1. Rendezze a $3, 12, 1, 34, 4, 6, 0$ számsorozatot beszúrásos rendezéssel! Hány összehasonlításra volt szükség amikor lineáris keresést használtunk és hányra amikor bináris keresést?

   Megoldás: A beszúrásos rendezésnek két fajtája van: amikor lineárisan és amikor binárisan keresünk. A beszúrásonként kapott sorozatok ugyanazok, de az összehasonlítások száma eltérhet.

   Az eljárás az, hogy az i-edik elemet szúrjuk be a már rendezett első $i-1$ elem közé, tehát a sorozat az egyes lépésekben így alakul: $(3, \mathbf{12}, 1, 34, 4, 6, 0) \longrightarrow (3, 12, \mathbf{1}, 34, 4, 6, 0) \longrightarrow (1, 3, 12, \mathbf{34}, 4, 6, 0) \longrightarrow (1, 3, 12, 34, \mathbf{4}, 6, 0) \longrightarrow (1, 3, 4, 12, 34, \mathbf{6}, 0) \longrightarrow (1, 3, 4, 6, 12, 34, \mathbf{0}) \longrightarrow (0, 1, 3, 4, 6, 12, 34)$.

   Az összehasonlítások száma lineáris keresésnél (itt feltesszük, hogy hátulról előre haladva keresünk): $1 + 2 + 1 + 3 + 3 + 6 = 16$, bináris keresésnél: $1 + 2 + 2 + 2 + 3 + 3 = 13$.

2. Rendezze a $16, 17, 2, 6, 11, 33, 28, 22$ sorozatot gyorsrendezéssel úgy, hogy mindig a tömb első elemét választja particionáló elemnek!

   Megoldás: $(\mathbf{16}, 17, 2, 6, 11, 33, 28, 22) \longrightarrow$
   $(\mathbf{6}, 11, 2 \mid 16 \mid \mathbf{17}, 33, 28, 22) \longrightarrow$
   $(2 \mid 6 \mid 11 \mid 16 \mid 17 \mid \mathbf{33}, 28, 22) \longrightarrow$
   $(2 \mid 6 \mid 11 \mid 16 \mid 17 \mid \mathbf{22}, 28 \mid 33) \longrightarrow$
   $(2 \mid 6 \mid 11 \mid 16 \mid 17 \mid 22 \mid 28 \mid 33)$

   Megoldás részletesebben: Nevezzük A-nak a tömböt. Kezdetben $s = 16$ (első elem) a particionáló elem. Az i pointer a második, 17-es elem indexétől (ami 2) indul, és addig nőhet, amíg $A[i] < s$. Tehát a 17-nél elakad, nem indul el. A j pointer az utolsó, 22-es elem indexétől (ami 8) indul, és addig csökkenhet, ameddig $A[j] > s$. Tehát a 11-es elemnél akad el.

   Most mindkét pointer elakadt, de $i \neq j$, tehát megcseréljük $A[i]$-t és $A[j]$-t:

   $$(\mathbf{16}, 11, 2, 6, 17, 33, 28, 22)$$

   Ezek után i-t (ami 2 volt) tovább növeljük, az előbbi szabály szerint a 17-es elem indexéig jut. Most már összeért j-vel, így $A[i-1]$-et és s-et megcseréljük. Mivel s-et mozgattuk, ő a helyére kerül:

   $$(6, 11, 2 \mid 16 \mid 17, 33, 28, 22)$$

   (a függőleges vonalak jelzik, hogy a 16-hoz képest már a kisebbek a bal, a nagyobbak a jobb oldalon vannak).

   Ezután rekurzívan a 16-tól balra lévő $6, 11, 2$ résztömbben (most ezt hívjuk A-nak és később is mindig az aktuálisan vizsgált tömböt, mindig 1-től indexelve) ugyanezt végrehajtjuk, így a 6 (első elem) lesz a particionáló elem. i a 11-nél, j a 2-nél indul és ott is maradnak. Mivel elakadtak, de nem értek össze, $A[i]$-t és $A[j]$-t megcseréljük: $\mathbf{6}, 2, 11$. Most i még eggyel nőhet, majd összeakad j-vel, így megcseréljük s-et $A[i-1]$-gyel, vagyis a $2 \mid 6 \mid 11$ particionált résztömböt kapjuk az előbbi jelöléssel.

   Ugyanezt megtesszük a 16-tól jobbra lévő $17, 33, 28, 22$ résztömbben, $s = 17$. i marad a 33 indexénél, míg j lemehet a 22 indexétől a 33-éhoz. Ekkor i és j összeért, tehát az $s = 17$-et megcseréljük $A[i-1]$-gyel, vagyis önmagával. Így látszólag nem csináltunk semmit, de valójában sikerült particionálnunk a 17-et is, tőle jobbra csak nála nagyobbak vannak (és tőle balra nyilván csak nála kisebbek):

   $$(2 \mid 6 \mid 11 \mid 16 \mid 17 \mid 33, 28, 22).$$

   A $33, 28, 22$ résztömbben $s = 33$ a particionáló elem. Mivel a 28 és a 22 is nagyobb 33-nál, i-t növelhetjük **a 22-n túl is**, így az indexe túlcsordul, 4-re nő. j viszont értelemszerűen el sem indul. Ekkor úgy tekintjük, hogy i és j már összeért, ezért megcseréljük $A[i-1]$-et, vagyis a 22-t, az $s = 33$-mal. Ezzel 33-at is particionáltuk:

   $$(2 \mid 6 \mid 11 \mid 16 \mid 17 \mid 22, 28 \mid 33).$$

Végül a 22-ből és 28-ból álló kételemű tömbben ugyanígy túlcsordultatjuk i-t, és így a 28-at megcserélhetjük a 22-vel:

$$(2 \mid 6 \mid 11 \mid 16 \mid 17 \mid 22 \mid 28 \mid 33).$$

Most már egyeleműek a partícióink, így készen vagyunk, a tömb rendezett.

3. Dr. Watson azzal állít be Sherlock Holmes-hoz, hogy olyan összehasonlítás-alapú rendezési algoritmust talált, ami úgy rendez akármekkora tömböt, hogy minden egyes tömbbeli szám legfeljebb 2025 összehasonlításban szerepel. Mivel indokolhatja Sherlock Holmes, hogy Watson téved?

   Megoldás: Ha minden elem legfeljebb 2025 összehasonlításban szerepel, akkor az algoritmus az n elemmel összesen legfeljebb $2025n/2$ összehasonlítást végez. De tudjuk, hogy minden összehasonlítás-alapú algoritmusnak kell legalább $\log n! = \Omega(n \log n)$ összehasonlítás, és $2025n/2 \notin \Omega(n \log n)$, tehát valóban nincs ilyen algoritmus.

---

4. Az $A[1 : n]$ tömbben számokat tárolunk. Határozza meg $O(n \log n)$ lépésben

   (a) azokat az értékeket, amelyek egynél többször fordulnak elő;

   (b) a leggyakoribb értékeket (vagyis azokat, amelyeknél többször semelyik másik szám sem fordul elő a tömbben)!

   Megoldás: (a) Ha a tömb rendezve van, akkor egyszerűbb a feladat, hiszen akkor az egyforma értékek egymás mellett vannak. *Algoritmus:* Először rendezzük a tömböt, például az összefésüléses rendezéssel, aminek lépésszáma $O(n \log n)$ (a többi eddig tanult rendező algoritmus nem lenne jó, mert azok nem $O(n \log n)$ lépésszámúak, ha nem csak az összehasonlításokat számoljuk). Majd végigmegyünk a tömbön, kiírjuk minden olyan elemet, ami határozottan nagyobb az előzőnél (vagy nincs előző) és egyenlő a következővel. *Jóság:* A rendezés miatt az egyforma elemek egy blokkban lesznek, minden blokk első elemét írjuk ki. Ha egy szám csak egyszer fordul elő, akkor nem fogjuk kiírni. *Lépésszám:* A rendezés $O(n \log n)$, utána a kiírás $n-1$ lépés, tehát összesen $O(n \log n)$ lépést végzünk.

   (b) Rendezzük a tömböt, mint előbb. Most azt kell meghatározni, hogy a rendezett tömbben melyik elem alkotja a leghosszabb blokkot. *Algoritmus:* Először rendezzük a tömböt. Utána végigmegyünk a tömbön és egy-egy számlálóval nyilvántartjuk, hogy mennyi az eddigi legnagyobb blokk hossza, ott milyen érték vagy értékek voltak, és számoljuk az aktuális blokk hosszát. Ha az aktuális blokk hossza nagyobb vagy egyenlő a korábbi legnagyobbnál, akkor frissítjük az első két számlálót. *Jóság:* Ha hosszabb blokk következik, akkor csak az ott levő értéket tartjuk már nyilván. Ha az eddigi leghosszabb blokkal egyenlőt találunk, akkor pedig az új értéket is megjegyezzük. *Lépésszám:* A rendezésen kívül itt is $O(n)$ sok lépés történik, ezért a lépésszám $O(n \log n)$.

5. Egy tömbön gyorsrendezést futtatva az első particionálás után az eredmény: $4, 2, 3, 1, 6, 8, 11$. Mi lehetett a particionáló elem?

   Megoldás: Olyan elemet kell keresni, hogy az előtte levők a nála kisebbek, az utána levők a nála nagyobbak. Tehát a 6, a 8 és a 11 is jó, de más nem.

6. Az $A[1 : n]$ tömbben levő elemekről tudjuk, hogy $A[1] \neq A[n]$. Adjon $O(\log n)$ összehasonlítást használó algoritmust, amely talál egy olyan i indexet, hogy $A[i] \neq A[i+1]$.

   Megoldás: Bináris kereséshez hasonló algoritmust használunk: legyen $i = \lceil n/2 \rceil$. Ha $A[i] \neq A[1]$, akkor az $A[1 : i]$ tömb is ugyanolyan tulajdonságú, mint az eredeti, elég ebben keresni. Ha viszont $A[i] = A[1]$, akkor nyilván $A[i] \neq A[n]$, és akkor a továbbiakban az $A[i : n]$ tömbben kereshetünk. Így ennél a lépésnél biztosan egy kisebb tömböt kapunk, ha $n \geq 3$.

   Az eljárás végén a tömb leszűkül egy 2 eleműre, azaz két szomszédos nem egyenlő elemre, ekkor készen vagyunk.

   A lépésszám, ahogy a bináris kereséséhez hasonlóan $O(\log n)$ lesz.

7. Legyen adott egy egészekből álló $A[1:n]$ tömb valamint egy b egész szám. Egy olyan $i, j \in \{1, \ldots, n\}$ indexpárt keresünk, melyre $A[i] + A[j] = b$. Hogyan lehet ezt $O(n \log n)$ időben megoldani?

Megoldás: Vegyük észre, hogy ha minden lehetséges indexpárt ki akarunk próbálni, akkor $\binom{n}{2} \in \Theta(n^2)$ lehetőség van, ami több, mint a kívánt lépésszám.

Itt is sokat segít a rendezettség, ezért rendezzük a tömböt $O(n \log n)$ lépésben, összefésüléses rendezéssel.

Ezután az $i = 1, 2, \ldots, n$ indexekre keressük a tömbben a $b - A[i]$ értéket. Tegyük fel, hogy $A[j] = b - A[i]$, ekkor egy jó indexpár a rendezett tömbben az (i, j). Meg kell még keresni ezeknek az $A[i]$ és $A[j]$ elemeknek az eredeti helyét az eredeti tömbben, ez további két lineáris kereséssel megkapható.

Ez az eljárás azért jó, mert $A[i] + A[j] = b$ pontosan akkor teljesül valami i és j indexpárra, ha $A[j] = b - A[i]$, azaz $b - A[i]$ benne van a tömbben.

Ha bináris keresést használunk $b - A[i]$ megkeresésére is, akkor mindegyik (összesen n darab) keresés $\Theta(\log n)$ lépésszámú, tehát a lépésszám (a végén levő két extra $O(n)$ lépésű kereséssel együtt) $O((n \log n + 2n) = O(n \log n)$.

Megjegyzés: a rendezés utáni rész lineárisan is megoldható. Kiindulunk az $i = 1$, $j = n$ indexekből és általában, ha $A[i] + A[j] < b$, akkor az i értékét növeljük eggyel, ha pedig $A[i] + A[j] > b$ akkor a j értékét csökkentjük eggyel. Akkor hagyjuk abba, ha $A[i] + A[j] = b$ vagy ha $i > j$ lenne – ez utóbbi esetben nincs megoldás. (Miért?)

Megoldás részletesebben: **Intuíció és jótanács.**

Amikor egy olyan problémával találkozunk, amely számokkal végzett műveleteket tartalmaz és a futásideje $O(n \log n)$, gyakran érdemes egy rendezéssel kezdeni, mert sok esetben a további lépések is hatékonyabbá válnak és pl. összefésüléses rendezéssel tudunk $O(n \log n)$ lépésben rendezni.

Ebben a feladatban egy olyan indexpárt keresünk, amelyre $A[i]+A[j] = b$. Ha minden lehetséges párt kipróbálnánk, az $O(n^2)$ időt venne igénybe, amely lassabb, mint a feladatban kért futásidő. Ehelyett rendezéssel kombinálva hatékonyabb megoldást találhatunk. Két, részben különböző megoldást is mutatunk.

1. Algoritmus menete

(a) **Rendezés:**

- Az $(A[i], i)$ párokat rendezzük $A[i]$ érték alapján összefésüléses rendezéssel. Pár tartalmazza az eredeti értékeket és az indexeiket az eredeti tömbben.

(b) **Iteráció minden elemre:**

- Végigmegyünk az új tömb minden egyes elemének (ne felejtsük, ezek párok) első értékein.
- Az aktuális $A[i]$ értékhez meghatározzuk a keresett párját:

$$b'[i] = b - A[i]$$

(c) **Bináris keresés:**

- Bináris kereséssel rákeressünk $b'[i]$-re a tömbben. Tehát egy $(A[j], j)$ pár esetén az adott $b'[i]$ értéket mindig a pár első elemével, hasonlítjuk össze a bináris keresés folyamán.
- Ha a feladatra úgy gondolunk, hogy a megtalált számok indexeinek különbözőnek kell lennie, akkor az esetlegesen megtalált szám, $A[j]$ eredeti indexe (j) nem egyezhet meg i-vel, ami az $A[i]$ eredeti indexe.
- Ha a feladatra úgy gondolunk, hogy a megtalált számok indexei lehetnek egyenlőek, akkor a megtalált index j megegyezhet i-vel.

(d) **Döntés:**

- Ha találunk megfelelő $A[i]$-t és $A[j]$-t, akkor (i, j) indexpár az output.
- Ha végigmentünk az egész tömbön és nem találtunk ilyen párt, visszatérünk azzal, hogy `NINCS MEGOLDÁS`.

Helyesség bizonyítása

A rendezés után biztosak lehetünk abban, hogy minden $A[i]$ érték esetén, ha létezik $b-A[i]$ a tömbben (tárolt párok első értékei közt), akkor azt a bináris keresés meg fogja találni. És futtathatjuk a bináris keresést, hiszen a tömb $A[i]$-k alapján rendezve van.

Lépésszám elemzése

(a) Rendezés: $O(n \log n)$

(b) Minden elemre végrehajtott bináris keresés: $n \cdot O(\log n) = O(n \log n)$

(c) Összesen: $O(n \log n)$

2. Algoritmus: Kétmutatós Keresés

A második algoritmus is rendezést használ, majd a rendezett tömbben egy kétmutatós (two-pointer) keresési módszert alkalmaz, amely a rendezés után csak $O(n)$ lépést vesz igénybe.

Algoritmus Lépései

(a) **Rendezés:**

- Az eredeti tömb elemeit rendezzük növekvő sorrendbe (például összefésüléses rendezéssel) és jelöljük ezt az új tömböt B-vel.

(b) **Kétmutatós keresés:**

- Állítsuk be az alsó mutatót i-re a tömb elejére, és a felső mutatót j-re a tömb végére.
- Amíg $i < j$ (vagy $i \leq j$, attól függően, hogy lehet-e kétszer ugyanaz az elem):
  i. Számoljuk ki az aktuális összeget: $i \leftarrow 1$ és $j \leftarrow n$ $S = B[i] + B[j]$
  ii. Ha $S = b$, akkor megtaláltuk a megfelelő két elemet.
  iii. Ha $S < b$, akkor az összeg növeléséhez növeljük az alsó mutatót: $i \leftarrow i + 1$.
  iv. Ha $S > b$, akkor az összeg csökkentéséhez csökkentjük a felső mutatót: $j \leftarrow j - 1$.

(c) **Eredmény meghatározása:**

- Ha a kétmutatós keresés során találunk két olyan elemet, melyek összege pontosan b, akkor lineáris kereséssel kikeressük őket az eredeti tömbből és visszaadjuk az indexet.
- Ha az i és j mutatók kereszteződnek (vagyis $i \geq j$ vagy feladat értelmezésétől függően: $j < i$), és nem találtunk megfelelőt, akkor visszatérünk `NINCS MEGOLDÁS` outputtal.

Helyességi Bizonyítás

Tegyük fel, hogy van megoldás, de átsiklottunk felette, azaz létezne olyan $i < j$ (ezt írjuk ki, a feladat azon értelmezésének helyessége, amikor lehetnek egyenlő indexek, értelemszerű módosításokkal megkapható), hogy

$$b = B[i] + B[j],$$

de nem találtuk meg a megoldást.

Nézzük meg, hogy egy adott lépésben mi történik a tömbben:

Legyenek az algoritmus kétmutatós keresésénél a bal oldali mutató i' és a jobb oldali mutató j'.

- Ha $B[i'] + B[j'] < b$: Mivel B rendezett, minden $k < j'$ esetén $B[k] \leq B[j']$, így a lépés után nem állhat elő olyan helyzet, hogy $B[i']$ b-t adna összegként valamelyik későbbi jobb oldali mutatóhoz kapcsolódó számmal.
- Ha $B[i'] + B[j'] > b$: Hasonlóan az előzőhöz: mivel B rendezett, minden $i' < k$ esetén $B[i'] \leq B[k]$, így a lépés után nem állhat elő olyan helyzet, hogy $B[j']$ b-t adna összegként valamelyik későbbi bal oldali mutatóhoz kapcsolódó számmal.

Így amikor a mutatók közül valamelyik először elérik i-t vagy j-t, ott kell maradjon, míg a másik "beér a helyére", hogy b-t adjanak együtt.

$$i' < i.$$

Lépésszám Elemzés

(a) **Rendezés:** $O(n \log n)$

(b) **Kétmutatós keresés:** A rendezést követően a keresési lépések $O(n)$ időben hajthatók végre.

(c) **Index visszakeresése:** lineáris keresés $2 \cdot O(n) = O(n)$.

(d) **Összesen:** A teljes algoritmus futási ideje $O(n \log n)$.

---

8. Az eredetileg növekvő $a_1, \ldots, a_n$ sorozatban egy elem értéke megváltozott, de nem tudjuk melyik. Hogyan lehet $O(n)$ lépésben újra növekvő sorrendbe rendezni az elemeket?

   Megoldás: Vegyük észre, hogy ha a változtatás után is $a_i < a_{i+1}$ minden i-re, akkor bár nem tudjuk megállapítani, melyik változott, de erre nincs is szükség, a sorozat továbbra is rendezett.

   Ha viszont van olyan i, melyre $a_i > a_{i+1}$, akkor ez úgy keletkezett, hogy a_i és a_{i+1} valamelyike változott, a_i nőtt vagy a_{i+1} csökkent. A legegyszerűbb (nem feltétlenül leggyorsabb) módszer, ha ezt a két problémás elemet beszúrjuk a többi elem által alkotott növekvő sorozatba. Ezt végezhetjük lineáris kereséssel, amikor $O(n)$ lépést végzünk. (Bináris keresésnél az összehasonlítások száma logaritmikus, de a mozgatásokra akkor is elmehet lineárisan sok lépés.)

   Megjegyzés: Egyetlen beszúrással is megoldható, hogyan?

9. Adjon minél kevesebb összehasonlítást használó algoritmust, ami n elem közül megtalálja a két legkisebbet!

   Megoldás: Előbb a legkisebb, majd a többiből megint a legkisebb meghatározása $n - 1 + n - 2 = 2n - 3$ összehasonlítás. Ennél jobbat is kaphatunk. Gondoljunk egy kieséses bajnokságra. A győztes legyen a legkisebb. A második nyilván csak olyan lehet, akit a későbbi győztes vert ki (a többieknél van legalább kettő nagyobb). Tehát a másodikra körönként egyetlen jelölt van. A bajnokság lebonyolítható $\lceil \log n \rceil$ körben, ennyi közül kell kiválasztani a másodikat, tehát a feladat összesen $n + \lceil \log n \rceil - 2$ lépésben megoldható.

10. Tudjuk, hogy az $a_1, \ldots, a_n$ sorozat olyan, hogy egy darabig növekszik, utána csökken. Adjon $O(n)$ összehasonlítást használó algoritmust, ami növekvő sorrendbe rendezi az elemeit!

    Megoldás: Ötlet: a legnagyobb elemnél kettévágjuk a sorozatot. A kapott két rész mindegyike rendezett, ezeket össze tudjuk fésülni egyetlen rendezett sorozattá.

    Algoritmus: Sorban az a_i – a_{i+1} szomszédokat összehasonlítva megtalálhatjuk a legnagyobb elemet: Ha $a_{x-1} < a_x$ és $a_x > a_{x+1}$ akkor a_x a legnagyobb elem. Ha nincs ilyen, vagyis $a_{i-1} < a_i$ minden i-re teljesül akkor nincs további tennivalónk, hiszen a sorozat növekszik. Eddig $n - 1$ összehasonlítást használunk. Az $a_1, \ldots, a_x$ és az $a_n, a_{n-1}, \ldots, a_{x+1}$ növekvő sorozatokat a tanult módon legfeljebb $n-1$ összehasonlítással összefésüljük, és így legfeljebb $2n - 2 \in O(n)$ összehasonlítással készen vagyunk. (Az algoritmus mozgatja is az elemeket, de összesen ez is $O(n)$ lépés.)

Megoldhatjuk az a_x előzetes megkeresése nélkül is. Elég azt képzelni, hogy két sorozatunk van, az egyik első eleme a_1, a másiké a_n (és kezdetben nem tudjuk, hol érnek véget). Legyen $i = 1$ és a $j = n$, és hasonlítsuk össze az a_i és az a_j elemet. A kisebb a készülő rendezett lista következő eleme, és ha $a_i < a_j$, akkor az i értékét növeljük, különben a j értékét csökkentjük. A továbbiakban is így, az összefésüléshez hasonlóan folytatjuk. Akkor lesz vége, ha $i = j$ (és akkor ez a legnagyobb elem). Ez az eljárás legfeljebb $n - 1 \in O(n)$ összehasonlítást használ.

Megjegyzés: mindkét eljárás helyesen működik abban az esetben is, amikor a legnagyobb elem az a_1 vagy az a_n, akkor eredetileg is egy rendezett sorozatunk van (növekvő vagy csökkenő).

11. Az n méretű (nem feltétlenül rendezett) A tömb elemei különböző pozitív számok. Adjon algoritmust, amely meghatároz egy $1 \leq k \leq n$ számot és kiválaszt k különböző elemet az A tömbből úgy, hogy a kiválasztott elemek összege nem több mint k^3. Ha nincs ilyen k, akkor az algoritmus jelezze ezt a tényt. Az algoritmus lépésszáma legyen $O(n \log n)$.

Megoldás: Vegyük észre, hogy egy rögzített k esetén, ha van megoldás, akkor a tömb legkisebb k eleme is megoldás. Ha ezek nem jók, akkor pedig nincs megoldás. Tehát azt kell vizsgálni, hogy van-e olyan k, hogy a legkisebb k elem összege legfeljebb k^3.

Ehhez rendezzük a tömböt $O(n \log n)$ lépésben, például összefésüléses rendezéssel. Legyen $k = 1$ és $T = A[1]$. Ha $T \leq k^3$, akkor készen vagyunk, különben növeljük k-t és adjuk T-hez az $A[k]$ számot. Ha egyik $k \leq n$ sem jó, akkor nincs megoldás.

A rendezés utáni rész n összehasonlítást és $O(n)$ összeadást használ, ezért összesen a lépésszám $O(n \log n)$.

Megoldás részletesebben: **Intuíció** Az algoritmus célja, hogy eldöntse, van-e az adott tömbben olyan részhalmaz, amelynek összege nem haladja meg az adott elemszám köbének értékét. Az alapötlet az, hogy a legkisebb elemekből képezett részhalmazok mindig kisebb összegeket adnak, mint ha véletlenszerűen választanánk elemeket. Ezért az algoritmus az elemeket növekvő sorrendbe rendezi, majd fokozatosan vizsgálja, hogy a legkisebb k elem összege kisebb-e, mint k^3. Ha ilyet találunk, azonnal leállunk, különben végignézzük az összes lehetőséget, és ha egyik sem felel meg a feltételnek, akkor kijelenthetjük, hogy nem létezik megfelelő részhalmaz. Fontos technikai dolog a végén, hogy egy ún. rollign sum változót is bevezetünk annak érdekében, hogy "benne maradjunk az adott futásidőben".

Az algoritmus leírása

(a) Rendezzük a tömböt növekvő sorrendbe
(például Összefésüléses Rendezés algoritmussal), és nevezzük el ezt a rendezett tömböt B-nek.

(b) Hozzunk létre egy új S tömböt, amelynek mérete n, és kezdetben minden eleme nulla. Ebben az S tömbben az első k legkisebb elem összegét fogjuk tárolni, vagyis $S[k]$ jelentése:

$$S[k] = \sum_{i=1}^{k} B[i], \quad \text{ahol } 1 \leq k \leq n.$$

(c) Kezdeti feltételezés: Legyen $S[1] := B[1]$. Ha $S[1] \leq 1$, akkor az algoritmus visszatér az **IGEN** válasszal, mert találtunk egy olyan részhalmazt, amelynek összege nem haladja meg az engedélyezett értéket ($1^3 = 1$). Ellenkező esetben folytatjuk a vizsgálatot.

(d) Ismételjük az alábbi lépéseket $k = 2$-től n-ig:

i. Számítsuk ki az aktuális összegértéket:

$$S[k] := S[k-1] + B[k].$$

ii. Ha $S[k] \leq k^3$, akkor az algoritmus azonnal visszatér az **IGEN** válasszal, mert találtunk k darab elemet, amelyek összege legfeljebb k^3.

iii. Ha $k \leq n-1$ és $S[k] > k^3$, akkor növeljük k értékét, és folytatjuk a vizsgálatot.

iv. Ha $k = n$ és $S[n] > n^3$, akkor az algoritmus visszatér a **NEM** válasszal, mert egyik k elemű részhalmaz sem teljesíti a feltételt.

Helyesség bizonyítása Az algoritmus helyessége azon a megfigyelésen alapul, hogy ha létezik olyan k elemű részhalmaz, amelynek összege legfeljebb k^3, akkor a legkisebb k elem összege is teljesíti ezt a feltételt. Más szóval, ha egy tetszőleges k elem kiválasztásával elérhető egy ilyen összeg, akkor ennek teljesülnie kell a legkisebb k elem esetében is, mivel ezek adják a lehető legkisebb összegű kombinációt. Ezért elegendő a legkisebb k elem összegét vizsgálni minden lehetséges k értékre.

Futási idő

(a) Az algoritmus első lépése egy rendezés, amely Összefésüléses Rendezés alkalmazásával $O(n \log n)$ időigényű.

(b) Az $S[k]$ értékek kiszámítása minden k esetén egy egyszerű összeadás, amely konstans időben ($O(1)$) elvégezhető. Ezért az iterációs lépés teljes futási ideje $O(n)$.

(c) Az összes lépés együtt:

$$O(n \log n) + O(n) = O(n \log n).$$

Így az algoritmus teljes futásideje $O(n \log n)$.

12. Adott a síkon n pont, melyek koordinátái (a_1, b_1), $(a_2, b_2), \ldots, (a_n, b_n)$. Olyan $P = (x, y)$ pontot keresünk a síkon, amire a $\sum_{i=1}^{n}(|a_i - x| + |b_i - y|)$ összeg minimális. Adjon algoritmust, amely $O(n \log n)$ lépésben meghatároz egy ilyen P pontot!

Megoldás: A felírt összeget minimalizálhatjuk úgy, hogy külön meghatározzuk az x koordinátában és külön az y koordinátában a minimumot. Tehát például olyan x kell, amire a $\sum_{i=1}^{n} |a_i - x|$ összeg minimális. Ehhez képzeljük el az a_i pontokat a számegyenesen. Ha x kisebb, mint a legkisebb a_i, akkor x-et növelve az összeg csökken. Amíg kevesebb a_i van ami kisebb x-nél, mint ami nagyobb nála, addig x-et növelve az összeg tovább csökken. Akkortól nő, amikor már több a_i van ami x-nél kisebb, mint ahány nagyobb. Tehát az optimális x a rendezés szerinti középső a_i, ha n páratlan. Páros esetben mindegy, hogy hol van a középső intervallumon belül, például az $x = a_{n/2}$ választás jó.

Ezek után az algoritmus: rendezzük az a_i értékeket, és legyen x a rendezés szerinti $\lceil n/2 \rceil$ szám. Ugyanezt megcsináljuk y-ra: rendezzük a b_i értékeket, és legyen y a rendezés szerinti $\lceil n/2 \rceil$ szám.

Az algoritmus lépésszáma, összefésüléses rendezés esetén $O(n \log n)$.

13. Adott a számegyenesen n intervallum, $[a_1, b_1], \ldots, [a_n, b_n]$. Azt akarjuk tudni, hogy együtt milyen hosszú részt fednek le a számegyenesből (azaz, hogy mennyi $\cup_{i=1}^{n}[a_i, b_i]$ összhossza). Adjon $O(n \log n)$ lépéses algoritmust ennek a hossznak a meghatározására!

Megoldás: Egy intervallum hossza $b_i - a_i$, de az unió lehet ezek összegénél kevesebb, ha az intervallumok metszik egymást. Ezeket az átfedéseket kell valahogy kezelni, pontosabban megtalálni azokat a diszjunkt intervallumokat, amik az uniót alkotják.

Rendezzük a végpontokat (mind a $2n$ számot) úgy, hogy tároljuk, melyik volt egy intervallum kezdőpontja és melyik egy intervallum végpontja. Az unió első része a legkisebb értéknél kezdődik (és ez szükségszerűen egy intervallum a_j kezdete). Menjünk a rendezett sorozatban addig, amíg először megegyezik a kezdő és végpontok száma. Ez egy végpontnál kövezkezik be (b_k), és vegyük észre, hogy itt ér véget az unió első része, ennek a résznek a hossza $b_k - a_j$. Ha még maradtak a sorozatban elemek, ugyanígy folytathatjuk.

A lépésszámhoz azt kell látni, hogy a rendezés után már csak végig kell menni a sorozaton, számlálva a kezdő- és végpontokat. A hossz meghatározásához legfeljebb n kivonás kell, majd ezek eredményét

kell összeadni. A rendezés lépésszáma $O(2n \log(2n)) = O(n \log n)$, az utána következőké $O(n)$, tehát ez összesen $O(n \log n)$.

14. Adjon minél kevesebb összehasonlítást használó algoritmust, ami n elem közül megtalálja a legkisebbet és a legnagyobbat is!

Megoldás: Külön a legkisebbet és a legnagyobbat is meg lehet találni $n-1$ összehasonlítással, azaz megoldható $2n-2$ összehasonlítással. (Sőt $(2n-3)$-mal is. Miért?)

Ennél jobb a következő: előbb hasonlítsuk össze az elsőt a másodikkal, a harmadikat a negyedikkel, stb. Ez után vegyük minden párból a kisebb elemet. A legkisebb közülük kerül ki. Használjuk ezekre a minimumot kereső algoritmust. Hasonlóan a páronkénti nagyobb elemek között találjuk meg a maximumot. Ha páratlan sok elem van, akkor az utolsó pár helyett 3 elemből határozzuk meg a kicsit és nagyot.

A lépésszám: $\lfloor n/2 \rfloor + 2 + 2(\lfloor n/2 \rfloor - 1)$, ami kb. $1,5n$. Pontosabban, ha n páros, akkor $1,5n-2$, páratlan esetben pedig $(n-1)/2 + 2 + 2(n-1)/2 - 2 = 1,5(n-1)$.

Megjegyzés: meggondolható, hogy $1,5n-2$ kell is. Legyen B azoknak az elemeknek a száma, amelyek még lehetnek legkisebbek és legnagyobbak is, I, ami már csak legkisebb lehet (volt már nála nagyobb), A, ami már csak legnagyobb lehet (volt nála kisebb, E pedig a többi, aminél már volt kisebb és nagyobb is. Kezdetben $|B| = n$, a többi üres, a végén B üres, $|I| = |A| = 1$.

Két B-beli összehasonlításakor B kettővel csökken, I és A eggyel-eggyel nő. Ha egy B-belit nem B-belivel hasonlítunk, akkor B csökken és mindig lehet olyan eredmény, amikor I és A egyike nő, a másik nem csökken (azaz I és A uniója nem csökken). Ha két I-belit vagy két A-belit hasonlítunk, akkor egyikük átkerül az E-be. A többi esetben mindig lehet olyan válasz, hogy sem I sem A nem változik. Ahhoz, hogy a B kiürüljön ezek szerint kell legalább $n/2$ összehasonlítás és eközben I és A uniója nem változik. A B-ből minden elem előbb az I-be vagy A-ba kerül, ahonnan viszont egyszerre csak egy tud kikerülni, tehát ezek (majdnem) kiürítéséhez kell összesen legalább $n-2$ összehasonlítás. Tehát együtt van legalább $1,5n-2$.