Complexitatea algoritmilor

Complexitatea algoritmilor este reprezentată de spaţiul de memorare (*complexitate de spaţiu*) şi puterea de calcul (*complexitatea de timp*) necesare implementării unui algoritm. Determinarea necesarului de memorie şi a timpului de execuţie al algoritmului poartă numele de analiză a algoritmului.

Analiza complexităţii unui algoritm se poate face printr-o **măsurare statică** sau **dinamică**. Astfel, măsurarea statică presupune analiza structurii algoritmului (analiza apriorică a algoritmului) şi are ca rezultat determinarea funcţiei care mărgineşte superior timpul de execuţie al algoritmului, în timp ce măsurarea dinamică este bazată pe calitatea algoritmului, comportamentul acestuia în funcţie de volumul datelor de intrare (testul ulterior). Este necesară stabilirea unui număr suficient de teste pentru determinarea comportamentului algoritmului. De asemenea, pentru determinarea experimentală a complexităţii, înainte de a analiza algoritmul este necesar să fie cunoscut modelul de tehnologie ce urmează a fi utilizată: spaţiul de memorare disponibil şi procesorul sistemului de calcul.

Se porneşte de la stabilirea unui model de maşină de calcul se fac următoarele măsurători: se determină timpul de execuţie în cazul defavorabil (timpul de execuţíe al operaţiilor de bază în cazul introducerii celui mai defavorabil set de date de intrare), timpul de execuţie în cel mai bun caz(timpul de execuţíe al operaţiilor de bază în cazul introducerii celui mai favorabil set de date de intrare) şi timpul mediu de execuţie (timpul în care sunt executate în medie operaţiile de bază). Determinarea timpului mediu de execuţie este de multe ori o sarcină dificilă necesitând

cunoaşterea distribuţiei statistice a posibilelor date de intrare, pentru a putea calcula media ponderată a complexităţilor.

Interesantă în studiul complexităţii de timp a algoritmilor este analiza creşterii timpului de execuţie la valori mari şi foarte mari ale intrărilor. Această analiză este exprimată de cele mai multe utilizând notaţia big O (O), notaţie ce reţine doar termenul dominant care tinde cel mai repede +∞ la creşterea valorii lui n (unde n reprezintă mărimea datelor de intrare). Astfel, complexitatea de timp a algoritmului poate fi definită ca fiind *T(n)*=O*(f(n))*, dacă termenul dominant are forma *c·f(n)*, *c* constantă. Determinarea lui *T(n)* este o sarcină dificilă, chiar imposibil de realizat, dar se pot determina *c1*, *c2* şi *n0* constante astfel încât $c1 \cdot f(n) \leq T(n) \leq c2 \cdot f(n)$, $(\forall) n > n0$.

Notăm *T(n)*=O*(f(n)),* dacă (∃)*n0* şi *c* constante astfel încât $T(n) \leq c \cdot f(n)$, $(\forall) n > n0$ semnificând că *T* nu creşte mai repede decât *f* eventual multiplicat cu o constantă. Putem concluziona că O desemnează marginea asimptotică superioară a unei funcţii.

Tipuri de probleme în funcţie de complexitatea lor:

Complexitate	Denumire
O(1)	algoritm cu timp de execuție constant
O(log n)	algoritm logaritmic
O(n)	algoritm liniar
$O(n^2)$	algoritm pătratic
$O(n^3)$	algoritm cubic
$O(n^k)$	algoritm polinomial
$O(2^n)$	algoritm exponențial

$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(n^k) < O(2^n)$

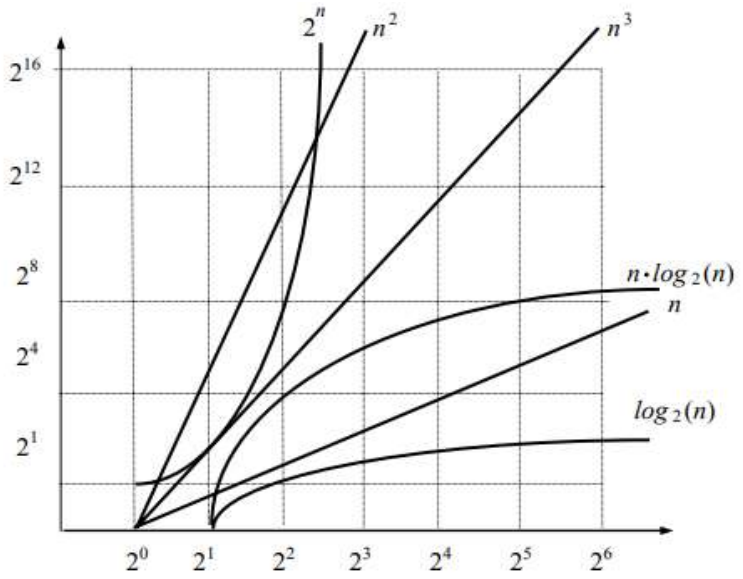

Figura 7. Reprezentarea grafică a funcţiilor care determină ordinului de complexitate

Analizând comportamentul funcţiilor care determină ordinul de complexitate putem concluziona că în cele mai multe cazuri algoritmii polinomiali sunt de preferat celor exponenţiali, dar uneori un algoritm exponenţial poate fi mai eficient decât unul polinomial pentru valori mici ale lui n şi un algoritm exponenţial se poate comporta acceptabil pentru unele probleme particulare (I.Odăgescu,1998).

Exemple de rezolvare a problemelor prin elaborarea de algoritmi eficienţi din punct de vedere al spaţiului şi al timpului de execuţie:

Problema propusă la simularea examenului de bacalaureat sesiunea 2017:

Fișierul ”bac.in” conține pe prima linie două numere naturale din intervalul $[2,10^4]$, m și n, iar pe fiecare dintre următoarele două linii câte un șir de m, respectiv n numere naturale din intervalul $[0,10^9]$, ordonate strict crescător. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu. Se cere să se afișeze pe ecran, în ordine strict descrescătoare,

numerele pare care apar în cel puțin unul dintre cele două șiruri. Numerele afișate sunt separate prin câte un spațiu, iar dacă nu există nici un astfel de număr, se afișează pe ecran mesajul nu exista. Pentru determinarea numerelor cerute se va utiliza un algoritm eficient din punctul de vedere al timpului de executare.

Exemplu: dacă fișierul conține numerele

5 6

1 4 8 9 10

2 4 10 12 15 18

se afișează pe ecran

18 12 10 8 4 2

La prima vedere, soluţia la care se gândesc majoritatea elevilor este aceea de a salva cele două şiruri în 2 tablouri unidimensionale, de a le concatena, apoi de a le sorta şi în finalde a afişa doar valorile pare. Dar care este complexitatea de timp a algoritmului? Având în vedere că metodele de sortare cel mai des utilizate sunt sortarea prin metoda bulelor şi sortarea prin interschimbare putem spune că algoritmul va avea complexitatea de timp de $O(n^2)$. În acest moment al lecţiei, în încercarea de a descoperi un algoritm mai eficient, prin intrebări ajutătoare, elevii descoperă că utilizarea algoritmul de interclasare pentru a reuni, ordonate elementele celor 2 tablouri ar conduce la o complexitate de tip mai bună, respectiv O(nlogn).

Un alt exemplu ar putea fi următoarea problemă dată la examenul de bacalaureat din sesiunea iunie-iulie din 2011:

Se citesc de la tastatură două numere naturale s1 si s2 ($0<s1\leq 18$, $0\leq s2\leq 18$) si se cere scrierea în fisierul BAC.TXT, fiecare pe câte o linie, în ordine strict crescătoare, a tuturor numerelor naturale cu exact 5 cifre, pentru care suma primelor două cifre este egală cu s1, iar suma ultimelor

două cifre este egală cu s2. Pentru determinarea numerelor indicate se utilizează un algoritm eficient din punct de vedere al timpului de executare.

Exemplu: dacă s1=8, iar s2=7, atunci 35725 este unul dintre numerele care respect proprietatea cerută (3+5=8 si 2+5=7).

În general prima soluţie pe care o sugerează elevii mei presupune parcurgerea numerelor de 5 cifre, adică din intervalul [10000,99999], determinarea utilizând algoritmi cu cifrele unui număr sumei ultimelor 2 cifre, respectiv a primelor 2 cifre pentru fiecare număr din interval şi afişarea doar a celor care îndeplinesc condiţia din problemă. După discutarea punctelor slabe ale acestui algoritm (90000 de numere ce trebuie descompuse în cifre, pentru fiecare dintre numere verificarea sumei primelor 2 cifre cu s1, a ultimelor 2 cifre cu s2), elevii descoperă dirijat soluţia de compunere a numerelor din cifre:

```
natural s1,s2,c1,c2,c3,c4,c5,m1,m2;
citeşte s1,s2;
Pentru c1←1,m1 execută
 c2←s1-c1
Pentru c3←0,9 execută
   Pentru c4=0,m2 execută
      c5←s2-c4
      scrie c1 · 10^4+c2 · 10^3+c3 · 10^3+c4 · 10+c5
```

unde m1 este valoarea minimă între s1 şi 9, iar m2 este valoarea minimă între s2 şi 9. Avantajul acestui algoritm este acela că numărul maxim de repetări este 9*10*10, comparativ cu 90000 în primul caz.