Búsqueda Binaria + Dos punteros

Emanuel Lupi

Universidad Nacional de Córdoba

emanuel.lupi91@gmail.com

21 de Julio

Overview

Breve mirada sobre los algoritmos de ordenamiento

Es un algoritmo re-acomoda los elementos de una lista o un vector en una secuencia dada por una relación de orden.

Breve mirada sobre los algoritmos de ordenamiento

Es un algoritmo re-acomoda los elementos de una lista o un vector en una secuencia dada por una relación de orden.

- Insertion sort $O(n^2)$.
- Selection Sort $O(n^2)$.
- Heap Sort $O(n \log(n))$.
- Merge Sort $O(n \log(n))$.
- Quick Sort $O(n \log(n))$.

Breve mirada sobre los algoritmos de ordenamiento

Es un algoritmo re-acomoda los elementos de una lista o un vector en una secuencia dada por una relación de orden.

- Insertion sort $O(n^2)$.
- Selection Sort $O(n^2)$.
- Heap Sort $O(n \log(n))$.
- Merge Sort $O(n \log(n))$.
- Quick Sort $O(n \log(n))$.

$$\begin{pmatrix} -1 & 2 & -2 & 3 & 0 & 10 \end{pmatrix}$$

Breve mirada sobre los algoritmos de ordenamiento

Es un algoritmo re-acomoda los elementos de una lista o un vector en una secuencia dada por una relación de orden.

- Insertion sort $O(n^2)$.
- Selection Sort $O(n^2)$.
- Heap Sort $O(n \log(n))$.
- Merge Sort $O(n \log(n))$.
- Quick Sort $O(n \log(n))$.

$$\begin{pmatrix} -2 & -1 & 0 & 2 & 3 & 10 \end{pmatrix}$$

Búsqueda lineal

La búsqueda lineal consiste en iterar sobre todos los elementos para verificar si el valor objetivo está en el arreglo.

1	5	7	9	14	27	37	38	39	50	55	67	70	71	75

busquemos: $v = 1$ (1 iteración)

Búsqueda lineal

La búsqueda lineal consiste en iterar sobre todos los elementos para verificar si el valor objetivo está en el arreglo.

1	5	7	9	14	27	37	38	39	50	55	67	70	71	75

busquemos: $v = 5$ (2 iteraciones)

Búsqueda lineal

La búsqueda lineal consiste en iterar sobre todos los elementos para verificar si el valor objetivo está en el arreglo.

1	5	7	9	14	27	37	38	39	50	55	67	70	71	75

busquemos: $v = 70$ (13 iteraciones)

Búsqueda lineal

La búsqueda lineal consiste en iterar sobre todos los elementos para verificar si el valor objetivo está en el arreglo.

1	5	7	9	14	27	37	38	39	50	55	67	70	71	75

busquemos: $v = 3$ (15 iteraciones)

Definición aproximada de búsqueda binaria

Búsqueda binaria:

- Es un algoritmo de búsqueda que ecuentra (si existe) una posición de un valor en un array **ordenado** o punto en **intervalo**.

Definición aproximada de búsqueda binaria

Búsqueda binaria:

- Es un algoritmo de búsqueda que ecuentra (si existe) una posición de un valor en un array **ordenado** o punto en **intervalo**.
- La busqueda la realiza comparando el valor del elemento del medio del array (o intervalo) que se está observando.

Definición aproximada de búsqueda binaria

Búsqueda binaria:

- Es un algoritmo de búsqueda que ecuentra (si existe) una posición de un valor en un array **ordenado** o punto en **intervalo**.
- La busqueda la realiza comparando el valor del elemento del medio del array (o intervalo) que se está observando.
- si es el valor esperado se retorna el valor (*).

Definición aproximada de búsqueda binaria

Búsqueda binaria:

- Es un algoritmo de búsqueda que ecuentra (si existe) una posición de un valor en un array **ordenado** o punto en **intervalo**.
- La busqueda la realiza comparando el valor del elemento del medio del array (o intervalo) que se está observando.
- si es el valor esperado se retorna el valor (*).
- si no son iguales, la mitad en la cual el **valor puede estar** es utilizada para la siguiente iteración de la busqueda.

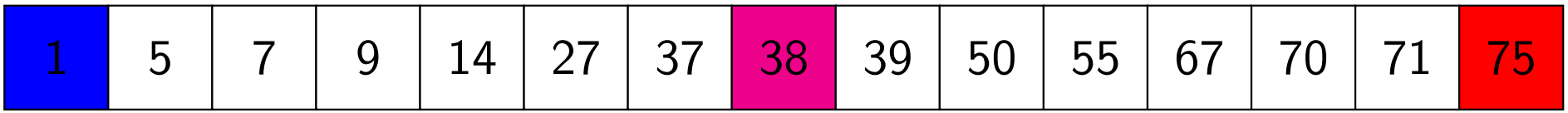
Definición aproximada de búsqueda binaria

Búsqueda binaria:

- Es un algoritmo de búsqueda que ecuentra (si existe) una posición de un valor en un array **ordenado** o punto en **intervalo**.
- La busqueda la realiza comparando el valor del elemento del medio del array (o intervalo) que se está observando.
- si es el valor esperado se retorna el valor (*).
- si no son iguales, la mitad en la cual el **valor puede estar** es utilizada para la siguiente iteración de la busqueda.
- tiene complejidad $O(long(n))$.

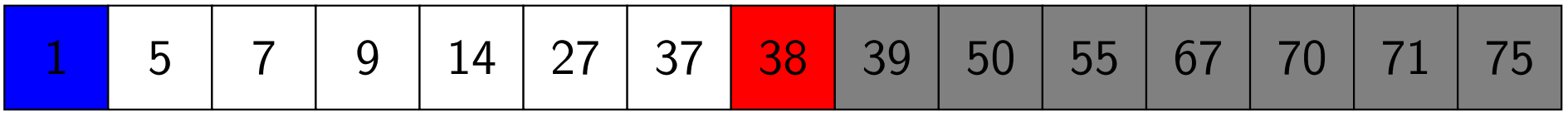
Algoritmo en acción

Busquemos el 7



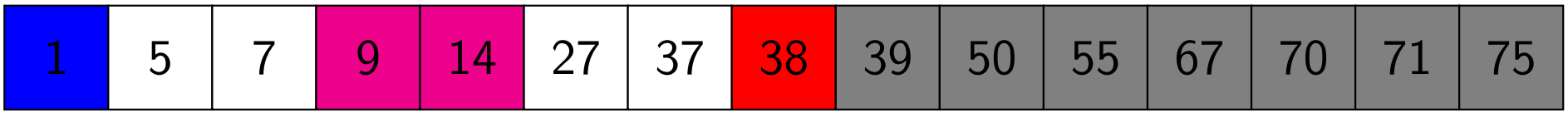
Algoritmo en acción

Busquemos el 7



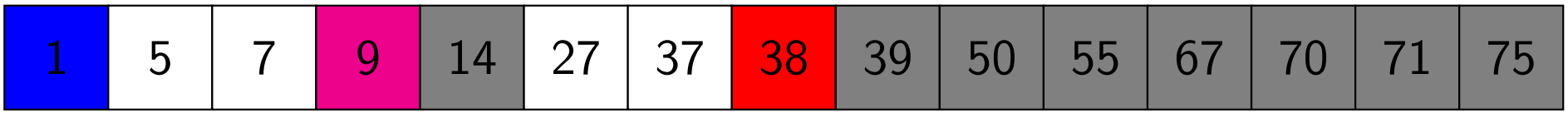
Algoritmo en acción

Busquemos el 7



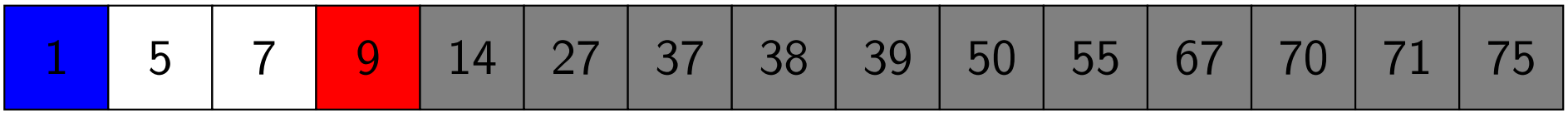
Algoritmo en acción

Busquemos el 7



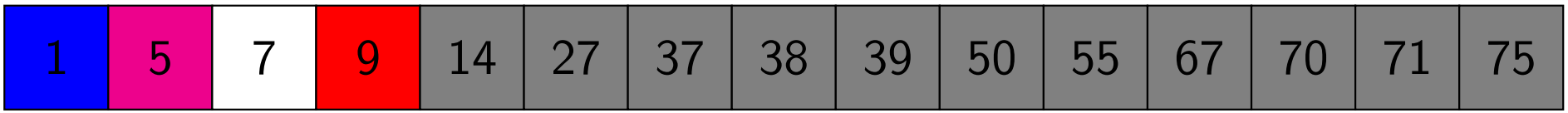
Algoritmo en acción

Busquemos el 7



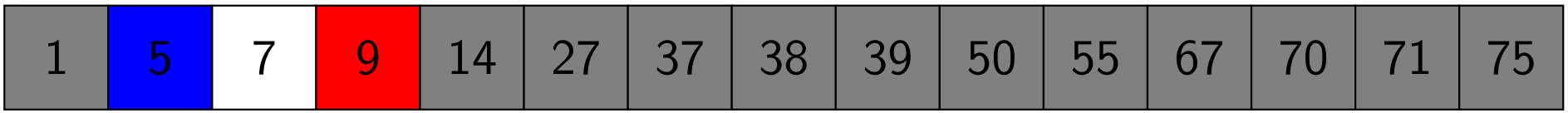
Algoritmo en acción

Busquemos el 7



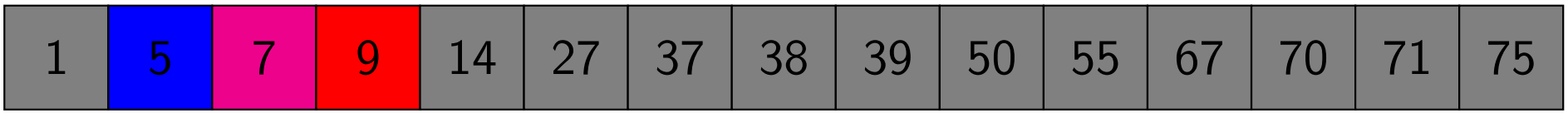
Algoritmo en acción

Busquemos el 7



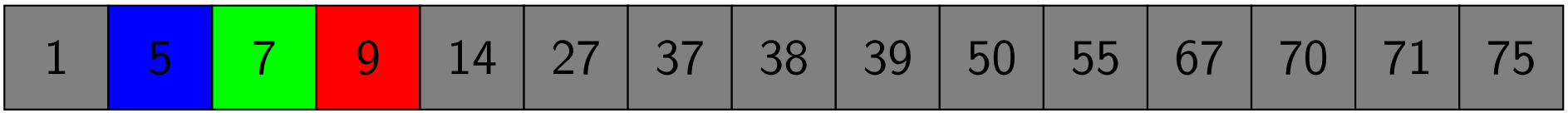
Algoritmo en acción

Busquemos el 7



Algoritmo en acción

Busquemos el 7

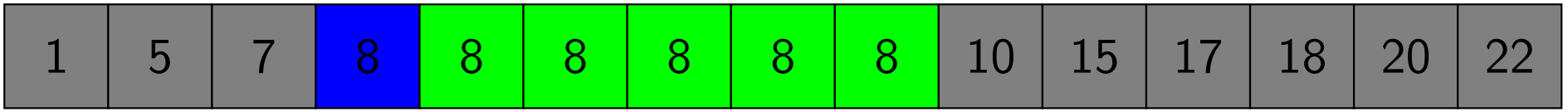


Implementación

```
1  int bsearch(vector<int> & a, int value){
2      int lo = 0;
3      int hi = a.size();
4      while(lo + 1 < hi){
5          int mid=(lo+hi)/2;
6          if(value < A[mid])
7              hi = mid;
8          else
9              lo = mid;
10     }
11     //if a[lo] is value
12     if(a[lo] == value)
13         return lo;
14     else
15         return -1;
16 }
```

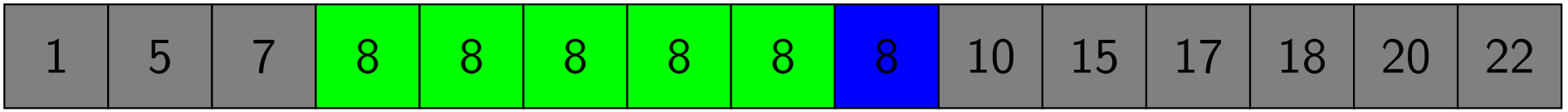
Detalles

Qué pasa si buscamos la primera aparición de un número?



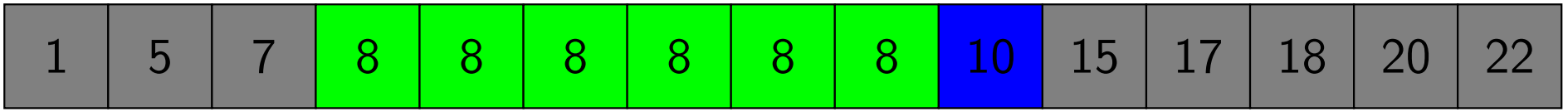
Detalles

Qué pasa si buscamos la última aparición de un número?



Detalles

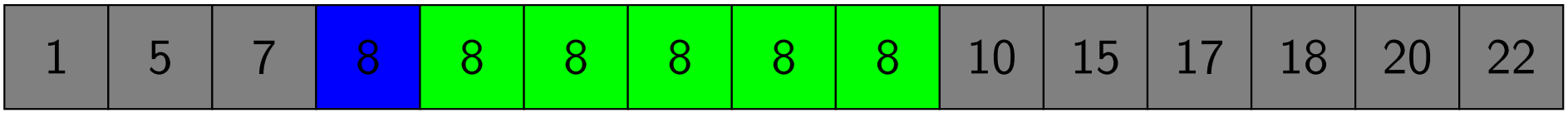
Qué pasa si buscamos el inmediatamente mayor de un número?



Detalles

Esos casos también pueden ser resueltos con busqueda binaria. Solo hay que ajustar un poco el algoritmo. En C++ y Java hay funciones que hacen lo que necesitamos.

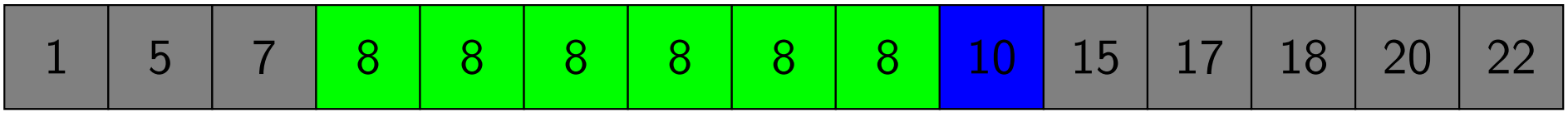
- lower_bound: da el primer iterador cuyo valor no sea menor que el valor de búsqueda (o sea, el primero mayor o igual).
  $lower\_bound(v.begin(), v.end(), 8);$



Detalles

Esos casos también pueden ser resueltos con busqueda binaria. Solo hay que ajustar un poco el algoritmo. En C++ y Java hay funciones que hacen lo que necesitamos.

- upper_bound: da el primer iterador cuyo valor sea mayor que el valor de búsqueda. $upper\_bound(v.begin(), v.end(), 8)$;



Implementación lower bound

```
1  int bs_lower_bound(int a[], int n, int x) {
2      int l = 0;
3      int h = n; // Not n - 1
4      while (l < h) {
5          int mid =  l + (h - l) / 2;
6          if (x <= a[mid]) {
7              h = mid;
8          } else {
9              l = mid + 1;
10         }
11     }
12     return l;
13 }
```

Implementación upper bound

```
1  int bs_upper_bound(int a[], int n, int x) {
2      int l = 0;
3      int h = n; // Not n - 1
4      while (l < h) {
5          int mid =  l + (h - l) / 2;
6          if (x >= a[mid]) {
7              l = mid + 1;
8          } else {
9              h = mid;
10         }
11     }
12     return l;
13 }
```

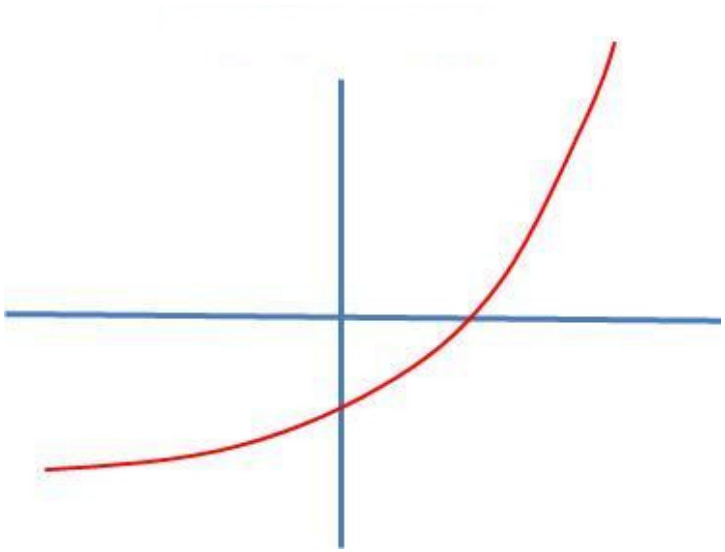
Aplicaciones

La búsqueda binaria puede ser utilizada para **buscar** soluciones o valores a proposiciones cumplan lo siguiente.

$$\left(\begin{array}{cccccc} P(x_1) = F & P(x_2) = F & \ldots & P(x_{n-1}) = F & P(x_n) = T & \ldots \end{array} \right)$$

Aplicaciones

Como buscar el "0" funciones con segmento crecientes (o decrecientes)



Implementación

```
1 long double binSearch( long double hi ){
2   long double lo = 0 ;
3   while(hi - lo >= 1e-7){
4     long double mid = ( lo + hi ) / 2.0;
5     if(f(mid) > 0) hi = mid;
6     else lo = mid;
7   }
8   return lo;
9 }
```

Aplicaciones

Se acerca mi cumpleaños y siempre sirvo pie. No solo un pie, no, tengo varios N. F de mis amigos van a venir a mi fiesta y cada uno de ellos recibe un trozo de pie. Esto debe ser una porción de un pie, no varias porciones pequeñas, ya que se ve desordenado. Sin embargo, esta porción puede ser un pastel completo.
Cuál es el tamaño de porción más grande posible que todos podemos obtener?

Aplicaciones - Solución

- En este problema sabemos que para ciertos tamaños de porciones se pueden repartir a los amigos, como por ejemplo dar una porción de tamaño 0.

Aplicaciones - Solución

- En este problema sabemos que para ciertos tamaños de porciones se pueden repartir a los amigos, como por ejemplo dar una porción de tamaño 0.
- Sabemos que el máximo tamaño posible de porción es el del pie más grande. Aunque puede que no alcance para todos.

Aplicaciones - Solución

- En este problema sabemos que para ciertos tamaños de porciones se pueden repartir a los amigos, como por ejemplo dar una porción de tamaño 0.
- Sabemos que el máximo tamaño posible de porción es el del pie más grande. Aunque puede que no alcance para todos.
- Sabemos que para un cierto tamaño M ya no podemos servir una porción más grande que esa.

Aplicaciones - Solución

- En este problema sabemos que para ciertos tamaños de porciones se pueden repartir a los amigos, como por ejemplo dar una porción de tamaño 0.
- Sabemos que el máximo tamaño posible de porción es el del pie más grande. Aunque puede que no alcance para todos.
- Sabemos que para un cierto tamaño M ya no podemos servir una porción más grande que esa.
- Lo ven o no lo ven!.

Aplicaciones - Solución

Entonces el problema se podría ver como 2 sub-problemas más sencillos.

- Definir el predicado $P(t)$ que sea *true* si se puede repartir porciones de tamaño t.

Aplicaciones - Solución

Entonces el problema se podría ver como 2 sub-problemas más sencillos.

- Definir el predicado $P(t)$ que sea *true* si se puede repartir porciones de tamaño t.
- Hacer una busqueda binaria buscando el mayor t que satisfaga P.

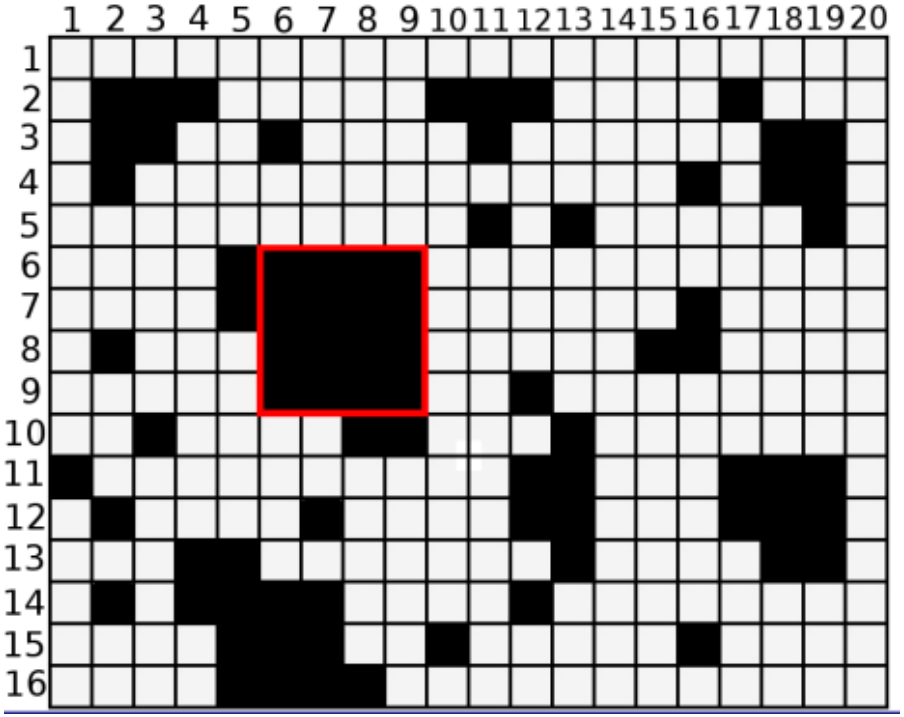
P(t_1)=T	P(t_2)=t	$\cdots$	P(t_{n-1})=T	P(t_n)=F	$\cdots$

Implementación

```
1 long double binSearch( long double hi ){
2   long double lo = 0 ;
3   while(hi - lo >= 1e-7){
4     long double mid = ( lo + hi ) / 2.0;
5     if(pred(mid)) lo = mid;
6     else hi = mid;
7   }
8   return lo;
9 }
```

Aplicaciones - problema 2

Tenemos un tablero de mn (m filas y n columnas). Algunas casillas del tablero están pintadas de blanco y otras de negro. Buscamos encontrar el tamaño del lado del subtablero cuadrado más grande de casillas negras.



Búsqueda binaria

Implementación

```
1 int binSearch(int hi){
2   int lo = 0 ;
3   while(lo + 1 < hi){
4     int mid = (lo + hi) / 2;
5     if(pred(mid)) lo = mid;
6     else hi = mid;
7   }
8   return lo;
9 }
```

Preguntas

Ventana deslizante

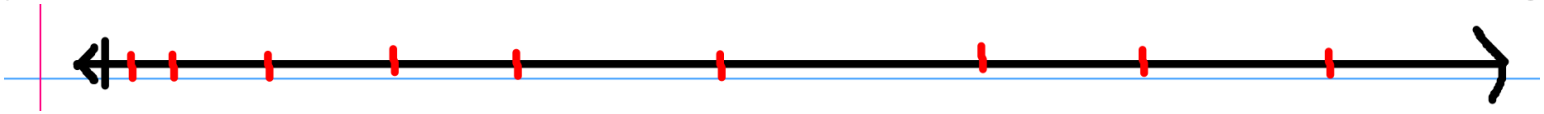
En muchos problemas se puede utilizar esta técnica para evitar tener un doble for loop. La idea es mantener dos índices (o un índices y el tamaño de la ventana) e ir haciendo crecer ambos según el problema lo requiera. Estos índices nunca "retroceden".

Ejercicio

Emanuel tiene la ropa en el tendedero que está en el patio. Ve que se viene una tormenta. A Emanuel no le importa que se le moje un poco la ropa ya que luego sale el sol y se seca. Pero si le interesa guardar K prendas, ya que de no guardar se quedará sin ropa. Como Emanuel es perezoso quiere juntar K prendas caminando la mínima distancia posible cargando ropa.

Ejercicio

Emanuel tiene la ropa en el tendedero que está en el patio. Ve que se viene una tormenta. A Emanuel no le importa que se le moje un poco la ropa ya que luego sale el sol y se seca. Pero si le interesa guardar K prendas, ya que de no guardar se quedará sin ropa. Como Emanuel es perezoso quiere juntar K prendas caminando la mínima distancia posible cargando ropa.



$$(\ -2 \quad -1 \quad 1 \quad 7 \quad 10 \quad 17 \quad 29 \quad 35 \quad 40 \ )$$

Implementación

```
1  int minWalk(vector<int> A, int k){
2    sort(A.begin(), A.end());
3    int result = -1
4    for(int i=0 ; i + k - 1<A.size() ; i++){
5      if(result == -1 || A[i + k - 1] - A[i] + 1 < result){
6        result = A[i + k - 1] - A[i] + 1
7      }
8    }
9    return result;
10 }
```

Preguntas

Dos Punteros

- Se utilizan 2 iteradores (o punteros) l y u.

Dos Punteros

- Se utilizan 2 iteradores (o punteros) l y u.
- l está al inicio del intervalo observado.

Dos Punteros

- Se utilizan 2 iteradores (o punteros) l y u.
- l está al inicio del intervalo observado.
- u está al final del intervalo observado.

Ejemplo

Dado un arreglo de n números **positivos**, y un número k, nos interesa saber si existe un subarreglo cuya suma sea k.
Queremos saber si existe un sub-array de suma $k = 12$

Implementación mala $O(n^2)$

Implementación en $O(n^2)$

```
1  int search(vector<int> & a, int k){
2    for(int i=0; i<a.size() ; i++){
3      int s = 0;
4      for(int j=i ; j<a.size() ; j++){
5        s += a[j];
6        if(s == k)
7          return true;
8      }
9    }
10   return false;
11 }
```

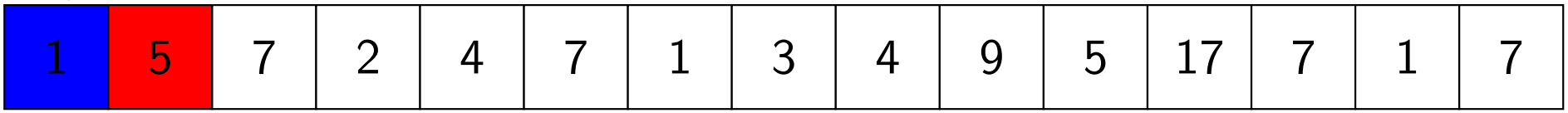
Ejemplo de ejecución con dos punteros

La ejecución es de la siguiente manera:
Objetivo suma $k = 12$

Ejemplo de ejecución con dos punteros

La ejecución es de la siguiente manera:

Objetivo suma $k = 12$



$Suma = 1$ $l = 0$ $u = 1$

Ejemplo de ejecución con dos punteros

La ejecución es de la siguiente manera:

Objetivo suma $k = 12$

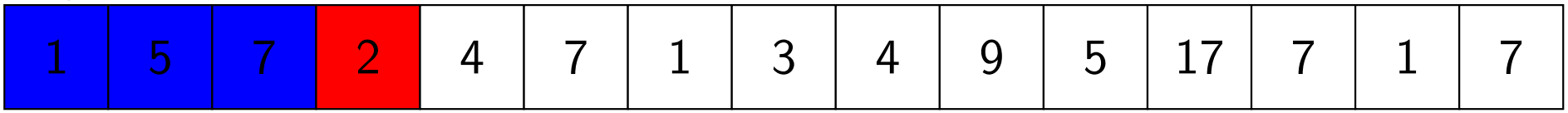
1	5	7	2	4	7	1	3	4	9	5	17	7	1	7

$Suma = 6$ $l = 0$ $u = 2$

Ejemplo de ejecución con dos punteros

La ejecución es de la siguiente manera:

Objetivo suma $k = 12$

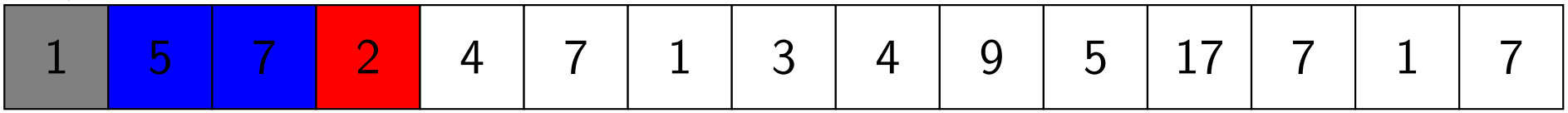


$Suma = 13$ $l = 0$ $u = 3$

Ejemplo de ejecución con dos punteros

La ejecución es de la siguiente manera:

Objetivo suma $k = 12$



$Suma = 12$ $l = 1$ $u = 3$

Ejemplo

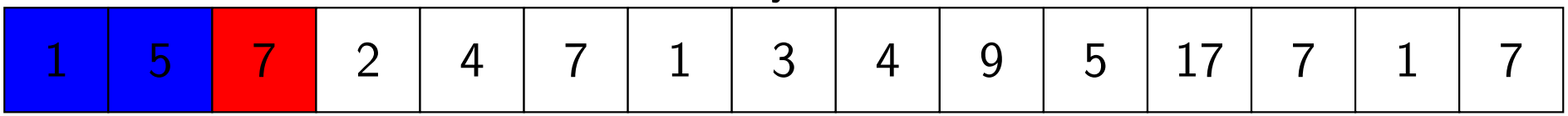
Queremos saber si existe un sub-array de suma $k = 21$

1	5	7	2	4	7	1	3	4	9	5	17	7	1	7

$Suma = 1$ $l = 0$ $u = 1$

Ejemplo

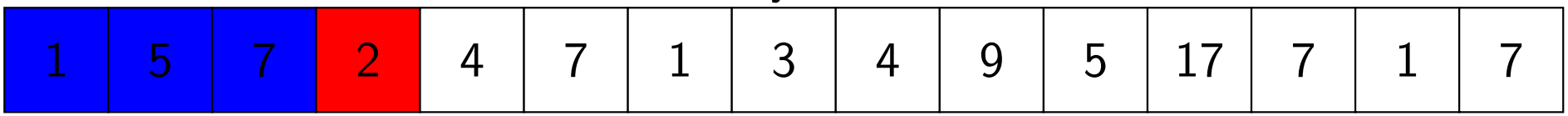
Queremos saber si existe un sub-array de suma $k = 21$



$Suma = 6$ $l = 0$ $u = 2$

Ejemplo

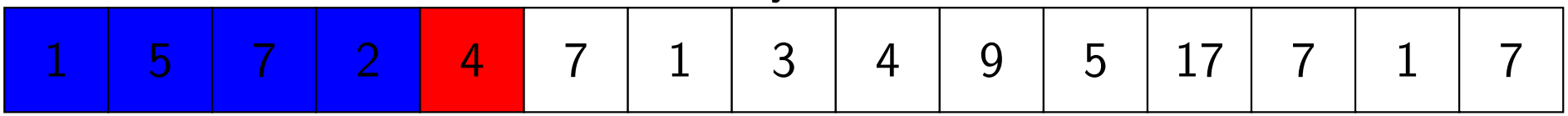
Queremos saber si existe un sub-array de suma $k = 21$



$Suma = 13$ $l = 0$ $u = 3$

Ejemplo

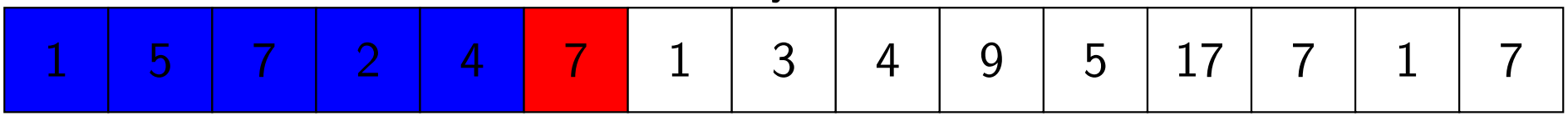
Queremos saber si existe un sub-array de suma $k = 21$



$Suma = 15$ $l = 0$ $u = 4$

Ejemplo

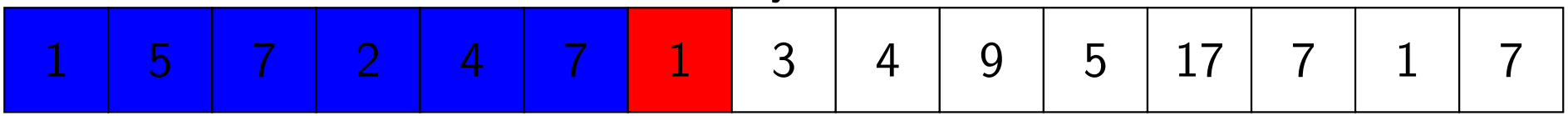
Queremos saber si existe un sub-array de suma $k = 21$



$Suma = 19$ $l = 0$ $u = 5$

Ejemplo

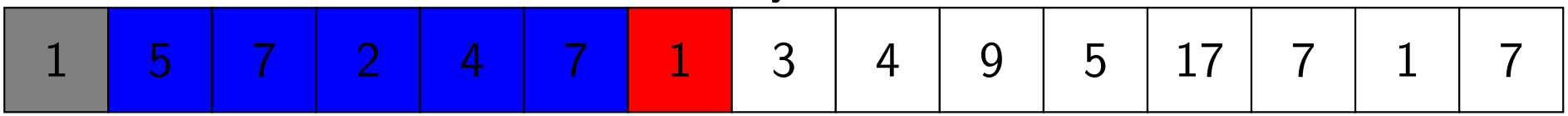
Queremos saber si existe un sub-array de suma $k = 21$



$Suma = 26$ $l = 0$ $u = 6$

Ejemplo

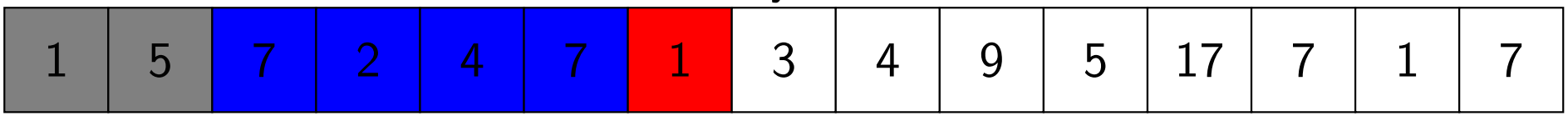
Queremos saber si existe un sub-array de suma $k = 21$



$Suma = 25$ $l = 1$ $u = 6$

Ejemplo

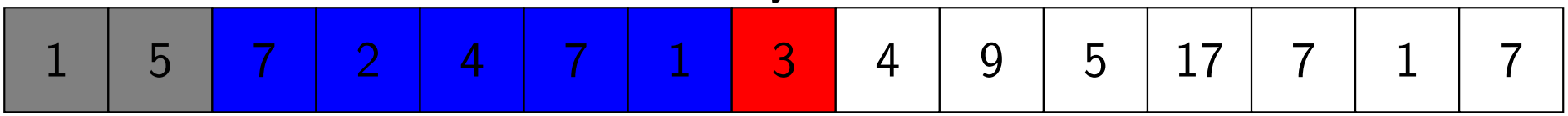
Queremos saber si existe un sub-array de suma $k = 21$



$Suma = 20$ $l = 2$ $u = 6$

Ejemplo

Queremos saber si existe un sub-array de suma $k = 21$



$Suma = 21$ $l = 3$ $u = 6$

Notas

- En cada iteración vemos que: o crece l o crece u.
- No puede pasar que no se mueva ni l ni u.

Implementación buena

Implementación en $O(n)$

```
1  int search(vector<int> & a, int k){
2    int l=0, u=0, s=0;
3    while(l<a.size() && s != k){
4      if(s > k || u == a.size())
5        s -= a[l++];
6      else
7        s += a[u++];
8    }
9    return s == k;
10 }
```

2-Sum problema

Enunciado:

Dado un arreglo de n números y un número x. Queremos encontrar dos números del arreglo que sumen x, o reportar que no hay tal par.

Notas

- La forma fácil de resolverlo es utilizando un doble for $O(n^2)$

Notas

- La forma fácil de resolverlo es utilizando un doble for $O(n^2)$
- Este problema se puede resolver con un hash_map en $O(n)$.

Notas

- La forma fácil de resolverlo es utilizando un doble for $O(n^2)$
- Este problema se puede resolver con un hash_map en $O(n)$.
- Igualmente lo vamos a resolver con dos punteros $O(n \log(n))$.

Implementación mala $O(n^2)$

```
1 bool search(vector<int> & a, int k){
2   for(int i=0 ; i<a.size() ; i++){
3     for(int j=i+1 ; j<a.size() ; j++){
4       if(a[i] + a[j] == k) return true;
5     }
6   }
7   return false;
8 }
```

Observación

Observación

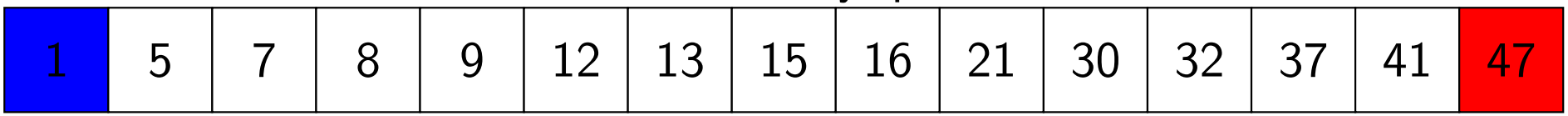
Es importante notar que el orden de los números en el arreglo A no juega ningún rol. Entonces podemos reordenar A. En particular, podemos ordenar el arreglo A de forma creciente $O(n \log(n))$.

Observación

Queres ver si existen 2 números del array que sumen $k = 15$

Observación

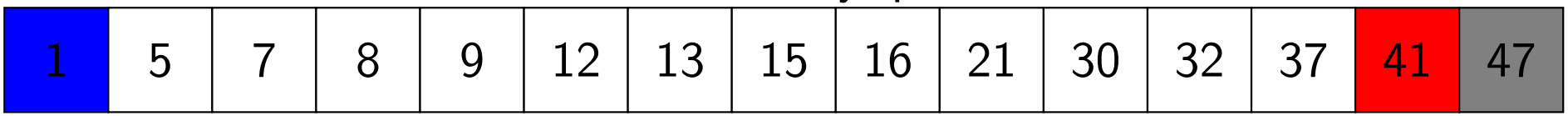
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 48$

Observación

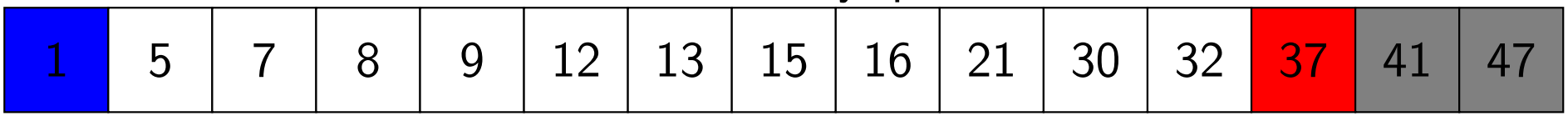
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 42$

Observación

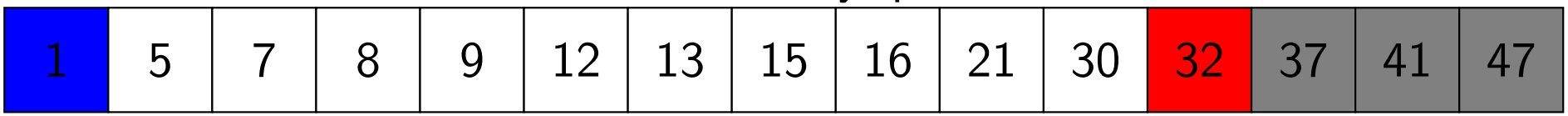
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 38$

Observación

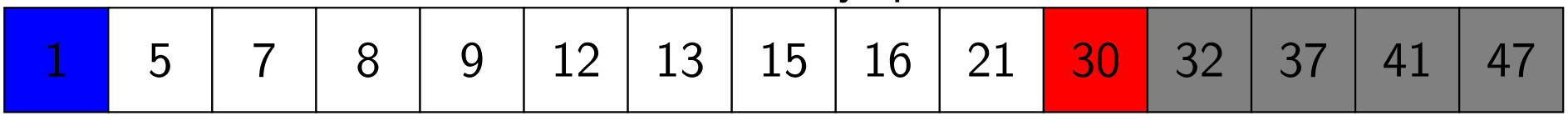
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 33$

Observación

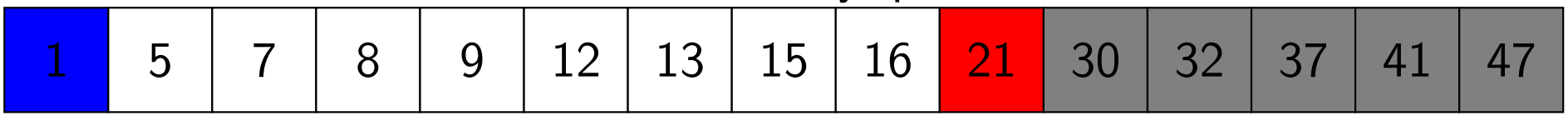
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 31$

Observación

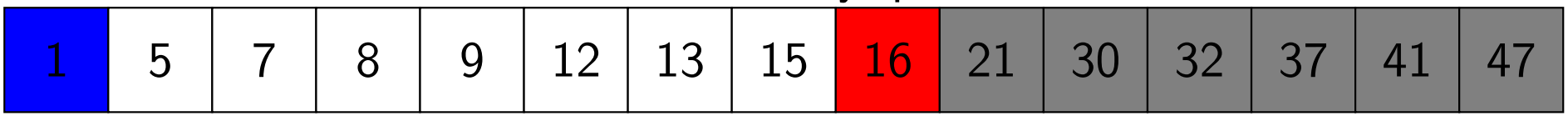
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 22$

Observación

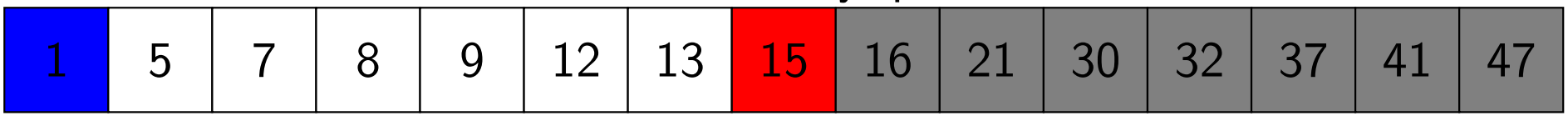
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 17$

Observación

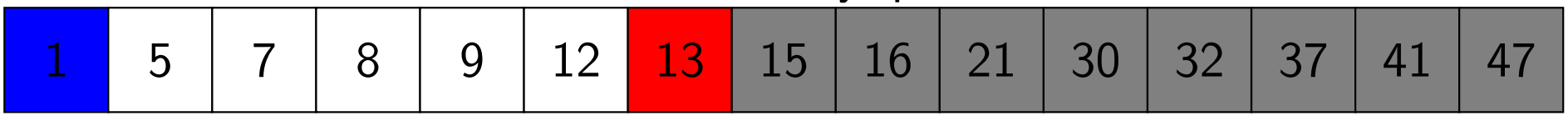
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 16$

Observación

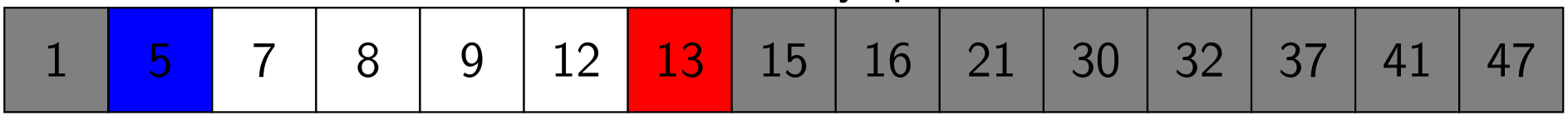
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 14$

Observación

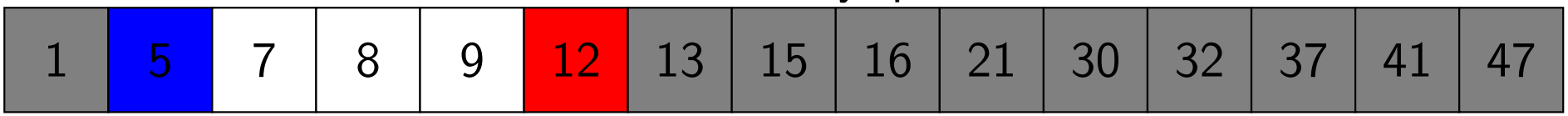
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 18$

Observación

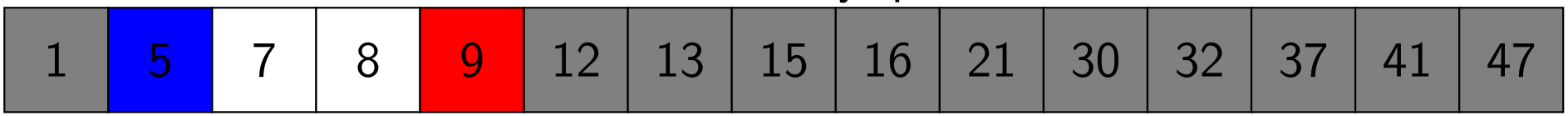
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 17$

Observación

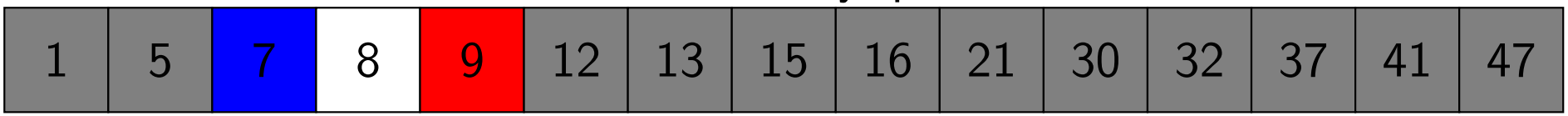
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 14$

Observación

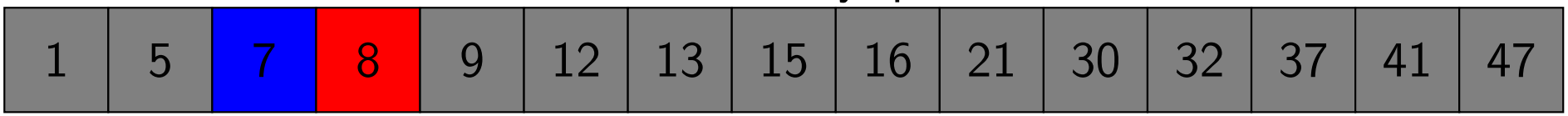
Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 16$

Observación

Queres ver si existen 2 números del array que sumen $k = 15$



$Suma = 15$

Referencias

SPOJ

PIE problem: https://www.spoj.com/problems/PIE/

.