8 Sortarea rapidă

Sortarea rapidă este un algoritm de sortare care, pentru un şir de n elemente, are un timp de execuţie $\Theta(n^2)$, în cazul cel mai defavorabil. În ciuda acestei comportări proaste, în cazul cel mai defavorabil, algoritmul de sortare rapidă este deseori cea mai bună soluţie practică, deoarece are o comportare medie remarcabilă: timpul său mediu de execuţie este $\Theta(n \lg n)$, şi constanta ascunsă în formula $\Theta(n \lg n)$ este destul de mică. Algoritmul are avantajul că sortează pe loc (în spaţiul alocat şirului de intrare) şi lucrează foarte bine chiar şi într-un mediu de memorie virtuală.

În secţiunea 8.1 sunt descrişi algoritmul şi un subalgoritm important, folosit de sortarea rapidă pentru partiţionare. Deoarece comportarea algoritmului de sortare rapidă este foarte complexă, vom începe studiul performanţei lui cu o discuţie intuitivă în secţiunea 8.2, şi lăsăm analiza precisă la sfârşitul capitolului. În secţiunea 8.3 sunt prezentate două variante ale algoritmului de sortare rapidă, care utilizează un generator de numere aleatoare. Aceşti algoritmi aleatori au multe proprietăţi interesante. Timpul lor mediu de execuţie este bun şi nu se cunosc date de intrare particulare pentru care să aibă comportarea cea mai proastă. O variantă aleatoare a algoritmului de sortare rapidă este studiată în secţiunea 8.4 şi se demonstrează că timpul lui de execuţie, în cazul cel mai defavorabil, este $O(n^2)$, iar timpul mediu $O(n \lg n)$.

8.1. Descrierea sortării rapide

Algoritmul de sortare rapidă, ca de altfel şi algoritmul de sortare prin interclasare, se bazează pe paradigma "divide şi stăpâneşte", introdusă în secţiunea 1.3.1. Iată un proces "divide şi stăpâneşte" în trei paşi, pentru un subşir $A[p..r]$.

Divide: Şirul $A[p..r]$ este împărţit (rearanjat) în două subşiruri nevide $A[p..q]$ şi $A[q+1..r]$, astfel încât fiecare element al subşirului $A[p..q]$ să fie mai mic sau egal cu orice element al subşirului $A[q+1..r]$. Indicele q este calculat de procedura de partiţionare.

Stăpâneşte: Cele două subşiruri $A[p..q]$ şi $A[q+1..r]$ sunt sortate prin apeluri recursive ale algoritmului de sortare rapidă.

Combină: Deoarece cele două subşiruri sunt sortate pe loc, nu este nevoie de nici o combinare, şirul $A[p..r]$ este ordonat.

Descrierea algoritmului este următoarea:

QUICKSORT(A, p, r)
1: **dacă** $p < r$ **atunci**
2: $q \leftarrow$ PARTIŢIE(A, p, r)
3: QUICKSORT(A, p, q)
4: QUICKSORT$(A, q+1, r)$

Pentru ordonarea întregului şir A, iniţial se apelează QUICKSORT(A, 1, *lungime*[A]).

Partiţionarea şirului

Cheia algoritmului este procedura PARTIŢIE, care rearanjează pe loc subşirul $A[p..r]$.

```
PARTIŢIE(A, p, r)
 1: x ← A[p]
 2: i ← p − 1
 3: j ← r + 1
 4: cât timp ADEVĂRAT execută
 5:    repetă
 6:       j ← j − 1
 7:    până când A[j] ≤ x
 8:    repetă
 9:       i ← i + 1
10:    până când A[i] ≥ x
11:    dacă i < j atunci
12:       interschimbă A[i] ↔ A[j]
13:    altfel
14:       returnează j
```

În figura 8.1 este ilustrat modul de funcţionare a procedurii PARTIŢIE. Întâi se selectează un element $x = A[p]$ din şirul $A[p..r]$, care va fi elementul "pivot", în jurul căruia se face partiţionarea şirului $A[p..r]$. Apoi, două subşiruri $A[p..i]$ şi $A[j..r]$ cresc la începutul şi respectiv, sfârşitul şirului $A[p..r]$, astfel încât fiecare element al şirului $A[p..i]$ să fie mai mic sau egal cu x, şi orice element al şirului $A[j..r]$, mai mare sau egal cu x. La început $i = p - 1$ şi $j = r + 1$, deci cele două subşiruri sunt vide.

În interiorul ciclului **cât timp**, în liniile 5–7, indicele j se decrementează, iar i se incrementează până când $A[i] \geq x \geq A[j]$. Presupunând că inegalităţile de mai sus sunt stricte, $A[i]$ este prea mare ca să aparţină primului subşir (cel de la început), iar $A[j]$ prea mic ca să aparţină celui de al doilea subşir (cel de la sfârşit). Astfel, interschimbând $A[i]$ cu $A[j]$ (linia 12), cele două părţi cresc. (Interschimbarea se poate face şi în cazul în care avem inegalităţi stricte.)

Ciclul **cât timp** se repetă până când inegalitatea $i \geq j$ devine adevărată. În acest moment, întregul şir $A[p..r]$ este partiţionat în două subşiruri $A[p..q]$ şi $A[q+1..r]$, astfel încât $p \leq q < r$ şi nici un element din $A[p..q]$ nu este mai mare decât orice element din $A[q+1..r]$. Procedura returnează valoarea $q = j$.

De fapt, procedura de partiţionare execută o operaţie simplă: pune elementele mai mici decât x în primul subşir, iar pe cele mai mari decât x în subşirul al doilea. Există câteva particularităţi care determină o comportare interesantă a procedurii PARTIŢIE. De exemplu, indicii i şi j nu depăşesc niciodată marginile vectorului $A[p..r]$, dar acest lucru nu se vede imediat din textul procedurii. Un alt exemplu: este important ca elementul $A[p]$ să fie utilizat drept element pivot x. În schimb, dacă se foloseşte $A[r]$ ca element pivot, şi, întâmplător, $A[r]$ este cel mai mare element al vectorului $A[p..r]$, atunci PARTIŢIE returnează procedurii QUICKSORT valoarea $q = r$, şi procedura intră într-un ciclu infinit. Problema 8-1 cere să se demonstreze corectitudinea procedurii PARTIŢIE.

Timpul de execuţie al procedurii PARTIŢIE, în cazul unui vector $A[p..r]$, este $\Theta(n)$, unde $n = r - p + 1$ (vezi exerciţiul 8.1-3.).

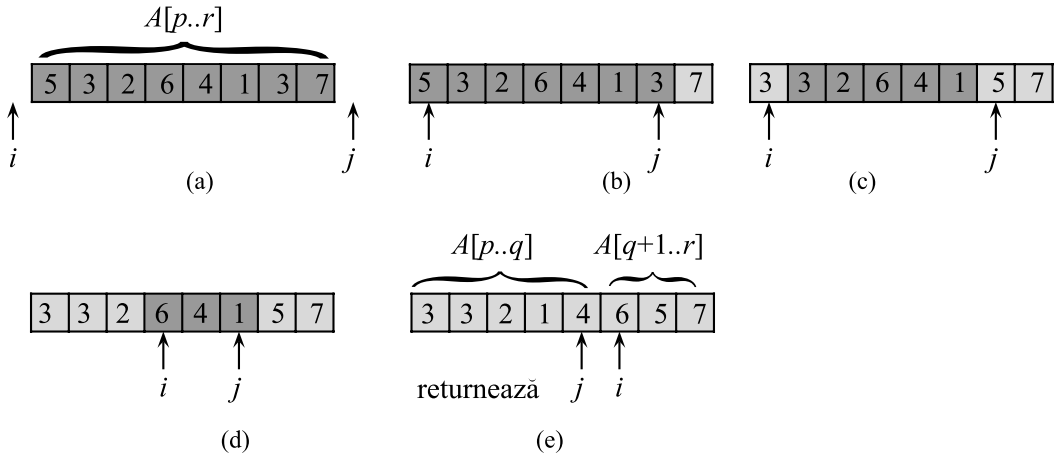

Figura 8.1 Operaţiile efectuate de procedura PARTIŢIE pe un exemplu. Elementele haşurate în gri deschis sunt deja plasate în poziţiile lor corecte, iar cele haşurate închis încă nu. (**a**) Şirul de intrare, cu valorile iniţiale ale variabilelor i şi j, care puncteză în afara şirului. Vom face partiţionarea în jurul elementului $x = A[p] = 5$. (**b**) Poziţiile lui i şi j în linia 11 a algoritmului, după prima parcurgere a ciclului **cât timp**. (**c**) Rezultatul schimbului de elemente descris în linia 12. (**d**) Valorile lui i şi j în linia 11 după a doua parcurgere a ciclului **cât timp**. (**e**) Valorile lui i şi j după a treia şi ultima iteraţie a ciclului **cât timp**. Procedura se termină deoarece $i \geq j$ şi valoarea returnată este $q = j$. Elementele şirului până la $A[j]$, inclusiv, sunt mai mici sau egale cu $x = 5$, iar cele de după $A[j]$, sunt toate mai mici sau egale cu $x = 5$.

Exerciţii

8.1-1 Folosind figura 8.1 drept model, să se ilustreze operaţiile procedurii PARTIŢIE în cazul vectorului $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21 \rangle$.

8.1-2 Ce valoare a lui q returnează procedura PARTIŢIE, dacă toate elementele vectorului $A[p..r]$ sunt egale.

8.1-3 Să se argumenteze, pe scurt, afirmaţia că timpul de execuţie al procedurii PARTIŢIE, pentru un vector de n elemente, este $\Theta(n)$.

8.1-4 Cum trebuie modificată procedura QUICKSORT pentru a ordona descrescător?

8.2. Performanţa algoritmului de sortare rapidă

Timpul de execuţie al algoritmului de sortare rapidă depinde de faptul că partiţionarea este echilibrată sau nu, iar acesta din urmă de elementele alese pentru partiţionare. Dacă partiţionarea este echilibrată, algoritmul asimptotic este la fel de rapid ca sortarea prin interclasare. În cazul în care partiţionarea nu este echilibrată, algoritmul se execută la fel de încet ca sortarea prin inserare. În această secţiune vom investiga, fără rigoare matematică, performanţa algoritmului de sortare rapidă în cazul partiţionării echilibrate.

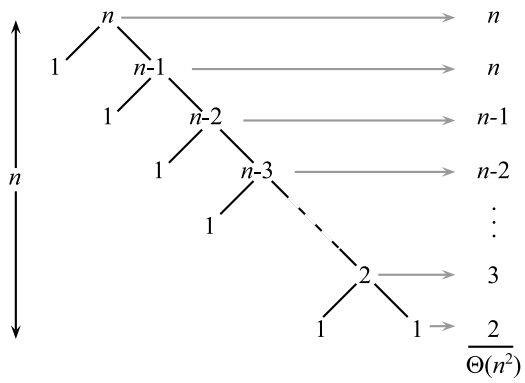

Figura 8.2 Arborele de recursivitate pentru QUICKSORT când procedura PARTIŢIE pune întotdeauna într-o parte a vectorului numai un singur element (cazul cel mai defavorabil). Timpul de execuţie în acest caz este $\Theta(n^2)$.

Partiţionarea în cazul cel mai defavorabil

Comportarea cea mai defavorabilă a algoritmului de sortare rapidă apare în situaţia în care procedura de partiţionare produce un vector de $n-1$ elemente şi unul de 1 element. (Demonstraţia se face în secţiunea 8.4.1) Să presupunem că această partiţionare dezechilibrată apare la fiecare pas al algoritmului. Deoarece timpul de partiţionare este de $\Theta(n)$, şi $T(1) = \Theta(1)$, formula recursivă pentru timpul de execuţie a algoritmului de sortare rapidă este:

$$T(n) = T(n-1) + \Theta(n).$$

Pentru evaluarea formulei de mai sus, observăm că $T(1) = \Theta(1)$, apoi iterăm formula:

$$T(n) = T(n-1) + \Theta(n) = \sum_{k=1}^{n} \Theta(k) = \Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2).$$

Ultima egalitate se obţine din observaţia că $\sum_{k=1}^{n} k$ este o progresie aritmetică (3.2). În figura 8.2 este ilustrat arborele de recursivitate pentru acest cel mai defavorabil caz al algoritmului de sortare rapidă. (Vezi secţiunea 4.2 pentru alte detalii privind arborii recursivi.)

Dacă partiţionarea este total dezechilibrată la fiecare pas recursiv al algoritmului, atunci timpul de execuţie este $\Theta(n^2)$. Deci timpul de execuţie, în cazul cel mai defavorabil, nu este mai bun decât al algoritmului de sortare prin inserare. Mai mult, timpul de execuţie este $\Theta(n^2)$ chiar şi în cazul în care vectorul de intrare este ordonat – caz în care algoritmul de sortare prin inserare are timpul de execuţie $O(n)$.

Partiţionarea în cazul cel mai favorabil

Dacă algoritmul de partiţionare produce doi vectori de $n/2$ elemente, algoritmul de sortare rapidă lucrează mult mai repede. Formula de recurenţă în acest caz este:

$$T(n) = 2T(n/2) + \Theta(n)$$

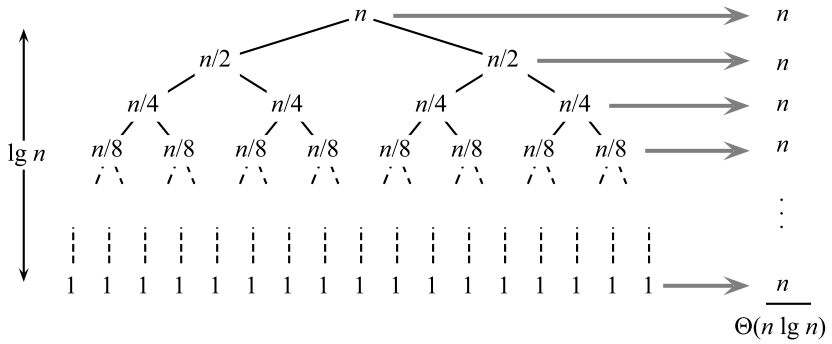

Figura 8.3 Arborele de recurenţă pentru QUICKSORT când procedura PARTIŢIE produce întotdeauna părţi egale (cazul cel mai favorabil). Timpul de execuţie rezultat este $\Theta(n \lg n)$.

a cărei soluţie este $T(n) = \Theta(n \lg n)$ (după cazul 2 al teoremei 4.1). Deci partiţionarea cea mai bună produce un algoritm de sortare mult mai rapid. În figura 8.3 se ilustrează arborele de recursivitate pentru acest cel mai favorabil caz.

Partiţionarea echilibrată

Analiza din secţiunea 8.4 va arăta că timpul mediu de execuţie a algoritmului de sortare rapidă este mult mai apropiat de timpul cel mai bun decât de timpul cel mai rău. Pentru a înţelege de ce este aşa, ar trebui să studiem efectul partiţionării echilibrate asupra formulei recursive care descrie timpul de execuţie.

Să presupunem că procedura de partiţionare produce întotdeauna o împărţire în proporţie de 9 la 1, care la prima vedere pare a fi o partiţionare dezechilibrată. În acest caz, formula recursivă pentru timpul de execuţie al algoritmului de sortare rapidă este:

$$T(n) = T(9n/10) + T(n/10) + n$$

unde, pentru simplificare, în loc de $\Theta(n)$ s-a pus n. Arborele de recurenţă corespunzător se găseşte în figura 8.4. Să observăm că la fiecare nivel al arborelui costul este n până când la adâncimea $\log_{10} n = \Theta(\lg n)$ se atinge o condiţie iniţială. În continuare, la celelalte niveluri, costul nu depăşeşte valoarea n. Apelul recursiv se termină la adâncimea $\log_{10/9} n = \Theta(\lg n)$. Costul total al algoritmului de sortare rapidă este deci $\Theta(n \lg n)$. Ca urmare, cu o partiţionare în proporţie de 9 la 1 la fiecare nivel al partiţionării (care intuitiv pare a fi total dezechilibrată), algoritmul de sortare rapidă are un timp de execuţie de $\Theta(n \lg n)$ – asimptotic acelaşi ca în cazul partiţionării în două părţi egale. De fapt, timpul de execuţie va fi $O(n \lg n)$ şi în cazul partiţionării într-o proporţie de 99 la 1. La orice partiţionare într-o proporţie *constantă*, adâncimea arborelui de recursivitate este $\Theta(\lg n)$ şi costul, la fiecare nivel, este $O(n)$. Deci timpul de execuţie este $\Theta(n \lg n)$ la orice partiţionare într-o proporţie constantă.

Intuirea comportării medii

Pentru a avea o idee clară asupra comportării medii a algoritmului de sortare rapidă, trebuie să facem presupuneri asupra frecvenţei anumitor intrări. Cea mai evidentă presupunere este că

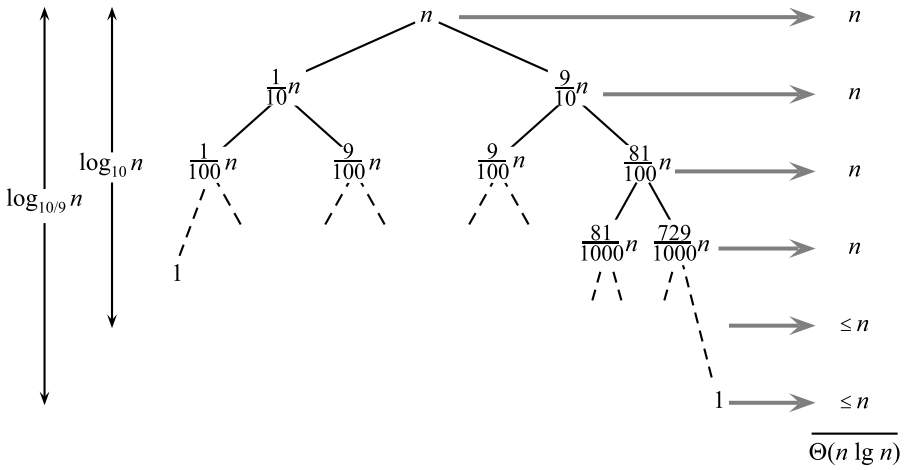

Figura 8.4 Arborele de recurenţă pentru QUICKSORT, când procedura PARTIŢIE produce întotdeauna părţi în proporţie de 9 la 1, rezultând un timp de execuţie de $\Theta(n \lg n)$.

toate permutările elementelor de intrare sunt la fel de probabile. Vom discuta această presupunere în secţiunea următoare, aici vom exploata doar câteva variante.

În situaţia în care algoritmul de sortare rapidă lucrează pe o intrare aleatoare, probabil că nu va partiţiona la fel la fiecare nivel, cum am presupus în discuţiile anterioare. Este de aşteptat ca unele partiţionări să fie echilibrate, altele nu. De exemplu, exerciţiul 8.2-5 cere să se demonstreze că procedura PARTIŢIE produce în 80% din cazuri o partiţionare mai echilibrată decât proporţia de 9 la 1, şi, numai în 20% din cazuri, una mai puţin echilibrată.

În cazul mediu, procedura PARTIŢIE produce un amestec de partiţionări "bune" şi "rele". Într-un arbore de recurenţă pentru cazul mediu al procedurii PARTIŢIE, partiţionările bune şi rele sunt distribuite aleator. Să presupunem, totuşi, pentru simplificare, că partiţionările bune şi rele alternează pe niveluri, şi că partiţionările bune corespund celui mai bun caz, iar cele rele celui mai defavorabilcaz. În figura 8.5 sunt prezentate partiţionările la două niveluri consecutive în arborele de recursivitate. Costul partiţionării la rădăcina arborelui este n, iar vectorii obţinuţi sunt de dimensiune $n-1$ şi 1: cazul cel mai defavorabil. La nivelul următor, vectorul de $n-1$ elemente se împarte în doi vectori de $(n-1)/2$ elemente fiecare, potrivit cazului celui mai bun. Să presupunem că pentru un vector de dimensiune 1 (un element) costul este 1.

Combinarea unei partiţionări rele şi a uneia bune produce trei vectori de dimensiune 1, $(n-1)/2$ şi respectiv $(n-1)/2$, cu un cost total de $2n-1 = \Theta(n)$. Evident, această situaţie nu este mai rea decât cea prezentată în figura 8.5(b), adică cea cu un singur nivel, care produce un vector de $(n-1)/2+1$ elemente şi unul de $(n-1)/2$ elemente, cu un cost total de $n = \Theta(n)$. Totuşi, situaţia din urmă este aproape echilibrată, cu siguranţă mult mai bună decât proporţia 9 la 1. Intuitiv, o partiţionare defavorabilă de un cost $\Theta(n)$ poate fi absorbită de una bună tot de un cost $\Theta(n)$, şi partiţionarea rezultată este favorabilă. Astfel timpul de execuţie al algoritmului de sortare rapidă, când partiţionările bune şi rele alternează, este acelaşi ca în cazul partiţionărilor bune: tot $O(n \lg n)$, doar constanta din notaţia O este mai mare. O analiză riguroasă a cazului mediu se va face în secţiunea 8.4.2.

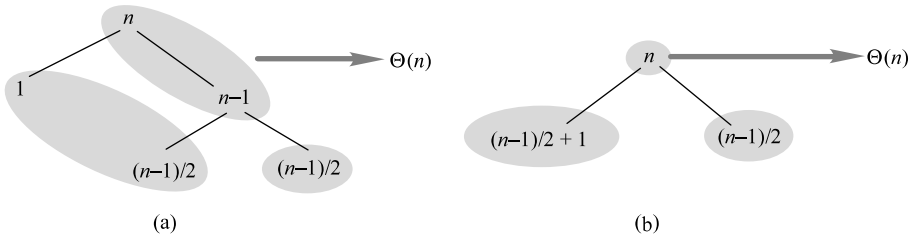

Figura 8.5 **(a)** Două niveluri ale arborelui de recurentă pentru algoritmul de sortare rapidă. Partiţionarea la nivelul rădăcinii consumă n unităţi de timp şi produce o partiţionare "proastă": doi vectori de dimensiune 1 şi $n-1$. Partiţionarea unui subşir de $n-1$ elemente necesită $n-1$ unităţi de timp şi este o partiţionare "bună": produce două subşiruri de $(n-1)/2$ elemente fiecare. **(b)** Un singur nivel al arborelui de recurenţă care este mai rău decât nivelurile combinate de la (a), totuşi foarte bine echilibrat.

Exerciţii

8.2-1 Demonstraţi că timpul de execuţie al algoritmului QUICKSORT, în cazul unui vector A cu toate elementele egale între ele, este $\Theta(n \lg n)$.

8.2-2 Demonstraţi că, în cazul unui vector A având n elemente distincte, ordonate descrescător, timpul de execuţie al algoritmului QUICKSORT este $\Theta(n^2)$.

8.2-3 Băncile, de obicei, înregistrează tranzacţiile pe un cont după data tranzacţiilor, dar multora dintre clienţi le place să primească extrasele de cont, ordonate după numărul cecurilor, deoarece ei primesc cecurile grupate de la comercianţi. Problema convertirii ordonării după data tranzacţiei în ordonare după numărul de cec este problema sortării unui vector aproape ordonat. Să se argumenteze de ce, în acest caz, algoritmul SORTEAZĂ-PRIN-INSERŢIE este mai bun decât algoritmul QUICKSORT.

8.2-4 Să presupunem că partiţionarea la fiecare nivel, în cadrul algoritmului de sortare rapidă, se face într-o proporţie de $1-\alpha$ la α, unde $0 < \alpha \leq 1/2$ şi α este constantă. Demonstraţi că, în arborele de recurenţă, adâncimea minimă a unui nod terminal este aproximativ $-\lg n / \lg \alpha$, pe când adâncimea maximă este aproximativ $-\lg n / \lg(1-\alpha)$. (Nu se va ţine cont de rotunjiri.)

8.2-5 $\star$ Argumentaţi afirmaţia că, pentru orice constantă $0 < \alpha \leq 1/2$, probabilitatea ca pentru o intrare aleatoare procedura PARTIŢIE să producă o partiţionare mai echilibrată decât proporţia $1-\alpha$ la α, este aproximativ $1-2\alpha$. Pentru ce valoare a lui α partiţionarea mai echilibrată are şanse egale cu cea mai puţin echilibrată?

8.3. Variantele aleatoare ale sortării rapide

În analiza comportării medii a algoritmului de sortare rapidă, am presupus că toate permutările elementelor de intrare sunt la fel de probabile. Mulţi consideră că, dacă această presupunere este adevărată, algoritmul de sortare rapidă, este cea mai bună soluţie pentru a sorta vectori de dimensiuni mari. Totuşi, nu ne putem aştepta ca, în practică aceasta să se întâmple întotdeauna

aşa (vezi exerciţiul 8.2-3). În această secţiune, vom introduce noţiunea de algoritm aleator şi vom prezenta două variante aleatoare ale algoritmului de sortare rapidă, care fac inutilă presupunerea că toate permutările elementelor de intrare sunt la fel de probabile.

O alternativă, la *a presupune* o distribuţie de date de intrare, este de *a impune* o astfel de distribuţie. De exemplu, să presupenem că, înainte să sorteze vectorul de intrare, algoritmul de sortare rapidă permută aleator elementele pentru a asigura astfel proprietatea că fiecare permutare are aceeaşi probabilitate. (Exerciţiul 8.3-4 cere scrierea unui algoritm care permută aleator elementele unui vector de dimensiune n în $O(n)$ unităţi de timp.) Această modificare nu îmbunătăţeşte timpul de execuţie a algoritmului în cazul cel mai defavorabil, dar asigură ca timpul de execuţie să nu depindă de ordinea elementelor de intrare.

Un algoritm se numeşte ***aleator*** dacă comportarea lui depinde nu numai de valorile de intrare, ci şi de valorile produse de un ***generator de numere aleatoare***. Vom presupune că dispunem de un generator de numere aleatoare numit RANDOM. Un apel al procedurii RANDOM(a, b) produce un număr întreg între a şi b inclusiv. Fiecare număr întreg din acest interval $[a, b]$ este la fel de probabil. De exemplu, RANDOM(0,1) produce 0 şi 1 cu aceeaşi probabilitate 1/2. Fiecare întreg generat de RANDOM este independent de valorile generate anterior. Ne-am putea imagina că RANDOM este un zar cu $(b - a + 1)$ feţe. (Majoritatea mediilor de programare posedă un ***generator de numere pseudoaleatoare***, un algoritm determinist care produce numere care "par" a fi aleatoare.)

Această variantă a algoritmului de sortare rapidă (ca de altfel mulţi alţi algoritmi aleatori) are interesanta proprietate că, practic, *pentru nici o intrare nu are comportarea cea mai defavorabilă*. Din contră, cazul cel mai defavorabil depinde de generatorul de numere aleatoare. Chiar dacă dorim, nu reuşim să generăm un vector de intrare prost, deoarece permutarea aleatoare face ca ordinea datelor de intrare să fie irelevantă. Algoritmul aleator poate avea o comportare proastă numai dacă generatorul de numere aleatoare produce o intrare nefericită. În exerciţiul 13.4-4 se arată că pentru aproape toate permutările elementelor de intrare, algoritmul de sortare rapidă are o comportare similară cu cazul mediu şi această comportare este apropiată de cea mai defavorabilă doar pentru *foarte* puţine permutări.

O strategie aleatoare este de obicei utilă când există multe moduri în care un algoritm se poate executa, dar este dificil să determinăm un mod care să fie cu siguranţă bun. Dacă multe din aceste alternative sunt bune, alegerea pur şi simplu a uneia, la întâmplare, poate reprezenta o strategie bună. Deseori, în timpul execuţiei sale, un algoritm trebuie să ia multe decizii. Dacă beneficiile unei decizii bune sunt mult mai mari decât costurile unor decizii rele, o selecţie aleatoare de decizii bune şi rele poate determina un algoritm eficient. În secţiunea 8.2 am arătat că o combinaţie de partiţionări bune şi rele asigură un timp de execuţie bun pentru algoritmul de sortare rapidă, şi acest fapt sugerează că şi algoritmul aleator va avea o comportare bună.

Prin modificarea procedurii PARTIŢIE, se poate obţine o variantă nouă a algoritmului de sortare rapidă care foloseşte această strategie de alegere aleatoare. La fiecare pas al sortării rapide, înainte de partiţionarea vectorului, interschimbăm elementul $A[p]$ cu un element ales aleator din vectorul $A[p..r]$. Această modificare asigură ca elementul pivot $x = A[p]$ să fie, cu aceeaşi probabilitate, orice element dintre cele $r - p + 1$ elemente ale vectorului. Astfel, partiţionarea vectorului de intrare poate fi, în medie, rezonabil de echilibrată. Algoritmul aleator, bazat pe permutarea elementelor de intrare, are şi el o comportare medie bună, dar ceva mai dificil de studiat decât această versiune.

Modificările asupra procedurilor PARTIŢIE şi QUICKSORT sunt minore. În noua procedură de partiţionare inserăm pur şi simplu interschimbarea celor două elemente înainte de partiţionare:

PARTIŢIE-ALEATOARE(A, p, r)
1: $i \leftarrow$ RANDOM(p, r)
2: interschimbă $A[p] \leftrightarrow A[i]$
3: **returnează** PARTIŢIE (A, p, r)

Noul algoritm de sortare rapidă foloseşte în loc de PARTIŢIE procedura PARTIŢIE-ALEATOARE:

QUICKSORT-ALEATOR(A, p, r)
1: **dacă** $p < r$ **atunci**
2: $q \leftarrow$ PARTIŢIE-ALEATOARE(A, p, r)
3: QUICKSORT-ALEATOR(A, p, q)
4: QUICKSORT-ALEATOR$(A, q + 1, r)$

Analiza algoritmului se va face în secţiunea următoare.

Exerciţii

8.3-1 De ce, în loc să studiem comportarea cea mai defavorabilă a unui algoritm aleator, studiem comportarea medie?

8.3-2 De câte ori se apelează generatorul de numere aleatoare RANDOM în timpul execuţiei procedurii QUICKSORT-ALEATOR în cazul cel mai defavorabil? Cum se schimbă răspunsul în cazul cel mai favorabil?

8.3-3 $\star$ Descrieţi o implementare a algoritmului RANDOM(a, b), care foloseşte numai aruncări de monede perfecte. Care va fi timpul mediu de execuţie a procedurii?

8.3-4 $\star$ Scrieţi un algoritm aleator care rulează în $\Theta(n)$ unităţi de timp, are ca intrare un vector $A[1..n]$ şi produce la ieşire o permutare a elementelor de intrare.

8.4. Analiza algoritmului de sortare rapidă

Rezultatul studiului intuitiv din secţiunea 8.2 ne face să credem că algoritmul de sortare rapidă are o comportare bună. În această secţiune vom face o analiză mai riguroasă. Începem cu analiza cazului celui mai defavorabil, atât pentru algoritmul QUICKSORT cât şi pentru algoritmul QUICKSORT-ALEATOR, şi terminăm cu analiza cazului mediu pentru algoritmul QUICKSORT-ALEATOR.

8.4.1. Analiza celui mai defavorabil caz

Am văzut, în secţiunea 8.2, că împărţirea cea mai rea la fiecare nivel ne conduce la timpul de execuţie al algoritmului $\Theta(n^2)$, care, intuitiv, reprezintă comportarea în cazul cel mai defavorabil. În continuare vom demonstra această afirmaţie.

Folosind metoda substituţiei (vezi secţiunea 4.1), se poate demonstra că timpul de execuţie al algoritmului QUICKSORT este $O(n^2)$. Fie $T(n)$ timpul de execuţie al algoritmului de sortare

rapidă, în cazul cel mai defavorabil, pentru o intrare de dimensiunea n. Avem următoarea formulă de recurenţă

$$T(n) = \max_{1 \le q \le n-1} (T(q) + T(n-q)) + \Theta(n), \tag{8.1}$$

unde parametrul q parcurge valorile între 1 şi $n-1$, deoarece procedura PARTIŢIE produce două subşiruri, având fiecare, cel puţin un element. Presupunem că $T(n) \le cn^2$, pentru o constantă c. Înlocuind în formula (8.1), obţinem următoarele:

$$T(n) \le \max_{1 \le q \le n-1} (cq^2 + c(n-q)^2) + \Theta(n) = c \cdot \max_{1 \le q \le n-1} (q^2 + (n-q)^2) + \Theta(n).$$

Expresia $q^2 + (n-q)^2$ atinge valoarea sa maximă în domeniul de valori $1 \le q \le n-1$ la una din extreme, deoarece derivata a doua, după q a expresiei, este pozitivă (vezi exerciţiul 8.4-2). Deci, avem

$$\max_{1 \le q \le n-1} (q^2 + (n-q)^2) \le 1^2 + (n-1)^2 = n^2 - 2(n-1).$$

Continuând majorarea lui $T(n)$, obţinem:

$$T(n) \le cn^2 - 2c(n-1) + \Theta(n) \le cn^2,$$

deoarece constanta c poate fi aleasă suficient de mare, astfel încât termenul $2c(n-1)$ să domine termenul $\Theta(n)$. Rezultă că timpul de execuţie (cel mai defavorabil) al algoritmului de sortare rapidă este $\Theta(n^2)$.

8.4.2. Analiza cazului mediu

Am explicat deja, intuitiv, de ce timpul mediu de execuţie al algoritmului QUICKSORT-ALEATOR este $\Theta(n \lg n)$: dacă procedura PARTIŢIE-ALEATOARE împarte vectorul, la fiecare nivel, în aceeaşi proporţie, atunci adâncimea arborelui de recurenţă este $\Theta(\lg n)$, şi timpul de execuţie la fiecare nivel este $\Theta(n)$. Pentru a putea analiza mai exact timpul mediu de execuţie al algoritmului QUICKSORT-ALEATOR, va trebui să înţelegem esenţa procedurii de partiţionare. În continuare, vom putea găsi o formulă de recurenţă pentru timpul mediu de execuţie, necesar pentru sortarea unui vector de dimensiune n. Ca parte a procesului de rezolvare a recurenţei, vom găsi margini tari pentru o sumă interesantă.

Analiza partiţionării

La început, facem câteva observaţii asupra procedurii PARTIŢIE. Când în linia 3 a procedurii PARTIŢIE-ALEATOARE este apelată procedura PARTIŢIE, elementul $A[p]$ este deja interschimbat cu un element ales aleator din vectorul $A[p..r]$. Pentru simplificare, să presupunem că toate elementele de intrare sunt distincte. Dacă elementele de intrare nu sunt distincte, timpul mediu de execuţie al algoritmului rămâne tot $O(n \lg n)$, dar acest caz necesită o analiză ceva mai complexă.

Prima noastră observaţie este că valoarea q, returnată de procedura PARTIŢIE, depinde numai de rangul elmentului $x = A[p]$ în vectorul $A[p..r]$. (***Rangul*** unui element într-o mulţime este egal cu numărul elementelor care sunt mai mici sau egale cu el.) Dacă notăm cu $n = r - p + 1$ numărul elementelor vectorului $A[p..r]$, şi interschimbăm elementul $x = A[p]$ cu un element ales

aleator din vectorul $A[p..r]$, probabilitatea ca rang$(x) = i$ (unde $i = 1, 2, \ldots, n$) este egală cu $1/n$.

Vom calcula, în continuare, probabilitatea diferitelor rezultate ale procedurii de partiţionare. Dacă rang$(x) = 1$, atunci, în procedura PARTIŢIE, la prima execuţie a ciclului **cât timp** din liniile 4–14, variabilele i şi j vor primi aceeaşi valoare p. Deci, când procedura returnează valoarea $q = j$, în primul subşir, singurul element va fi $A[p]$. Probabilitatea acestui caz este $1/n$, deoarece aceasta este probabiltatea ca rang(x) să fie egal cu 1.

Dacă rang$(x) \geq 2$, atunci, există cel puţin un element care este mai mic decât $x = A[p]$. În consecinţă, la prima trecere prin ciclul **cât timp** variabila i va avea valoarea egală cu p, iar j nu va atinge valoarea p. Deci, prin interschimbarea a două elemente între ele, elementul $A[p]$ ajunge în al doilea subşir. Când procedura PARTIŢIE se termină, fiecare dintre cele rang$(x) - 1$ elemente, care se află în primul subşir, vor fi mai mici sau egale cu x. Deci, când rang$(x) \geq 2$, probabilitatea ca în primul subşir să fie i elemente (pentru orice $i = 1, 2, \ldots, n-1$) este egală cu $1/n$.

Combinând cele două cazuri, putem trage concluzia că dimensiunea $q-p+1$, a primului subşir, este egală cu 1, cu probabilitatea $2/n$, iar această dimensiune este egală cu i ($i = 2, 3, \ldots, n-1$), cu probabilitatea $1/n$.

O relaţie de recurenţă pentru cazul mediu

Vom da în continuare o formulă de recurenţă pentru timpul mediu de execuţie al algoritmului QUICKSORT-ALEATOR. Fie $T(n)$ timpul mediu necesar ordonării unui vector având n elemente. Un vector de 1 element poate fi ordonat cu procedura QUICKSORT-ALEATOR în timp constant, deci $T(1) = \Theta(1)$. Procedura QUICKSORT-ALEATOR partiţionează vectorul $A[1..n]$ de dimensiune n în $\Theta(n)$ unităţi de timp. Procedura PARTIŢIE returnează un indice q, iar QUICKSORT-ALEATOR este apelat recursiv pentru un vector de q elemente şi pentru unul cu $n-q$ elemente. Deci timpul mediu pentru ordonarea unui vector, având n elemente, se poate exprima astfel:

$$T(n) = \frac{1}{n}\left(T(1) + T(n-1) + \sum_{q=1}^{n-1}(T(q) + T(n-q))\right) + \Theta(n). \tag{8.2}$$

Distribuţia valorii q este aproape uniformă, exceptând cazul $q = 1$ care este de două ori mai probabil decât celelalte valori, cum am văzut anterior. Folosind valorile $T(1) = \Theta(n)$ şi $T(n-1) = O(n^2)$ (din analiza celui mai defavorabil caz), se poate scrie:

$$\frac{1}{n}(T(1) + T(n-1)) = \frac{1}{n}(\Theta(1) + O(n^2)) = O(n),$$

şi astfel termenul $\Theta(n)$, în formula (8.2), poate absorbi expresia $\frac{1}{n}(T(1) + T(n-1))$. Astfel, formula de recurenţă (8.2) poate fi scrisă sub forma:

$$T(n) = \frac{1}{n}\sum_{q=1}^{n-1}(T(q) + T(n-q)) + \Theta(n). \tag{8.3}$$

Se poate observa că, pentru $k = 1, 2, \ldots, n-1$, fiecare termen $T(k)$ al sumei apare de două ori, o dată ca $T(q)$, iar altă dată ca $T(n-q)$. Reducând aceşti termeni, formula finală va fi:

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1}T(k) + \Theta(n). \tag{8.4}$$

Rezolvarea ecuaţiei de recurenţă

Vom putea rezolva ecuaţia de recurenţă (8.4) folosind metoda substituţiei. Să presupunem că $T(n) \leq an \lg n + b$, pentru un $a > 0$ şi $b > 0$. Valorile a şi b pot fi alese destul de mari, astfel încât $an \lg n + b$ să fie mai mare decât $T(1)$. Pentru $n > 1$ avem:

$$\begin{aligned} T(n) &= \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n) \leq \frac{2}{n}\sum_{k=1}^{n-1}(ak \lg k + b) + \Theta(n) = \\ &= \frac{2a}{n}\sum_{k=1}^{n-1} k \lg k + \frac{2b}{n}(n-1) + \Theta(n). \end{aligned}$$

Vom arăta, mai târziu, că ultima sumă poate fi majorată astfel:

$$\sum_{k=1}^{n-1} k \lg k \leq \frac{1}{2}n^2 \lg n - \frac{1}{8}n^2. \tag{8.5}$$

Folosind această majorare, se obţine:

$$\begin{aligned} T(n) &\leq \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \frac{2b}{n}(n-1) + \Theta(n) \leq \\ &\leq an \lg n - \frac{a}{4}n + 2b + \Theta(n) = an \lg n + b + \left(\Theta(n) + b - \frac{a}{4}n\right) \leq an \lg n + b, \end{aligned}$$

deoarece valoarea lui a poate fi aleasă astfel încât $\frac{a}{4}n$ să domine expresia $\Theta(n) + b$. Se poate deci trage concluzia că timpul mediu de execuţie a algoritmului de sortare rapidă este $O(n \lg n)$.

Margini strânse pentru însumarea cu chei

Trebuie să mai demonstrăm marginea (8.5) a sumei $\sum_{k=1}^{n-1} k \lg k$.

Deoarece fiecare termen al sumei este cel mult $n \lg n$, se poate scrie:

$$\sum_{k=1}^{n-1} k \lg k \leq n^2 \lg n,$$

formulă care reprezintă o estimare destul de precisă, abstracţie făcând de o constantă, dar nu este suficient de puternică pentru a obţine $T(n) = O(n \lg n)$ ca soluţie a recurenţei. Pentru a obţine soluţia de mai sus, avem nevoie de o margine de forma $\frac{1}{2}n^2 \lg n - \Omega(n^2)$.

Această margine se poate obţine folosind metoda de împărţire a sumei în două, ca în secţiunea 3.2. Vom obţine:

$$\sum_{k=1}^{n-1} k \lg k = \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg k.$$

Termenul $\lg k$ din prima sumă a membrului drept se poate majora prin $\lg(n/2) = \lg n - 1$, iar termenul $\lg k$ din cea de a doua sumă prin $\lg n$. Astfel se obţine

$$\sum_{k=1}^{n-1} k \lg k \leq (\lg n - 1)\sum_{k=1}^{\lceil n/2 \rceil - 1} k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k = \lg n \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k \leq$$

$$\leq \frac{1}{2}n(n-1)\lg n - \frac{1}{2}\left(\frac{n}{2}-1\right)\frac{n}{2} \leq \frac{1}{2}n^2\lg n - \frac{1}{8}n^2$$

dacă $n \geq 2$. Deci am obţinut chiar formula (8.5).

Exerciţii

8.4-1 Demonstraţi că timpul de execuţie al algoritmului de sortare rapidă, în cazul cel mai favorabil este $\Omega(n \lg n)$.

8.4-2 Demonstraţi că expresia $q^2 + (n-q)^2$, unde $q = 1, 2, \ldots, n-1$, atinge maximul pentru $q = 1$ sau $q = n-1$.

8.4-3 Demonstraţi că timpul mediu de execuţie al lui QUICKSORT-ALEATOR este $\Omega(n \lg n)$.

8.4-4 În practică, timpul de execuţie al algoritmului de sortare rapidă se poate îmbunătăţi considerabil, ţinând cont de execuţia mai rapidă a algoritmului de sortare prin inserţie pentru date de intrare "aproape" ordonate. Când algoritmul de sortare rapidă este apelat pentru un subşir având mai puţin de k elemente, impunem ca acesta să returneze vectorul fără a-l sorta. După revenirea din apelul iniţial, să se execute algoritmul de sortare prin inserţie pentru întregul vector (care este "aproape" ordonat). Argumentaţi afirmaţia că acest algoritm de sortare are un timp mediu de execuţie $O(nk + n\lg(n/k))$. Cum trebuie aleasă valoarea lui k, teoretic şi practic?

8.4-5 $\star$ Demonstraţi următoarea identitate:

$$\int x \ln x dx = \frac{1}{2}x^2 \ln x - \frac{1}{4}x^2,$$

şi, folosind metoda de calculul aproximativ al integralei, daţi o margine mai bună decât cea din (8.5) pentru $\sum_{k=1}^{n-1} k \lg k$.

8.4-6 $\star$ Modificaţi procedura PARTIŢIE, alegând aleator trei elemente din vectorul A, iar dintre acestea elementul de mijloc ca valoare pivot. Aproximaţi probabilitatea ca, în cazul cel mai defavorabil, să se obţină o partiţionare în proporţie de α la $(1-\alpha)$, unde $0 < \alpha < 1$.

Probleme

8-1 Corectitudinea partiţiei

Argumentaţi afirmaţia că procedura PARTIŢIE din secţiunea 8.1 este corectă. Demonstraţi următoarele:

- ***a.*** Indicii i şi j nu se referă niciodată la elemente ale lui A din afara intervalului $[p..r]$.
- ***b.*** La terminarea procedurii PARTIŢIE, indicele j nu va fi niciodată egal cu r (astfel, partiţionarea nu este niciodată trivială).
- ***c.*** La terminarea partiţionării fiecare element din $A[p..j]$ este mai mic sau egal cu orice element din $A[j+1..r]$.

8-2 Algoritmul de partiţionare a lui Lomuto
Considerăm următoarea variantă a procedurii PARTIŢIE, dată de N. Lomuto. Vectorul $A[p..r]$ se împarte în vectorii $A[p..i]$ şi $A[i+1..j]$ astfel încât fiecare element din primul vector este mai mic sau egal cu $x = A[r]$ şi fiecare element din cel de al doilea vector este mai mare ca x.

```
PARTIŢIE-LOMUTO(A, p, r)
 1: x ← A[r]
 2: i ← p − 1
 3: pentru j ← p, r execută
 4:    dacă A[j] ≤ x atunci
 5:       i ← i + 1
 6:       interschimbă A[i] ↔ A[j]
 7: dacă i < r atunci
 8:    returnează i
 9: altfel
10:    returnează i − 1
```

a. Argumentaţi că procedura PARTIŢIE-LOMUTO este corectă.

b. Cel mult, de câte ori poate fi mutat un element în procedura PARTIŢIE? Dar în procedura PARTIŢIE-LOMUTO?

c. Argumentaţi că timpul de execuţie al procedurii PARTIŢIE-LOMUTO, ca de altfel şi al procedurii PARTIŢIE, pe un subşir de n elemente, este $\Theta(n)$.

d. Cum afectează, înlocuirea procedurii PARTIŢIE cu procedura PARTIŢIE-LOMUTO, timpul de execuţie al algoritmului de sortare rapidă QUICKSORT când toate datele de intrare sunt egale?

e. Definiţi o procedură PARTIŢIE-LOMUTO-ALEATOARE care interschimbă elementul $A[r]$ cu un element ales aleator din vectorul $A[p..r]$, apoi apelează procedura PARTIŢIE-LOMUTO. Demonstraţi că probabilitatea ca procedura PARTIŢIE-LOMUTO-ALEATOARE să returneze valoarea q, este aceeaşi cu probabilitatea ca procedura PARTIŢIE-ALEATOARE să returneze valoarea $p + r - q$.

8-3 Sortare circulară
Profesorii Howard, Fine şi Howard au propus următorul algoritm "elegant" pentru sortare:

```
SORTEAZĂ-CIRCULAR(A, i, j)
 1: dacă A[i] > A[j] atunci
 2:    interschimbă A[i] ↔ A[j]
 3: dacă i + 1 ≥ j atunci
 4:    revenire
 5: k ← ⌊(j − i + 1)/3⌋                ▷ Trunchiere.
 6: SORTEAZĂ-CIRCULAR(A, i, j − k)     ▷ Primele două-treimi.
 7: SORTEAZĂ-CIRCULAR(A, i + k, j)     ▷ Ultimele două-treimi.
 8: SORTEAZĂ-CIRCULAR(A, i, j − k)     ▷ Primele două-treimi din nou.
```

a. Demonstraţi că SORTEAZĂ-CIRCULAR($A, 1,$*lungime*$[A]$) ordonează corect vectorul de intrare $A[1..n]$, unde $n =$ *lungime*$[A]$.

b. Determinaţi o formulă de recurenţă, pentru timpul de execuţie al procedurii SORTEAZĂ-CIRCULAR, şi o margine asimptotică tare (folosind notaţia Θ), pentru timpul de execuţie, în cazul cel mai defavorabil.

c. Comparaţi timpul de execuţie al algoritmului SORTEAZĂ-CIRCULAR, în cazul cel mai defavorabil, cu timpii corespunzători ai algoritmilor de sortare prin inserare, sortare prin interclasare, heapsort şi sortare rapidă. Merită profesorii să fie felicitaţi?

8-4 Adâncimea stivei pentru algoritmul de sortare rapidă

Algoritmul QUICKSORT, prezentat în secţiunea 8.1, se autoapelează de două ori. După apelul procedurii PARTIŢIE, este sortat recursiv primul subşir, apoi subşirul al doilea. Al doilea apel recursiv din QUICKSORT nu este neapărat necesar, el poate fi evitat prin folosirea unei structuri de control iterative. Această tehnică, numită ***recursivitate de coadă***, este oferită în mod automat de compilatoarele mai bune. Se consideră următoarea variantă a algoritmului de sortare rapidă, care simulează recursivitatea de coadă:

QUICKSORT$'(A, p, r)$
1: **cât timp** $p < r$ **execută**
2: $\triangleright$ Partiţionarea şi sortarea primului subşir
3: $q \leftarrow$ PARTIŢIE(A, p, r)
4: QUICKSORT$'(A, p, q)$
5: $p \leftarrow q + 1$

a. Arătaţi că algoritmul QUICKSORT$'(A, 1,$*lungime*$[A])$ sortează corect vectorul A.

Compilatoarele, de obicei, execută apelurile recursive, folosind o ***stivă*** care conţine informaţii pertinente, inclusiv valorile parametrilor la fiecare apel. Informaţiile referitoare la cel mai recent apel se găsesc în vârful stivei, pe când informaţiile referitoare la primul apel se păstrează la baza stivei. Când se apelează o procedură, informaţiile sale se depun în vârful stivei (operaţia ***push***), iar la terminarea execuţiei, aceste informaţii se scot din stivă (operaţia ***pop***). Deoarece presupunem că parametrii vectorului se reprezintă cu ajutorul pointerilor, informaţiile necesare fiecărui apel necesită un spaţiu de stivă de $O(1)$ unităţi. ***Adâncimea stivei*** este spaţiul maxim utilizat de stivă, în timpul execuţiei algoritmului.

b. Descrieţi un scenariu în care, pentru un vector de intrare de dimensiune n, adâncimea stivei algoritmului QUICKSORT$'$ este $\Theta(n)$.

c. Modificaţi textul algoritmului QUICKSORT$'$ astfel încât adâncimea stivei, în cazul cel mai defavorabil, să fie $\Theta(\lg n)$. Păstraţi valoarea $O(n \lg n)$ pentru timpul mediu de execuţie a algoritmului.

8-5 Partiţionare mijlocul-din-3

O metodă prin care procedura QUICKSORT-ALEATOR poate fi îmbunătăţită este de a face partiţionarea în jurul unui element x, ales mai atent decât aleator. O metodă obişnuită poate fi metoda ***mijlocul-din-3***, care constă în a alege elementul mijlociu ca mărime, dintre trei elemente

alese la întâmplare din vector. Să considerăm, pentru problema de faţă, că toate elementele vectorului de intrare $A[1..n]$ sunt distincte şi că $n \geq 3$. Să notăm cu $A'[1..n]$ vectorul sortat rezultat. Fie $p_i = \Pr\{x = A'[i]\}$ probabilitatea ca elementul pivot x, ales cu metoda mijlocul-din-3, să fie al i-lea element în vectorul de ieşire.

a. Daţi o formulă exactă pentru p_i în funcţie de n şi i, pentru $i = 2, 3, \ldots, n-1$. (Să observăm că $p_1 = p_n = 0$.)

b. Cu cât creşte probabilitatea alegerii ca pivot a elementului $x = A'[\lfloor (n+1)/2 \rfloor]$, mijlocul vectorului $A[1..n]$ faţă de implementarea originală? Calculaţi limita acestei probabilităţi când $n \to \infty$.

c. Dacă numim partiţionare "bună" acea partiţionare care foloseşte ca pivot elementul $x = A'[i]$, unde $n/3 \leq i \leq 2n/3$, atunci cu cât creşte probabilitatea obţinerii unei împărţiri bune faţă de implementarea originală? (*Indicaţie:* Aproximaţi suma cu o integrală.)

d. Arătaţi că metoda mijlocul-din-3 afectează numai factorul constant din expresia $\Omega(n \lg n)$ a timpului de execuţie al algoritmului de sortare rapidă.

Note bibliografice

Algoritmul de sortare rapidă a fost inventat de Hoare [98]. Sedgewick [174] oferă o prezentare bună a detaliilor de implementare a acestui algoritm. Avantajele algoritmilor aleatori au fost tratate de Rabin [165].