Összefésüléses rendezés alkalmazásai

Szerző: Nikházy László
Előadó: Varga Péter

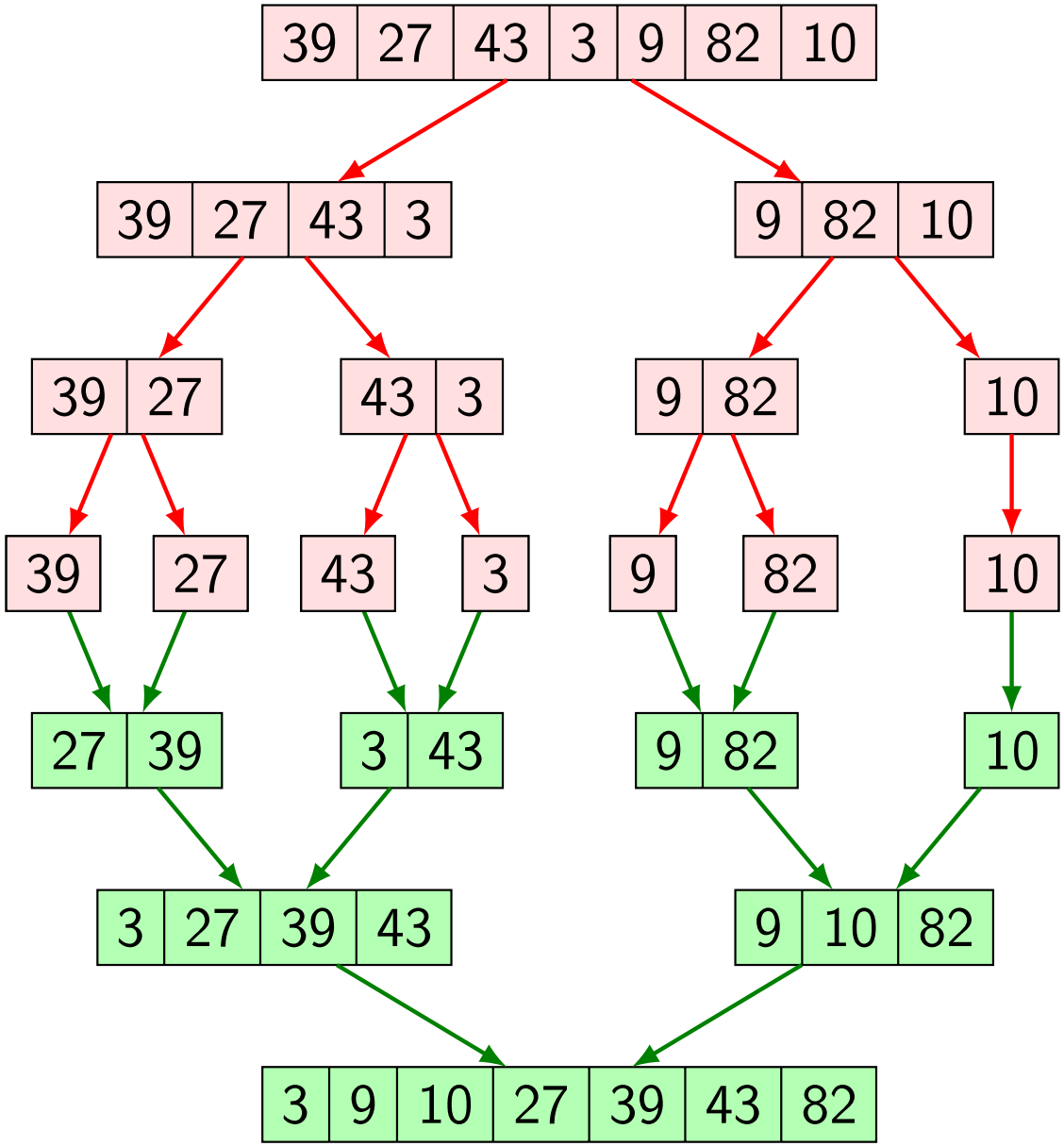
2025. október 20.

Összefésüléses rendezés (Merge Sort) - alapelv

Oszd meg és uralkodj módszer

- A sorozatot két egyenlő (vagy közel egyenlő) hosszú részre osztjuk.
- A két részt sorbarendezzük, rekurzívan, ugyanezzel a módszerrel. Egy elem önmagában rendezett.
- A két rendezett sorozatrészt összefésüljük egy rendezett sorozattá.

Összefésüléses rendezés (Merge Sort)

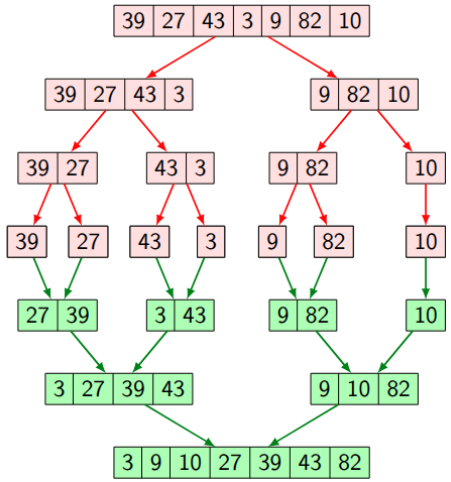


Merge Sort – C++ kód

```
1   void merge(vector<int>& a, int l, int m, int r) {
2       vector<int> left(a.begin() + l, a.begin() + m + 1);
3       vector<int> right(a.begin() + m + 1, a.begin() + r + 1);
4
5       int i = 0, j = 0, k = l;
6       while (i < left.size() && j < right.size()) {
7           if (left[i] <= right[j]) a[k++] = left[i++];
8           else a[k++] = right[j++];
9       }
10      while (i < left.size()) a[k++] = left[i++];
11      while (j < right.size()) a[k++] = right[j++];
12  }
13
14  void mergeSort(vector<int>& a, int l, int r) {
15      if (l >= r) return;
16      int m = (l + r) / 2;
17      mergeSort(a, l, m);
18      mergeSort(a, m + 1, r);
19      merge(a, l, m, r);
20  }
21
22  int main() {
23      vector<int> v = {39, 27, 43, 3, 9, 82, 10};
24      mergeSort(v, 0, v.size() - 1);
25      for (int x : v) cout << x << "␣";
26  }
```

Merge Sort - Műveletigény

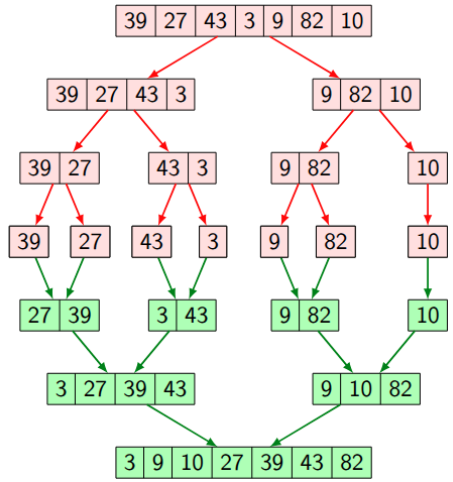
Legrosszabb esetben mennyi műveletet végzünk, az elemszám (n) függvényében? Hogy lehet ezt indokolni?



Merge Sort - Műveletigény

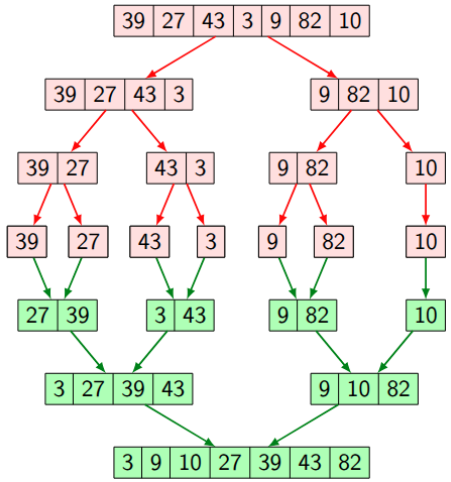
Legrosszabb esetben mennyi műveletet végzünk, az elemszám (n) függvényében? Hogy lehet ezt indokolni?

- A rekurzió $\log n$ mélységű, mivel minden szinten felezzük a méretet.
- Minden szinten az összefésülendő részek elemszáma **összesen** n.
- Az összefésülés lineáris (az elemek számával arányos)
- Tehát a teljes komplexitás $O(n \log n)$.



Az összefésülés miért lineáris?

Merge Sort - Műveletigény



Legrosszabb esetben mennyi műveletet végzünk, az elemszám (n) függvényében? Hogy lehet ezt indokolni?

- A rekurzió $\log n$ mélységű, mivel minden szinten felezzük a méretet.
- Minden szinten az összefésülendő részek elemszáma **összesen** n.
- Az összefésülés lineáris (az elemek számával arányos)
- Tehát a teljes komplexitás $O(n \log n)$.

Az összefésülés miért lineáris?

- Minden lépésben a két összefésülendő rész valamelyikén (vagy mindkettőn) léptetjük az indexet. (Érdemes megnézni az előző dián a ciklust.)

A konstans faktoron lehet javítani például a vector másolások kiküszöbölésével (ha két tömböt használunk), vagy akár rekurzió nélküli (iteratív) megvalósítással.

Merge Sort – Két tömbbel

```
1   void mergeArrays(vector<int>& src, vector<int>& dst, int left, int mid, int right) {
2       int i = left, j = mid + 1, k = left;
3       while (i <= mid && j <= right) {
4           if (src[i] < src[j]) dst[k++] = src[i++];
5           else dst[k++] = src[j++];
6       }
7       while (i <= mid) dst[k++] = src[i++];
8       while (j <= right) dst[k++] = src[j++];
9   }
10
11  void mergeSort(vector<int>& src, vector<int>& dst, int left, int right) {
12      if (left == right) { dst[left] = src[left]; return; }
13      int mid = (left + right) / 2;
14      mergeSort(dst, src, left, mid);
15      mergeSort(dst, src, mid + 1, right);
16      mergeArrays(src, dst, left, mid, right);
17  }
18
19  void mergeSort(vector<int>& a) {
20      vector<int> b = a;
21      mergeSort(b, a, 0, a.size() - 1);
22  }
23
24  int main() {
25      vector<int> v = {39, 27, 43, 3, 9, 82, 10};
26      mergeSort(v);
27      for (int x : v) cout << x << "␣";
28  }
```

Merge Sort – Iteratív (rekurzió nélküli) verzió

```
1   void mergeArrays(vector<int>& src, vector<int>& dst, int left, int mid, int right) {
2       int i = left, j = mid + 1, k = left;
3       while (i <= mid && j <= right) {
4           if (src[i] <= src[j]) dst[k++] = src[i++];
5           else dst[k++] = src[j++];
6       }
7       while (i <= mid) dst[k++] = src[i++];
8       while (j <= right) dst[k++] = src[j++];
9   }
10
11  void mergeSortIterative(vector<int>& a) {
12      int n = a.size();
13      vector<int> buf[2];
14      buf[0] = a;
15      buf[1].resize(n);
16      int cur = 0, run = 1;
17      while (run < n) {
18          int left = 0;
19          while (left < n) {
20              int mid = min(left + run - 1, n - 1);
21              int right = min(mid + run, n - 1);
22              if (mid < right)
23                  mergeArrays(buf[cur], buf[1 - cur], left, mid, right);
24              else
25                  for (int i = left; i <= right; ++i)
26                      buf[1 - cur][i] = buf[cur][i];
27              left = right + 1;
28          }
29          run *= 2;
30          cur = 1 - cur; // swap buffers
31      }
32      a = buf[cur];
33  }
```

Feladat 1: Inverziók száma (Inversion Count)

https://www.spoj.com/problems/INVCNT/

Adott egy $A[0 \ldots n-1]$ tömb n darab különböző pozitív egész számmal. Ha $i < j$ és $A[i] > A[j]$, akkor az (i, j) párt **inverziónak** nevezzük. Készíts programot, amely meghatározza a tömbben található inverziók számát!

Bemenet

A standard bemenet első sora egy t számot tartalmaz — a tesztesetek számát.
Ezután minden teszteset az alábbi formátumban érkezik:

- egy sorban az n elem száma ($1 \leq n \leq 200\,000$),
- a következő n sor mindegyike egy $A[i]$ számot tartalmaz ($1 \leq A[i] \leq 10^7$),
- ezután egy üres sor következik.

Kimenet

Minden tesztesetre egy sorban írjuk ki az inverziók számát!

Példa

Bemenet	Kimenet
2 3 3 1 2 5 2 3 8 6 1	2 5

Korlátok

$1 \leq t \leq 10$, $1 \leq n \leq 200\,000$, $1 \leq A[i] \leq 10^7$
Időlimit: 1.5 másodperc
Memórialimit: 1536 MB

Inverziók száma

Megoldási ötletek?

Inverziók száma

Megoldási ötletek?

Oszd meg és uralkodj: Az inverziók száma három részre bontható:

- az inverziók a **bal oldali** részben,
- az inverziók a **jobb oldali** részben,
- és a **keresztinverziók**, amikor az egyik elem balról, a másik jobbról származik.

Inverziók száma

Megoldási ötletek?

Oszd meg és uralkodj: Az inverziók száma három részre bontható:

- az inverziók a **bal oldali** részben,
- az inverziók a **jobb oldali** részben,
- és a **keresztinverziók**, amikor az egyik elem balról, a másik jobbról származik.

A bal és jobb részeket rekurzívan rendezzük és számoljuk az inverziókat, a **keresztinverziókat pedig az összefésülés során** határozzuk meg.

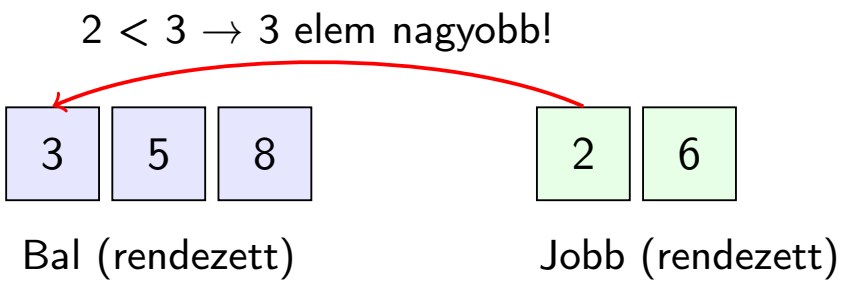
Inverziók száma – Keresztinverziók az összefésülésnél

Cél: Az összefésülés során számoljuk meg, hány **keresztinverzió** keletkezik.

Inverziók száma – Keresztinverziók az összefésülésnél

Cél: Az összefésülés során számoljuk meg, hány **keresztinverzió** keletkezik.

Ötlet: Amikor a jobb rész egy eleme kerül be a végső sorozatba, miközben a bal rész aktuális eleme nagyobb nála, akkor a bal részben **minden hátralevő elem inverziót alkot** ezzel a jobboldali elemmel.



Mivel $2 < 3$, a bal részben a **3, 5, 8** mind nagyobb, tehát ezek mind inverziót alkotnak 2-vel.

$\Rightarrow$ Hozzáadunk **3 inverziót** a számlálóhoz.

Inverziók száma – Pszeudokód

```
1  countInversions(A, left, right):
2      if left >= right:
3          return 0
4      mid = (left + right) / 2
5      inv = countInversions(A, left, mid)
6          + countInversions(A, mid + 1, right)
7          + mergeAndCount(A, left, mid, right)
8      return inv
```

Megfigyelés: Az algoritmus **szerkezete teljesen megegyezik** az **összefésüléses rendezésével**. A különbség csupán annyi, hogy a `mergeAndCount` függvény az összefésülés közben számolja az inverziókat.

- Minden olyan párnál, ahol $A[i] > A[j]$ és i a bal, j a jobb részben van, növeljük a számlálót `mid - i + 1`-gyel.

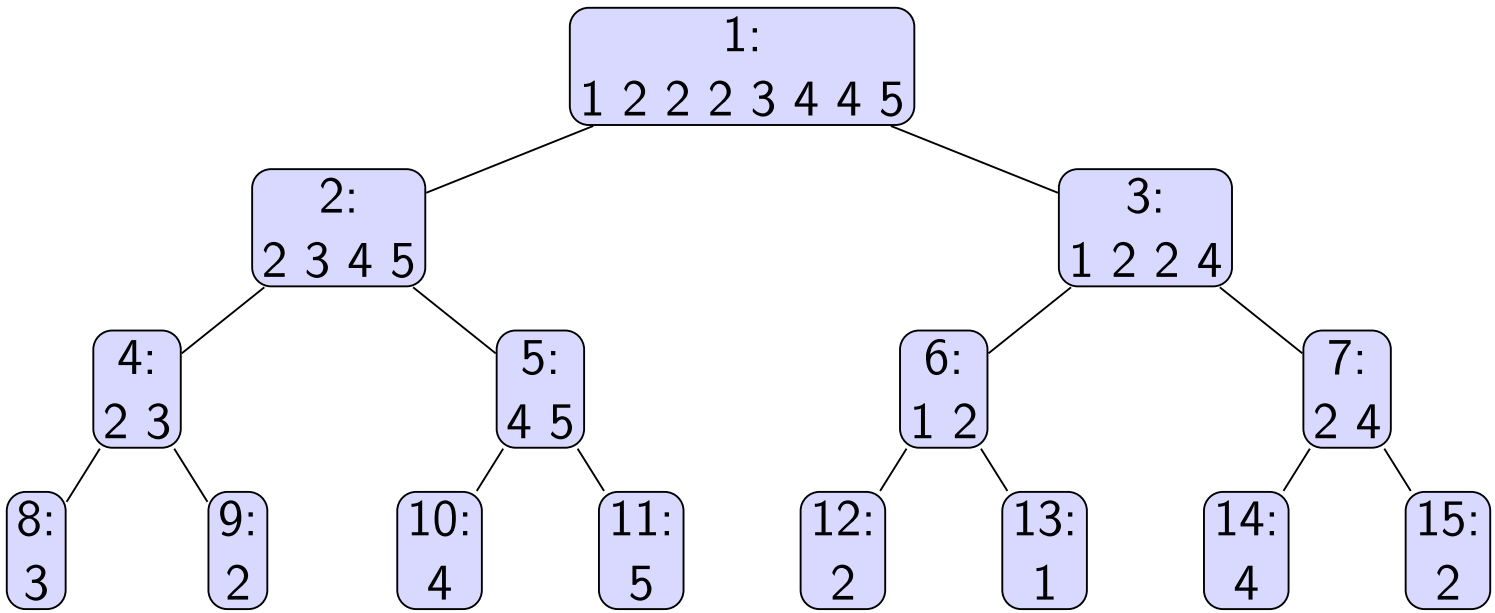
Teljes időkomplexitás: $O(n \log n)$

Merge Sort Tree

Merge Sort Tree

Ötlet: Az összefésüléses rendezés közben eltárolunk minden köztes vectort egy bináris fában.

- A fa csúcsait szokásosan indexeljük: az 1-es a gyökér, egy v csúcs balgyereke $2v$, jobbgyereke $2v + 1$.
- Tárhely igény: minden szinten n elem: $O(n \log n)$.

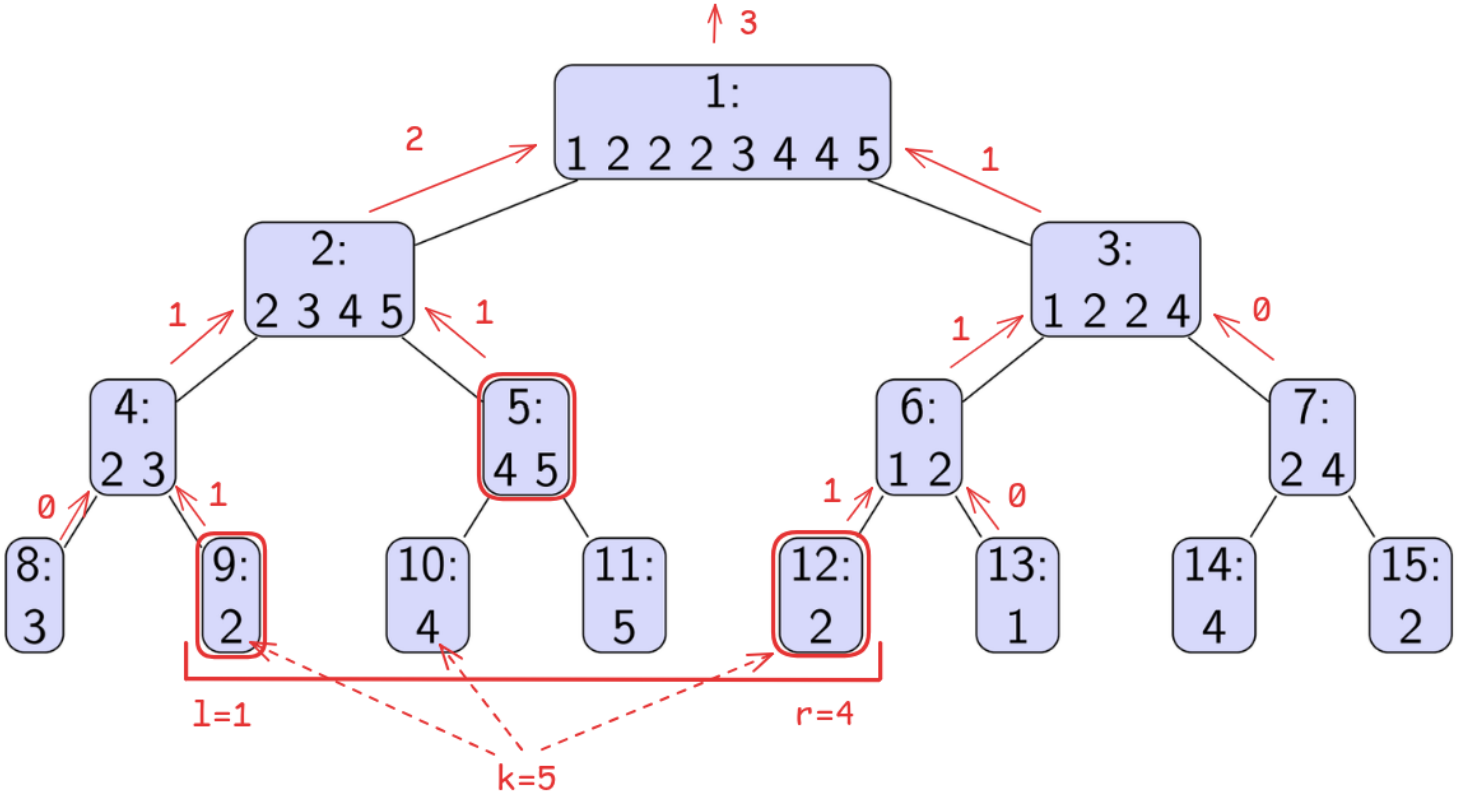


Merge Sort Tree - Felépítés

```
1  const int MAXN = 100000;
2  vector<int> tree[MAXN * 4];
3  vector<int> a;
4
5  // Build a Merge Sort Tree
6  void buildTree(int node, int start, int end) {
7      if (start == end) { tree[node].push_back(a[start]); return; }
8      int mid = (start + end) / 2;
9      buildTree(2 * node, start, mid);
10     buildTree(2 * node + 1, mid + 1, end);
11     merge(tree[2*node].begin(), tree[2*node].end(),
12           tree[2*node+1].begin(), tree[2*node+1].end(),
13           back_inserter(tree[node]));
14 }
15
16 int main() {
17     int n = 8; a = {3,2,4,5,2,1,4,2};
18     buildTree(1, 0, n-1);
19 }
```

Merge Sort Tree – Intervallum-kérdések

Kérdés: Hány k-nál kisebb elem van az $[l, r]$ intervallumban?



Példa lekérdezés számítása a fán: az [1,4] intervallumon 5-nél kisebb elemek száma 3.

- Az intervallumot a fa csúcsai fedik le részben vagy teljesen.
- Ha a csúcs teljesen benne van, egyszerűen számoljuk a kisebb elemeket a rendezett vectorban.
- Ha részben fedi, rekurzívan megyünk le a gyerekekhez.

Merge Sort Tree – Kisebb elemek lekérdezése

```cpp
const int MAXN = 100000;
vector<int> tree[MAXN * 4];
vector<int> a;

void buildTree(int node, int start, int end) { ... }

// Count elements from a[l] to a[r] with value smaller than k.
int querySmaller(int node, int start, int end, int l, int r, int k) {
    if (l > end || start > r) return 0;
    if (l <= start && r >= end) {
        return lower_bound(tree[node].begin(), tree[node].end(), k) - tree[node].begin();
    }
    int mid = (start + end) / 2;
    return querySmaller(2 * node, start, mid, l, r, k) +
           querySmaller(2 * node + 1, mid + 1, end, l, r, k);
}

int main() {
    int n = 8; a = {3,2,4,5,2,1,4,2};
    buildTree(1, 0, n - 1);
    for (int i = 1; i <= 6; i++)
        cout << querySmaller(1, 0, n - 1, 1, 4, i) << "␣";
    // Output: 0 0 2 2 3 4
}
```

Feladat 2: Intervallumon k-adik legkisebb elem

https://judge.yosupo.jp/problem/range_kth_smallest

Adott egy $a_0, a_1, \ldots, a_{N-1}$ egész számokból álló sorozat és Q darab lekérdezés, mindegyik lekérdezés a következő formátumú: $l_i\ r_i\ k_i$.
Feladat: írjuk ki a $(a_{l_i}, a_{l_i+1}, \ldots, a_{r_i-1})$ részintervallum $k_i + 1$-dik legkisebb elemét.

Bemenet

A standard bemenet:

- Egy sorban az N és Q számok ($1 \leq N \leq 200\,000$, $1 \leq Q \leq 200\,000$).
- Egy sorban N szám: $a_0, a_1, \ldots, a_{N-1}$ ($1 \leq a_i \leq 10^9$).
- Ezután Q sor, mindegyik: $l_i\ r_i\ k_i$ ($0 \leq l_i < r_i \leq N$, $0 \leq k_i < r_i - l_i$).

Kimenet

Minden lekérdezésre egy sorban írjuk ki a k-adik legkisebb elemet!

Példa

Bemenet	Kimenet
5 3 1 4 0 1 3 0 5 2 1 3 1 3 4 0	1 4 1

Korlátok

$1 \leq N \leq 200\,000$, $1 \leq Q \leq 200\,000$, $1 \leq a_i \leq 10^9$

$0 \leq l_i < r_i \leq N$, $0 \leq k_i < r_i - l_i$

Időlimit: 5 másodperc

Memórialimit: 1024 MB

Intervallumon k-adik legkisebb elem - Ötlet

Megoldási ötletek?

Intervallumon k-adik legkisebb elem - Ötlet

Megoldási ötletek?

Átalakítás: Rendezzük az (a_i, i) párokat, így megkapjuk az indexeket az értékek szerint növekvő sorrendben. Erre a sorozatra építsünk merge sort tree -t. (Ebben a fában tulajdonképpen az indexek a nagyságrendi sorrendből kiindulva a saját helyükre rendeződnek).

- Hogy módosul a lekérdezés? Hogyan válaszoljuk meg a fa segítségével?

Intervallumon k-adik legkisebb elem - Ötlet

Megoldási ötletek?

Átalakítás: Rendezzük az (a_i, i) párokat, így megkapjuk az indexeket az értékek szerint növekvő sorrendben. Erre a sorozatra építsünk merge sort tree -t. (Ebben a fában tulajdonképpen az indexek a nagyságrendi sorrendből kiindulva a saját helyükre rendeződnek).

- Hogy módosul a lekérdezés? Hogyan válaszoljuk meg a fa segítségével?
- A sorozatban k-adik olyan elem kell, ami az (l, r) intervallumban van.
- Nézzük meg, hogy a bal felében hány ilyen van $\rightarrow$ kiderül, hogy melyik felében kell keresni.
- Egy-egy csúcsban tárolt rendezett sorozatban két bináris kereséssel meghatározható, hogy hány elem esik az (l, r) intervallumba.

Intervallumon k-adik legkisebb elem - Példa

Legyen $n = 7$, $a = (2, 3, 1, 6, 5, 3, 5)$ és a lekérdezés: $l = 2$, $r = 6$, $k = 2$, ekkor az $(1, 6, 5, 3)$ részben keressük a harmadik legkisebb elemet, tehát a válasz $a_4 = 5$ lesz.

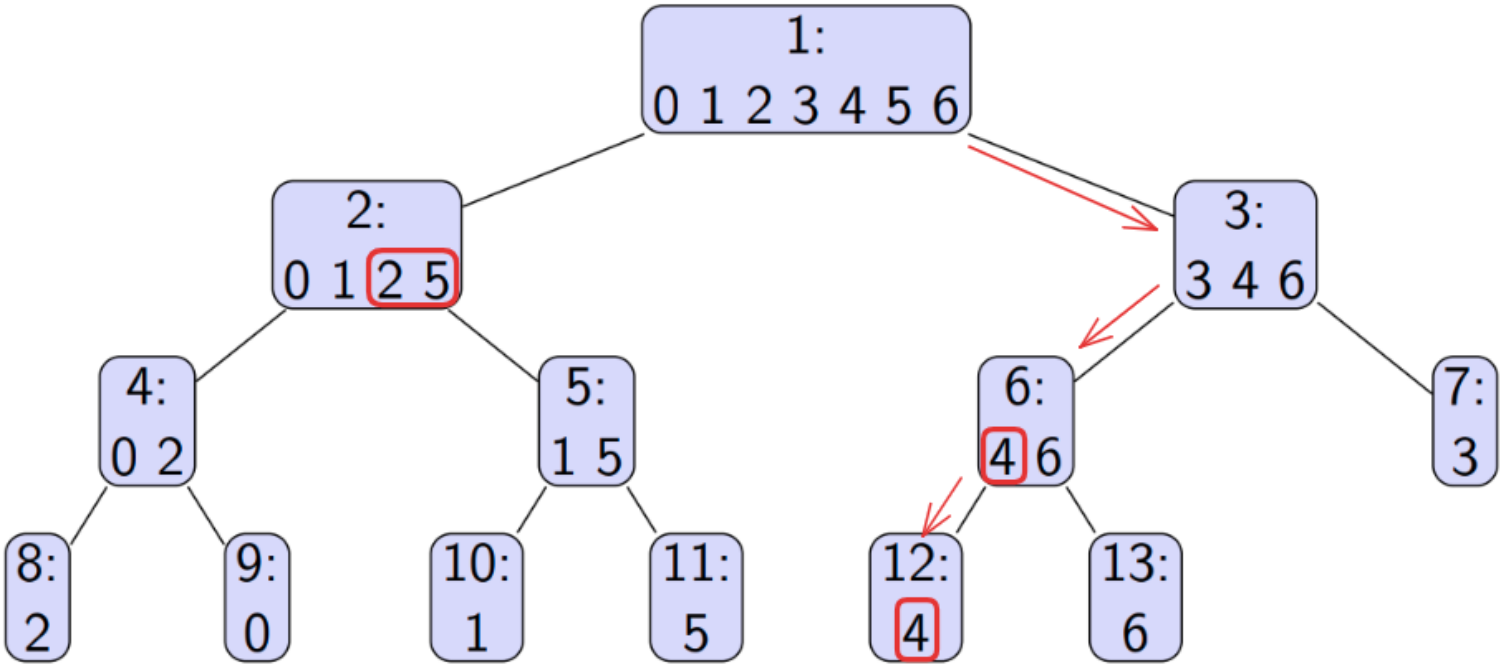
a_i	2	3	1	6	5	3	5
i	0	1	2	3	4	5	6

Rendezve:

a_i	1	2	3	3	5	5	6
i	**2**	0	1	**5**	**4**	6	**3**

A legalsó indexsorozatban a kérdés úgy alakul át, hogy melyik a k-adik elem, ami a $[2, 6)$ intervallumban van. Erre fogjuk alkalmazni a merge sort tree-t.

Intervallumon k-adik legkisebb elem - Példa



A gyökér balgyerekében csak két elem van, ami a $[2, 6)$ intervallumban van, mi a harmadikat keressük, ezért jobbra lépünk, ahol már az első elemet keressük majd ebben az intervallumban.

Feladatok

Inversion Count
https://www.spoj.com/problems/INVCNT/

Merge Sort
https://codeforces.com/contest/873/problem/D

Range Kth Smallest
https://judge.yosupo.jp/problem/range_kth_smallest

Hold the Line
https://codeforces.com/gym/102452/problem/H