Creat de Nagy Ildikó, Liceul Teoretic „Arany János” Salonta

DIVIDE ET IMPERA[1]

Oszd meg és uralkodj módszer

A módszer lényege:

- **Felbontjuk a feladatot két vagy több részfeladatra**
- A részfeladatokat továbbbontjuk kisebb részfeladatokra
- ……..
- Addig ismételjük, amíg a részfeladatok nagyon egyszerűekké (azonnalikká) válnak
- Megoldjuk a részfeladatokat.
- **Felépítjük a részmegoldásokból a feladat teljes megoldását.**
- A **rekurzív** programozási módszert használjuk.

Maximumkeresés

Adott egy n elemű tömb, határozzuk meg divide et impera módszerrel a legnagyobb elemét.

A módszer lényege:

- Felosszuk a tömböt két egyforma részre (a felénél)
- Megkeressük az első felének a legnagyobb elemét (a)
- Megkeressük a második felének a legnagyobb elemét (b)
- Összehasonlítjuk ezt a két értéket (a<b), és a megfelelőt térítjük vissza

Hogyan keressük a felosztott részek legnagyobb elemét? (Kiadjuk parancsba) vagyis
Ugyanazt megcsináltatjuk a két fél tömbbel is (felosztjuk, megkeressük, megkeressük, összehasonlítjuk)
…
És ezt addig folytatjuk (vagyis osszuk), amíg a felosztott rész, már csak egy elemből áll!
Nah, akkor pont az az egy elem lesz a **maximum!**
A megoldás módszere: REKURZIÓ

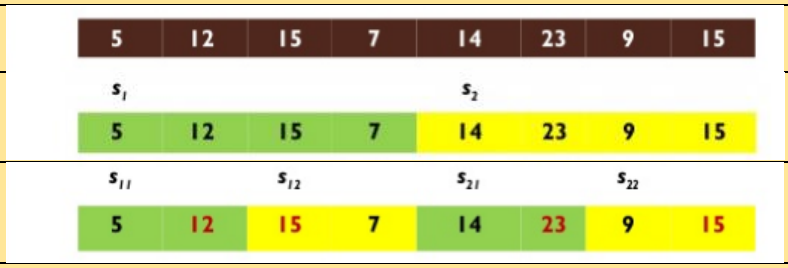
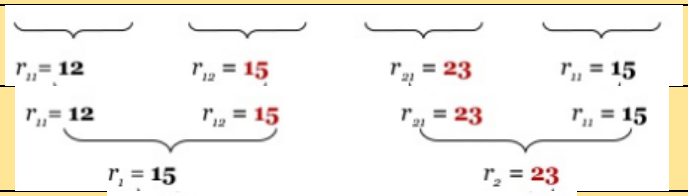
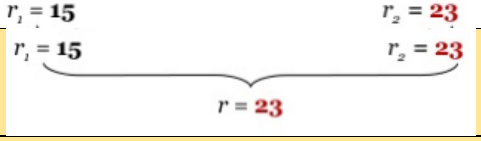
A **STOP** feltétel: amikor már csak egy elemből áll a rész.

[1] A latin divide et impera kifejezés, magyarul „Oszd meg és uralkodj!”, a Római Birodalom politikai módszerét kifejező elv, mely később többek közt XI. Lajos francia uralkodó mottója is lett.

A módszer értelmében, ha az ellenfelek közül egyeseknek ígérünk bizonyos előnyt, míg másoknak nem, ezáltal az ellenfelek sorai megbonthatóak, az ellenfelek külön-külön könnyebben legyőzhetőek, illetve ellenőrzés alatt tarthatóak.

Creat de Nagy Ildikó, Liceul Teoretic „Arany János” Salonta

Szemléltetés:

A tömb	
Felosztjuk két részre	
Tovább osztjuk:	
Tovább osztjuk, de már csak egy- egy elemből fog állni:	
Összehasonlítjuk és kiválasztjuk a nagyobbat	
Ezeket is összehasonlítjuk, és kiválasszuk a nagyobbat	
Tovább hasonlítjuk	

A program:

int v[100],n;	Értelmezzük a tömböt

int max(int i ,int j)
{ int a,b;
if (i==j) return v[i] ;
else
{
a=max(i, (i+j)/2);
b=max((i+j)/2+1,j);
if (a>b) return a;
else return b;
}
}
	Az **alprogram** i az intervallum kezdetének, j a végének az indexe
	a,b helyi változó, ők lesznek az aktuális maximumok
	i==j, azt jelenti, hogy már csak egy elem van
	
	az intervallum felének az indexe: (i+j)/2
	a az első résznek less a maximum i-től a feléig
	b a második résznek a fele+1-től a végéig
	összehasonlítjuk őket, és a nagyobbikat térítjük vissza

int main( )
{
cout<<”n=";
cin>>n;
for (int i=1;i<=n;i++)
{
cout<<”v[”<<i<<”]=”;
cin>>v[i];
}
cout<<”max=”<<max(1,n);
}
	Főprogram
	
	Beolvassuk a tömb:
	- dimenzióját
	- elemeit
	
	Meghívjuk a függvényt az 1-től n-ig intervallumra, kiírjuk a megoldást

Versenyzőknek: https://docplayer.hu/7818052-Katai-zoltan-algoritmusok-felulnezetbzl.html
Egér-sajt feladat 103 oldal

Creat de Nagy Ildikó, Liceul Teoretic „Arany János” Salonta

Bináris keresés

Adott egy n elemű tömb, elemei növekvő sorba helyezve, és egy x szám.
Határozzuk meg a **binaries** keresés módszerével, hogy a szám megtalálható-e a tomb elemei között. Használjuk a dvide et impera módszert.

Magyarázat:

- A tömböt két egyenlő részre osszuk a felénél
- Teszteljük, hogy a keresett szám, egyenlő-e kisebb vagy nagyobb a középen lévő számnál, s ennek megfelelően:
  - Visszatérítjük a poziciót, ahol megtaláltuk
  - Keressük a számot a tömb első felében
  - Keressük a számot a tömb második felében
- STOP FELTÉTEL: i>j

A rekurzív függvény:

```
int keres (int i, int j)
{
   if(i>j)
      return -1;
   else
      {
         m =(i+j)/2;
         if (x==v[m])
            return m;
         if (x<v[m])
            return keres(i,m-1);
         else
            return keres(m+1,j);
      }
}
```

Creat de Nagy Ildikó, Liceul Teoretic „Arany János” Salonta

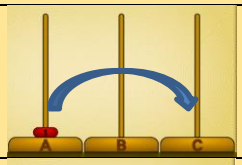
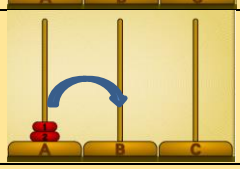
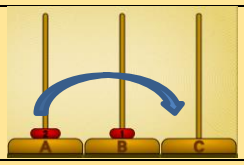
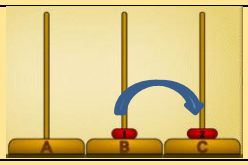
Hanoi tornyai[2]

Adott három rúd. Az egyik rúdon n darab, különböző átmérőjű korongokat található. Ezeket kell átpakolni egy másik rúdra, az alábbi szabályok betartása mellett:

- egyszerre csak egy korong mozgatható;
- mindig csak a legfelső korongot szabad levenni és áttenni egy másik rúdra;
- egy korong nem rakható nála kisebb átmérőjű korongra.

https://www.novelgames.com/en/tower/

Magyarázat:

n=1		**A → C**	1 lépés
n=2	  	A → B **A → C** B → C	3 lépés
n=3	3 korong esetén a feladat megoldása nem túl bonyolult, https://www.novelgames.com/en/tower/	A → C A → B C → B **A → C** B → A B → C A → C	7 lépésből megoldható.
n	n-1 korongot átpakolunk B –re C segítségével 1 korongot C -re n-1 korongot átpakolunk C –re A segítségével		2^n-1 lépés

2 Ez úgy is ismert, mint **Brahma tornyai**, vagy világvége feladvány.
A játékot 1883-ban Édouard Lucas francia matematikus találta fel.
Az ötletet egy legendából kapta, ami szerint a világ megteremtésekor egy 64 korongból álló hanoi torony feladványt kezdtek el „játszani” Brahma szerzetesei. A szabályok azonosak. A legenda szerint, amikor a szerzetesek végeznek majd a korongok átjuttatásával a harmadik rúdra, a kolostor összeomlik, és a világunk megszűnik létezni.
Forrás, levezetés gif: https://hu.wikipedia.org/wiki/Hanoi_tornyai

Creat de Nagy Ildikó, Liceul Teoretic „Arany János” Salonta

A rekurzív függvény:

$$H(n,a,b,c)=\begin{cases} ab, & n=1 \\ H(n-1,a,c,b),ab,H(n-1,c,b,a), & n>1 \end{cases}$$

A program:

```
#include <iostream>
using namespace std;

long k;
void Hanoi(int n, char a, char b, char c)
{
   k++;
  if (n == 1)
{
    cout << "Pakold az " << n << " korongot:  " << a << " rol " << b << " re "<<endl;
  }
else
{
    Hanoi(n-1, a,c,b);
    cout << "Pakold az " << n << " korongot:  " << a << " rol " << b << " re "<<endl;
    Hanoi(n-1,c,b,a);
  }
}

int main()
{
  int n;
  cout << "Hány korongod van: " << endl;
  cin >> n;
  Hanoi(n, 'A', 'B', 'C');
  cout << "A pakolasok szama " << k << endl;
  return 0;
}
```