5-MA’RUZA. ALGORITMLAR TAHLILI. ALGORITMLAR SAMARADORLIGINI BAHOLASH

Tahlil nima? Algoritm tahlilini, qo‘yilgan masalani ushbu algoritm bilan yechish qancha vaqt talab qilishi deb tasavvur qilish mumkin. Har bir qaralayotgan algorimtni N o‘lchovli boshlang‘ich ma’lumotlar massividagi masalalaning qanchalik tez yechilishi bilan baholaymiz.

Masalan, saralash algoritmi N ta qiymatdan iborat ro‘yxatni o‘sish tartibida joylashtirish uchun qancha taqqoslash talab qiladi yoki N*N o‘lchamli ikkita matritsani ko‘paytirishda qancha arifmetik amallar zarurligini hisoblash.

Bitta masalani turli algoritmlar bilan yechish mumkin. Algoritmlar tahlili bizga algoritmni tanlash uchun qurol bo‘ladi. To‘rtta qiymatdan eng kattasini tanlaydigan ikkita algoritmni qaraymiz:

```
largest = a
if b > largest then
largest = b
end if
return a
if s > largest then
largest = s end if if d > largest then
largest = d end if return largest
if a > b then if a > s then I f a > d then
return a else
return d end if else
if s > d then
return s else
return d end if end if else
if b > s then if b > d then
return b else
return d end if else
if s > d then
return s else
```

```
return d end if end if end if
```

Ko‘rinib turibdiki, qaralayotgan algoritmlarning har birida uchta taqqoslash bajariladi. Birinchi algoritmni o‘qish va tushunish oson, ammo kompyuterda bajarilish nuqtai nazaridan ularning murakkablik darajalari teng. Bu ikki algoritm vaqt nuqtai nazaridan teng, lekin birinchi algoritm largest nomli qo‘shimcha o‘zgaruvchi hisobiga ko‘proq xotira talab qiladi. Agarda son yoki belgilar taqqoslansa, ushbu qo‘shimcha o‘zgaruvchi katta ahamiyatga ega bo‘lmaydi, lekin boshqa turdagi ma’lumotlar bilan ishlaganda bu muhim ahamiyatga ega. Ko‘plab zamonaviy dasturlash tillari katta va murakkab obyektlarni yoki yozuvlarni taqqoslash operatorlarini aniqlash imkonini beradi. Bunday hollarda qo‘shimcha o‘zgaruvchilarni joylashtirish katta joy talab qiladi. Algoritmlarning effektivligini tahlil qilishda bizni birinchi navbatda vaqt masalasi qiziqtiradi, ammo xotira muhim rol o‘ynaydigan vaziyatda uni ham muhokama qilamiz.

Algoritmlaring turli xossalari bitta masalani yechuvchi ikki turdagi algoritmlarning effektivligini taqqoslash uchun xizmat qiladi. Biz shuning uchun hech qachon matritsalarni ko‘paytirish algoritmi bilan saralash algoritmini emas, balki ikkita turli saralash algoritmlarini bir-biri bilan taqqoslaymiz.

Algoritm tahlilining natijasi – belgilangan algoritmning kompyuterdan qancha vaqt yoki takrorlash talab qilishini aniq hisoblovchi formula emas.

Bunday ma’lumot muhim emas, bu holatda kompyuter turi, u bitta yoki undan ortiq foydalanuvchi tomonidan ishlatilyaptimi, uning protsessori va chastotasi qanaqa, protsessor chipida komandalar to‘liqmi va kompilyator bajarilayotgan kodni qay darajada amalga oshirmoqda kabi tomonlarni nazarda tutish kerak.

Bu shartlar algoritm bajarilish natijasida dasturning ishlash tezligiga ta’sir qiladi. Yuqoridagi shartlar hisobiga dasturni boshqa tez ishlaydigan kompyuterga o‘tkazilganda algoritm yaxshi ishlaganday bajarilishi tezroq amalga oshadi. Aslida esa unday emas, biz shuning uchun tahlilimizda kompyuterning imkoniyatlarini inobatga olmaymiz.

Oddiy va katta bo‘lmagan dasturlarda bajariladigan amallar sonini N ning funksiyasi ko‘rinishida aniq hisoblash mumkin. Biz algoritmlarni bajariladigan amallar soniga qarab tahlil qilamiz.

Algoritm tomonidan bajariladigan jarayonlar borki, biz ularning hammasini hisoblab o‘tirmaymiz, buning sababi shundaki, hatto uning eng kichik sozlashi ham samaradorlikning sezilmas yaxshilanishiga olib keladi. Fayldagi turli belgilar sonini hisoblovchi algoritmni qaraymiz. Bu masala yechimi uchun algoritmning taxminiy ko‘rinishi quyidagicha bo‘ladi:

```
    for all 256 belgilarni do
    hisoblagichni nolga tenglash end for
    while agar faylda belgi qolsa do
    navbatdagi belgini ko‘rsat va hisoblagichni bittaga oshir end while
    do
    hisoblagichni nolga tenglashtirish end for
    while faylda belgi mavjud bo‘lsa do
    navbatdagi belgini ko‘rsat va hisoblagichni bittaga oshir
    end while
```

Ushbu algoritmni ko‘rib chiqamiz. U takrorlanish bajarilishida 256 ta o‘tish qiladi. Agar berilgan faylda N ta belgi bo‘lsa unda ikkinchi takrorlanishda *N* ta o‘tish qilinadi. “Bu qanday hisoblash?” degan savol tug‘iladi. **For** siklida avval sikl o‘zgaruvchisi bajariladi, keyin xar bir o‘tishda uning sikl chegarasidan chiqmayotganligi tekshiriladi va o‘zgaruvchi qiymatini oshiradi. Bu esa sikl bajarilishida 257 yuklash bajariladi (biri sikl o‘zgaruvchisi, 256 tasi hisoblagich uchun), ya’ni 256 ta oshirish va 257 ta sikl chegarasidan chiqmaganligini tekshirish (bitta amal siklni to‘xtatish uchun qo‘shilgan). Ikkinchi siklda *N* + 1 marta shart tekshiriladi (+1 fayl bo‘sh bo‘lgandagi oxirgi tekshiruv), va *N* hisoblagichni oshirish. Jami amallar:

Oshirish	N + 256

Yuklash	257
Shartlarni tekshirish	N + 258

Shunday qilib 500 belgidan iborat fayl berilsa algoritmda 1771 ta amal bajariladi, ulardan 770 tasi natija beradi (43%). Endi N ning qiymati oshganda nima bo‘lishini ko‘ramiz. Agar fayl 50 000 belgidan iborat bo‘lsa, unda algoritm 100 771 amal bajaradi, ularning 770 tasi natija uchun (jami amallar sonining 1% ini tashkil etadi). yechimga qaratilgan amallar soni oshmayapti, lekin *N* katta bo‘lganda ularning foizi juda kam.

Endi boshqa tomoniga e’tibor qaratamiz. Kompyuterda ma’lumotlar bilan shunday ishlashga mo‘ljallanganki, katta xajmdagi ma’lumotlar blokini ko‘chirish va yuklash bir xil tezlikda amalga oshiriladi. Shuning uchun, biz avval 16 ta hisoblagichga boshlang‘ich qiymat 0 ni yuklaymiz, keyin qolgan hisoblagichlarni to‘ldirish uchun shu blokdan 15 ta nusxa olamiz. Bu esa sikl bajarilish davomida tekshirishlar sonini 33 ga, yuklashlar sonini 33 va oshirishlar sonini 31 ga kamayishiga olib keladi. Demak, amal bajarilishlar soni 770 dan 97 gacha kamaydi, ya’ni 87%. Agar erishilgan natijani 50000 belgidan iborat fayl ustida bajarsak, tejamkorlik 0.7% ni tashkil qiladi (100771 ta amal o‘rniga 100098 amal bajaramiz).

Agarda barcha amallarni sikldan foydalanmay 31 ta yuklashlar orqali bajarganimizda, vaqtni yanada tejagan bo‘lardik, ammo bu usul 0.07 foyda keltiradi. Ishimiz unumli bo‘lmaydi.

Ko‘rib turganimizdek, algoritmning bajarilish vaqti bilan bog‘liq barcha amallar befoyda. Tahlil tili bilan aytganda, boshlang‘ich ma’lumotlar hajmining ortishiga alohida e’tibor qaratish kerak.

Avvalgi ishlarda algoritmlarni tahlil qilishda algoritmlarni Tyuring mashinasida hisoblash aniqlangan. Tahlilda masalani yechish uchun zarur bo‘lgan o‘tishlar soni hisoblangan. Bu turdagi tahlil to‘g‘ri bo‘lib, ikki algoritmning nisbiy tezliklarini aniqlash imkonini beradi, ammo uning amaliyotda qo‘llanilishi kam, chunki ko‘p vaqt talab qiladi.

Avval bajariladigan algoritmning Tyuring mashinasidagi o‘tish funksiyalarini yozish, keyin esa bajarilish vaqti hisoblanadi.

Har doim biz bitta aniq algoritmni ko‘rib chiqishimizga to‘g‘ri keladi – unda birdan ortiq funksiya yoki programma fragmenti kiritilgan bo‘ladi, shuning uchun yuqorida keltirilgan tillarning tezligi umuman muhim emas. Psevdokodlardan foydalanishimizning sababi shunda.

Ko‘plab dasturlash tillarida mantiqiy ifodaning qiymatlari qisqartirilgan shaklda hisoblanadi. Bu A and V ifodadagi V hadning qiymati qachonki A rost bo‘lsagina hisoblanadi, aks holda natija V ga bog‘liq bo‘lmagan tarzda yolg‘on bo‘ladi. Xuddi shunday A or V ifodada A ning qiymati rost bo‘lsa, V hadning qiymati hisoblanmaydi. Ko‘rinib turibdiki, murakkab shartlarning 1 yoki 2 ga tengligidagi taqqoslashlarining sonini hisoblash shart emas. SHuning uchun bu bobni o‘rganishda mantiqiy ifodalarning qiymatini qisqartirib hisoblanishini e’tiborsiz qoldiramiz.

Kiruvchi berilganlar sinfi. Algoritmlarning tahlilida kiruvchi ma’lumotlarning roli yuqori, chunki algoritm harakatlarining ketma-ketligi kiruvchi ma’lumotlar bilan belgilanadi. Masalan, *N* ta elementdan tashkil topgan ro‘yxatning eng katta elementini topish uchun quyidagi algoritmdan foydalanish mumkin:

```
largest = list [1]
for i = 2 to N do
if (list [i] > largest) then
 largest = list[i]
end if
end for
```

Agar ro‘yxat kamayish tartibida bo‘lsa, u holda sikl boshlanishidan avval bitta o‘zlashtirish bajariladi, sikl tanasida esa o‘zlashtirish bo‘lmaydi. Agar ro‘yxat o‘sish tartibida bo‘lsa, u holda N ta o‘zlashtirish bajariladi (sikl boshlanishidan avval bitta va N-1 ta siklda). Biz tahlil qilish davomida kiruvchi qiymatlar to‘plamining turli imkoniyatlarini ko‘rib chiqishimiz kerak, agar bitta to‘plam bilan chegaralansak, bu yechim eng tez (yoki eng sekin) bo‘lgan to‘plam bo‘lib chiqishi mumkin. Natijada

biz algoritm haqidagi yolg‘on tasavvurga ega bo‘lamiz. Buning o‘rniga kiruvchi to‘plamlar turining barchasini ko‘rib chiqamiz.

Biz kiruvchi to‘plamlarni har bir to‘plamdagi algoritm holatiga bog‘liq holda sinflarga bo‘lib chiqamiz. Bunday bo‘linish ko‘rib chiqilayotgan to‘plamlar miqdorini kamaytirish imkonini beradi. Masalan, 10 ta sondan iborat ro‘yxat uchun eng katta elementni topish algoritmini qo‘llaymiz. Birinchi soni eng katta bo‘lgan 362 880 kiruvchi to‘plamlar mavjud, ularni bitta sinfga joylashtirish mumkin. Agar qiymati bo‘yicha eng katta son ikkinchi o‘rinda turgan bo‘lsa, u holda algoritm ikkita o‘zlashtirishni amalga oshiradi. Eng kata son ikkinchi o‘rinda turgan to‘plam 362880. Ularni boshqa sinfga kiritish mumkin. 1 dan N gacha bo‘lgan sonlar orasida eng katta sonning o‘zgarish holida o‘zlashtirishlar sonining qanday o‘zgarishini ko‘rishimiz mumkin.

Shunday qilib, barcha kiruvchi to‘plamlarni bajarilgan o‘zlashtirishlar soni bo‘yicha N ta turli sinfga bo‘lish kerak. Ko‘rib turganingizdek, har bir sinfda joylashgan to‘plamlarni birma-bir yozish yoki yozib olish shart emas. Faqatgina har bir to‘plamdagi sinflar miqdori va ish hajmini bilish yetarli.

Kiruvchi berilganlarning mumkin bo‘lgan to‘plami N kattalashganda judayam katta bo‘lishi mumkin. Masalan, 10 ta turli soni ro‘yxatda 3 628 800 usulda joylashtirish mumkin. Bu usullarning barchasini ko‘rib chiqishning imkoni yo‘q. Biz buning o‘rniga algoritm bajarilishiga ko‘ra ro‘yxatni sinflarga bo‘lamiz. Yuqorida ko‘rsatilgan algoritm uchun bo‘linish eng katta qiymatning joylashishi o‘rniga asoslanadi. Natijada 10 ta turli sinf hosil bo‘ladi. Boshqa algoritm uchun, masalan, eng katta va eng kichik sonni topish algoritmida bo‘linish eng katta va eng kichik sonning joylashuviga asoslanadi. Bunday bo‘linishda 90 ta sinf bo‘ladi. Sinflarni ajratib bo‘lgach, har bir sinfdan bitta to‘plamda algoritm holatini ko‘rish mumkin. Agar sinflar to‘g‘ri tanlangan bo‘lsa, u holda bir sinfdagi kiruvchi berilganlar to‘plamida algoritm bir xil miqdordagi amallarni bajaradi, boshqa sinfning to‘plamlari uchun esa amallar miqdori boshqacha bo‘ladi.

Xotira bo‘yicha murakkablik. Biz asosan algoritmlarning vaqt bo‘yicha murakkabligini muhokama qilamiz, ammo ish bajarish uchun u yoki bu algoritmga

qancha xotira kerakligi haqida ham aytish mumkin. Kompyuter xotirasi (ham ichki, ham tashqi) hajmi chegaralangan. Kompyuterlar rivojlanishining dastlabki bosqichlarida bu tahlil uslubiy xarakterga ega edi. Barcha algoritmlar chegaralangan xotira yetarli yoki qo‘shimcha maydonni talab qiluvchi algoritmlarga bo‘linadi. Ko‘pincha dasturlovchilar xotirasiga ega va tashqi qurilmalar talab qilmaydigan sekin ishlovchi algoritmni tanlashar edi.

Kompyuter xotirasiga bo‘lgan talab juda katta edi, shuning uchun qaysi ma’lumotlar saqlanib qoladi, bunday saqlashning samarali usullari qanday kabi savollar o‘rganilar edi. Faraz qilaylik, masalan, biz -10 dan +10 gacha intervaldagi verguldan keyin bitta o‘nli belgiga ega bo‘lgan moddiy son yozyapmiz. Moddiy soni yozishda ko‘pchilik kompyuterlar 4 dan 8 baytgacha xotira sarflaydi, leki nagar bu soni avvaldan 10 ga ko‘paytirsak, -100 dan +100 gacha intervaldagi butun son hosil qilamiz va uni saqlash uchun bor yo‘g‘i bir bayt sarflanadi. Birinchi variant bilan solishtirsak, 3-7 bayt tejashga erishildi. 1000 ta shunday son saqlaydigan dastur 3000 dan 7000 baytgacha tejaydi. Agar o‘tgan asrning 80-yillarida kompyuterlarning xotirasi 65536 bayt bo‘lganligini e’tiborga olsak, jiddiy tejash ko‘zga tashlanadi. Aynan shu kompyuter dasturlarining uzoq yil ishlashi xotirani tejash zaruriyati Bilan bir qatorda 2000-yil muammo tug‘dirdi. Agar sizning dasturingiz turli sanalardan foydalansa, yilni yozish uchun 1999 o‘rniga 99 ifodasini saqlagan holda joyning yarmini tejasa bo‘ladi. 80-yillardagi dastur mualliflari mahsulotlari 2000-yilgacha yashashini taxmin ham qilishmagan edi. Hozirgi kunda bozorlarda taklif qilinayotgan dasturiy ta’minotga nazar tashlasak, xotiraning bunday tahlili o‘tkazilmaganligi ayon bo‘ladi. Oddiy dasturlar uchun zarur xotira hajmi megabaytlarda o‘lchanadi. Dastur tuzuvchilar joyni tejash ehtiyojini his qilmayotganga o‘xshaydilar, ularning fikricha, agar foydalanuvchida yetarli xotira bo‘lmasa, u dastur bajarilishi uchun yetmayotgan 32 yoki undan ortiq megabayt xotira yoki uni saqlash uchun yangi qattiq disk sotib oladi. Natijada kompyuterlar o‘zining belgilangan muddatidan avval yaroqsiz holga kelib qoladi.

Yaqinda tarqalgan cho‘ntak kompyuterlari (PDA – personal digital assistant) yangi ohang olib kirdi. Bunday qurilmaning xotirasi ham ma’lumotlar, ham dasturlar

uchun 2 dan 8 megabaytgacha. Shuning uchun ham ma’lumotlarni ixcham saqlashni ta’minlovchi kichik dasturlarni yaratish qiyin bo‘lib qolmoqda.

Nimani hisoblash va nimani inobatga olish lozim. Nimani hisoblash masalasi ikkita qadamdan iborat. Birinchi qadamda ahamiyatli jarayon yoki jarayonlar guruhi tanlanadi, ikkinchi qadamda shu jarayonlardan qay biri algoritmda joylashgan, qaysilari esa qo‘shimcha harajatlarni yoki ma’lumotlarni qayd eti shva hisobga olishga ketishi tashkil etadi. Ikki xil ahamiyatli jarayon turi mavjud: taqqoslash va arifmetik amallar. Barcha taqqoslash operatorlari ekvivalent hisoblanadi va ularni izlash hamda ajratuv algoritmlarida inobatga olinadi. Taqqoslash miqdori bunday algoritmlarning muhim elementi hisoblanadi, izlashda ushbu miqdor izlangan kattalik bilan mos tushadimi, saralashda esa berilgan oraliqdan chiqishi aniqlanadi. Solishtiruvchi operatorlar bir kattalikning ikkinchisi bilan teng yoki teng emasligi, katta yoki kichikligi, kichik yoki tengligi, katta yoki tengligini tekshiradi.

Biz arifmetik amallarni ikki guruhga bo‘lamiz: additiv va multiplikativ. Additiv operatorlar (qisqa qilib aytganda qo‘shuv) qo‘shish, ayirish, orttirish va qisqartirishni o‘z ichiga oladi. Multiplikativ operatorlar (yoki qisqacha ko‘paytirishlar) ko‘paytirish, bo‘lish va modul bo‘yicha qoldiq olishni o‘z ichiga oladi. Ikki guruhga ajratish ko‘paytirishning qo‘shishdan ko‘proq ishlatilishiga bog‘liq. Amaliyotda ba’zi algoritmlar ularda ko‘paytirish kam bo‘lsa, qo‘shishlar soni proporsional darajada o‘ssa ham afzalroq hisoblanadi. Biz o‘z kitobimizda yana bir, ko‘paytirishdan ham ko‘p vaqt talab qiluvchi amallar guruhini hosil qiluvchi logarifmlar va trigonametrik funksiyalardan foydalanuvchi algoritmlarga to‘xtalmadik (odatda kompyuterlar bu ifodalarni bir qatorga ajratish yordamida hisoblaydilar). Butun sonli ikki darajaga ko‘paytirish yoki bo‘lish alohida holatni tashkil qiladi. Bu jarayon siljishga olib keladi, keyingisi esa tezlik jihatdan qo‘shishning ekvivalenti hisoblanadi. Biroq, bu tafovut sezilarli bo‘lgan hollar juda kam, chunki 2 ga ko‘paytirish va bo‘lish birinchi navbatda taqqoslash operatorlari ahamiyatga ega bo‘lgan “taqsimla va boshqar” singari algoritmlarda uchraydi.

Kiruvchi ma’lumotlarning sinflari. Algoritmni tahlil qilishda kiruvchi ma’lumotlarni tanlash uning bajarilishiga ta’sir qilishi mumkin. Aytaylik, ba’zi saralash algoritmlari, agar kirish ro‘yxati saralangan bo‘lsa, juda tez ishlashi mumkin, boshqa algoritmlar shunday ro‘yxatda uncha katta bo‘lmagan natijani ko‘rsatadi. Tasodifiy ro‘yxatda esa natija buning teskarisi bo‘lishi mumkin. SHuning uchun biz ma’lumotlarning bir kirish ro‘yxatidagi algoritmlar harakatini tahlil qilish bilan chegaralanmaymiz. Biz algoritmni ham eng tez, ham eng sekin ishlashini ta’minlovchi ma’lumotlarni qidiramiz. Bundan tashqari, biz barcha mavjud ma’lumotlar to‘plamidagi algoritmlarning o‘rtacha samarasini ham baholaymiz.

Eng yaxshi holat. Bo‘limning nomlanishidan ham ko‘rinib turibdiki, algoritmlar uchun eng yaxshi holat bu qisqa vaqt ichida amalga oshiriladigan algoritmning ma’lumotlar jamlanmasi. Bunday jamlanma algoritm eng amal bajaradigan qiymatlar kombinatsiyasini ifodalaydi. Agar biz izlash algoritmini tekshirsak, izlangan qiymat birinchi algoritm tekshirayotgan katakka yozilgan bo‘lsa (odatda maqsadli qiymat yoki kalit deb ataladi), ma’lumotlar to‘plami eng yaxshi hisoblanadi. Bunday algoritmga uning murakkabligidan qatiy nazar, bitta taqqoslash kerak bo‘ladi. Shuni eslatish kerakki, ro‘yxatdan izlashda, uning qanchalik uzun bo‘lishidan qatiy nazar, eng yaxshi holat doimiy vaqtni talab qiladi. Umuman, eng yaxshi holatda algoritmni bajarish vaqti kichik yoki doimiy bo‘ladi, shuning uchun biz bunday tahlilni kam o‘tkazamiz.

Eng yomon holat. Eng yomon holatni tahlil qilish juda muhim, chunki u algoritm ishining maksimal vaqtini tasavvur qilishga yordam beradi. Eng yomon holatni tahlil qilganda algoritm eng ko‘p ish bajaradigan kirish ma’lumotlarini topish zarur. Izlovchi algoritm uchun bu kabi kiruvchi ma’lumotlar – bu shunday ro‘yxatki, unda izlangan kalit oxirida keladi yoki umuman bo‘lmaydi. Natijada N taqqoslash kerak bo‘ladi. Eng yomon holatning tahlili tanlangan algoritmga qarab dasturning ishlash vaqti uchun yuqori bahoni beradi.

O‘rtacha holat. O‘rta holatning tahlili eng murakkab hisoblanadi, chunki u ko‘pgina detallarni hisobga olishni talab qiladi. Tahlil asosini mavjud bo‘lgan kiruvchi ma’lumotlar to‘plamini bo‘lib chiqish lozim bo‘lgan turli guruhlarni

aniqlash tashkil qiladi. Ikkinchi qadamda kiruvchi ma’lumotlar to‘plami qaysi guruhga tegishli bo‘lish ehtimoli aniqlanadi. Uchinchi qadamda har bir guruhdagi ma’lumotlarga algoritmning ish vaqti hisoblanadi. Algoritmning bir guruhdagi hamma kiruvchi ma’lumotlar uchun ishlash vaqti bir xil bo‘lishi kerak, aks holda guruhni bo‘lish lozim. O‘rtacha ish vaqti

$$A(n)\sum_{i=1}^{m} p_i \bullet t_i, \qquad (5.1)$$

formula orqali hisoblanadi. Bu yerda n - kiruvchi ma’lumotlar o‘lchami, m - guruhlar soni, p_i - kiruvchi ma’lumotlarning i sonli guruhga tegishlilik ehtimoli, t_i – i sonli guruhdagi ma’lumotni qayta ishlash uchun algoritmga kerak bo‘ladigan vaqt deb belgilangan.

Ba’zi hollarda biz kiruvchi ma’lumotlarning har bir guruhga tushish ehtimolini bir xil deb taxmin qilamiz. Boshqacha aytganda, agar guruh 5 ta bo‘lsa, birinchi guruhga tushish extimoli ikkinchi yoki boshqa guruhga tushish extimolidek, ya’ni har bir guruhga tushish extimoli 0,2 ga teng. Bu holda ishning o‘rtacha vaqtini avvalgi formula Bilan yoki unga ekvivalent soddalashtirilgan barcha guruhlarning teng extimolligida haqiqiy bo‘lgan

$$A(n)\sum_{i=1}^{m} p_i \bullet t_i, \qquad (5.2)$$

formuladan foydalanishimiz mumkin.

Logarifmlar. Bizning tahlilimizda logarifmlar sezilarli o‘rin egallaydi, shuning uchun ularning xossalarini ko‘rib chiqishimiz lozim. *x* sonining *u* asos bo‘yicha logarifmi deb shunday darajaga aytiladiki, *x* ni olish uchun *u* ni ko‘tarish kerak bo‘ladi. Masalan, $\log_{10}45$ taxminan 1,653 ga teng, chunki $10^{1.653} \sim 45$. Logarifmning asosi ixtiyoriy son bo‘lishi mumkin, lekin bizning tahlilimizda ko‘proq 10 va 2 asosli logarifmlar uchraydi.

Logarifm – o‘sib boruvchi funksiya. Bu degani, agar $X > Y$ bo‘lsa, har bir V asos uchun $\log_B X > \log_B Y$. Logarifm – o‘zaro bir ifodali funksiya. Bu degani, agar $\log_B X = \log_B Y$ bo‘lsa $X=U$ bo‘ladi. SHuningdek, logarifmning unga kiruvchi

o‘zgaruvchilarning musbat qiymatida to‘g‘ri bo‘lgan quyidagi muhim xossalarini bilish lozim:

$$\log_B 1 = 0; \quad (5.3)$$

$$\log_B B = 1; \quad (5.4)$$

$$\log_B(XY) = \log_B X + \log_B Y; \quad (5.5)$$

$$\log_B X^Y = Y\log_B X; \quad (5.6)$$

$$\log_A X = \frac{\log_B X}{\log_B A} \quad (5.7)$$

shu xossalar yordamida funksiyani soddalashtirish mumkin. (5.7) xossasi logarifmning asosini almashtirish imkonini beradi. Ko‘plab kalkulyatorlar 10 asosli logarifmlar va natural logarifmlarni hisoblaydi. $\log_{42}75$ ni hisoblash uchun nima qilasiz? (5.7) tengligi yordamida 1.155 javobini olishingiz mumkin.

Binar daraxtlar. Binar daraxti shunday tuzilishga egaki, undagi har bir tugun ikkita tugundan ortiq bo‘lmagan bir ajdod nasldan iborat bo‘ladi. Daraxtning eng yuqori tuguni yagona ajdodsiz tugun hisoblanadi; u ildizli tugun deb ataladi. N tugunli binar daraxti kam $[\log_2 N+1]$ tugunga ega (tugunlarning maksimal zichligida). Masalan, 15 tugunli to‘la binar daraxtida bir ildiz, ikkinchi darajada 2 ta tugun, 3-darajada 4 ta tugun va 4-darajada 8 ta tugun bor; bizning tengligimiz ham $[\log_2 15]+1=[3.9]+1=4$ darajani beradi. Daraxtga yana bir tugunning qo‘shilishi yangi darajaning hosil bo‘lishiga olib keladi va ularning soni teng bo‘ladi $[\log_2 16] + 1 = [4] + 1 = 5$. N tugunli eng katta binar daraxti N darajaga ega: bu daraxtning har bir tugunida bitta nasl bor (daraxtning o‘zi ham oddiy ro‘yxat ko‘rinishiga ega).

Agar daraxtning darajalarini raqamlab chiqsak, ildiz 1 darajada deb hisoblab, K raqamli darajada 2^{K-1} tuguni yotadi. J darajali (1 dan J gacha raqamlangan) to‘la binar daraxtida hamma barglar J raqamli darajada yotadi va har bir tugunda birdan J-1 darajada ikkita to‘g‘ridan – to‘g‘ri nasl bor. J darajali to‘la binar daraxtida $2^J - 1$ tugun bor. Bu axborot keyinchalik ham sizga asqotishi mumkin. Bu formulalarni

yaxshiroq tushunish uchun binar daraxtlarini chizishni va natijalaringizni yuqoridagi formulalar bilan solishtirishingizni maslahat beramiz.

Ehtimolliklar. Biz algoritmlarni kiruvchi ma’lumotlarga ko‘ra tahlil qilmoqchimiz, buning uchun esa u yoki bu kiruvchi ma’lumotlar to‘plami qanchalik ko‘p uchrashini baholashimiz kerak. Shu bilan birga, biz kiruvchi ma’lumotlar u yoki bu sharoitlarga to‘gri kelish extimolligi bilan ishlashimizga to‘g‘ri keladi. U yoki bu hodisaning extimolligi nol va bir oralig‘idagi sondan iborat, 0 extimolligi hodisa hech qachon sodir bo‘lmasligi, 1 ehtimoli esa bo‘lishi mumkinligini bildiradi. Agar bizga turli kiruvchi qiymatlarning soni aniq 10 ga tengligi ma’lum bo‘lsa, ishonch Bilan aytishimiz mumkinki, har qanday bunday kirishning extimolligi 0 va 1 oralig‘ida bo‘ladi, barcha extimolliklarning yig‘indisi 1 ga teng, chunki ulardan bittasi amalga oshishi mumkin. Agar har bir kirishning amalga oshish extimolligi bir xil bo‘lsa, ulardan har birining extimolligi 0.1 ga teng bo‘ladi (10 dan 1 yoki 1/10).

Bizning tahlilimiz, asosan barcha imkoniyatlarni ko‘rib chiqishdan iborat bo‘ladi, keyin esa biz ularning hammasi teng extimolli deb faraz qilamiz. Agar imkoniyatlarning umumiy soni N ga teng bo‘lsa, ulardan har birining amalga oshishi extimolligi 1/N ga teng bo‘ladi.

Qo‘shish formulalari. Algoritmlarni tahlil qilganda biz ba’zi kattaliklar yig‘indisini qo‘shishimizga to‘g‘ri keladi. Aytaylik, bizda siklik algoritm bor. Agar sikl o‘zgaruvchisi 5 qiymatini olsa, sikl 5 marta bajariladi, agar uning qiymati 20 ga teng bo‘lsa 20 bo‘ladi. Agar sikl o‘zgaruvchisi M ga teng bo‘lsa, sikl M marta bajariladi. Agar sikl o‘zgaruvchisi 1 dan N gacha hamma qiymatlarga o‘tsa, sikl bajarilishining jami soni 1 dan N gacha bo‘lgan hamma natural sonlar yig‘indisiga teng bo‘ladi. Bu yig‘indini biz $\sum_{i=1}^{N} i$ ko‘rinishida yozamiz. Yig‘indi belgisining pastki qismida o‘zgaruvchi yig‘indining boshlang‘ich qiymati, yuqori qismida esa – oxirgi qiymati turibdi. Bunday ifodalanish bizni qiziqtirgan yig‘indi bilan qanday bog‘liqligi tushunarli.

Agar biror qiymat shu kabi yig‘indi ko‘rinishida yozilsa, natijani boshqa shu kabi ifodalar bilan solishtirish mumkin bo‘lishi uchun uni soddalashtirish kerak. Ikki

sondan kattasi $\sum_{i=11}^{N}(i^2-i)u\sum_{i=0}^{N}(i^2-20i)$. Shuning uchun yig‘indini soddalashtirish uchun biz quyidagi formulalardan foydalanamiz, bunda *S* - *i* ga bog‘liq bo‘lmagan o‘zgaruvchi.

$$\sum_{i=1}^{N} Ci = C\sum_{i=1}^{N} i \qquad (5.8)$$

$$\sum_{i=L}^{N} i = \sum_{i=0}^{N-L}(i+L) \qquad (5.9)$$

$$\sum_{i=L}^{N} i = \sum_{i=0}^{N} i - \sum_{i=0}^{L-1} i \qquad (5.10)$$

$$\sum_{i=1}^{N}(A+B) = \sum_{i=1}^{N} A + \sum_{i=1}^{N} B \qquad (5.11)$$

$$\sum_{i=0}^{N}(N-i) = \sum_{i=0}^{N} i \qquad (5.12)$$

(5.12) tengligi 0 dan N gacha sonlar yig‘indisi N dan 0 gacha sonlar yig‘indisiga tengligini bildiradi. Ba’zi hollarda bu tenglikni qo‘llash hisoblashlarni yengillashtiradi.

$$\sum_{i=1}^{N} 1 = N \qquad (5.13)$$

$$\sum_{i=1}^{N} C = CN \qquad (5.14)$$

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \qquad (5.15)$$

(5.15) tengligini yodlab olish oson, agar *1* dan *N* gacha bo‘lgan sonlarni juftlikka ajratsak. *1* ni *N* ga qo‘shib, *2* ni *N – 1* ga va hokazo, biz har biri *N+1* ga teng sonlar to‘plamini olamiz. Bunday sonlar nechta bo‘ladi? Albatta, ularning soni juft qilib ajratilgan elementlar sonining yarmiga teng, ya’ni *N/2*. Shuning uchun barcha *N* sonlarining yig‘indisi (5.18) ga teng.

$$\frac{N}{2}(N+1)=\frac{N(N+1)}{2} \qquad (5.16)$$

$$\sum_{i=1}^{N} i^2=\frac{N(N+1)(2N+1)}{6}=\frac{2N^3+3N^2+N}{6} \qquad (5.17)$$

$$\sum_{i=0}^{N} 2^i=2^{N+1}-1 \qquad (5.18)$$

(5.17) tengligini ikki bo‘linadigan sonlar orqali eslab qolishi mumkin. 0 dan 10 gacha bo‘lgan ikki darajasining yig‘indisi 11111111111 ikkilangan soniga teng. Bu songa 1 ni qo‘shib 100000000000, ya’ni 2^{11} ni hosil qilamiz. Lekin bu natija noldan o‘nggacha bo‘lgan ikki darajalarining yig‘indisidan 1 ga ko‘p, shuning uchun yig‘indining o‘zi 2^{11}-1 ga teng. Endi agar 10 o‘rniga N qo‘ysak, biz (5.17) tenglikka kelamiz.

Har qanday son uchun

$$\sum_{i=1}^{N} A^i=\frac{A^{N+1}-1}{A-1} \text{ ixtiyoriy A soni uchun} \qquad (5.19)$$

$$\sum_{i=1}^{N} i2^i=(N-1)2^{N+1}+2 \qquad (5.20)$$

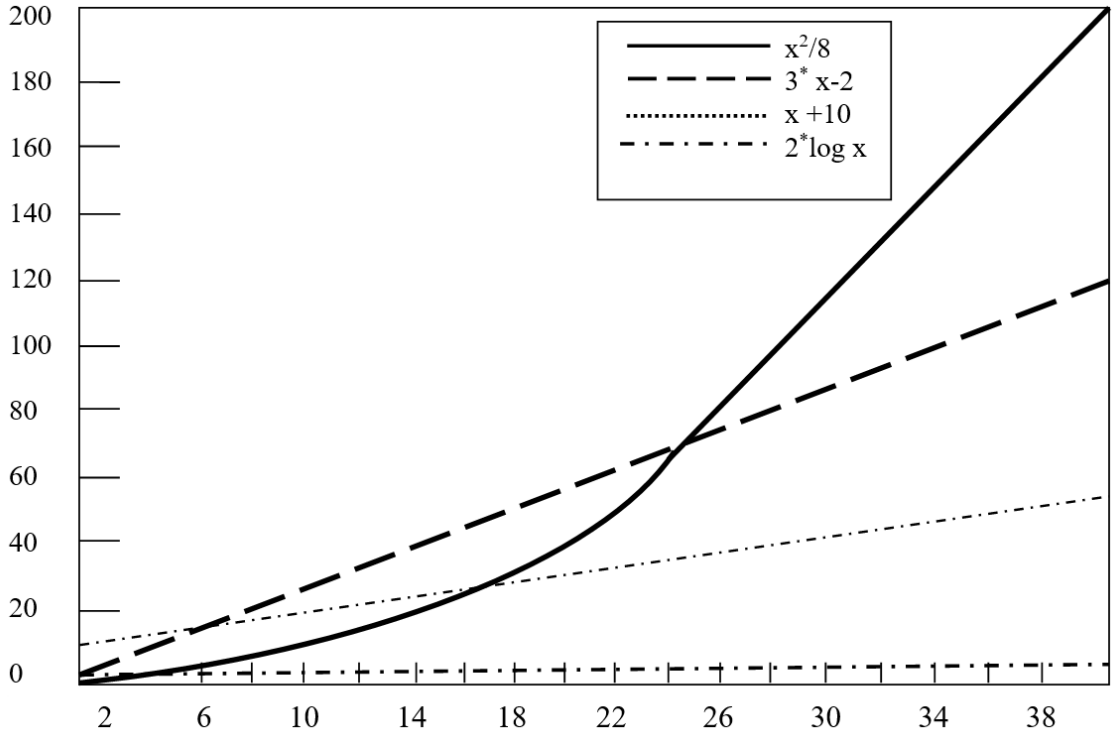
$$\sum_{i=1}^{N} \frac{1}{i}\approx \ln N \qquad (5.21)$$

$$\sum_{i=1}^{N} \log_2 i\approx N\log_2 N-1.5 \qquad (5.22)$$

Yig‘indilarni soddalashtirishda avval ularni (5.8)-(5.12) tenglik yordamida yanada oddiy sonlarga ajratish, so‘ngra yig‘indilarni boshqa ayniyatlar yordamida almashtirish mumkin.

O‘sish tezliklari. Algoritm bilan bajariladigan jarayonlar sonini aniq bilish algoritmlarni tahlil qilishda muhim rol o‘ynamaydi. Kiruvchi ma’lumotlarning hajmi ko‘payganida bu sonning o‘sish tezligi muhimroq hisoblanadi. U algoritmning **o‘sish tezligi** deb ataladi.

Agar 5.1-rasmga diqqat bilan qarasak, funksiya grafiklarining quyidagi xususiyatlarini ko‘rsatish mumkin. x^2 funksiya avval sekin o‘sadi, lekin x o‘sganda uning o‘sish tezligi ham oshadi. x funksiyasining o‘sish tezligi o‘zgaruvchining hamma qiymatlari oralig‘ida doimiydir. 2 log x funksiyasi umuman o‘smaydi, lekin bu yolg‘on tasavvur. Haqiqatda esa u o‘sadi, faqat juda sekin.



5.1-rasm. To‘rt funksiyaning grafigi.

Funksiya qiymatlarining nisbiy balandligi biz ko‘rib chiqayotgan o‘zgaruvchining qiymatlari kata yoki kichikligiga bog‘liq. x=2 da funksiya qiymatini taqqoslaymiz. Bu nuqtadagi eng kichik qiymatli funksiya $x^2/8$; eng kata qiymatli funksiya x+10 hisoblanadi. Biroq x ortishi bilan $x^2/8$ funksiya osha boshlaydi. Algoritmlarni tahlil qilish paytida bizni algoritmning o‘sish tezligi qiziqtiradi. Funksiyaning nisbiy “o‘lchami” bizni x o‘zgaruvchining katta qiymatlarida qiziqtiradi.

Ba’zi ko‘p uchraydigan funksiya sinflari 5.1-jadvalda keltirilgan. Bu jadvalda biz berilgan sinfning funksiya qiymatini erkin o‘zgaruvchan kattalik qiymatining keng diapazonida keltirdik. Ko‘rinib turibdiki, kiruvchi ma’lumotlarning oz o‘lchamlarida funksiya qiymatlari sezilarli farq qilmaydi, ammo bu o‘lchamlari o‘sganda farq sezilarli oshadi. bu jadval 5.1-jadvaldan taassurotni kuchaytiradi.

Shuning uchun biz kiruvchi ma’lumotlarning kata hajmlarida nima sodir bo‘lishini o‘rganamiz, kichik hajmlardagi farq ko‘rinmas bo‘lganligi sababli.

5.1-jadval va 5.1-rasmlardagi ma’lumotlar funksiyalarning yana bir xossasini namoyish etadi. Tez o‘suvchi funksiyalar sekin o‘suvchi funksiyalardan ustunlik qiladi. Shuning uchun, biz agar algoritm murakkabligi ikki yoki bir necha funksiyalar yig‘indisidan iborat bo‘lsa, tez o‘suvchi funksiyalardan tashqari barcha funksiyalarni olib tashlaymiz, masalan, agar biz algoritmni tahlil qilganda, uning x^3-30x taqqoslash qilishini bilsak, algoritmning murakkabligi x^3 kabi o‘sadi, deymiz. Buning sababi shundaki, 100 ta kiruvchi raqamlarda x^3 va orasidagi farq atiga 0,3 % ni tashkil qiladi. Keyingi bo‘limda biz bu fikrni formallashtiramiz.

O‘sish tezliklarini tasniflash. Algoritm murakkabligining o‘sish tezligi muhim rol o‘ynaydi va biz o‘sish tezligi formulasi kata ustunlikka ega hadi bilan aniqlanishini ko‘rdik. SHuning uchun biz sekin o‘sadigan kichik hadlarga e’tibor qaratmaymiz. Barcha kichik hadlarni olib tashlab, murakkablikning o‘sish tezligi hisoblanuvchi algoritm yoki funksiyaning tartibiga ega bo‘lamiz. Algoritmlarni ular murakkabligining o‘sish tezligiga qarab guruhlarga ajratish mumkin. Biz 3 toifani kiritamiz: murakkabligi mazkur funksiya kabi tez o‘suvchi algoritmlar, murakkabliklari o‘sha tezlikda o‘suvchi algoritmlar va murakkabligi bu funksiyadan sekin o‘suvchi algoritmlar. Funksiyalarning o‘sish sinflari 5.1-jadvalda keltirilgan.

5.1-jadval.

	$\log_2 n$	**n**	**$n\log_2 n$**	**n^2**	**n^3**	**$2n^2$**
1	0.0	1.0	0.0	1.0	1.0	2.0
2	1.0	2.0	2.0	4.0	8.0	4.0
5	2.3	5.0	11.6	25.0	125.0	32.0
10	3.3	10.0	33.2	100.0	1000.0	1024.0
15	3.9	15.0	58.6	225.0	3375.0	32768.0
20	4.3	20.0	86.4	400.0	8000.0	1048576.0
30	4.9	30.0	147.2	900.0	27000.0	1073741824.0

40	5.3	40.0	212.9	1600.0	64000.0	1099511627776.0
50	5.6	50.0	282.2	2500.0	125000.0	1125899906842620.0
60	5.9	60.0	354.4	3600.0	216000.0	1152921504606850000.0
70	6.1	70.0	429.0	4900.0	343000.0	1180591620717410000000.0
80	6.3	80.0	505.8	6400.0	512000.0	1208925819614630000000000.0
90	6.5	90.0	584.3	8100.0	729000.0	1237940039285380000000000000.0
100	6.6	100.0	664.4	10000.0	1000000.0	1267650600228230000000000000000.0

Katta omega. f singari tez o‘suvchi funksiyalar sinfini biz Ω(f) orqali belgilaymiz (katta omega deb o‘qiladi). Agar hamma qiymatlarda erkin o‘zgaruvchan kattalik n, ba’zi kata chegarada n_0 , $g(p) > cf(n)$ qiymati ba’zi musbat s son uchun bo‘lsa, g funksiyasi shu sinfga tegishli bo‘ladi. Ω(f) sinfi o‘zining quyi chegarasi bilan izohlansa, undagi hamma funksiyalar f kabi tez o‘sadi.

Biz algoritmlarning samaradorligi bilan qiziqamiz, shuning uchun Ω(f) sinfi bizni u darajada qiziqtirmaydi: masalan, Ω,(p^2) ga p^2 dan tez o‘suvchi hamma funksiyalar kiradi, aytaylik n^3 va2^n .

Katta O. Spektrning boshqa tarafida O(f) sinfi joylashgan (katta O deb o‘qiladi). Bu sinf f dan tez o‘smaydigan funksiyalardan tashkil topgan. Funksiya O(f) sinflari uchun yuqori chegarani hosil qiladi. Formal nuqtai nazardan f funksiyasi O(f) sinfiga tegishli, agar barcha n uchun $g(p) \leq cf(n)$, ba’zi chegara katta O va ba’zi musbat s konctanta uchun bo‘lsa.

Bu sinf biz uchun juda muhim. Ikkita algoritmdan qaysi birining murakkabligi katta O sinfiga tegishligi bizni qiziqtiradi.

Katta teta. v(Θ) orqali biz f singari tezlikda o‘suvchi funksiyalar sinfini belgilaymiz (katta teta deb o‘qiladi). Formal nuqtai nazardan bu sinf ikki avvalgi sinflarning kesishuvidan iborat, Θ (f) = Ω (f) ∩O(f).

Algoritmlarni taqqoslaganda bizni o‘rganilgan masalalardan tezroq yechuvchilari qiziqtiradi. Shuning uchun agar topilgan algoritm Θ sinfiga tegishli bo‘lsa, biz uni ko‘rb chiqmaymiz.

Katta O sinfining tavsifi. Berilgan funksiya O(f) ga tegishli ekanligini ikki xil yo‘l bilan tekshirish mumkin: yuqoridagi tavsif orqali yoki quyidagi tavsif yordamida: $g \in O(f), agar \lim_{n\to\infty} \frac{g(n)}{f(n)}$ = s ixtiyoriy s konstanta uchun. (5.23)

Bu shuni anglatadiki, *g(p)/f(n)* ning munosabatlar chegarasi mavjud bo‘lsa va u cheksizlikdan kichik bo‘lsa, g€O *(f)* bo‘ladi. Ba’zi funksiyalar uchun buni tekshirish oson emas. Lopital qonuniga ko‘ra, bunday holda funksiyalar chegarasini ularning hosilasi chegarasida almashtirish mumkin.

Belgilash. Θ(f),Ω(f), va O(f) sinflarining har biri to‘plam hisoblanadi, shuning uchun «g – shu to‘plam elementi» iborasi ahamiyatlidir. Biroq, tahlillarda g = O*(f)*deb yozishadi, aslida esa g€O *(f)***.**

Savol va topshiriqlar

1. Algoritmlar tahlili nima?
2. Algoritmlar tahlilining natijasini tushuntiring?
3. Xotira bo‘yicha murakkablikni tushuntiring?
4. Kiruvchi ma’lumotlarning sinflarini izohlang?
5. Funkisyalarning o‘sish sinflariga nimalar kiradi?
6. O‘sish sinflarini tasniflang?