Az algoritmusok hatékonyságának kérdése a programozás oktatásban

Pappné Palovics Éva

palovics.eva@avkf.hu
AVKF

Absztrakt. Sok esetben egy probléma megoldására nem csak egy algoritmus létezik, és a megoldások csak az algoritmus hatékonyságában különböznek egymástól. Jó megoldás-e a kevésbé hatékony megoldás? Érdemes-e informatika órán foglalkozunk azzal, hogy hatékonyság szempontjából is elemezzük az algoritmusokat? Hogyan tudjuk bemutatni a különböző korosztályok számára az algoritmusok hatékonyságát? Milyen feladatok, példák segíthetnek a diákoknak a hatékonyság fogalmának megértését, elsajátítását? Előadásomban ezeket a kérdéseket járom körbe.

„Az informatikaoktatás célja a praktikus alkalmazói tudás, a készség- és képességfejlesztés mellett a logikus, algoritmikus gondolkodás és a problémamegoldás tanítása. A műveltségi terület fontos feladata, hogy felkészítse a tanulókat az informatikai eszközök, információforrások önálló és csoportos használatára." [1/10813. o.]

Az algoritmusok hatékonysága az a témakör, melynek segítségével az algoritmizálás képességét továbbfejleszthetjük a diákokban, a problémamegoldás tanításának nagyszerű eszköze lehet. Az algoritmusok elemzésének témaköre a fenti célkitűzésnek megfelelően meg is jelenik a NAT-ban a 9-12. évfolyamon, de a kerettanterv már nem foglalkozik a témával. Az előadás során azt szeretném bemutatni, hogy miért fontos, hogy ez a témakör megjelenjen a közoktatásban, és hogyan vezethetjük be a hatékonysággal kapcsolatos fogalmakat.

Miért fontos foglalkozni az algoritmusok hatékonyságával?

A programozók számára nem kérdés, hogy egy algoritmus hatékonysága kulcsfontosságú a tervezési folyamatban. A különböző algoritmikus problémákra hatékony algoritmusok tervezése az informatika tudományának egyik legfontosabb kutatási területe. A jól megtervezett algoritmusok kulcsfontosságúak az üzleti szférában, hiszen egy alkalmazás akkor versenyképes, ha minél hatékonyabban oldja meg a feladatát.

Egyértelmű, hogy az algoritmusok elemzése az informatikai pályára készülő tanulók számára hasznos és fontos ismeretkör. A nem informatikai pályára készülő tanulók számára is számos előnnyel jár a témakör megjelenése a tanórákon, hiszen az algoritmikus gondolkozás, problémamegoldás fejlesztésének nagyon jó eszköze lehet, ezek fejlesztése pedig minden tanuló számára egyformán fontos. Az algoritmusok elemzése segít megérteni magát az algoritmusokat, ötleteket adhat azok javítására. Sok esetben az elemzési folyamat során az algoritmus egyszerűbbé, elegánsabbá válik.

A tanulók viszont sokszor nem látják a hatékony algoritmus tervezésének fontosságát. Ennek egyik oka lehet, hogy a modern számítógépek sebessége, memóriája, tárhely kapacitása folyamatosan nő, így számukra nem érzékelhető az általuk tervezett egyszerű algoritmusoknál a hatékonyság problémája. A másik oka, hogy ha az órákon téma is az algoritmusok elemzése, a számonkérésben szinte soha nem jelenik meg a programok hatékonysága. Az érettségi programozási feladataiban a megoldás jó és maximális pontszámot ér, bármilyen algoritmussal is oldja meg

a diák a feladatot, a hatékonyság alapján nem tesznek különbséget, így nincs motiváció arra, hogy törekedjenek a feladat optimális megoldására.

Algoritmusok elemzése a tanórákon

A cél, hogy a tanulók megértsék a hatékonyság fogalmát, ennek fontosságát a programtervezésben, képesek legyenek hatékonyság szempontjából elemezni az algoritmusokat, és a rendelkezésükre álló eszközökkel hatékony algoritmusokat készítsenek.

A hatékonyság fogalma

Az algoritmusok elemzésének lényege, hogy megvizsgáljuk, milyen erőforrásokra lesz szüksége. Elemzésük segítségével tudjuk eldönteni, hogy ugyanazt a feladatot elvégző algoritmusok közül melyik a hatékonyabb, azaz melyiknek van kevesebb erőforrásra szüksége.

A különböző erőforrások szempontjából vizsgálhatjuk az algoritmusok hatékonyságát, például a számítási idő, a szükséges memória mérete, a kommunikációs sávszélesség, energia felhasználás, stb.

A hatékonyság témáját a fiatalabb korosztály számára különösebb fogalom-meghatározások nélkül is taníthatjuk, a későbbiekben viszont érdemes bevezetni a témakör alapfogalmait, a futási idő számítási modellen alapuló meghatározását, a legrosszabb és átlagos eset fogalmát, a növekedési rend és az algoritmusok komplexitásának definícióját.

Példák a hatékonyság fogalmának bevezetésére a kisiskolás korosztály számára

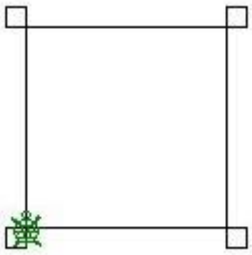
Hatékonyság a Logo programozásban

A programozás tanításának kezdetén is érdemes foglalkozni a hatékonyság kérdésével, de magát a fogalmat nem szükséges bevezetni, hiszen itt még a programozás is csak játékos formában jelenik meg. A hatékonysággal is játékos formában foglalkozhatunk.

A Logo nyelv sajátosságánál fogva definiálhatjuk úgy a programok hatékonyságát, hogy a vizsgált erőforrás a teknőc „energiája”, azaz annál hatékonyabb a program, minél kevesebb energiába kerül a teknőc számára ugyanannak a feladatnak a megoldása, minél rövidebb utat kell bejárnia ugyanannak a feladatnak a végrehajtásához. Érdemes egyszerű feladatok segítségével vizsgálni a kérdést.

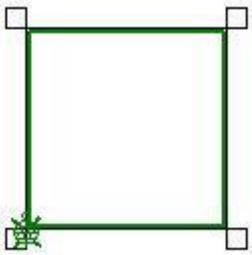
Ez a hatékonyság-fogalom minden további magyarázat nélkül érthető a kisebb gyerekek számára. Az algoritmusok elemzését megkönnyíti, hogy ha lépésekre bontva hajtjuk végre a feladatot. Így pontosan láthatjuk, hogyan hajtja végre az utasításokat a teknőc, azaz látjuk a megtett utat.

Nézzük meg a következő ábra rajzolásának két megoldását:



1. ábra Logo példa a hatékonyság bevezetésére.

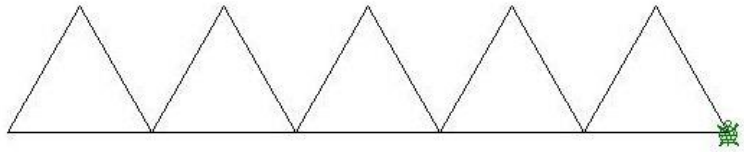
```
Négyzet rajzolása
      tanuld négyzet :oldal
      ism 4 [e :oldal j 90]
Első program:
      négyzet 100
      ism 4 [tf e 100 b 90 tl négyzet 10 b 180]
Második program
      ism 4 [e 100 b 90 négyzet 10 b 180]
```



2. ábra: Többször bejárt út az első programban.

Míg az első program esetén a minden négyzet megrajzolásához a négyzetrajzoló eljárást használjuk, így a nagy négyzet megrajzolása után mindig el kell mennie a teknőcnek a kis négyzetek kezdőpontjáig. Azaz a nagy négyzet oldalait kétszer járta be (A 2. ábrán zölddel jelölt a többször bejárt útvonal.). A második program esetén csak a kis négyzeteket rajzoljuk az eljárással, így a középső alakzat oldalait akkor rajzolja a teknőc, amikor az egyik kis négyzettől a másikhoz megy. Nyilván így a nagy négyzet oldalait csak egyszer járja be, azaz sokkal rövidebb utat jár be a teknőc, így a második program a hatékonyabb megoldás. Ez a kódból is nagyon egyszerűen észrevehető.

Egy újabb példa a rövidebb út megtalálásához:



3. ábra: Újabb Logo példa.

```
tanuld sorminta1 :hanyszor
      ism :hanyszor [j 30 ism 3 [e 100 j 120] j 60 e 100
b 90]
vége
tanuld sorminta2 :hanyszor
      ism :hanyszor [j 30 e 100 j 120 e 100 b 150]
      b 90 e 100*:hanyszor
vége
```

Látványosabbá tehetjük a példát, ha mindig beiktatunk egy azonos hosszúságú pihenőt, miután a teknőc előre ment 100-at, így könnyebben megfigyelhetjük, vagy akár időméréssel versenyeztethetjük is a teknősöket, melyik tudja gyorsabban végrehajtani a feladatát. Jósoljuk meg, melyik algoritmus lesz a gyorsabb! Így a feladat szórakoztató játékká válik a tanulók számára, a továbbiakban is motiváltabbak lesznek a feladat megoldási algoritmusának átgondolására, javítására.

```
tanuld sorminta1 :hanyszor
      ism :hanyszor [j 30 ism 3 [e 100 várj 1000 j 120] j
60 e 100 várj 1000 b 90]
vége
tanuld sorminta2 :hanyszor
      ism :hanyszor [j 30 e 100 várj 1000 j 120 e 100
várj 1000 b 150]
b 90 ism :hanyszor [e 100 várj 1000]
vége
```

Hasonló példák és feladatok segítségével már a programozással való foglalkozás kezdetén, játékos formában megtaníthatjuk a gyerekeknek, hogy ugyanarra a feladatra több megoldás is lehet, melyek között van jobb, gyorsabb, hatékonyabb megoldás.

A Logo vizuális környezete nagy segítség az algoritmusok hatékonyságának látványos bemutatására. Több olyan programnyelv van, ami a fiatalabb korosztályokat célozza meg, vizuális programozási környezet segítségével ismerteti meg a programozás alapfogalmaival a diákokat. Ilyen például a Scratch és az Alice is.

Jó példa lehet a vizuális programozási környezetekben a különböző rendezési algoritmusok bemutatása. Hétköznapi példa lehet a tornasor felállítása, azaz a diákok sorba rendezése magasság szerint. A lenti ábrán az Alice programozási környezetben készített példa kezdőképe látható. Két különböző algoritmust leprogramozva a példára, bemutathatjuk, hogy a rendezések futási ideje különböző. Például az adott tornasor rendezése egyszerű cserés rendezés esetén hét cseré-

vel hajtható végre, míg maximum kiválasztással történő rendezés esetén csak három csere szükséges. Látva a folyamatot, könnyen eldönthetik a tanulók, melyik a gyorsabb, jobb megoldás a feladat végrehajtására.

4. ábra: Példa a rendezés vizualizációjára Alice programozási környezetben.

Hatékonyság fogalma az algoritmizálás bevezetésével párhuzamosan

A Nemzeti Alaptanterv szerint az 5-8. osztályban megjelenik informatika órán az algoritmus fogalma, az algoritmuselemek, az algoritmus leíró eszközök használata. Amint algoritmust írunk egy feladat megoldására, foglalkozhatunk azok hatékonyságával. Érdemes korán bevezetni a hatékonyság fogalmát, mivel a téma motiváció lehet arra, hogy a diákok újabb és újabb algoritmusokat vizsgáljanak meg egy feladat megoldása során, hogy, elemezzék és folyamatosan újragondolják a megoldásokat.

A fogalom korai bevezetése azonban problémákat is felvet. A programozás tanításának kezdetén nagyon egyszerű feladatokat oldunk meg, így a hatékonyság fontossága nem egyértelmű a diákok számára. Sokszor azzal a téves elképzeléssel találkozunk, hogy a mai számítógépek olyan gyorsak, hogy az, hogy hány lépésben oldja meg a feladatot egy algoritmus, lényegtelen. Arra, hogy mennyi memória szükséges egy-egy feladat megoldásához, ritkán gondolnak.

Meg kell teremtenünk a motivációt a hatékony algoritmusok készítésére. Ezt úgy tehetjük meg a legegyszerűbben, ha megvilágítjuk, milyen problémákat oldanak meg a professzionális programozók, ahol a futási idő és a memóriahasználat kritikus fontosságú. Néhány példa erre:

- A New York-i tőzsde részvényeinek nagy része elektronikus rendszeren keresztül cserélik. Egy nap kb. 1 TB adatot generál a forgalom. A futball világbajnokság döntőjének napján a facebookon összesen 1 milliárd poszt, hozzászólás, like keletkezett. Ezek mind valós idejű adatok, melyeket algoritmusok segítségével dolgoznak fel. Az elektronikus kereskedés különösen időérzékeny folyamat, tizedmásodperccel gyorsabb adatáramlás hatalmas üzleti előnyt jelenthet.
- Egyes tudományterületek kutatásai nagy adatmennyiségekkel dolgoznak. A meteorológia, a csillagászat, a genetika mind-mind hatalmas mennyiségű adatot használ fel a kutatásaiban. Az időjárás-előrejelzés a légkör matematikai modelljén alapul. Ez a matematikai modell ha-

talmas adatmennyiséget kíván ahhoz, hogy előre jelezhessék, hogyan változik a légkör állapota. Minél pontosabb eredményt szeretnének kapni, annál nagyobb adatmennyiséggel kell dolgozniuk. A genetika területén is hatalmas adatmennyiségekkel dolgoznak. Az emberi genom a petesejt vagy hímivarsejt teljes genetikai tartalma, ami 23 kromoszómapárból és körülbelül 3 milliárd DNS bázispárból áll. Azt, hogy milyen betegségekre vagyunk hajlamosak, vagy egyszerűen azt, hogy hogy nézünk ki, több ezer gén befolyásolja. Ahhoz, hogy a tudósok egyes genetikai megbetegedéseket okozó géneket feltárják, hatalmas számítási kapacitásra van szükségük. Az emberi genetikai állomány első szekvenálása 13 évet vett igénybe, de ma is napokig tartó feladat.

- Az eddigi példák mutatják, hogy a modern tudomány algoritmusaiban az idő és a memóriahasználat kritikus kérdés. Az, hogy ez akár a legegyszerűbb programozási feladatokban is fontos lehet, a hétköznapi életből vett példával tudjuk legjobban szemléltetni. Ilyen például a weboldalak területe. Számos kutatás készült arról, hogy mi befolyásolja a felhasználókat egy weboldal, alkalmazás használatában. Ezekből egyértelműen kiderül, hogy egy weboldal sikerességében is nagyon fontos szerepet játszik az idő. Ha egy weboldal nem tölt be 3 másodpercen belül, a felhasználók 40%-továbblép róla. Egy mobilalkalmazást törölnek a felhasználók, ha 6 másodperc alatt nem tölt be. A felhasználói élmény, azaz egy program sikeressége nagyban múlik az algoritmusok számítási idején.

A számítási idő mérése

Ha diákok megértették, miért fontos a számítási idő az algoritmusoknál, felmerül a kérdés, hogy hogyan mérhetjük ezt? Hogyan hasonlíthatunk össze két algoritmust számítási idő szempontjából? Az algoritmusunk hatékonysága nem a diákok számára természetes időtől függ, hiszen nem mérhetjük le stopperrel, hogy mennyi idő alatt végzi el a feladatot a számítógépünk. Olyan módszerre van szükségünk, mely független a programozási nyelvtől, az operációs rendszertől, a számítógép-architektúrától, a bemenő adatoktól, stb., ami tisztán az algoritmus hatékonyságát vizsgálja. Egy olyan számítási modellre van szükség, melyen programként valósítjuk meg az algoritmusunkat. Ez a modell legyen a Neumann-elvű gép, hiszen a legtöbb számítógép ezen a modellen alapul. Az algoritmus futási ideje legyen a bemenetre végrehajtott lépések, azaz alapműveletek száma.

Elemezzünk egy egyszerű algoritmust! Az algoritmus feladata, hogy összegezze sorozat n elemét. Az ábrán láthatjuk, hogy az algoritmus egyes részei hány lépést jelentenek a végrehajtás során. Az eredmény: n+2 lépés szükséges az algoritmus lefuttatásához. Azaz az algoritmus futási ideje attól függ, milyen hosszú a sorozatunk, azaz a bemenettől.

```
Eljárás Összegzés                Lépésszám
összeg:=0                        1
Ciklus i:=1-től n-ig
    összeg:=összeg+A[i]          n
Ciklus vége
Ki: összeg                       1
Eljárás vége
```

Egyszerű példák a számítási idő csökkentésére

Ha a diákok megismerték az algoritmusok futási idejének fogalmát, a gyakorlatban is felhasználhatjuk ezt. Néhány egyszerű feladat segítségével példát hozhatunk az algoritmusok lépésszámának csökkentésére.

Egyik ilyen feladat lehet egy szám osztóinak megadása. Egyszerű, kevés matematikai ismerettel hatékonyabbá tehető az algoritmus.

```
Eljárás Osztók(n)
Ciklus i:=1-től n-ig
  Ha n mod i=0 akkor kiír(i)
  Elágazás vége
Ciklus vége
Eljárás vége
```

n lépés

```
Eljárás Osztók(n)
Ciklus i:=1-től n div 2-ig
  Ha n mod i=0 akkor kiír(i)
  Elágazás vége
Ciklus vége
Eljárás vége
```

[n/2] lépés

```
Eljárás Osztók(n)
Ciklus i:=1-től [sqrt(n)]-ig
  Ha n mod i=0 akkor kiír(i)
  Elágazás vége
Ciklus vége
Eljárás vége
```

$[\sqrt{2}]$ lépés

A másik ilyen feladat annak megállapítása, prím-e egy szám. A feladatnál érdemes lehet megemlíteni, hogy egy szám prím voltának megállapítása fontos kérdés a kódolási algoritmusokban.

```
Eljárás Prím(n)
Prím:=igaz
Ciklus i:=2-től n-1-ig
    Ha n mod i = 0 akkor Prím:=hamis
    Elágazás vége
Ciklus vége
```

n-2 lépés

```
Eljárás Prím(n)
Prím:=igaz
i:=2
Ciklus amíg i<=n-1 és prím
     Ha n mod i = 0 akkor Prím:=hamis
     Elágazás vége
Ciklus vége
Eljárás vége
```

A ciklus legrosszabb esetben n-2-szer fut le. A ciklusfeltételben n-2-szer vizsgáljuk a prím változó értékét, a ciklusmagban pedig n-2-szer az n mod i = 0 feltételt. Azaz **2*(n-2)+2** a lépésszám legrosszabb esetben.
De ha pl. páros a szám, 3 lépésben megállapítható, hogy nem prím.

```
Eljárás Prím(n)
i:=2
Ciklus amíg i<=n-1 és n mod i<>0
     i:=i+1
Ciklus vége
Prím:=i>n-1
Eljárás vége
```

Tovább finomítva a feladatot lecsökkenthetjük a ciklusban szükséges vizsgálatok számát, így **n** lesz a lépésszám legrosszabb esetben.

```
Eljárás Prím(n)
i:=2
Ciklus amíg i<=[sqrt(n)] és n mod i<>0
     i:=i+1
Ciklus vége
Prím:=i>n-1
Eljárás vége
```

Csökkenthető a lépésszám az előző példából ismert módon, ha a ciklus csak **n div 2**-ig vagy **[sqrt(n)]**-ig fut.

A feladat jó példa az eldöntés tételére. A hatékonyság kérdésének bevezetésével jobban megérthetik az eldöntés algoritmusának felépítését, ennek segítségével jobban érthetővé válik, hogy miért kerül be a ciklusfeltételbe a tulajdonság vizsgálata, ami tapasztalataim szerint a programozás tanulásának kezdetén nehézséget szokott okozni a diákoknak.

A feladat jó példa arra is, hogy az algoritmus futási ideje függ nem csak a bemenet nagyságától (n), hanem a bemenet egyéb tulajdonságaitól is. Azaz van egy legrosszabb eset, ebben az esetben az, ha a szám nem prím. Pl. a második és harmadik eljárásban, ha a szám nem prím. Ekkor n-2-szer fut le a ciklus, legjobb esetben pedig 3-szor.

A legrosszabb eset

Ha felmerült, hogy bizonyos bemenetekre gyorsabb, bizonyos bemenetekre lassabb az algoritmus, célszerű megállapodnia diákokkal, hogy milyen esetet vizsgáljunk. A hatékonyság tanítása során általában a legrosszabb eset vizsgálata a célravezető.

Az biztos, hogy ugyanolyan méretű bemenet esetén az algoritmus sosem fog tovább tartani, mint a leghosszabb futási idő. Sok algoritmus esetén pedig a legrosszabb eset sokszor fordul elő (pl. a szám prím vagy keresési feladatnál nincs a keresett elem a tömbben). Az átlagos futási idő

pedig nem egyértelmű a diákok számára, kiszámításához valószínűségi elemzés szükséges és ez már túllépi a középiskolai matematikai kereteket.

Rendezési algoritmusok

A rendezési algoritmusok tökéletes példák a hatékonyság kérdésének tanítására, hiszen ugyanarra a feladatra, egy tömb sorba rendezésére, több algoritmust is tanítunk. Vizsgáljunk meg két különböző rendezési algoritmus futási idejét!

```
Egyszerű cserés rendezés(n,x):
Ciklus i=1-től n-1-ig
       Ciklus j=i+1-től n-ig
              Ha x[i]>x[j] akkor
Csere(x[i],x[j])
              Elágazás vége
       Ciklus vége
Ciklus vége
Eljárás vége
```

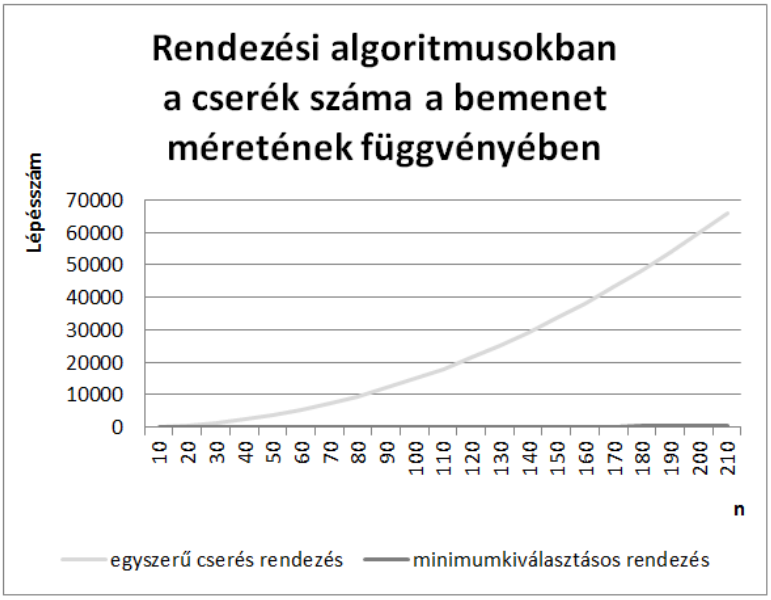
hasonlítások száma: n*(n-1)/2
mozgatások száma legrosszabb esetben: 3*n*(n-1)/2
(egy csere 3 mozgatást igényel)

```
Minimumkiválasztásos rendezés(n,x):
Ciklus i=1-től n-1-ig
       min:=i
       Ciklus j=i+1-től n-ig
          Ha X[min]>X[j] akkor min:=j
              Elágazás vége
       Ciklus vége
       Csere(x[i],x[min])
Ciklus vége
Eljárás vége
```

hasonlítások száma: n*(n-1)/2
mozgatások száma: 3*(n-1)/2

Algoritmusok komplexitása

Ha a fenti rendezési algoritmusok futási idejét megvizsgáljuk, látjuk, hogy a mozgatások száma a legrosszabb esetben az egyszerű cserés algoritmusnál $3*n*(n-1)/2=3*(n^2-n)/2$, a minimumkiválasztásos rendezésnél $3*(n-1)/2$. Az algoritmus komplexitásának fogalmát könnyen bevezethetjük, ha rajzolunk egy grafikont, mely megmutatja, hogyan változik az algoritmusok futási ideje, ha változik a bemenet mérete. Egy táblázatkezelő program segítségével a diákok maguk is megtehetik ezt.



5. ábra: Rendezési algoritmusok növekedési rendje.

A diagramon jól látható, hogy míg az egyszerű cserés rendezés esetén „sokkal jobban", négyzetesen nő a cserék száma, vagyis ha a bemenet mérete kétszeresére nő, akkor a cserék száma a négyszeresére. A minimumkiválasztásos rendezés esetén lineárisan nő a cserék száma, azaz ha a bemenet mérete kétszeresére nő, a cserék száma is csak a kétszerese lesz. Ez azt jelenti, hogy az egyszerű cserés rendezés növekedési sebessége nagyobb. Az, hogy milyen az algoritmus növekedési sebessége, az algoritmus komplexitásának nevezzük. Azaz az egyszerű cserés rendezés komplexitása négyzetes (jelölése ($O(n^2)$), a minimumkiválasztásos rendezés komplexitása lineáris ($O(n)$).

Összefoglalás

A fenti példák alapján láthatjuk, hogy az informatika tudományának, az algoritmusok tervezésének egyik legfontosabb kérdését már a programozás oktatás bevezetésénél megismertethetjük a diákokkal. Nem igényel különösebb matematikai ismereteket, mégis olyan szemléletmódot közvetíthetünk a diákok számára, ami az informatikában tovább tanuló diákok számára elengedhetetlen lesz, a többiek számára pedig érdekes, és segíti a logikus, algoritmikus gondolkodás képességének fejlesztését.

Irodalom

1. *Nemzeti alaptanterv* (2012) www.ofi.hu/nat/mk-nat-2012
2. Cormen, T. H.; Leierson, C. E.; Rives, R. L.; Stein, C.: *Új Algoritmusok.* Scolar Kiadó. (2003)
3. Gal-Ezer, J.; Zur, E.: *The concept of „algorithm efficiency" int he high school CS curriculum*. In: Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods. (2002) 161-170
4. Horowitz, E.; Sahni, S.: *Fundamentals of Computer Algorithms*. Computer Science Press. (1978)

5. MIT Teaching and Learning Facility: *Algorithm Efficiency*.
http://tll.mit.edu/help/algorithm-efficiency