2.6. Logaritmic

Performanța *O*(log *N*) este mai greu de explicat printr-o simplă secvență de cod.

In paragraful următor se va încerca explicarea acesteia prin folosirea ca și exemplu a căutării binare.

Căutarea binară este o tehnică folosită pentru a căuta seturi de date **sortate**. Funcționează selectând elementul de mijloc al setului de date pe care îl compară cu o valoare țintă. Dacă valorile se potrivesc, aceasta va returna succesul. Dacă valoarea țintă este mai mare decât valoarea elementului sondă, va lua jumătatea superioară a setului de date și va efectua aceeași operație pentru ea. De asemenea, dacă valoarea țintă este mai mică decât valoarea elementului țintă, va efectua operațiunea față de jumătatea inferioară. Va continua să reducă la jumătate setul de date cu fiecare iterație până când valoarea a fost găsită sau până când nu va mai putea împărți setul de date.

Acest tip de algoritm este descris ca *O*(log *N*).

Reducerea iterativă la jumătate a seturilor de date descrise în exemplul de căutare binară produce o curbă de creștere care se ridică la început și se aplatizează lent pe măsură ce dimensiunea seturilor de date crește. De exemplu: un set de date de intrare care conține 10 elemente durează o secundă pentru a fi completat, un set de date care conține 100 de articole durează două secunde și un set de date care conține 1000 de elemente va dura trei secunde. Dublarea dimensiunii setului de date de intrare are un efect redus asupra creșterii sale, deoarece după o singură iterație a algoritmului, setul de date va fi redus la jumătate și, prin urmare, la egalitate cu un set de date de intrare având jumătate din dimensiune. Acest lucru face ca algoritmi precum căutarea binară să fie extrem de eficienți atunci când este vorba de seturi mari de date.

#### 2.7. Reguli de determinare a timpului de execuție T(n) și O(f(n))

1. Daca *T*1(*n*) și *T*2(*n*) sunt timpii de execuție a două secvențe de program *P*1 și *P*2, *T*1(*n*) fiind *O*(*f*(*n*)), iar *T*2(*n*) fiind *O*(*g*(*n*)), atunci timpul de executie *T*1(*n*)+*T*2(*n*), al secvenței *P*1 urmată de *P*2, va fi *O*(*max*(*f*(*n*),*g*(*n*))).
2. Timpul de execuție a unei instructiuni de asignare, citire sau scriere, este *O*(*n*0)=*O*(1);
3. Timpul de rulare a unei secvențe de instrucțiuni e determinat de regula de însumare, fiind proportional cu cel mai lung timp din cei ai instrucțiunilor secvenței;
4. Timpul de execuție a unei instrucțiuni if-then-else este suma dintre timpul de evaluare a condiției (*O*(1)) și cel mai mare dintre timpii de execuție ai instrucțiunilor pentru condiție adevarată sau falsă;
5. Timpul de execuție a unei instrucțiuni de ciclare este suma, pentru toate iterațiile, dintre timpul de execuție a corpului instrucțunii și cel de evaluare a condiției de terminare (O(1));
6. Pentru evaluarea timpului de execuție a unei funcții recursive, se asociază fiecarei iterații recursive un timp necunoscut *T*(*n*), unde *n* masoară argumentele funcției; se poate obține o relație recurentă pentru *T*(*n*), adică o ecuație pentru *T*(*n*), în termeni *T*(*k*), pentru diferite valori ale lui *k*;
7. Timpul de execuție poate fi analizat chiar pentru programele scrise in pseudocod; pentru secvențele care cuprind operații asupra unor TDA, se pot alege câteva implementări și astfel se poate face comparație între performanțele implementărilor, în contextul aplicației respective.

#### 2.8. Comparație

Prin determinarea eficienței O(f(n)) a diferiților algoritmi, ajungem să reducem, în practică, comparația eficienței acestor algoritmi la comparația funcțiilor *O*(*f*(*n*)) pentru fiecare algoritm. În această comparație, ținem cont ca cele mai obișnuite ordine de mărime ale notației *O* se află în relația:

O(1) < *O*(*log* *n*) < *O*(*n*) < *O*(*n* \* *log* *n*) < *O*(*n*2 ) < *O*(*n*3 ) <  ... < *O*(2*n* ) < *O*(10 *n* ) < *O*( *n* ! ) < O(*n* *n* )