Óbudai Egyetem
Neumann János Informatikai Kar

Szoftverfejlesztés párhuzamos és elosztott környezetben

Rendezési algoritmusok párhuzamosítása

Hallgató neve: **Héthy Zoltán**
Azonosítója: **JW3L34**
Szak megnevezése: **Mérnök Informatikus MSc**
Aktuális tanév: **2016/2017**

Rendezési algoritmusok

A rendezés az egyik legalapvetőbb programozási probléma, ahol adatokat kell valamilyen szempont alapján nagyság szerinti sorrendbe rendezni. Egy algoritmust akkor nevezünk rendezésnek, ha az valamilyen szempont alapján sorba állítja elemek egy listáját. Ezen rendezési algoritmusok használata elengedhetetlen a programozásban, mivel egy rendezett adathalmazzal több és hatékonyabb műveletek végezhetünk el, mint egy rendezetlennel. [1]

A rendezési algoritmusoknak két fő típusa van: az összehasonlító (melyek az elemek összehasonlítása alapján állítják elő a rendezett kimenetet) és a nem összehasonlító (melyek az elemek összehasonlítása nélkül képesek erre) rendezések.

Összehasonlító rendezések:

- Bubble Sort (Buborékrendezés)
- Insertion Sort (Beszúró rendezés)
- Selection Sort (Kiválasztó rendezés)
- Merge Sort (Összefésülő rendezés)
- Quick Sort (Gyorsrendezés)
- Heap Sort (Kupacrendezés)

Nemösszehasonlító rendezések:

- Bucket Sort (Edényrendezés)
- Counting Sort (Leszámláló rendezés)
- Radix Sort (Számjegyes rendezés)

[1]

A feladatom során csak az összehasonlító rendezési algoritmusokkal szerettem volna foglalkozni, ezek közül is a Bubble, a Selection, a Quick és a Merge Sort rendezéssel, mivel ezek a jól párhuzamosítható algoritmusok közé tartoznak.

Bubble Sort (Buborékrendezés)

A buborékrendezés egy egyszerű algoritmus, amelyet főleg az oktatásban használunk, mivel nem igazán hatékony. A buborékrendezésnél az egymás mellett álló elemeket hasonlítjuk össze, és szükség esetén sorrendjüket felcseréljük. Ezt mindaddig folytatjuk, amíg szükség van cserére. Jellemző rá a sok csere, és az elem lassan kerül a helyére. [2]

Selection Sort (Kiválasztó rendezés)

A kiválasztásos rendezés egy egyszerű, négyzetes időben futó rendezési algoritmus. Az alapötlet az, hogy kiválasztjuk a rendezendő tömb legkisebb elemét, és kicseréljük a tömb legelső elemével. Ezzel a tömb első eleme megkapta a végső értékét, és a feladat

egyszerűsödött a tömb maradékának rendezésére. Az algoritmust addig ismételjük a maradék tömbön, amíg csak egy elem marad. [3]

Quick Sort (Gyorsrendezés)

Az egyik legnépszerűbb rendezési algoritmus, amely átlagos esetben gyorsabb, mint a többi algoritmus, viszont hátránya hogy a legrosszabb esetben lassú, és nem stabil rendezés. Rekurzív algoritmus, kettéosztja a rendezendő listát egy kiemelt elemnél kisebb és nagyobb elemekre, majd a részekre külön-külön alkalmazza a gyorsrendezést. [1]

Merge Sort (Összefésülő rendezés)

Az összefésülő rendezés alapelve az összefésülés műveletén alapszik, amely két rendezett tömbből egy új rendezett tömböt állít elő. Az összefésüléskor mindkét tömbnek megvizsgáljuk az első elemét. A két elem közül a kisebbiket beírjuk az eredménytömb első szabad eleme helyére. A felszabaduló helyre újabb elemet veszünk abból a tömbből, ahonnan előzőleg a kisebbik elem jött. Ezt a tevékenységet folytatjuk mindaddig, míg valamelyik kiinduló tömbünk ki nem ürül. Ezután a még vizsgálat alatt lévő elemet, valamint a megmaradt másik tömb további elemeit sorba az eredménytömbhöz hozzáírjuk a végén. [2]

A tervezés koncepciója

A rendezési algoritmusok párhuzamosításához az alapötlet a rendezendő tömb tartományokra való bontása. Vagyis a párhuzamosítást adatdekompozíció segítségével valósítjuk meg. Ekkor különböző szálak ugyanazt a feladatot oldják meg párhuzamosan a kisebb résztömbökön. Ehhez először szükségünk van egy nagyméretű tömbre, amit véletlen számokkal töltöttünk fel, ez lesz a rendezendő tömb. Majd ezt a tömböt felosztjuk több tartományra (részsorozatra), amelyekre külön-külön, szálak segítségével meghívjuk a rendező eljárást. Ezzel csökkentve a rendezés bonyolultságát és az eljárás idejének hosszát is. Azonban a párhuzamosítás nem ennyire egyszerű, mivel a rendezések részeredményei rendezett sorozatok, melyek egyszerű uniója általában nem produkál teljesen rendezett végeredményt. Ezért alkalmaznunk kell még egy összefésülő metódust is, amely alkalmas két (vagy több) rendezett részsorozatból kevés plusz művelet ráfordítása mellett teljesen rendezett sorozatot készíteni, amely tartalmazza mindkét részsorozat minden elemét. [4]

Sajnos ez az eljárás azonban csak bizonyos rendezési algoritmusok esetén működik hatékonyan. Így tehát jól alkalmazható például a Bubble és a Selection Sortnál, azonban nincs sok értelme a Quick Sort vagy a Merge Sort esetében, mivel azok eleve nagyon gyors algoritmusok és csak vissza fognánk őket. Ezek párhuzamosítása más módszerrel történik, amelyek implementálása már nem egyszerű feladat.

Továbbá mérlegelni kell a megvalósítás során, hogy a rendezendő elemek mennyisége, vagy az összehasonlító művelet időigénye indokolja-e a párhuzamosítást. Ezért különböző tesztek segítségével összehasonlítjuk a soros és párhuzamos futásokat, amelyeknél figyelembe vesszük a rendezés költségét és futási idejét. Az algoritmus költségének meghatározásához két tényezőt veszünk figyelembe, az egyik az összehasonlítások száma, a másik az elemcserék száma. Továbbá lefuttatjuk a rendezési algoritmusokat különböző méretű és különböző esetű tömbökön. A legjobb esetben a tömbünk rendezett, a legrosszabb esetben fordítottan rendezett, átlagos esetben pedig véletlenszerűen van összekeverve. [5]

A feladat megvalósítása

A feladatot konzolos alkalmazásként implementáltam C# nyelven. Három osztályt tartalmaz: a `Program`, a `Tomb` és a `Rendezesek` osztályt.
A `Program` osztály a főosztály, ebben futtatom a programom. Ehhez elegendő létrehoznom egy tömböt, amire meghívom a rendezési algoritmusokat.

A `Tomb` osztály különböző tömbkezelő függvényeket és metódusokat tartalmaz, amelyekkel leegyszerűsíthető a programom kezelése. Tartalmaz egy konstruktort is, amely példányosításkor automatikusan létrehoz egy n elemű tömböt random értékekkel. Állíthatók benne a tömb paraméterei (méret, max elemszám, elemei). Tömbök egyszerű lekérését, másolását, kiíratását és legrosszabb esetbe rendezését teszi lehetővé.

A `Rendezések` osztály tartalmazza a rendezési algoritmusokat (1. ábra) soros és párhuzamos futáshoz is egyaránt. A Bubble Sort és a Selection Sort szekvenciálisan és párhuzamosan is implementálva lettek. Párhuzamos futás esetén 2 vagy 4 szálon futtathatók. A Quick Sort viszont csak szekvenciálisan lett implementálva, mivel ennek a párhuzamosítása túlságosan bonyolult volt számomra.
A rendező metódusok mindig egy tömböt várnak paraméterként és 3 referencia változót, amelyekkel a rendezés cseréinek száma, összehasonlításainak száma és a futási ideje adható át. A visszatérési értéke egy tömb lesz, amely a rendezett tömböt adja vissza. A rendezés előtt előbb lemásolódik az eredeti tömb az `Array.Copy()` függvénnyel. Abból a célból, hogy megmaradjon a rendezendő tömb a többi rendezési algoritmus számára is. Ezután párhuzamos futások esetén a `reszTomb()` metódus hívódik meg, amely szétosztja a rendezendő tömböt annyi darab résztömbre, ahány szálon szeretnénk futtatni az algoritmust. Aztán következhet a rendezés, ami előtt még elindítunk egy Stopwatchot is, hogy mérje az rendezés futási idejét. Párhuzamos futás esetén a rendezések külön metódusokban lettek implementálva, amiket meg kell hívni külön-külön a szálakra. Szálakat a `Thread` osztály segítségével hozhatunk létre, metódusokat pedig a legegyszerűbben a lambda kifejezések segítségével adhatunk át a szálaknak.

```
Thread t1 = new Thread(() => pBubbleSort(rtomb1, ref csereSz, ref osszhSz));
```

A `Thread` osztály `Start()` példánymetódus segítségével indítjuk el a szálak futását, majd a `Join()` metódussal a főszálat várakoztatjuk, míg az adott szálak be nem fejeződnek. Ezután a rendezett résztömböket a `merge()` metódus segítségével összefésüljük. Négy szál esetén többször is meg kell hívnunk az összefésülő eljárást, hogy az eredeti rendezett tömböt kapjuk. Az összefésülésnél is próbálkoztam párhuzamosítással, azonban ez minden esetben lassabb futási időt produkált. Szekvenciális összefésüléseknél néhány µs-ot vett igénybe a mergelés, párhuzamosnál viszont ms-okkal lassabb időt eredményezett. Ezért az összefésüléseknél ésszerűbb volt a szekvenciális kód használata. Az összefésülések után leállítjuk a Stopwatchot, ami ms-ban adja vissza nekünk, hogy meddig tartott az algoritmus futása. A rendezési algoritmus futás közben egy-egy változóban számolja az elemcserék és az összehasonlítások számát is. Több szál esetén előfordulhat, hogy a szálak egyszerre próbálják meg növelni a változó értékét, ezért csak egyszer inkrementálódik, és a végén hamis értéket ad vissza. Ezért ezeket a kódrészeket szinkronizálni kellett. Ezt meg lehetett volna oldani `lock` utasítással is, de ennél egy kényelmesebb megoldást biztosított az `Interlocked` statikus osztály `Increment()` metódusa. Ez a metódus a processzor hardveres támogatását kihasználva, atomi idő alatt inkrementálja a változó értékét, ami által szükségtelenné válik a zárolás.

```
// SOROS RENDEZÉSEK
// ---------------- Bubble Sort -------------------------
Bubble Sort
// ------------------------------------------------------
// ---------------- Selection Sort ----------------------
Selection Sort
// ------------------------------------------------------
// ---------------- Quick Sort --------------------------
Quick Sort
// ------------------------------------------------------

// PÁRHUZAMOS RENDEZÉSEK (2 szálon)
// ---------------- Bubble Sort -------------------------
Parallel Bubble Sort (2 szál)
// ------------------------------------------------------
// ---------------- Selection Sort ----------------------
Parallel Selection Sort (2 szál)
// ------------------------------------------------------

// PÁRHUZAMOS RENDEZÉSEK (4 szálon)
// ---------------- Bubble Sort -------------------------
Parallel Bubble Sort (4 szál)
// ------------------------------------------------------
// ---------------- Selection Sort ----------------------
Parallel Selection Sort (4 szál)
// ------------------------------------------------------
```

1. ábra: Rendezési algoritmusok

Tesztelés

A rendezési algoritmusok tesztelésénél három dolgot vettünk figyelembe: a rendezés futási idejét, az összehasonlítások számát és az elemcserék számát. Ezek alapján tudtuk meghatározni az algoritmus költségét és összehasonlítani a különböző rendezési algoritmusokat.

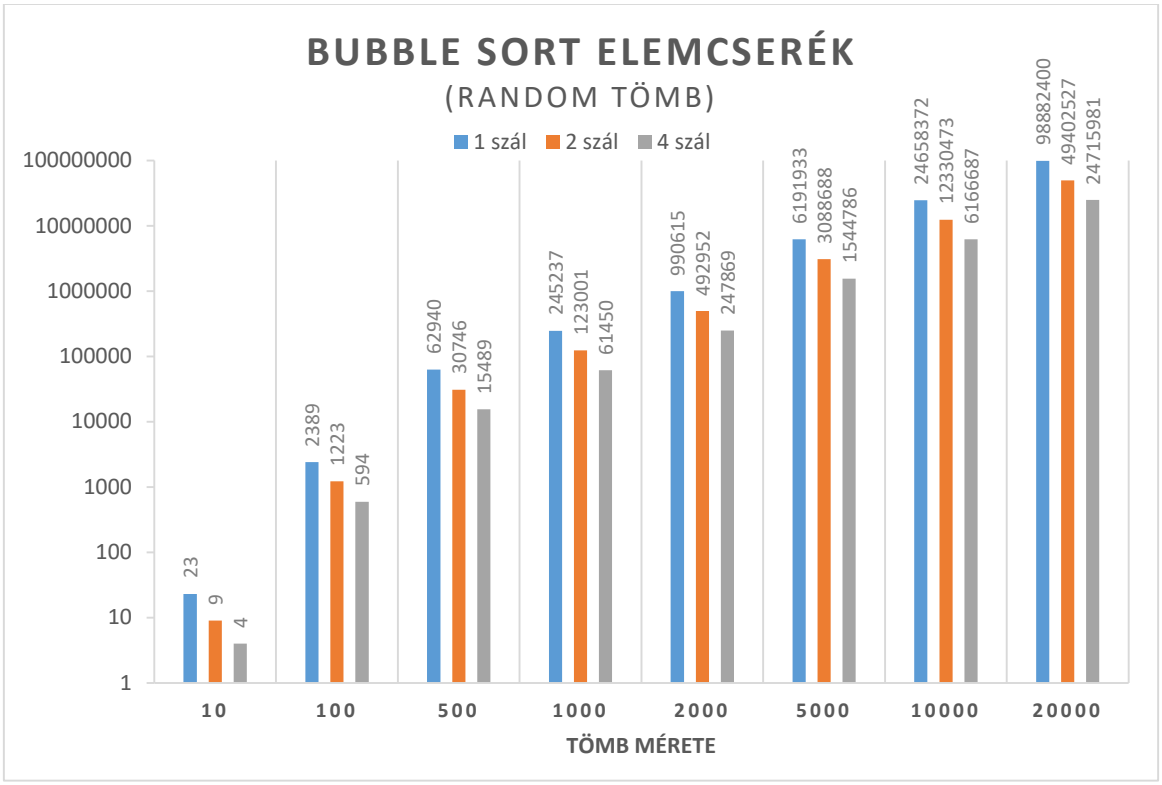
A teszteléseket egy erősnek számító laptopon (Lenovo Y510P – Intel Core i7-4700MQ processzor) végeztem. A Bubble és a Selection Sortot tesztelgettem és hasonlítottam össze soros és párhuzamos futások esetén 2 és 4 szálon is. A teszteléseket különböző méretű és típusú tömbökre végeztem el. A tömb elemek 0-tól 99-ig fordulhattak elő. Random, rendezett és fordítottan rendezett tömb esetekre futtattam az algoritmusokat 10, 100, 500, 1000, 2000, 5000, 10000, 20000 elemből álló tömböknél. Egy for ciklus segítségével minden esetre 10 különböző rendezést futtattam le más-más tömbökkel, és ezeknek a futási eredményeknek az átlagait (2. ábra) dokumentáltam. Az elemcserék és az összehasonlítások átlagai álltalában pontos eredményt adtak, de az átlagos futási időt gyakran manuálisan kellett utána számolni. Mivel párhuzamos rendezések esetén előfordult, hogy egy-egy rendezés futási ideje irreálisan hosszú ideig tartott. Ez gondolom betudható a kedvezőtlen CPU ütemezésnek. Ezeket a futási időket nem vettük figyelembe az átlag futási idő számításakor. A for ciklusos tesztelés előtt szükséges volt még egy 0. rendezés indítása is, mivel a legelső rendezés futási ideje szinte minden esetben lassabb volt a többitől. Ezt a jelenséget a .NET keretrendszer fordításának tudhattuk be, mivel futtatáskor a .NET először létrehoz egy közteskódot, ezt átalakítja gépi kóddá, amit a processzor futtatni tud, és ami egyszer már meg van neki gépi kódban, azt a későbbiekben is fel tudja használni. Továbbá nagyon fontos volt még, hogy ne debug módban futtassuk a programunkat, mert az belassíthatja az algoritmusok futását.

```
A rendezendő tömb mérete: 2000

----------------------------------------------------------------

Bubble Sort - Szekvenciálisan:
Átlagosan az elemcserék száma: 985655
Átlagosan az összehasonlítások száma: 3885256
Az átlagos futási idő: 27,281 ms

----------------------------------------------------------------

Bubble Sort - Párhuzamosan (2 szál):
Átlagosan az elemcserék száma: 492215
Átlagosan az összehasonlítások száma: 1923874
Az átlagos futási idő: 130,01904 ms

----------------------------------------------------------------

Bubble Sort - Párhuzamosan (4 szál):
Átlagosan az elemcserék száma: 247613
Átlagosan az összehasonlítások száma: 946453
Az átlagos futási idő: 117,4969 ms
```

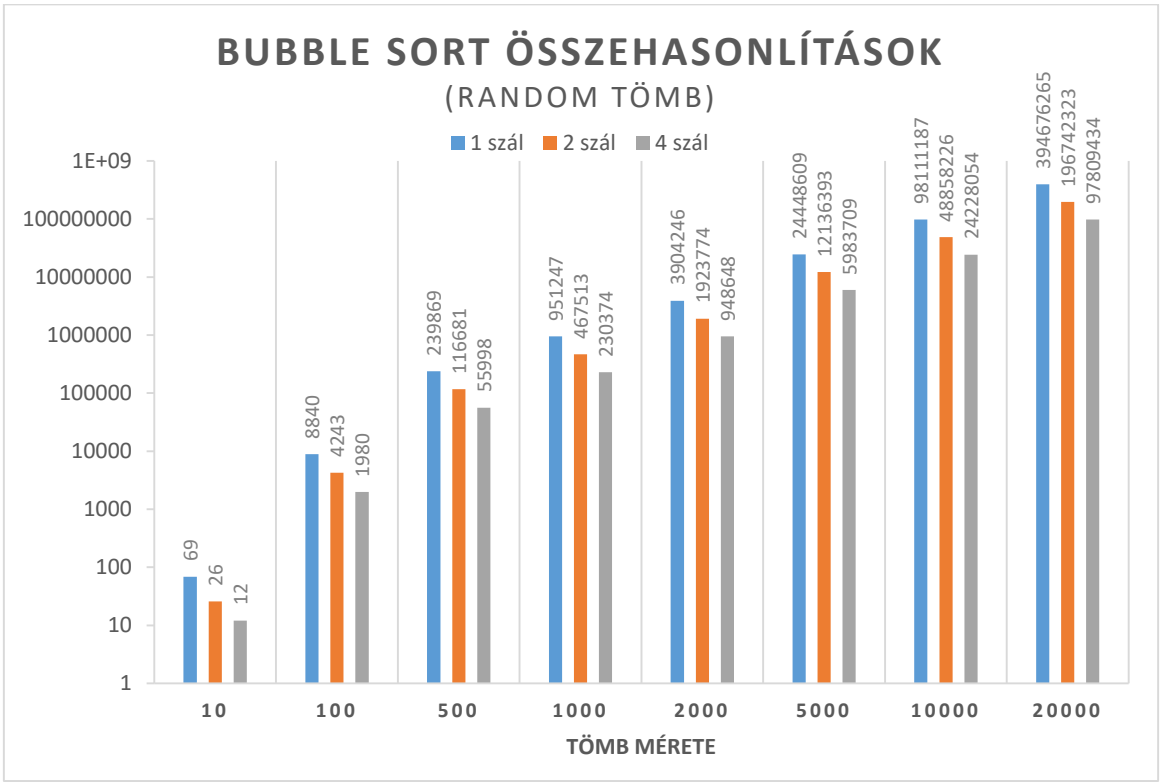
2. ábra: Futási eredmények

Mérési eredmények

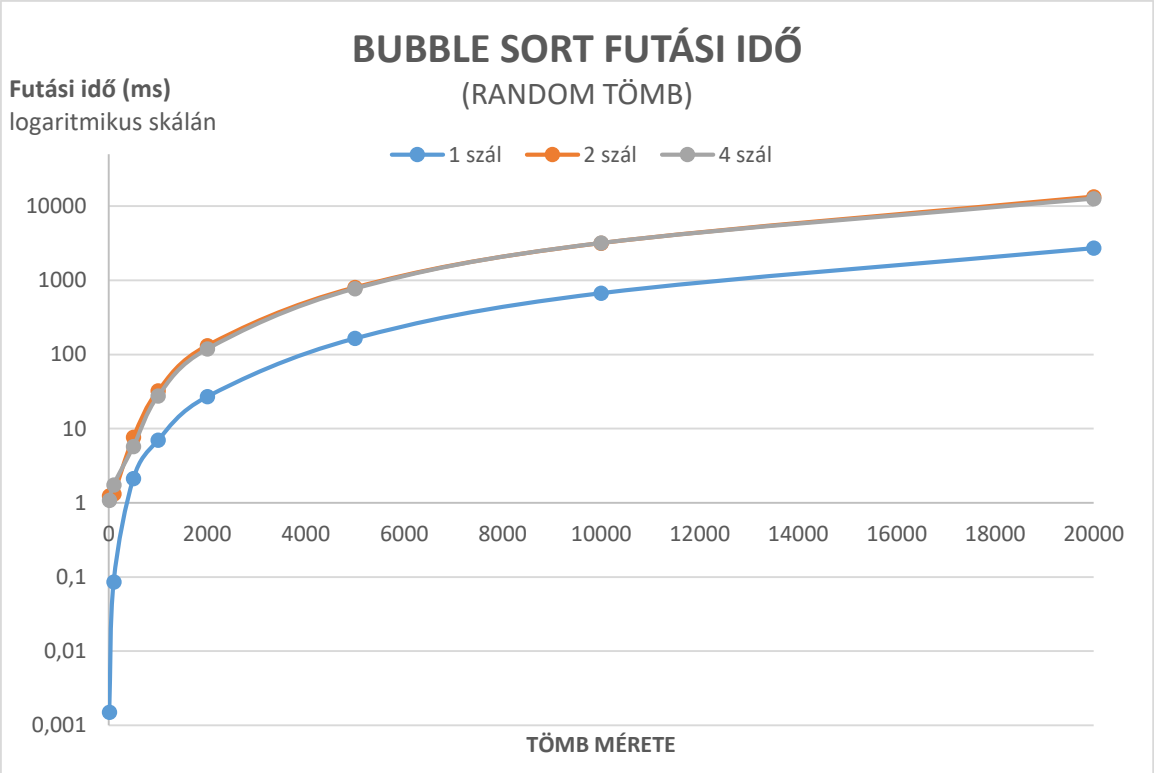
Bubble Sort (véletlenszerű tömb esetén)



3. ábra: Bubble Sort elemcserék (random)



4. ábra: Bubble Sort összehasonlítások (random)



5. ábra: Bubble Sort futási idők (random)

A 3. ábrán a Bubble Sort algoritmus elemcseréinek száma látható különböző tömbméretekre soros és párhuzamos futások esetén. Megfigyelhetjük, hogy soros futás esetén szükséges a legtöbb elemcsere, és 4 szálas futás esetén pedig a legkevesebb.

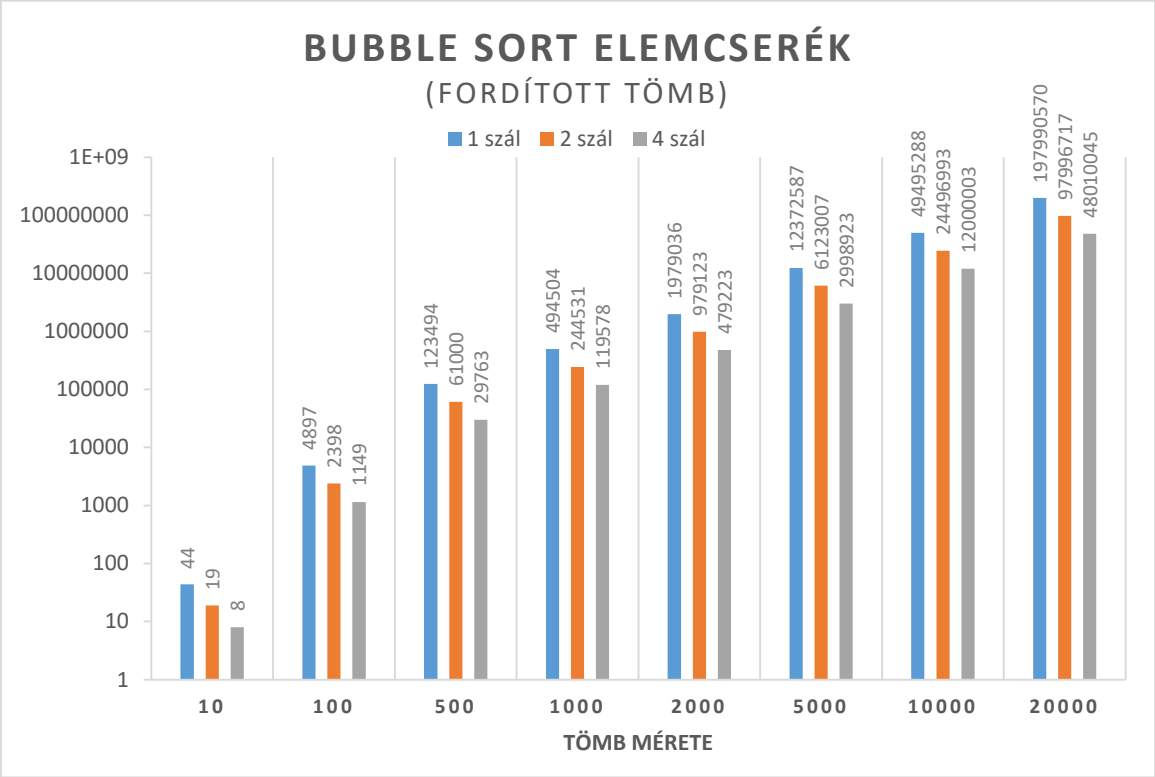
A 4. ábra a rendezéshez szükséges összehasonlítások számát ábrázolja. Ebben az esetben is a soros futásnál lesz szükségünk a legtöbb összehasonlításra, és szintén 4 szál esetén kapjuk a legkevesebb összehasonlítást.

Az 5. ábrán a mért futási idők láthatók. Azt várnánk, hogy a 4 szálas futás lesz a leggyorsabb, azonban ennél az algoritmusnál nem valami jól működik ez a párhuzamosítási megoldás. Megfigyelhetjük, hogy kezdetben kis tömbök esetén sokkal lassabb a párhuzamos algoritmus, azonban el kezd gyorsulni, ahogy növekszik a tömb mérete, de egy méret után megáll ez a gyorsulás, és körülbelül 4,7x lassabb lesz a párhuzamos algoritmus a szekvenciálisnál. A 4 szálas kezdetben lassabb a 2-nél, azonban 500-as tömb méret után pár milliszekundummal a 4 szálon futtatott algoritmus kicsivel mindig gyorsabb lesz.

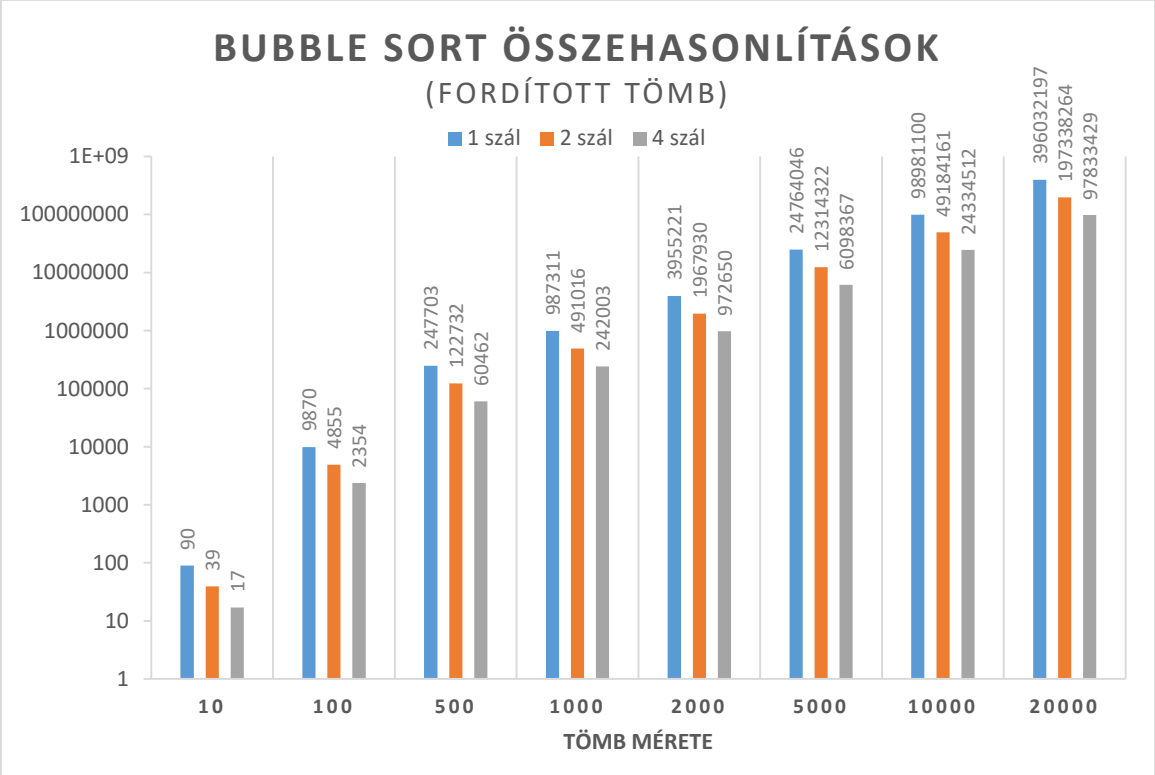
A következő táblázat szemlélteti, hogy mennyiszer gyorsabb a szekvenciális eljárás a 2 vagy 4 szálas párhuzamos rendezéstől:

Párhuzamos futás sebessége (lassabb) a szekvenciálishoz képest a tömb méretének függvényében									
tömb mérete		10	100	500	1000	2000	5000	10000	20000
idő	2 szál	833,3	15,3	3,6	4,6	4,9	4,9	4,7	4,9
	4 szál	720,0	20,3	2,7	3,9	4,4	4,7	4,7	4,6

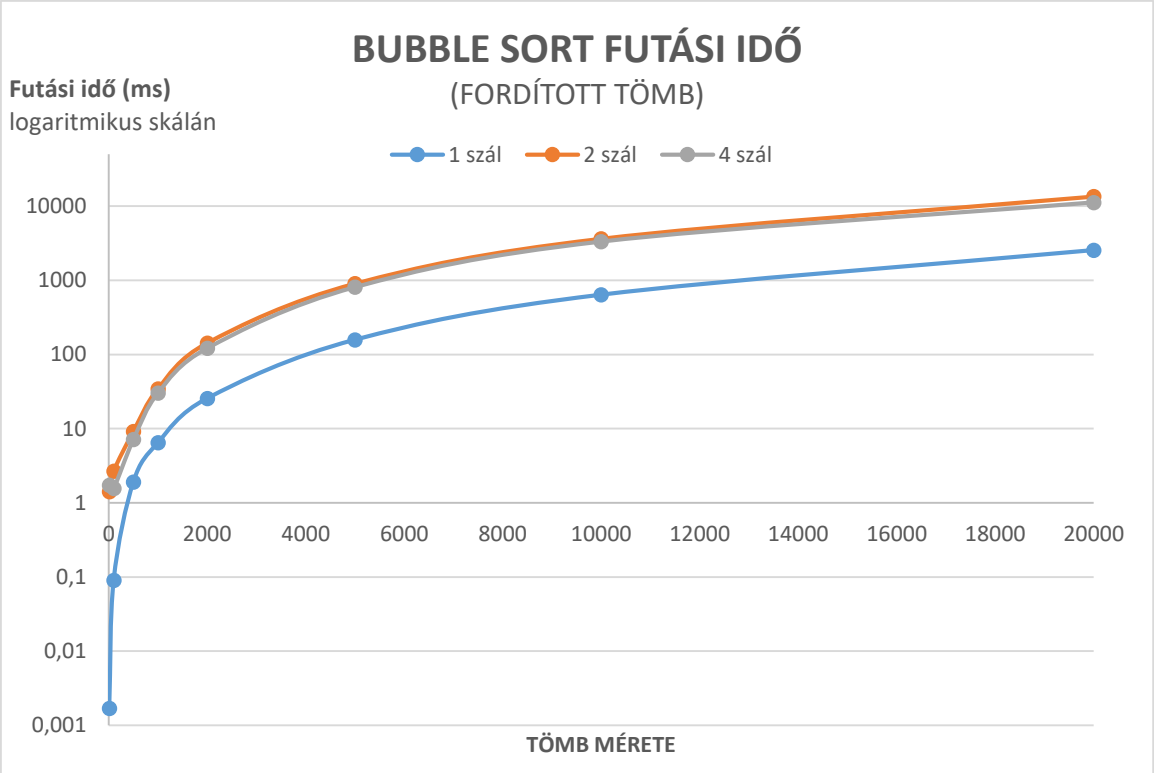
Bubble Sort (fordítottan rendezett tömb esetén)



6. ábra: Bubble Sort elemcserék (fordított)



7. ábra: Bubble Sort összehasonlítások (fordított)



8. ábra: Bubble Sort futási idők (fordított)

A 6. ábrán látható a rendezés szempontjából legrosszabb, fordítottan rendezett tömbökre alkalmazott Bubble Sort algoritmus elemcseréinek száma. Ebben az esetben is a soros rendezésnél van a legtöbb elemcserére szükségünk, és a 4 szálasnál pedig a legkevesebbre. Továbbá, ha jobban megfigyeljük, itt 2x annyi elemcserére lesz szükségünk, mint a véletlenszerűen generált tömböknél.

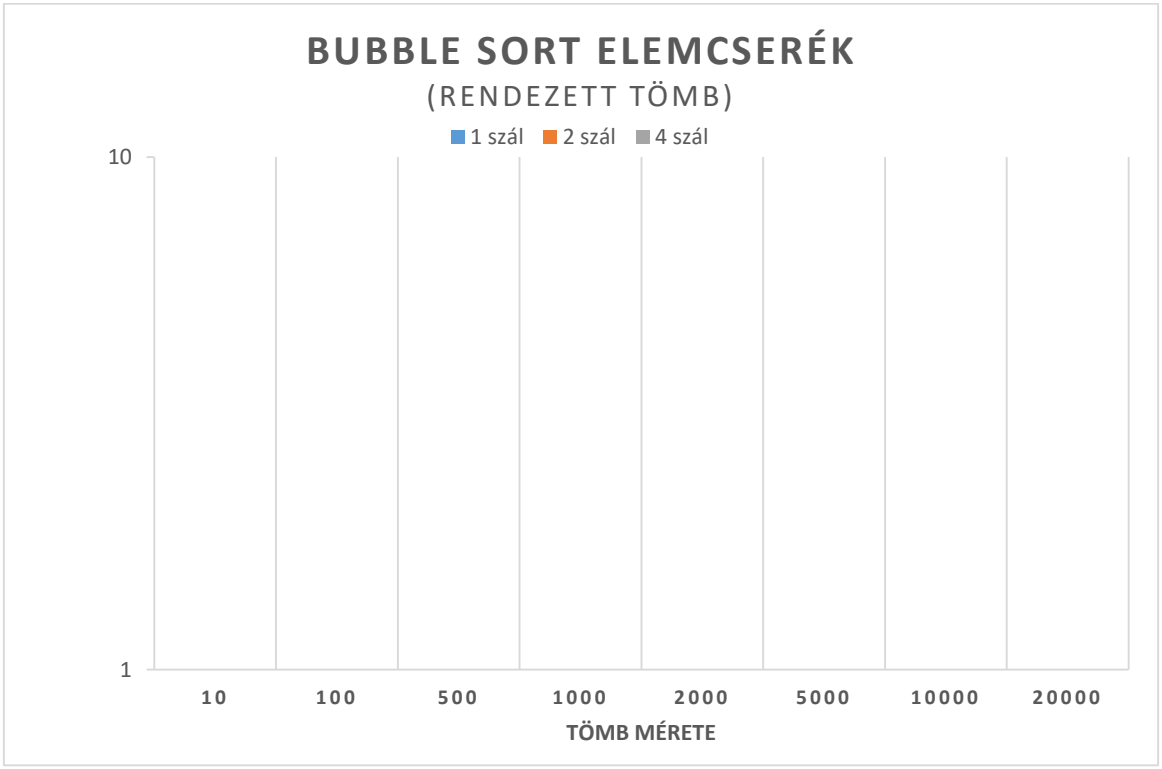
A 7. ábra az összehasonlítások számát szemlélteti erre a tömb esetre. Itt is megfigyelhető, hogy több összehasonlítás szükséges, azonban itt nincs olyan nagy a különbség, mint az elemcseréknél.

A 8. ábra a futási időket ábrázolja. Érdekes módon amennyivel több elemcserére van szükség itt, nem vehető észre a futási idő növekedés. Itt szinte minden esetben jobban működött a 4 szálas futás a 2 szálasnál, de még mindig a szekvenciális biztosítja magasan a legjobb időt.

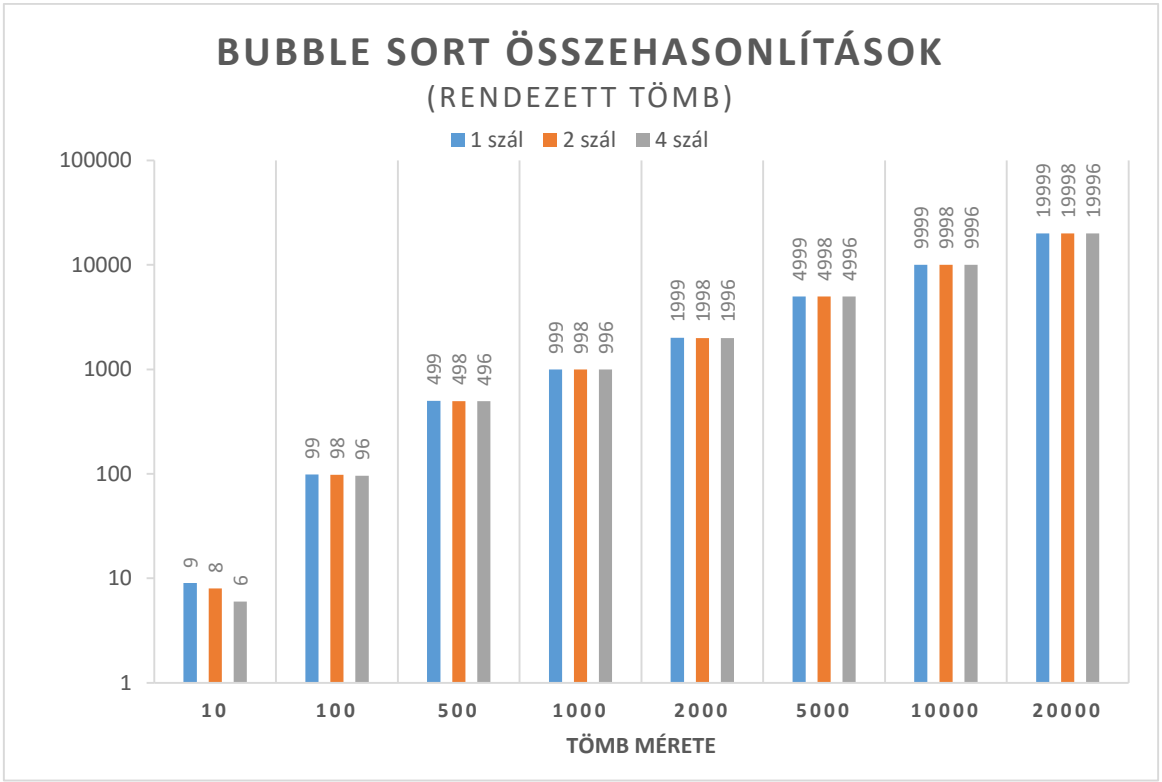
A szekvenciális futás gyorsasága a párhuzamos rendezésekhez viszonyítva:

Párhuzamos futás sebessége (lassabb) a szekvenciálishoz képest a tömb méretének függvényében									
tömb mérete		10	100	500	1000	2000	5000	10000	20000
idő	2 szál	835,3	29,7	4,8	5,3	5,6	5,7	5,7	5,3
	4 szál	1017,6	17,4	3,7	4,7	4,7	5,1	5,2	4,4

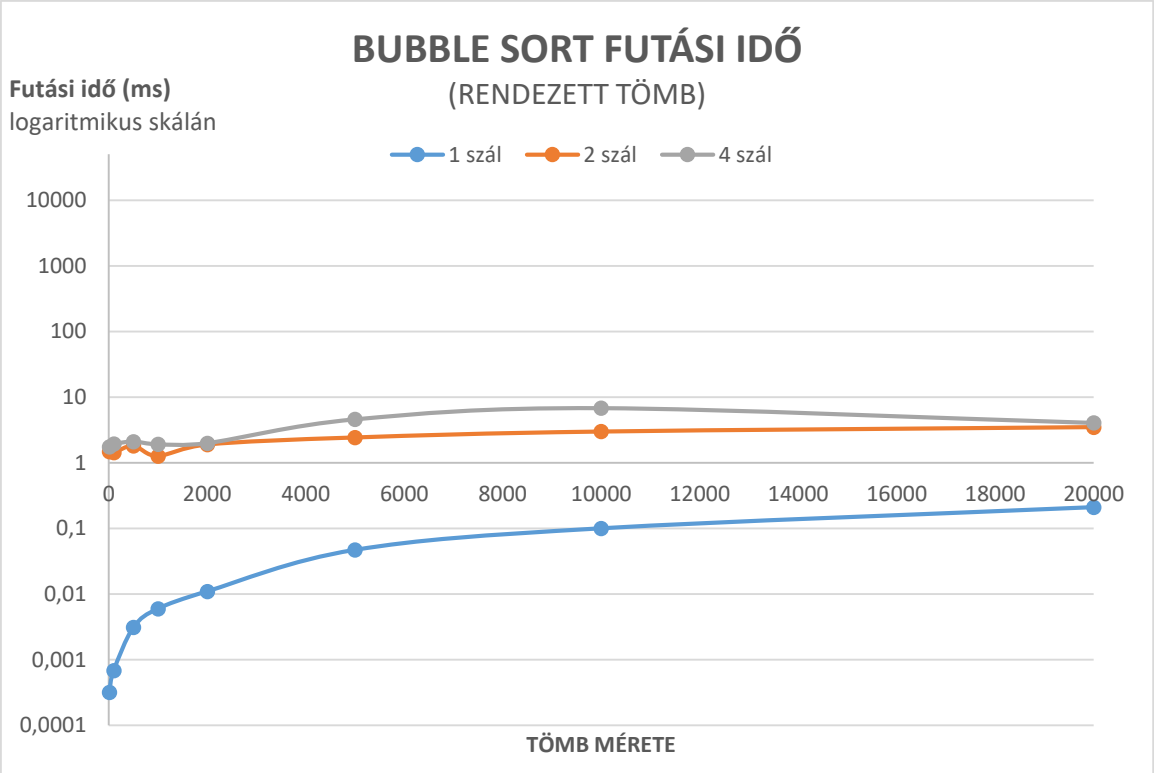
Bubble Sort (rendezett tömb esetén)



9. ábra: Bubble Sort elemcserék (rendezett)



10. ábra: Bubble Sort összehasonlítások (rendezett)



11. ábra: Bubble Sort futási idők (rendezett)

A 9. ábra azt mutatja meg, ha egy rendezett tömbre lefuttatjuk az algoritmust, akkor nem történik elemcsere.

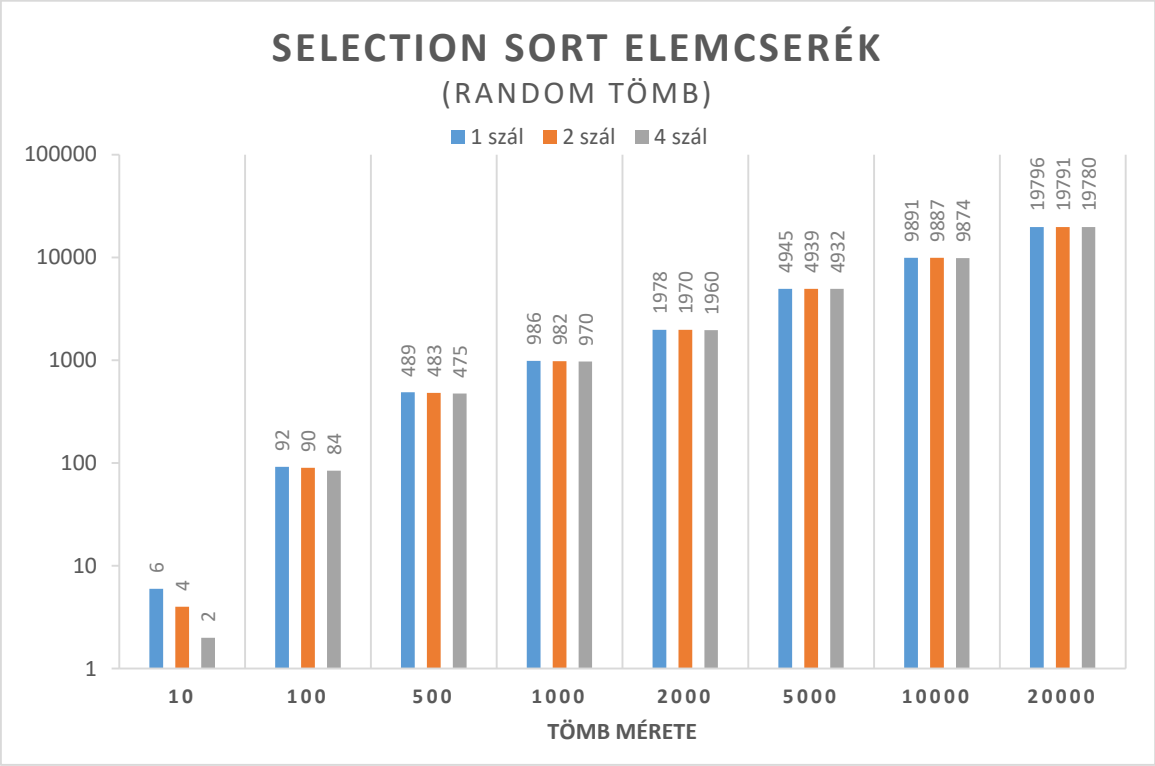
A 10. ábra az összehasonlítások számát ábrázolja rendezett tömbök esetén. Ezen a grafikonon azt figyelhetjük meg, hogy soros rendezés esetén n-1, két szál esetén n-2, négy szál esetén pedig n-4 összehasonlításra van szükség.

A 11. ábra a futási időket tartalmazza. Itt most nagy eltérést mutat az előző két tömb esettől, mivel itt szinte a tömb mérettől függetlenül azonos idő alatt nagyon gyorsan lefut az algoritmus ezekre a tömbökre. Ez annak köszönhető, hogy itt csupán egyszer megy végig és végez összehasonlításokat a tömbön. Bár itt szinte egyformán gyors mindegyik futás, a legjobban itt is a szekvenciális működik.

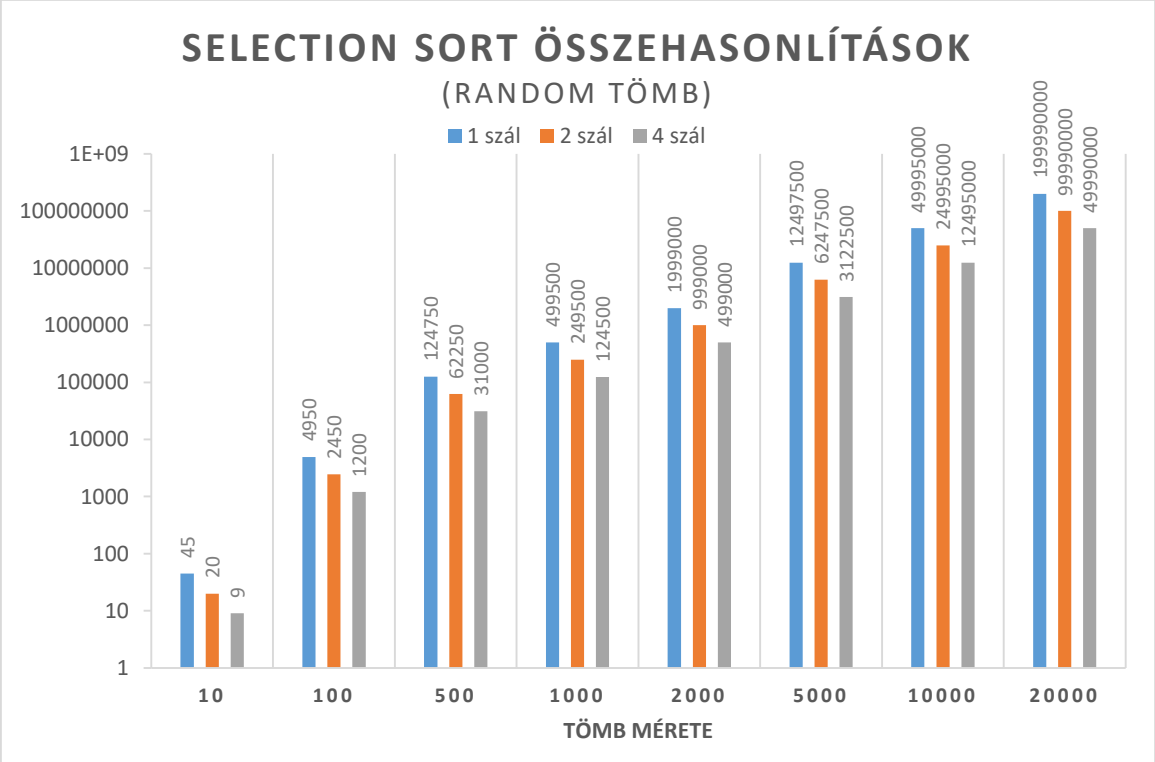
A szekvenciális futás gyorsasága a párhuzamos rendezésekhez viszonyítva:

Párhuzamos futás sebessége (lassabb) a szekvenciálishoz képest a tömb méretének függvényében									
tömb mérete		10	100	500	1000	2000	5000	10000	20000
idő	2 szál	4625,0	2102,9	583,9	211,7	172,7	51,7	29,9	16,6
	4 szál	5500,0	2838,2	677,4	318,3	180,9	97,7	68,0	19,3

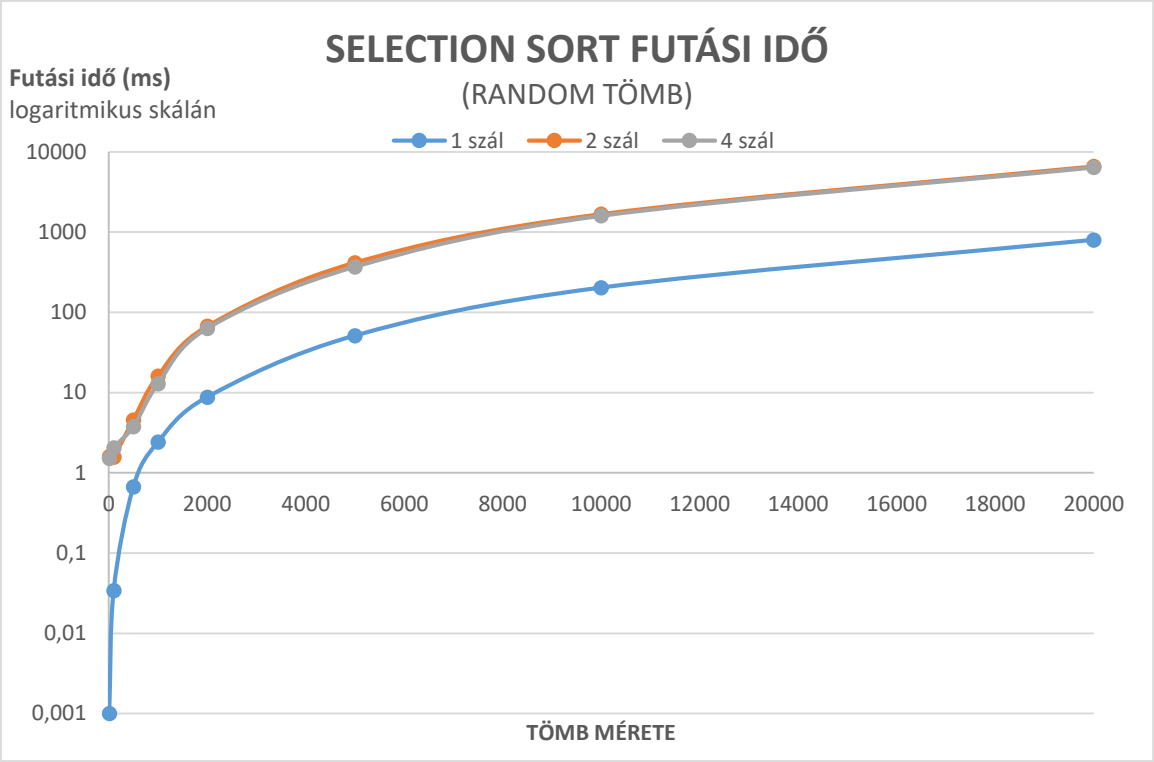
Selection Sort (random tömb esetén)



12. ábra: Selection Sort elemcserék (random)



13. ábra: Selection Sort összehasonlítások (random)



14. ábra: Selection Sort futási idők (random)

A 12. ábrán a Selection Sort algoritmus elemcseréinek száma látható. Itt azt figyelhetjük meg, hogy nagyjából azonos mennyiségű elemcsere szükséges soros és párhuzamos futás esetén is, de egy picivel itt is a többszálú futások eredményesebbek.

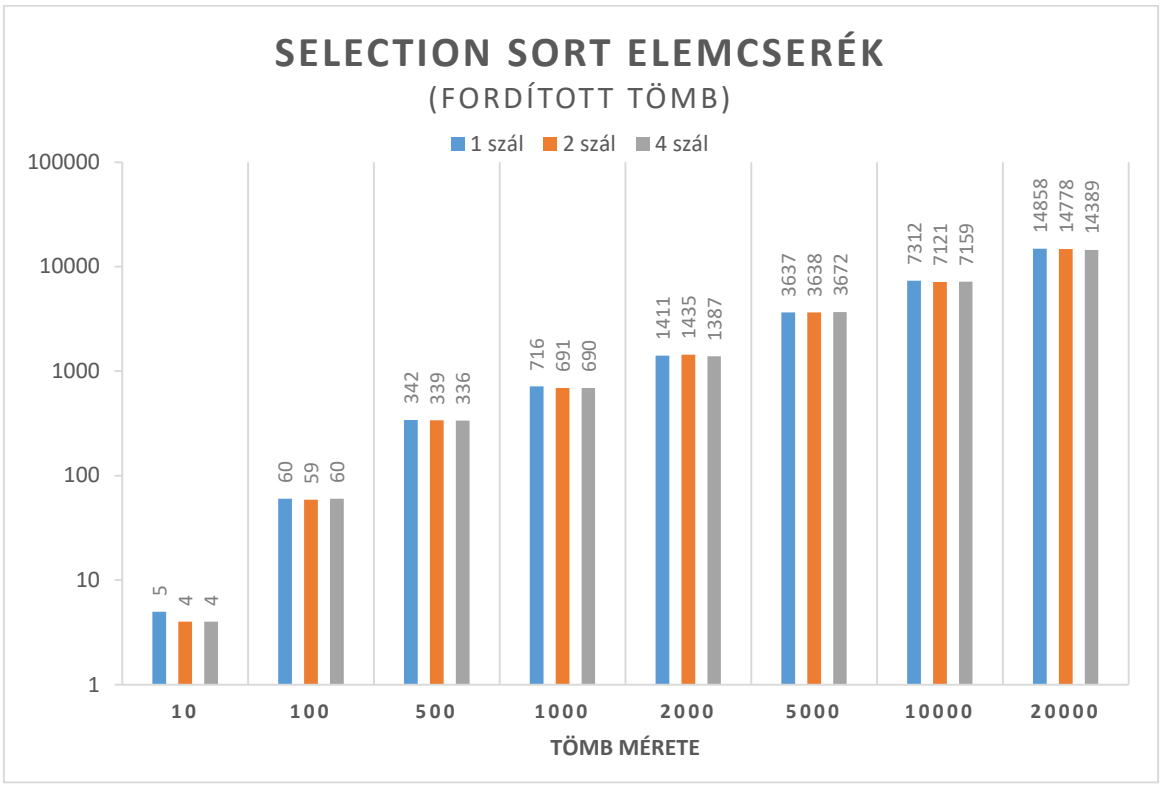
A 13. ábra a rendezés összehasonlításainak számát szemlélteti. Ebben az esetben a soros futásnál lesz a legtöbb összehasonlítás, a 4 szálas esetén pedig a legkevesebb összehasonlítás. Soros futásnál az összehasonlítások számát az $\frac{1}{2}\, n^2 - \frac{1}{2}\, n$ képlettel is megkaphatjuk.

A 14. ábrán a futási idők láthatók. Ebben az esetben sem a többszálas futás lesz a gyorsabb. Továbbá itt is megfigyelhető, hogy ahogy növekszik a tömb mérete, úgy gyorsul a párhuzamos futás, viszont itt a szekvenciális nagyjából 8x gyorsabb marad a párhuzamos algoritmusnál. A 4 szálas algoritmus itt is 500 feletti tömböknél lesz gyorsabb a 2 szálasnál. Ennél a rendezésnél a szekvenciális és a párhuzamos algoritmus futási ideje között nagyobb lesz a különbség, mint a Bubble Sortnál, azonban így is gyorsabb rendezést biztosít. Tehát ennél az algoritmusnál ez a párhuzamosítás még rosszabbul működik, mint a másiknál.

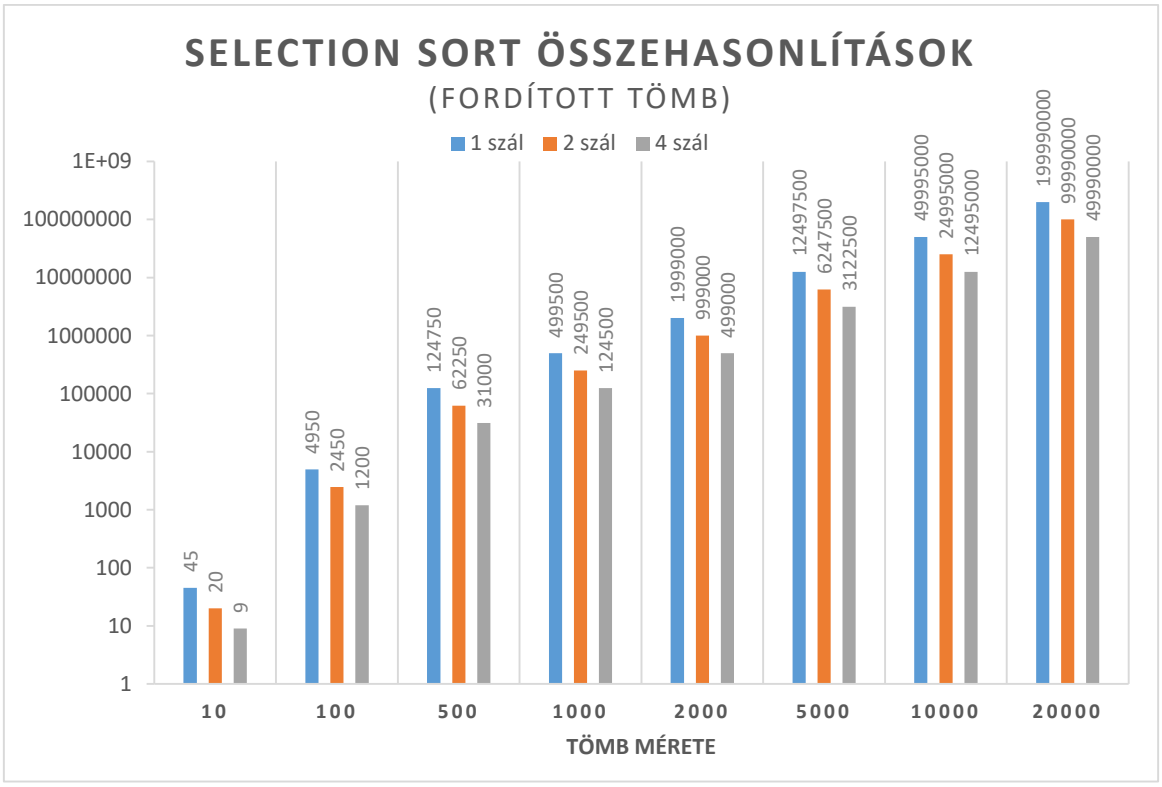
A szekvenciális futás gyorsasága a párhuzamos rendezésekhez viszonyítva:

Párhuzamos futás sebessége (lassabb) a szekvenciálishoz képest a tömb méretének függvényében									
tömb mérete		10	100	500	1000	2000	5000	10000	20000
idő	2 szál	1600,0	46,5	6,8	6,7	7,6	8,1	8,2	8,2
	4 szál	1520,0	60,6	5,6	5,3	7,2	7,2	7,9	8,0

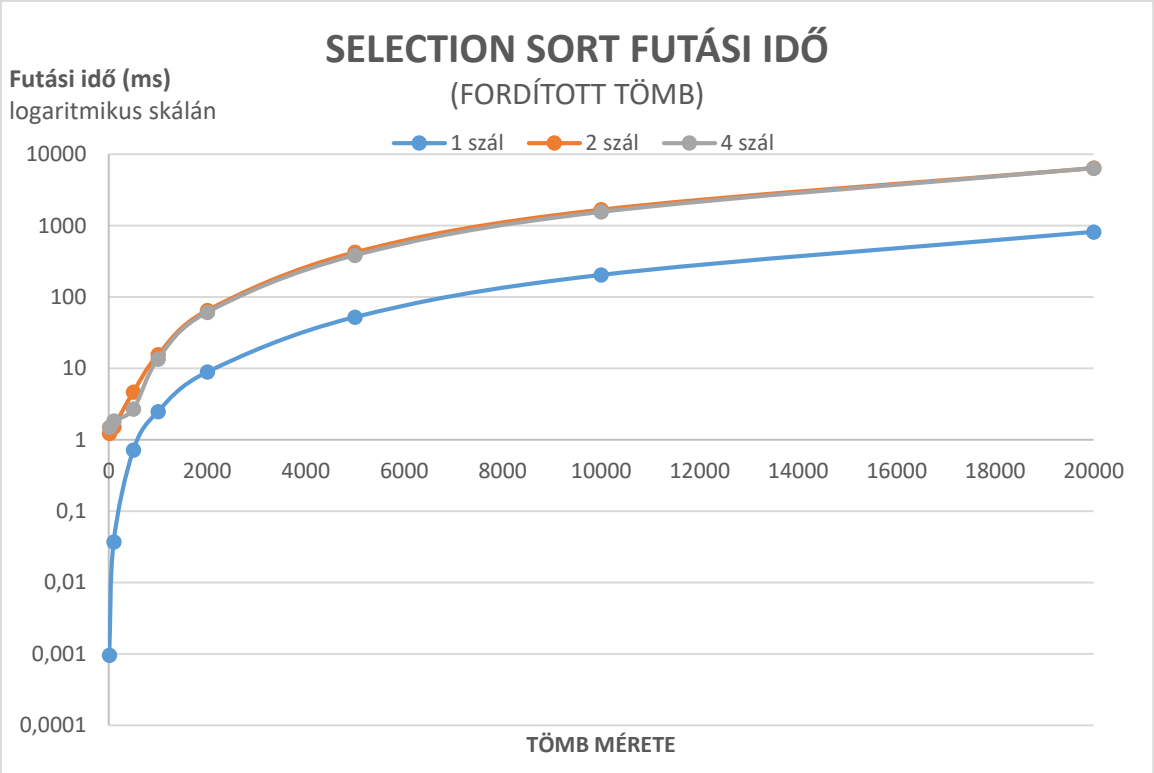
Selection Sort (fordítottan rendezett tömb esetén)



15. ábra: Selection Sort elemcserék (fordított)



16. ábra: Selection Sort összehasonlítások (fordított)



17. ábra: Selection Sort futási idők (fordított)

A 15. ábrán látható a fordítottan rendezett tömbökre alkalmazott Selection Sort algoritmus elemcseréinek száma. Érdekes módon, itt kevesebb elemcserére lesz szükség, mint egy random tömb esetében. Továbbá nem feltétlenül a soros rendezésnél lesz a legtöbb elemcsere. Változó, hogy melyiknél van a legkevesebb elemcsere, viszont a legvalószínűbb a 4 szálas futás esetén.

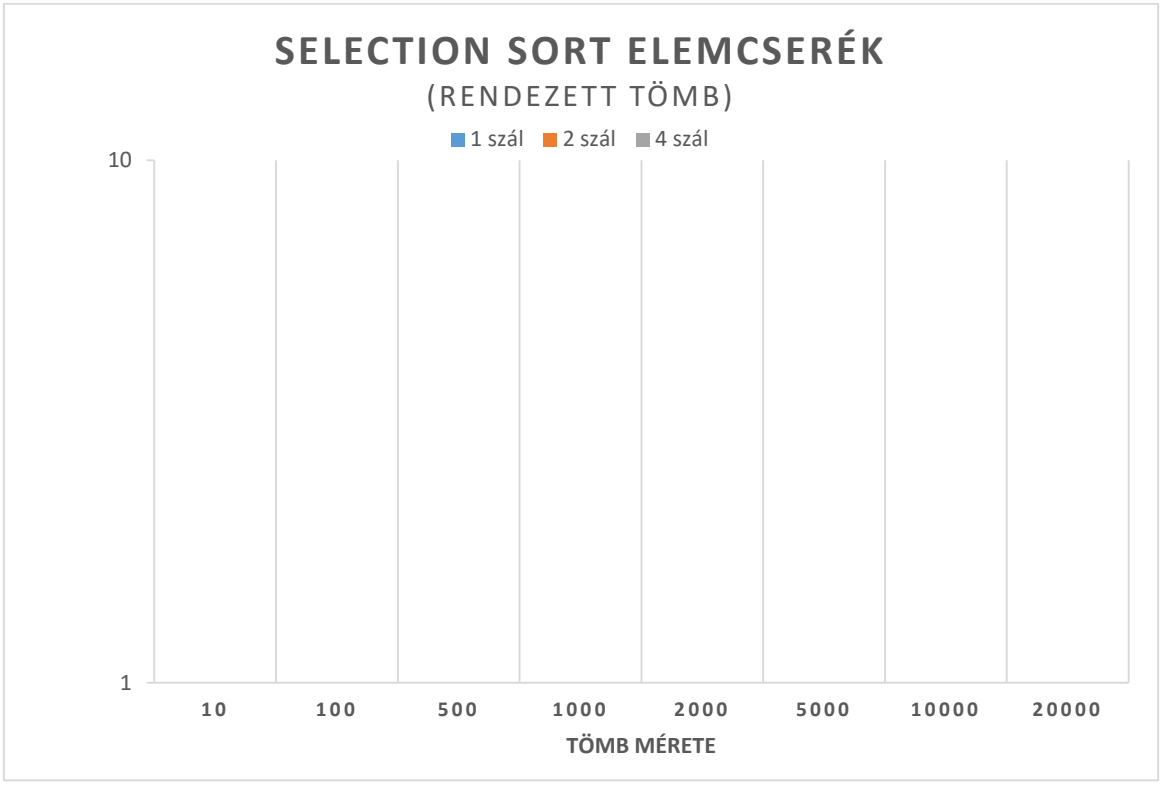
A 16. ábra az összehasonlítások számát szemlélteti erre a tömb esetre. Azonban ennél a rendezésnél nem számít, hogy milyen tipusú tömbre alkalmazzuk a rendezést, mindig ugyanannyi lesz az összehasonlítások száma, csak a tömb méretétől függ.

A 17. ábra a futási időket ábrázolja. Itt is a soros futás a leggyorsabb, de a 4 szálon futtatott rendezés csak egy picivel lesz gyorsabb a 2 szálon futtatott rendezésnél.

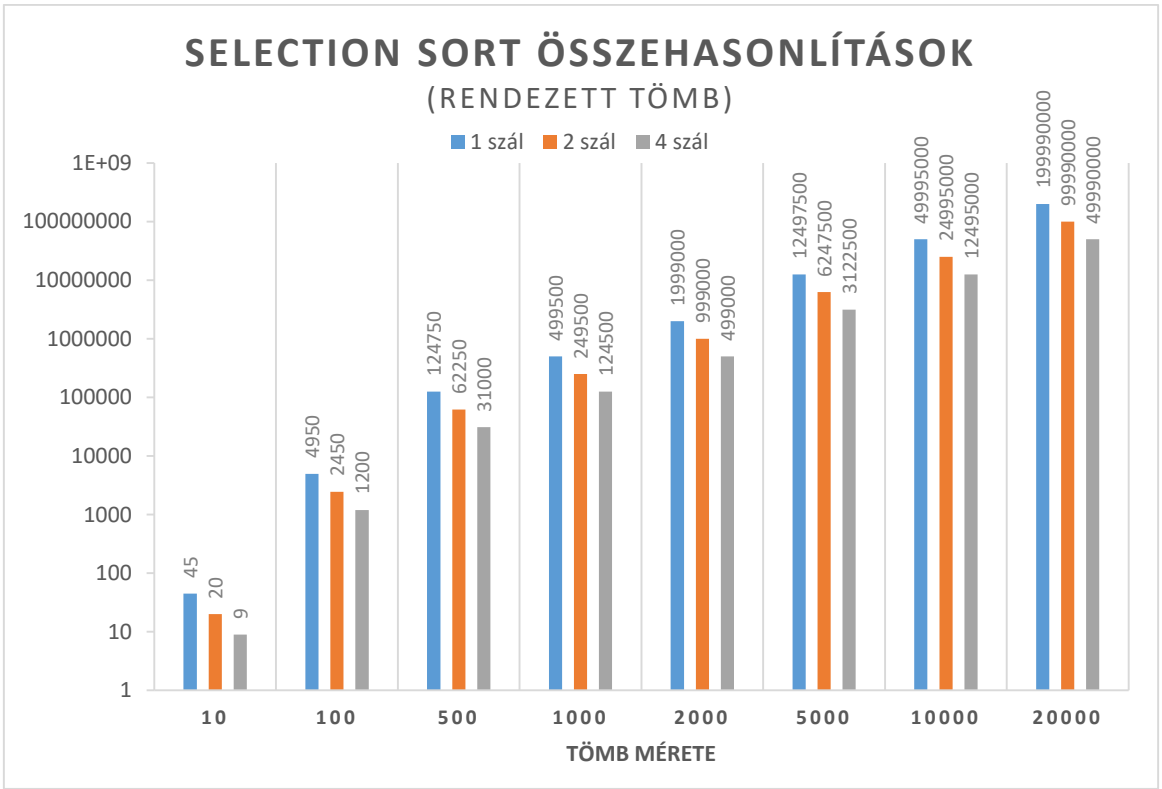
A szekvenciális futás gyorsasága a párhuzamos rendezésekhez viszonyítva:

Párhuzamos futás sebessége (lassabb) a szekvenciálishoz képest a tömb méretének függvényében									
tömb mérete		10	100	500	1000	2000	5000	10000	20000
idő	2 szál	1281,3	41,1	6,5	6,2	7,3	8,1	8,2	7,8
	4 szál	1562,5	50,0	3,7	5,4	6,8	7,3	7,6	7,8

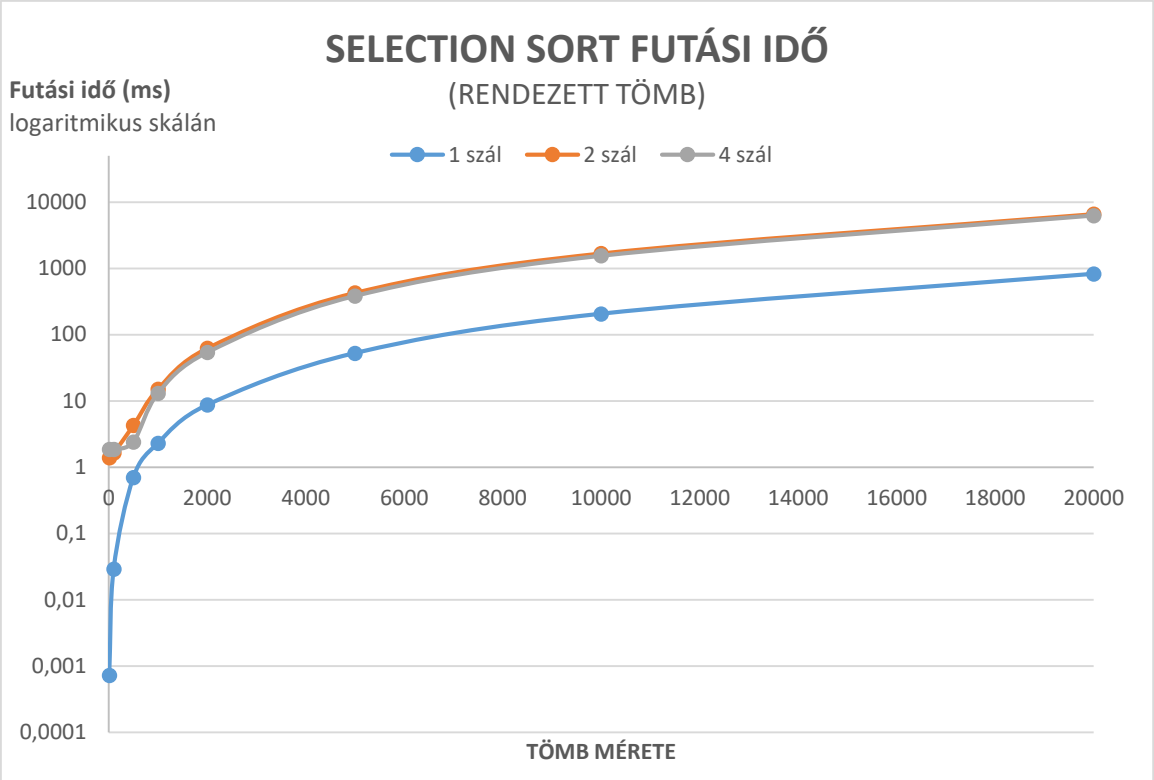
Selection Sort (rendezett tömb esetén)



18. ábra: Selection Sort elemcserék (rendezett)



19. ábra: Selection Sort összehasonlítások (rendezett)



20. ábra: Selection Sort futási idők (fordított)

A 18. ábrán azt láthatjuk, hogy ennél az algoritmusnál se lesz szükség elemcserére rendezett tömbök esetén.

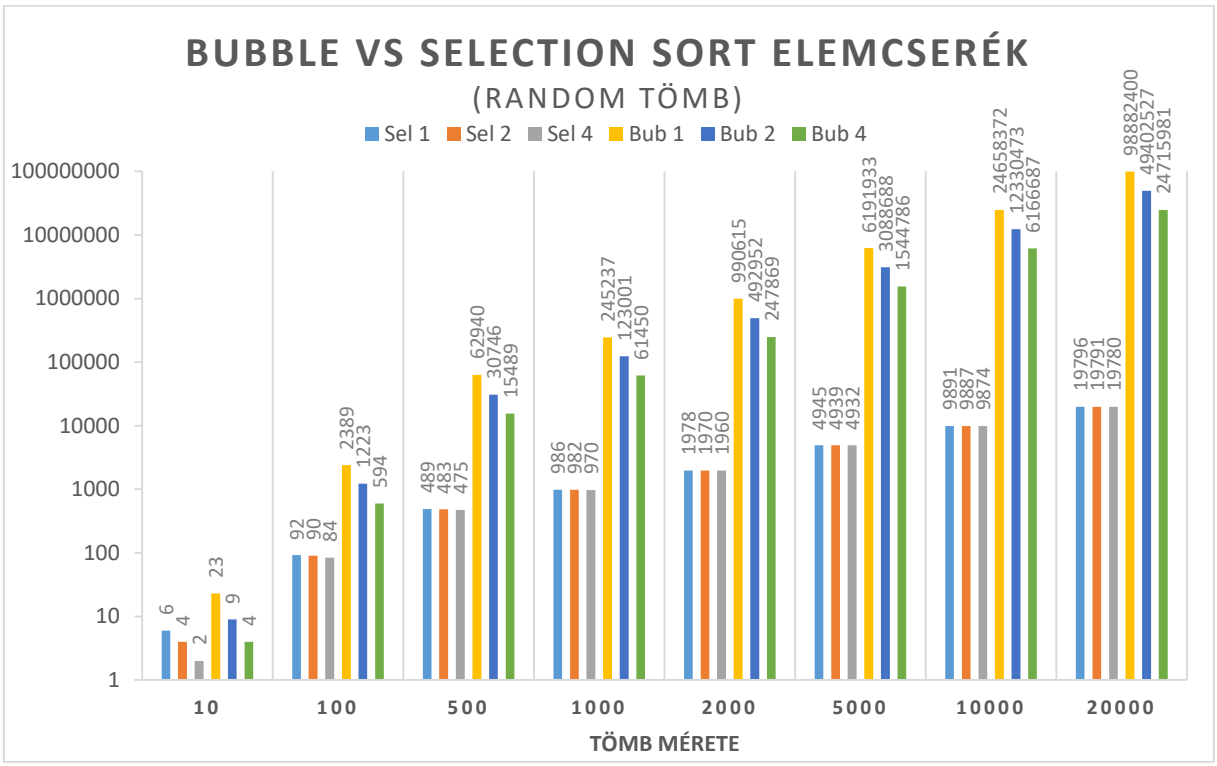
A 19. ábra az összehasonlítások számát ábrázolja rendezett tömbökre, amit már említettem, hogy tömbesettől függetlenül mindig azonos lesz az összehasonlítások száma, amit képlet segítségével is meghatározhatunk.

A 20. ábrán a futási idők láthatók. Nem nagy meglepetésre, a szekvenciális rendezés lesz a leggyorsabb. Azonban annak ellenére, hogy nincsenek elemcserék, a rendezés körülbelül ugyanannyi időt vesz igénybe, mint az összes többi tömb esetnél. A Selection Sort csupán ebben marad alul a Bubble Sortal szemben, minden másban jobb.

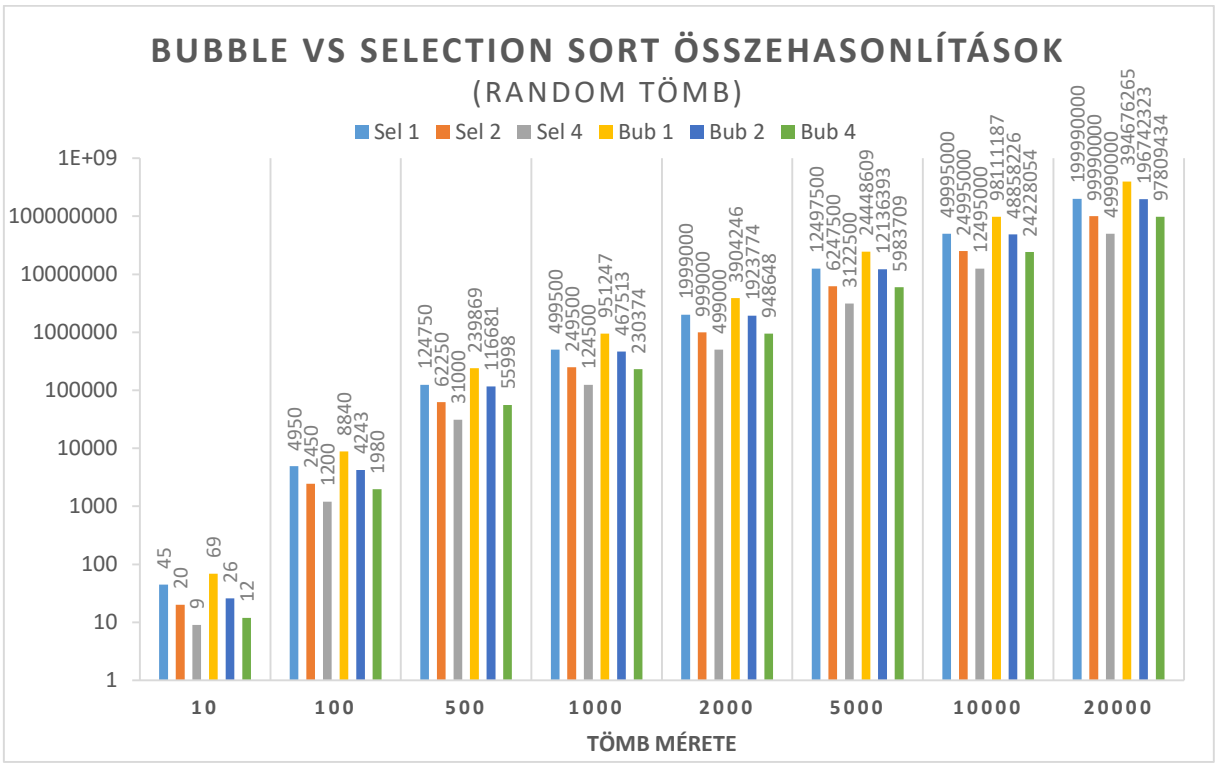
A szekvenciális futás gyorsasága a párhuzamos rendezésekhez viszonyítva:

Párhuzamos futás sebessége a szekvenciálishoz képest a tömb méretének függvényében									
tömb mérete		10	100	500	1000	2000	5000	10000	20000
idő	2 szál	1930,6	56,6	6,1	6,5	7,1	8,1	8,2	8,0
	4 szál	2583,3	64,1	3,4	5,7	6,2	7,3	7,5	7,6

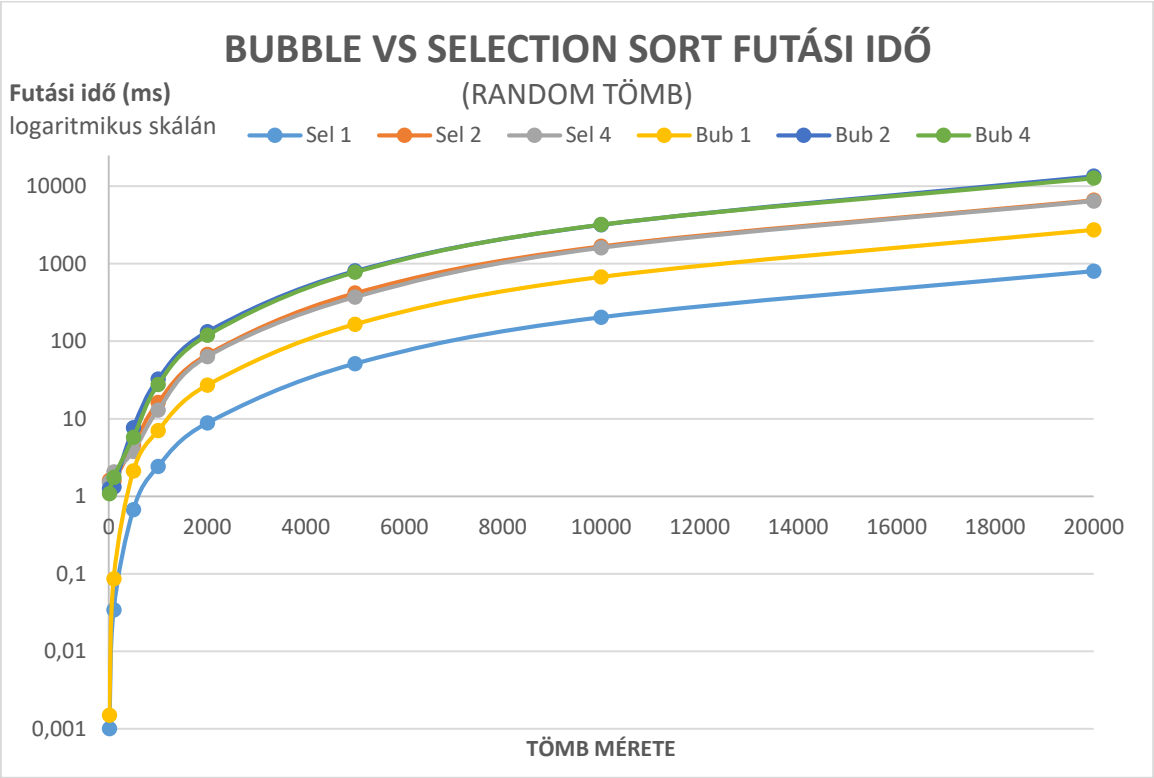
Összehasonlítás (véletlenszerű tömb esetén)



21. ábra: Algoritmusok elemcsere szerinti összehasonlításai (random)



22. ábra: Algoritmusok összehasonlítás szerinti összehasonlításai (random)



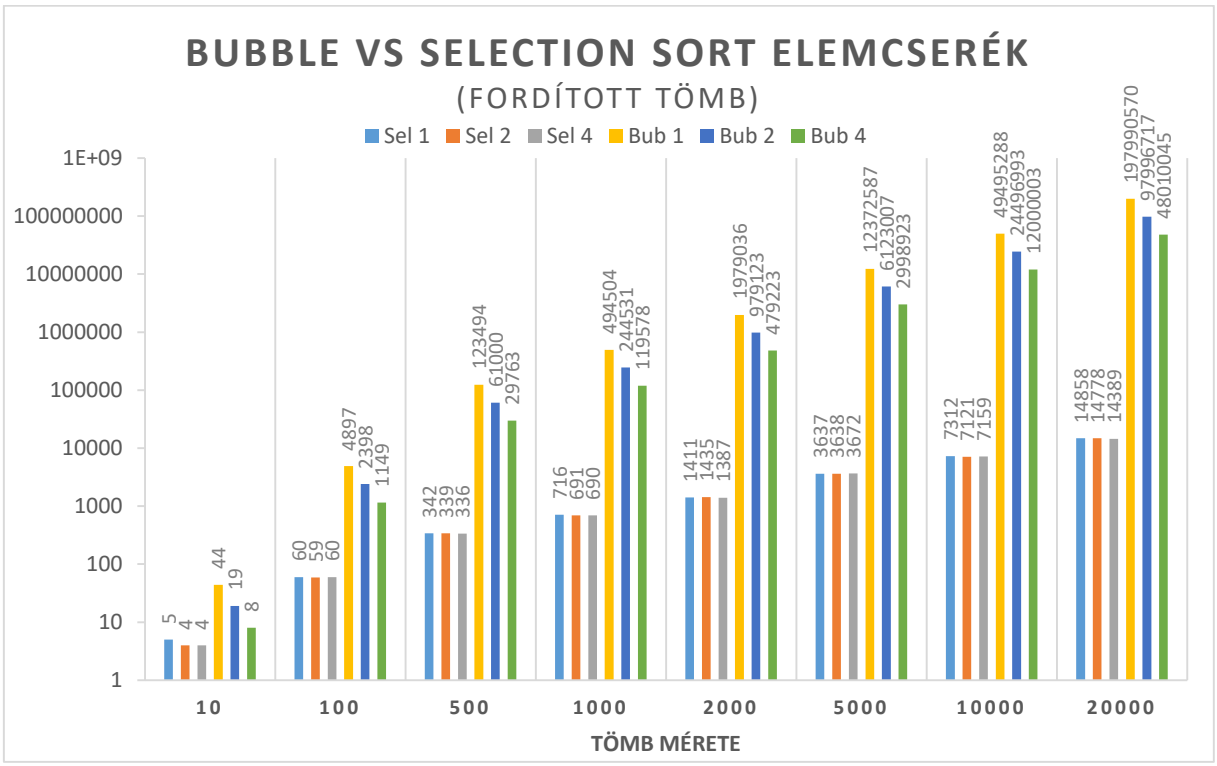
23. ábra: Algoritmusok futás idő szerinti összehasonlításai (random)

A 21. ábrán láthatjuk a Bubble és a Selection Sort összehasonlítását az elemcserék tekintetében random tömb esetre. Itt az figyelhető meg, hogy a Bubble Sortnak jóval több elemcserére van szüksége, mint a Selection Sortnak. Ez egyaránt igaz párhuzamos és szekvenciális futás esetén is.

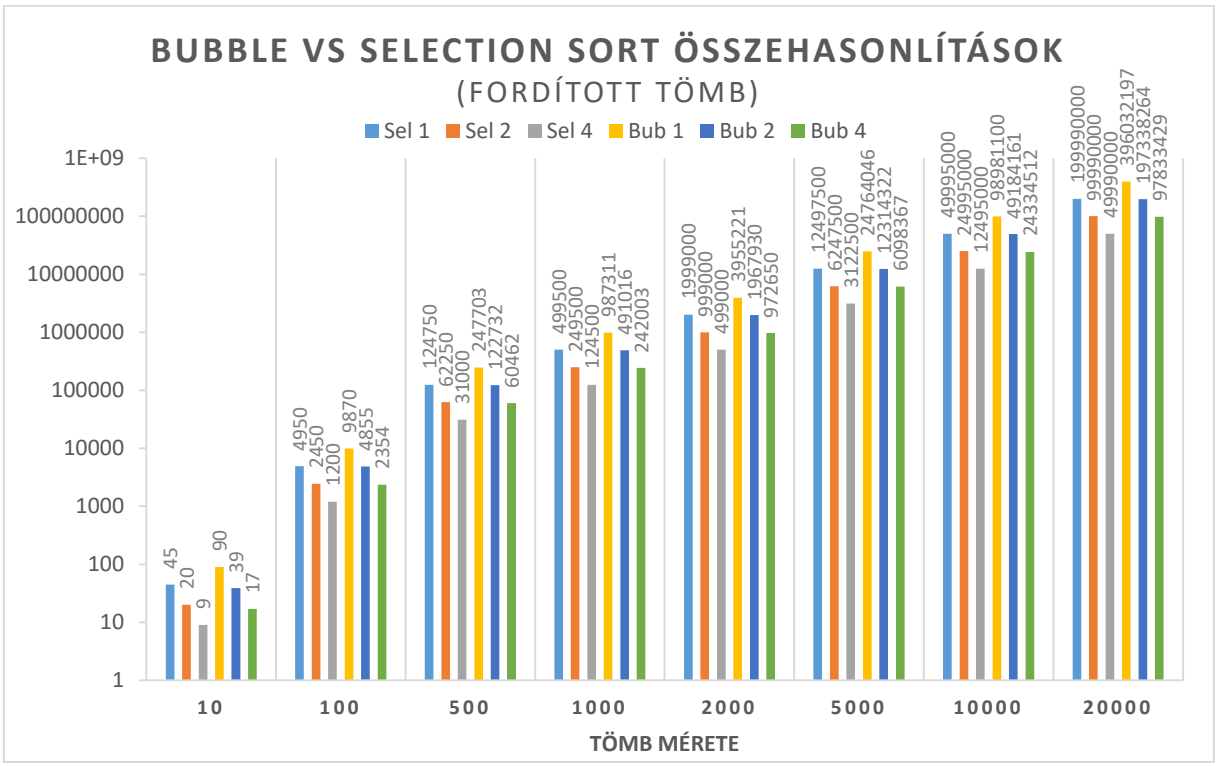
A 22. ábrán a két algoritmus szükséges összehasonlításainak számát láthatjuk. Itt is kevesebb összehasonlítás szükséges a Selection Sortnak egy tömb rendezéséhez, mint a Bubble Sortnak. Mind soros, mind párhuzamos rendezés esetén majdnem kétszer annyi összehasonlításra van szüksége a Bubble Sortnak.

A 23. ábrán a futási idők összehasonlítása látható. Ahogy az már előre látható volt az elemcserék és az összehasonlítások számából, a Selection Sort lesz a gyorsabb, párhuzamos és soros futás esetén is. Soros futás esetén egy 10000-es tömb rendezése 200 ms időt vesz igénybe, míg a Bubble Sortnál minimálisan 600 ms szükséges a rendezéshez. Párhuzamos futásnál 4 szál esetében a 10000-es tömböt 1,5 s alatt rendezi a Selectio Sort, míg a Bubble Sortnak ez is eltart legalább 3 másodpercig. Szekvenciális esetben 3x gyorsabb, míg párhuzamos esetben 2x gyorsabb a Selection a Bubble Sorttól.

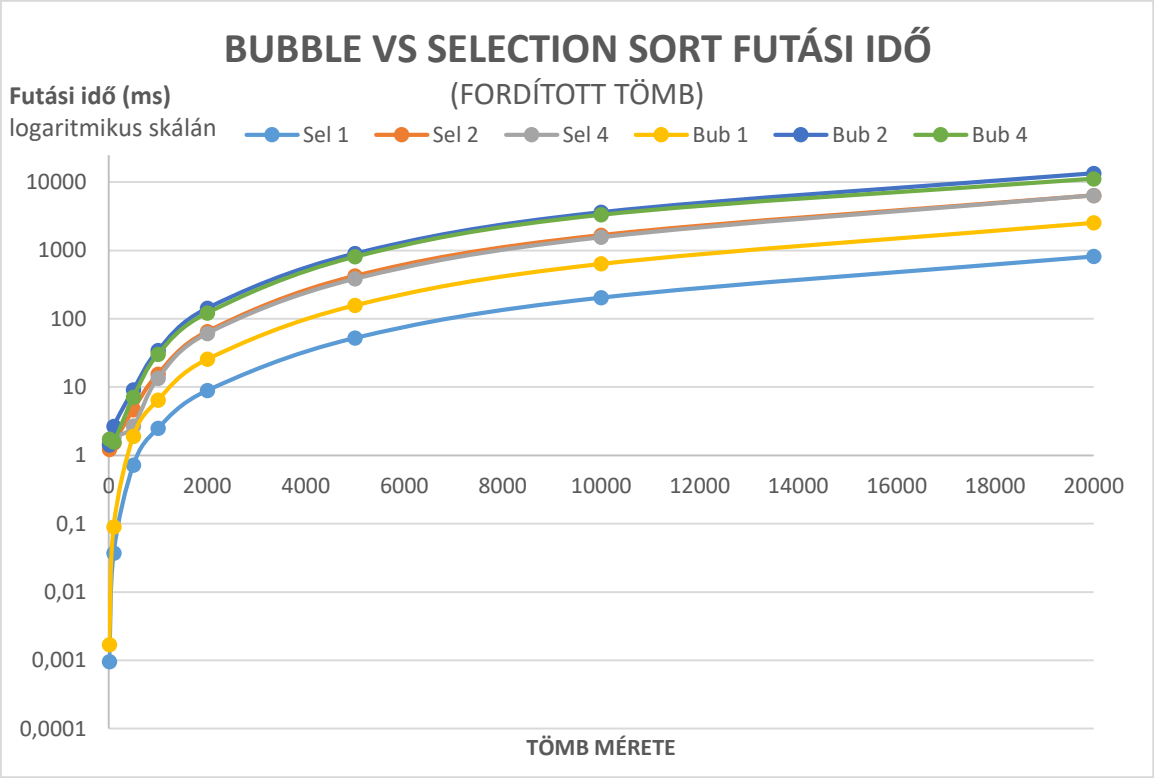
Összehasonlítás (fordítottan rendezett tömb esetén)



24. ábra: Algoritmusok elemcsere szerinti összehasonlításai (fordított)



25. ábra: Algoritmusok összehasonlítás szerinti összehasonlításai (fordított)



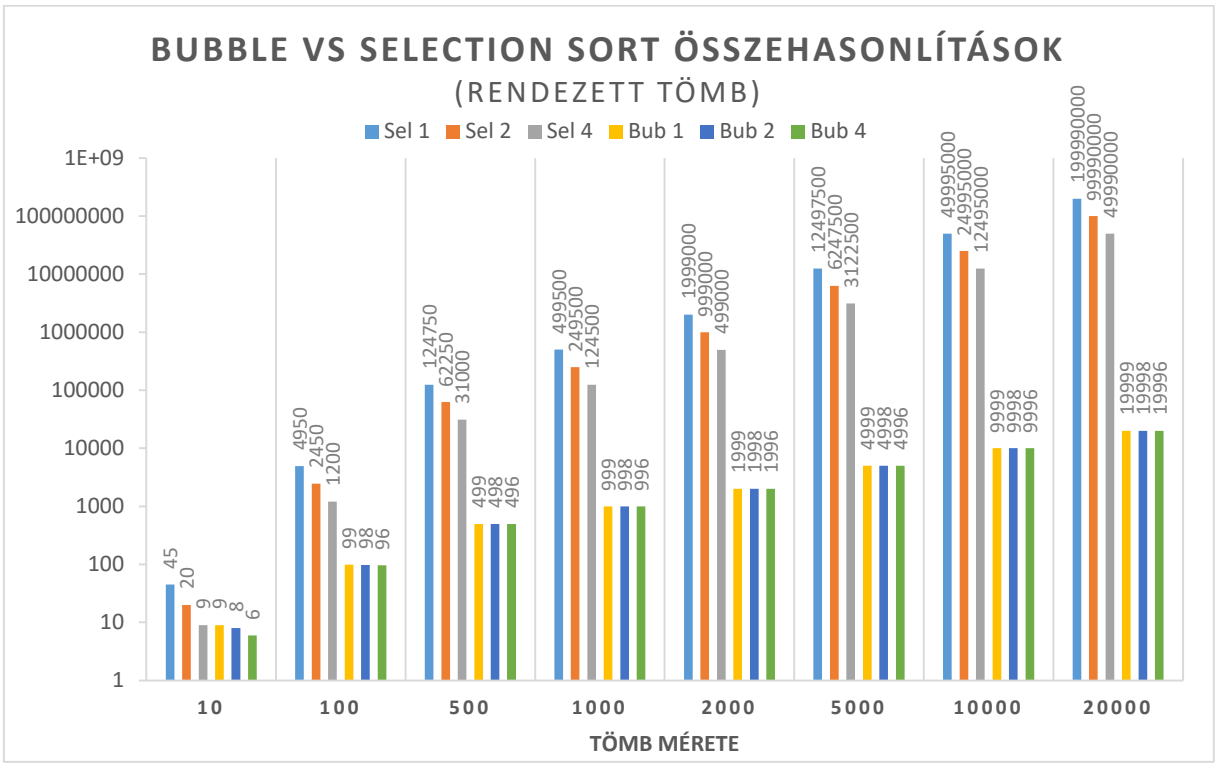
26. ábra: Algoritmusok futás idő szerinti összehasonlításai (fordított)

A 24. ábrán a Bubble és a Selection Sort algoritmus elemcseréinek számát láthatjuk fordítottan rendezett tömb esetekre. Itt is ugyan az a helyzet, mint a random tömbök esetében, hogy a Selection Sortnak sokkal kevesebb cserére van szüksége. Azonban itt a Bubble Sorthoz több csere lesz szükséges, a Selection Sorthoz viszont még kevesebb, mint random tömbök rendezésénél.

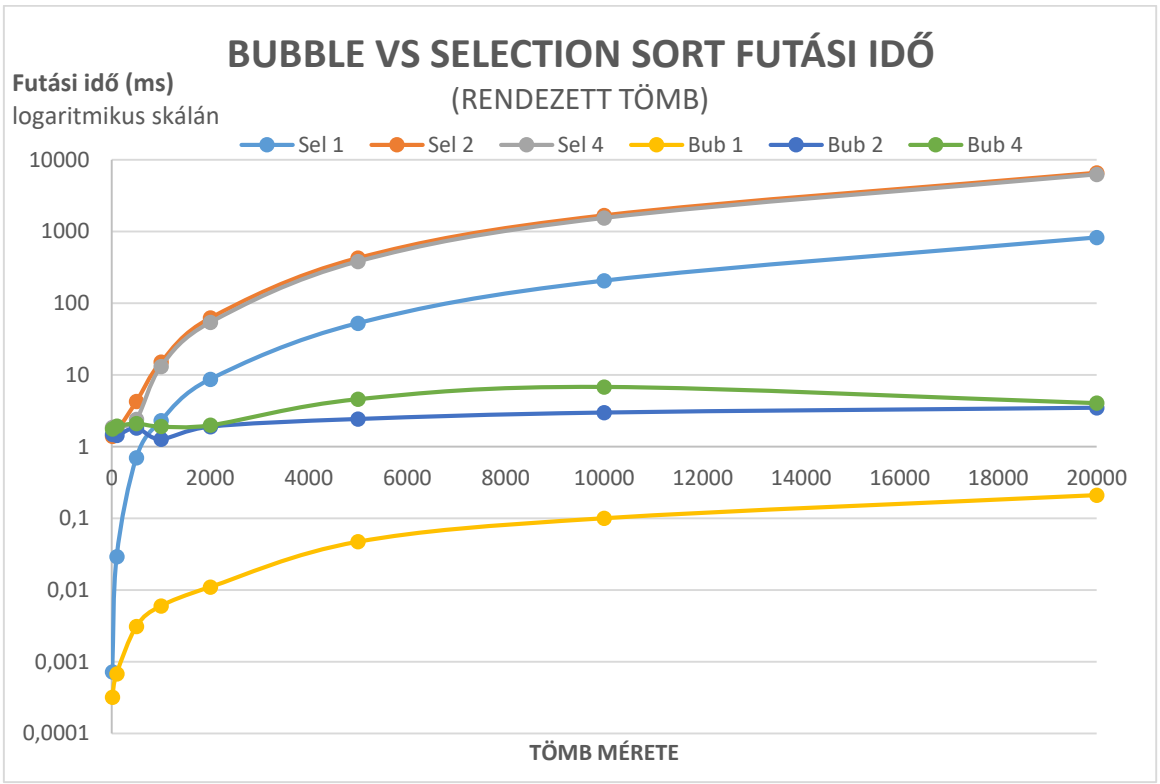
A 25. ábrán a szükséges összehasonlítások száma látható a két algoritmus összehasonlításához fordítottan rendezett tömb esetén. A Selection Sort összehasonlításainak száma nem változik, de a Bubble Sort összehasonlításainak száma itt viszont már a kétszerese lesz a Selection Sortnak.

A 26. ábrán a futási idők olvashatók le a két rendezési algoritmushoz. Ebben az esetben is a Selection bizonyul sokkal jobbnak, mind párhuzamos, mind soros rendezések esetén. Hasonlóan bizonyul jobbnak, mint a random tömbök esetében, szekvenciális futásnál itt is 3x gyorsabb, míg párhuzamosnál is 2x gyorsabbak az algoritmusok.

Összehasonlítás (rendezett tömb esetén)



27. ábra: Algoritmusok összehasonlítás szerinti összehasonlításai (rendezett)



28. ábra: Algoritmusok futás idő szerinti összehasonlításai (rendezett)

Rendezett tömb esetén nem volt értelme összehasonlítani az elemcserék számát, mivel egyik rendezésnél sincs szükség elemcserére.

Viszont ami itt sokkal lényegesebb, a 27. ábrán látható a Bubble és a Selection Sort összehasonlításainak száma rendezett tömbre. Ebben az egy esetben felülmúlja a Bubble Sort a Selection Sortot. Mivel a Selection Sort minden esetben ugyanannyi elemcserét használ, akár rendezett a tömb, akár nem. A Bubble Sort pedig rendezett tömb esetén csak egyszer megy végig a tömbön, ezért itt sokkal jobbnak fog bizonyulni a Selection Sortnál. Habár nincs sok értelme, rendezett tömböt rendezni, de majdnem rendezett tömbök esetén is hasonló lesz a helyzet.

A 28. ábrán láthatjuk a futási időket. Ahogy azt az elemcserék számából is kivehettük, sokkal gyorsabb lesz a Bubble Sort a Selection Sortnál ebben az esetben. Szekvenciális futás esetén ilyen esetekben verhetetlen ez az algoritmus, de párhuzamos futás esetén is egész gyorsnak bizonyul. 1000-nél nagyobb méretű tömbök esetén a párhuzamos Bubble Sort már gyorsabb lesz, mint a soros Selection Sort.

Összegzés:

Az utolsó tesztesetettől függetlenül a Selection Sort lesz a jobb rendezési algoritmus, mivel az előző 2 tesztesetnél az bizonyult sokkal jobbnak, továbbá rendezett vagy majdnem rendezett tömbökre ritkán alkalmazunk rendezést. Kevesebb összehasonlítás és elemcsere szükséges a rendezéshez, azáltal a futási ideje is jóval rövidebb lesz, mint a Bubble Sortnak.

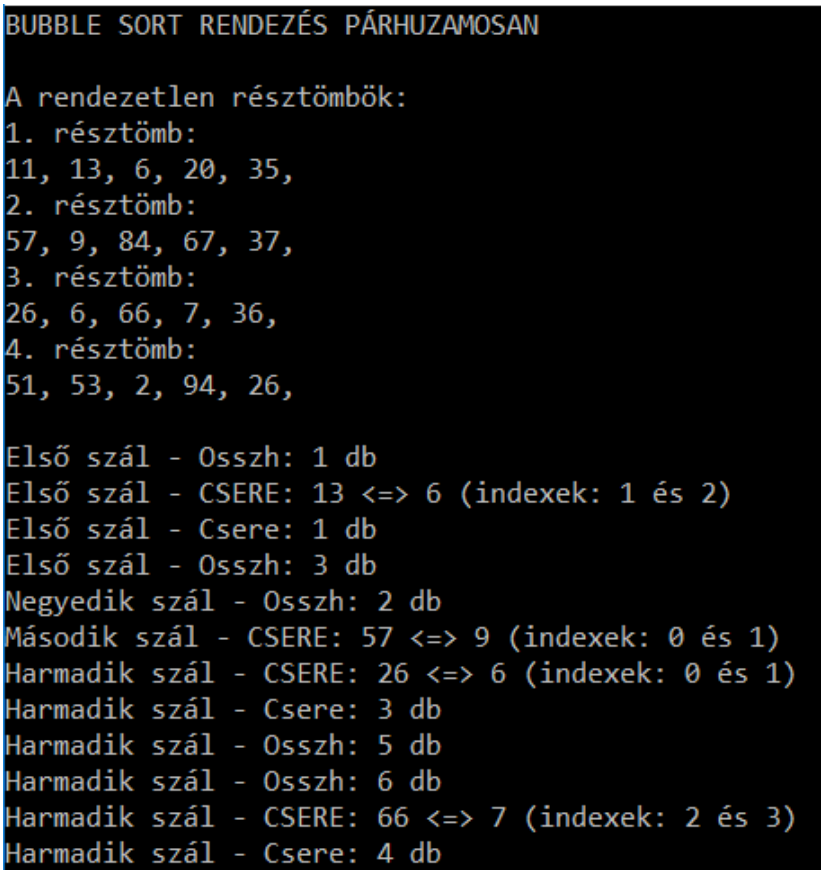
A párhuzamosítás valamivel jobban működött a Bubble Sortnál, de az sem érte el a kívánt hatást, nem tudtam megfelelően gyorsítani a szekvenciális algoritmusokhoz képest. Azonban léteznek más párhuzamosítási módszerek, amikkel minden bizonnyal remekül lehet párhuzamosítani ezeket az algoritmusokat.

Rendezési lépések ellenőrzése

Mivel a párhuzamos algoritmusok minden esetben lassabban futottak le, mint a szekvenciálisak, ezért készítettem két külön alkalmazást a rendezések lépésenkénti ellenőrzéséhez, hogy nem-e lettek hibásan implementálva az algoritmusok. Az egyik alkalmazásban a Bubble Sort algoritmust implementáltam sorosan, párhuzamosan 2 szálon és 4 szálon, a másikban a Selection Sortot ugyanígy. Ezek a konzolos alkalmazások annyiban különböztek az előző algoritmustesztelő alkalmazástól, hogy a rendezések során kiírták az egyes lépéseket a konzolra. (29. ábra)

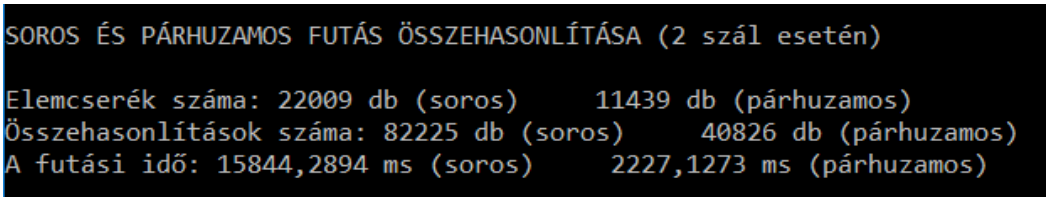
A következő lépéseket írattam ki vele a konzolra:

- a rendezendő tömböt az elején
- a rendezendő tömb résztömbökre bontását
- az összehasonlítások számának növelését
- a cserék számának növelését
- az elemek cseréjét
- minden iteráció után az aktuális tömböt
- ha befejeződött a szál futása
- a rendezett résztömböket
- a futási eredményeket (elemcsere, összehasonlítás, futási idő)
- az eredmények összehasonlítását soros és párhuzamos futás esetén

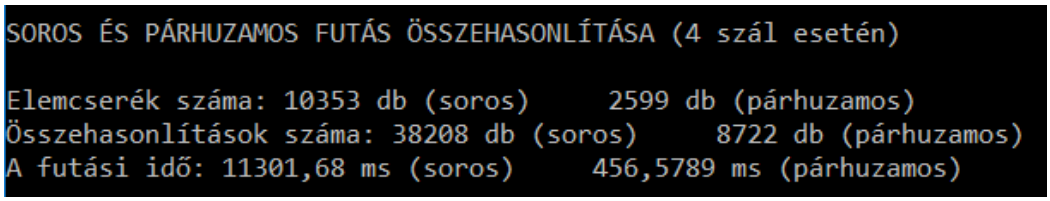


29. ábra: Rendezési lépések a konzolon

A meglepő az volt, hogy ezen az alkalmazáson rendkívül jól működött a párhuzamosítás. Már kis tömbök esetén is látható volt a teljesítményjavulás, nagy tömböknél még érezhetőbb volt. Sajnos ezekhez nem készítettem részletes teszteket, azonban az alábbi két ábra (30. és 31. ábra) segítségével éreztetni tudom, hogy ezeknél az alkalmazásoknál kifejezetten jól működtek a párhuzamosított algoritmusok. 2 szálas futás esetén 7x-es gyorsulást értünk el a szekvenciális futással szemben, 4 szál esetén pedig 24x-es volt a gyorsulás mértéke. Természetesen, ezek csak egy-egy tesztadatok, nem biztos, hogy minden esetben ilyen jól fog működni a párhuzamosítás. Azonban úgy néz ki, hogy ilyen esetekben, ahol rengeteg konzolra írás van, megéri párhuzamosítani.



30. ábra: 2 szálas futási eredmény



31. ábra: 4 szálas futási eredmény

Felhasznált irodalom

[1] WIKIPÉDIA. *Rendezés (programozás)*. 2015. [online]. Elérhető az interneten: https://hu.wikipedia.org/wiki/Rendez%C3%A9s_(programoz%C3%A1s)

[2] Házy, A. - Nagy, F. *Adatstruktúrák és algoritmusok*. 2009. [online]. Elérhető az interneten:http://www.tankonyvtar.hu/hu/tartalom/tamop425/0046_adatstrukturak_es_algoritmusok/ch04s02.html

[3] Programozás Wiki. *Kiválasztásos rendezés*. 2013. [online]. Elérhető az interneten: http://wiki.prog.hu/wiki/Kiv%C3%A1laszt%C3%A1sos_rendez%C3%A9s_(algoritmus)

[4] Hernyák, Z. *Communication Foundation*. 2011. [online]. Elérhető az interneten: http://aries.ektf.hu/~hz/pdf-tamop/pdf-03/html/ch06.html

[5] PROG.HU. *Saját tömb rendező algoritmus*. 2011. [online]. Elérhető az interneten: https://prog.hu/tarsalgo/134163/sajat-tomb-rendezo-algoritmus

Melléklet

A tesztelési eredményeket tartalmazó excel fájl:

- Tesztelési eredmények.xlsx

A féléves feladathoz készített alkalmazások:

- Féléves fel – Rendezési alg
  - Rendezesi_alg
  - Bubble
  - Selection