Competitive Programming Workshop: Divide y Vencerás

Giovanny Alfonso Chávez Ceniceros

Universidad Autónoma de Chihuahua
Facultad de Ingeniería

gchavezceniceros@acm.org

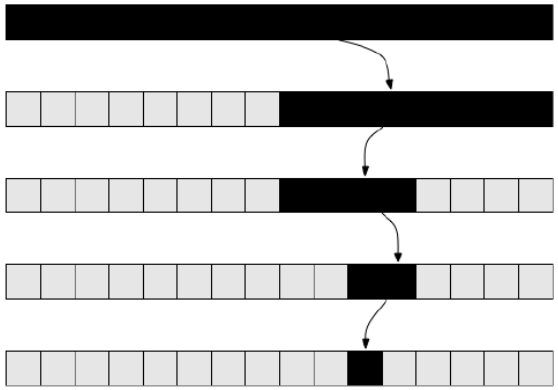
26 de marzo de 2019

Contenido

Divide y Vencerás

- Los algoritmos que emplean el enfoque de Divide y vencerás *dividen* el problema en ejemplares más pequeños del mismo problema (conjuntos más pequeños), luego resuelven (*vencen*) los ejemplares más pequeños de forma recursiva (o sea, empleando el mismo método) y por último *combinan* las soluciones para obtener la solución correspondiente a la entrada original.

Búsqueda en arreglo ordenado



- Dado un arreglo A que contiene n elementos en orden no decreciente, y dado un valor k, encontrar un indice i para el cual $k = A[i]$ o bien, si k no está en el arreglo, devolver -1 como respuesta.

Búsqueda binaria recursiva

```
1  int binarySearch(const vi &u, int low, int high, int key)
2  {
3      if(high < low) return -1;
4
5      int mid = low + (high - low)/2;
6
7      if(key == u[mid])
8          return mid;
9
10     else if(key < u[mid])
11         return binarySearch(u, low, mid - 1, key);
12     else
13         return binarySearch(u, mid + 1, high, key);
14 }
```

Búsqueda binaria iterativa

```
1  int itBinarySearch(const vi &u, int low, int high, int key)
2  {
3      while(low <= high)
4      {
5          int mid = low + (high - low)/2;
6
7          if(key == u[mid]) return mid;
8
9          else if(key < u[mid])
10             high = mid - 1;
11         else
12             low = mid + 1;
13     }
14     return -1;
15 }
```

Relaciones de recurrencia

- Una relación de recurrencia define una función sobre los números naturales, digamos $T(n)$, en términos de su propio valor con uno o más enteros menores que n. En otras palabras, $T(n)$ se define recursivamente.

Relaciones de recurrencia del tipo divide y vencerás

- Sea $T(n)$ el tiempo de ejecución de un problema de tamaño n. Si el tamaño del problema es lo suficientemente pequeño, digamos $n \leq c$ entonces la solución directa toma un tiempo constante $\Theta(1)$.
- Además, supongamos que nuestra división del problema arroja a subproblemas cuyo tamaño es $\frac{1}{b}$ veces el tamaño original. Entonces toma un tiempo $aT(\frac{n}{b})$ resolver todos y cada uno de los subproblemas generados.

Relaciones de recurrencia del tipo divide y vencerás

Si tomamos $D(n)$ el costo de dividir el problema y $C(n)$ el costo de combinar las soluciones generadas, tenemos que:

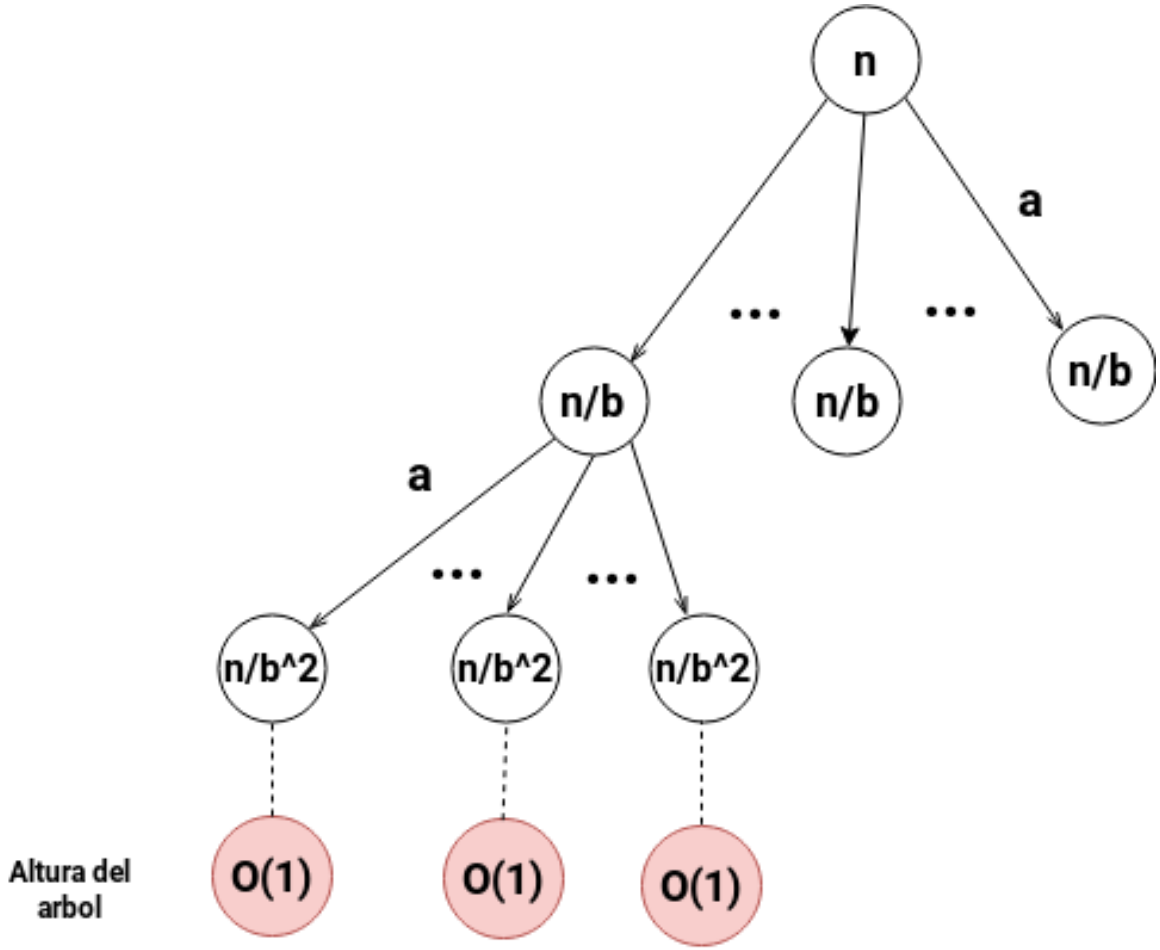
$$T(n) = \Theta(1)$$

para $n \leq c$ y

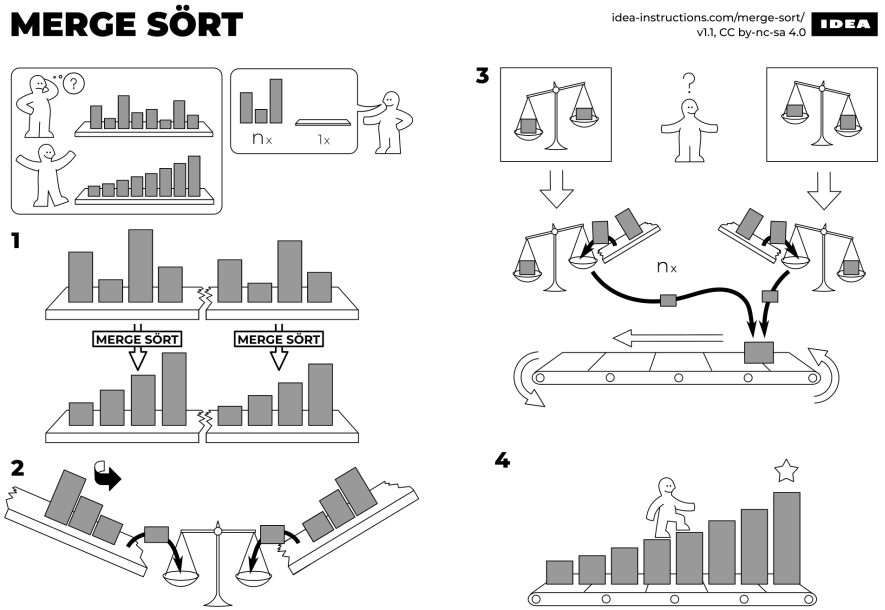
$$T(n) = aT(\frac{n}{b}) + D(n) + C(n)$$

en cualquier otro caso.

Árbol de recursión



Implementación del algoritmo mergesort

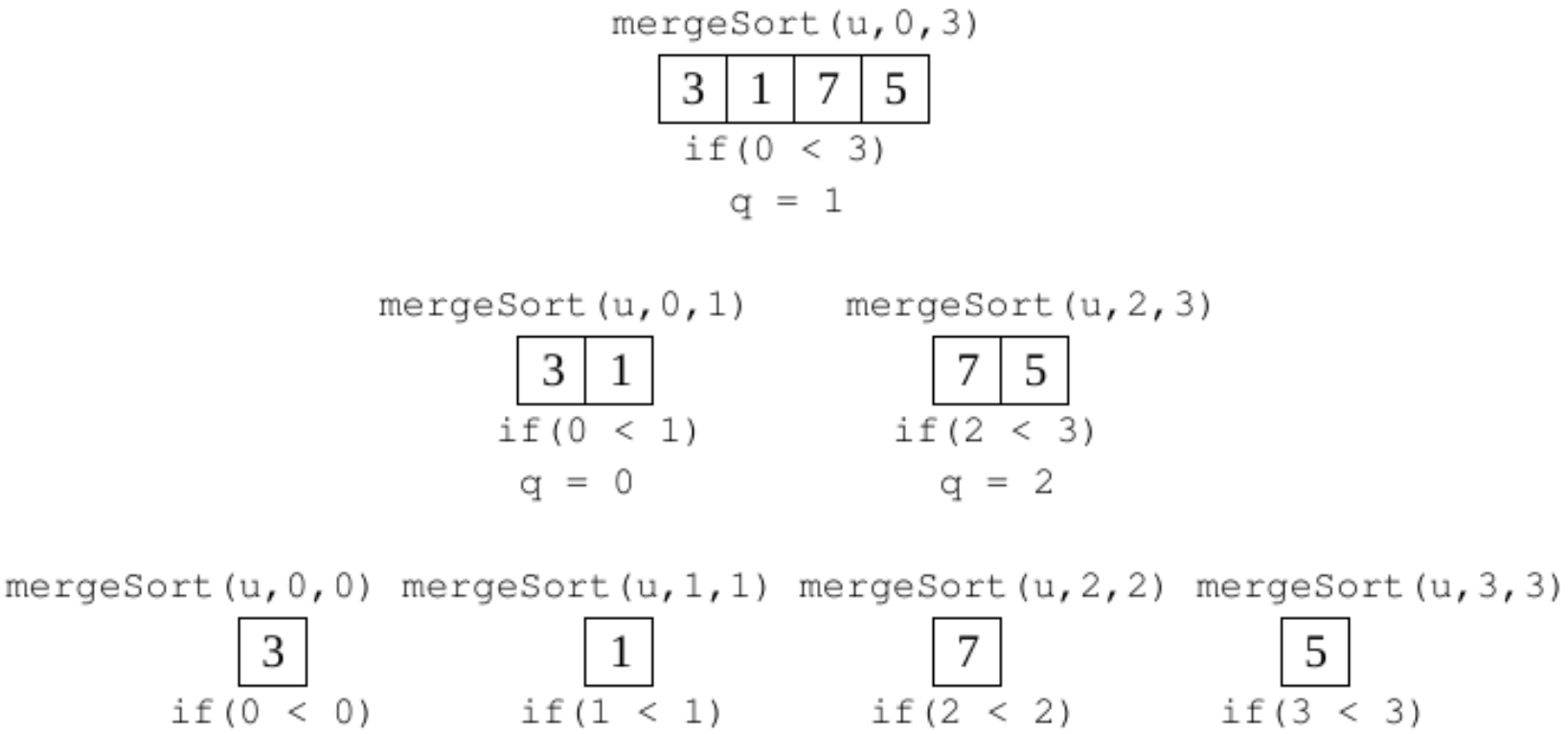


Mergesort es un algoritmo de ordenamiento basado en la técnica divide y vencerás. Simplemente parte un arreglo en dos partes y las ordena por separado para luego fusionar las partes ordenadas.

Implementación del algoritmo mergesort

```
1  /**
2   * @brief  Mergesort algorithm.
3   */
4  void mergesort(vi &u, int p, int r)
5  {
6      if(p < r)
7      {
8          int q = (p + r)/2;       // D(n)     => O(1)
9          mergesort(u, p, q);      // aT(n/b) => 2T(n/2)
10         mergesort(u, q + 1, r); //
11         merge(u, p, q, r);       // C(n)     => O(n)
12     }
13 }
```

Implementación del algoritmo mergesort

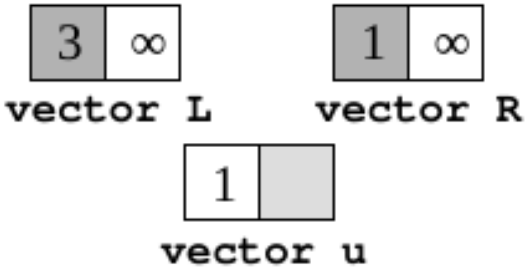


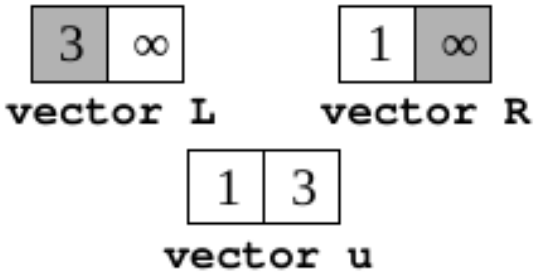
Algoritmo de mezclado implementando sentinelas

```
1  void merge(vi &u, int p, int q, int r)
2  {
3      vi L, R;
4      L.reserve(q - p + 1); R.reserve(r - q + 1 + 1);
5
6      for(int i = p; i <= q; ++i)   L.push_back(u[i]); // u[p..q]
7      for(int j = q+1; j <= r; ++j) R.push_back(u[j]); // u[q+1..r]
8
9      L.push_back(std::numeric_limits<int>::max()); // sentinel L
10     R.push_back(std::numeric_limits<int>::max()); // sentinel R
11
12     int i = 0, j = 0;
13     for(int k = p; k <= r; ++k)
14         u[k] = (L[i] <= R[j]) ? L[i++] : R[j++];
15 }
```

Algoritmo de mezclado implementando sentinelas

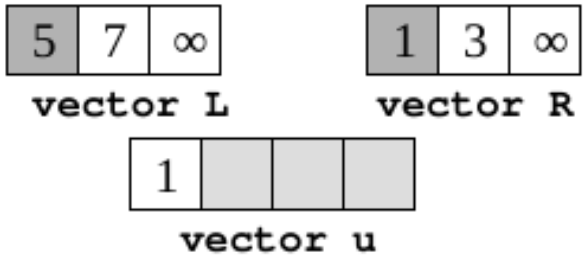
```
for(int k = 0; k <= 1; ++k){
    if(L[0] <= R[0])
        u[0] = L[0]
    else
        u[0] = R[0]
}
for(k = 1; k <= 1; ++k){
    if(L[0] <= R[1])
        u[1] = L[0]
    else
        u[1] = R[0]
}
```



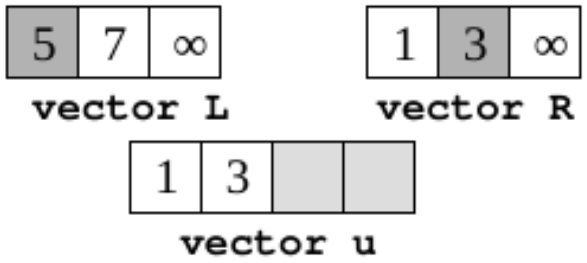


Algoritmo de mezclado implementando sentinelas

```
for(int k = 0; k <= 3; ++k){
    if(L[0] <= R[0])
        u[0] = L[0]
    else
        u[0] = R[0]
}
```

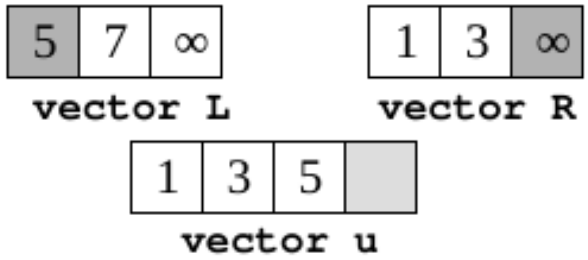



```
for(k = 1; k <= 3; ++k){
    if(L[0] <= R[0])
        u[0] = L[0]
    else
        u[1] = R[1]
}
```

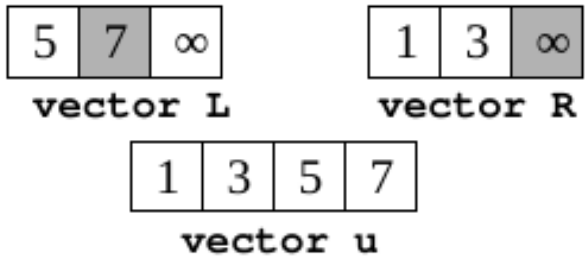


Algoritmo de mezclado implementando sentinelas

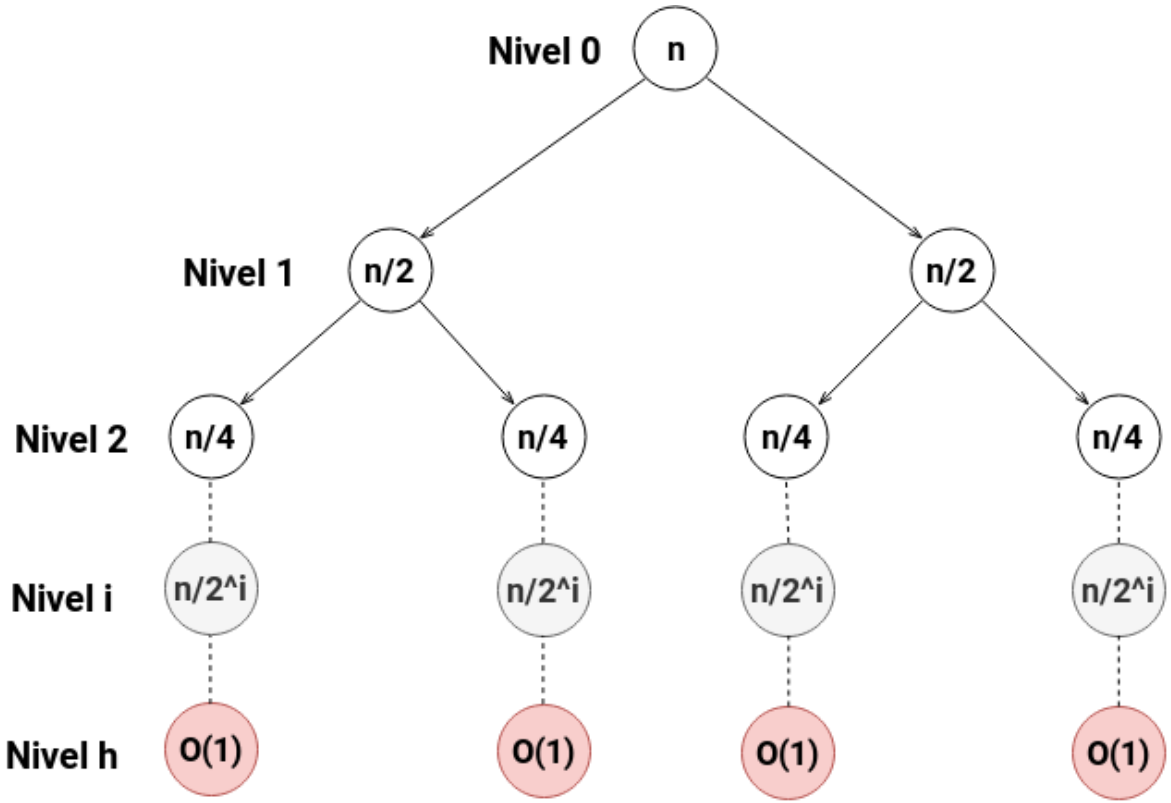
```
for(k = 2; k <= 3; ++k){
    if(L[0] <= R[0])
        u[2] = L[0]
    else
        u[0] = R[0]
}
```




```
for(k = 3; k <= 3; ++k){
    if(L[0] <= R[0])
        u[2] = L[0]
    else
        u[0] = R[0]
}
```



Crecimiento asintótico del mergesort



Crecimiento asintótico del mergesort

El i-ésimo término queda definido como $\frac{n}{2^i}$ por lo que la longitud en el nivel i del árbol se define como $i = lgn$ y $h = i + 1 = lgn + 1$
Como el costo por nivel n no varía, el costo total se obtiene como

$$\sum_{i=1}^{lgn+1} n = (lgn + 1)n = \Theta(nlgn)$$

Numero de inversiones en un arreglo

- El número de inversiones en un arreglo indica a qué distancia (o que tan cerca) está un arreglo de estar ordenado. Si el arreglo ya está ordenado, el conteo de inversiones es 0. Si el arreglo está ordenado en orden inverso, el conteo de inversiones es el máximo posible.
- Dos elementos $a[i]$ y $a[j]$ de un arreglo forman una inversión si $a[i] > a[j]$ e $i < j$

Algoritmo ingenuo

El siguiente algoritmo calcula el número de inversiones en un arreglo con una complejidad de $O(n^2)$.

```
1  int naiveInversions(const vi &u, int n)
2  {
3      int I = 0;
4      for(int i = 0; i < n - 1; i++)
5          for(int j = i + 1; j < n; j++)
6              if(u[i] > u[j]) I++;
7      return I;
8  }
```

Sin embargo es posible aplicar el paradigma divide y vencerás para obtener una complejidad $O(nlgn)$.

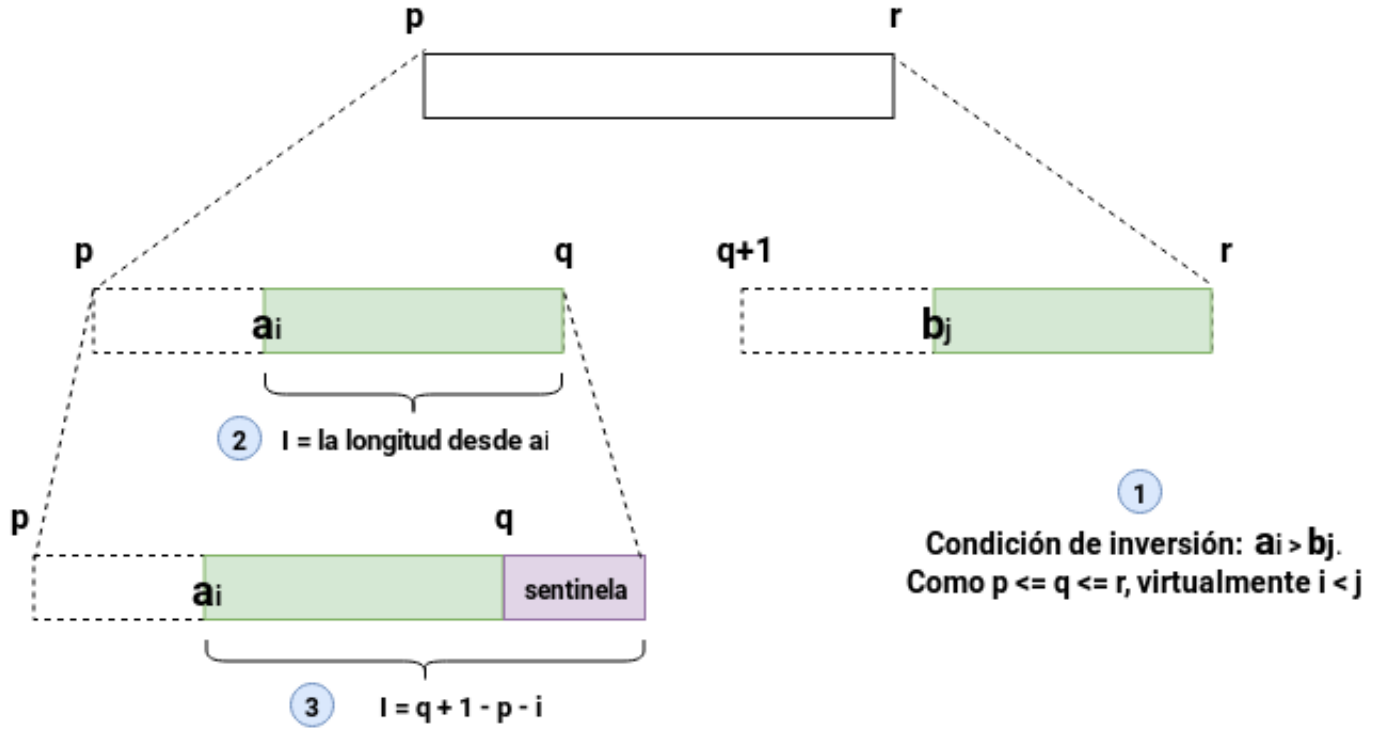
Número de inversiones usando mergesort

Supongamos que conocemos el número de inversiones en la mitad izquierda y en la mitad derecha del arreglo, ¿qué tipo de inversiones no se tienen en cuenta en $I_1 + I_2$?

```
1  int mergesort(vi &u, int p, int r)
2  {
3      int I = 0;
4      if(p <  r)
5      {
6          int q = (p + r)/2;
7          I  = mergesort(u, p, q);
8          I += mergesort(u, q + 1, r);
9          I += merge(u,  p, q, r);
10     }
11     return I;
12 }
```

Número de inversiones usando mergesort

Las inversiones que tenemos que contar durante el paso de mezclado.



Referencias


Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2009)
Introduction to algorithms


Skiena, S. S. (2003)
Programing Challenges. The Programming Contest Training Manual