return c; // c8 0 1/0

}

// Caso base (n = 1): T(n) = c1 + c2 ---> T(n) = k1

//

//

// Caso geral (n > 1): T(n) = c1 + c3 + [c4 + T(n/2)] + [c5 + T(n/2)] + c6 + {c7,c8}

//

// T(n) = T(n/2) + T(n/2) + c1 + c3 + c4 + c5 + c6 + cret

//

// T(n) = 2T(n/2) + (c1 + c3 + c4 + c5 + c6 + cret)

//

// T(n) = 2T(n/2) + k2

//

// ...

//

// T(n) = ?????

//

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main(){

int a[] = { 7, 11, 20, 4, 9, 13, 32, 0, 8, 5, 16, 23 };

int n = sizeof(a) / sizeof(int);

printf("%d\n", max\_it(a, n));

printf("%d\n", max1(a, 0, n));

printf("%d\n", max2(a, 0, n - 1));

return 0;

}