**OʻZBEKISTON RESPUBLIKASI OLIY TAʻLIM, FAN VA**

**INNOVATSIYALAR VAZIRLIGI**

**MIRZO ULUGʻBEK NOMIDAGI OʻZBEKISTON**

**MILLIY UNIVERSITETINING JIZZAX FILIALI**

****

Kompyuter ilmlari va dasturlash kafedrasi

Algoritmlar va berilganlar strukturasi fanidan

**MUSTAQIL ISH-3-4**

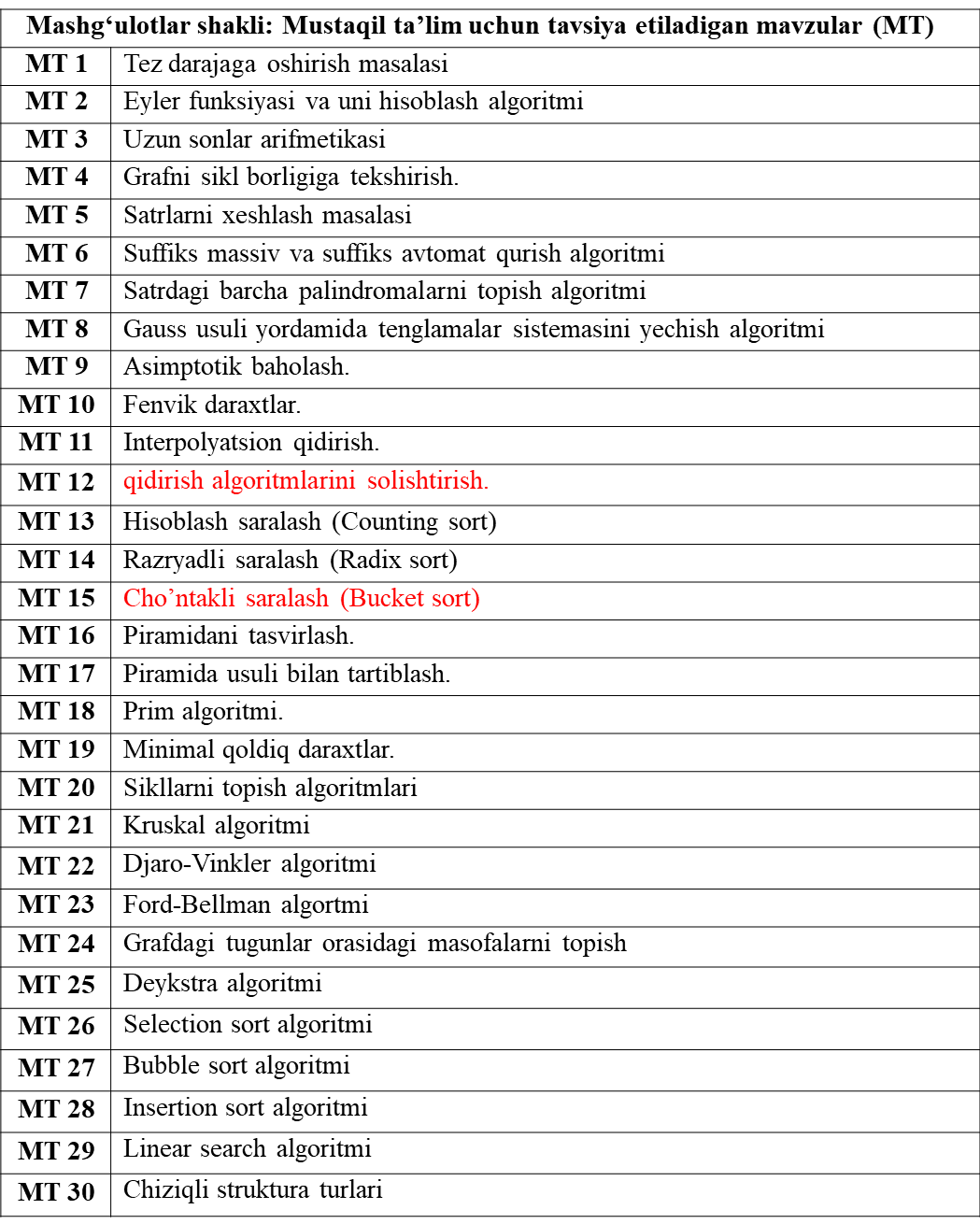
**Guruh:** 30-21 **Bajardi:** Bobomurotov Jafarbek

**Tekshirdi:** Tojiyev Maʻruf

*Jizzax\_2023*

Mavzu:Qidirish algoritmlarini solishtirish.

Reja:

1. Kirish. Algoritm tushunchasi
2. Qidirish algoritmlari haqida qisqacha ma’lumot.
3. Qidirish algoritmlarining turlari va ularga misollar.
4.  Xulosa.

Men hemis boyicha 6-variantda turaman

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No hemisdagi tartib buyicha har bir talabaga 4 tadan mustaqil ish | | | | |
| No | M1 | M2 | M3 | M4 |
| 1 | 1 | 20 | 2 | 30 |
| 2 | 2 | 8 | 4 | 27 |
| 3 | 3 | 2 | 6 | 24 |
| 4 | 4 | 19 | 8 | 21 |
| 5 | 5 | 6 | 10 | 18 |
| 6 | 6 | 14 | 12 | 15 |
| 7 | 7 | 19 | 14 | 12 |
| 8 | 8 | 12 | 16 | 9 |
| 9 | 9 | 5 | 18 | 6 |
| 10 | 10 | 15 | 20 | 3 |
| 11 | 11 | 27 | 22 | 6 |
| 12 | 12 | 20 | 24 | 7 |
| 13 | 13 | 21 | 26 | 8 |
| 14 | 14 | 16 | 28 | 9 |
| 15 | 15 | 19 | 30 | 10 |
| 16 | 16 | 16 | 3 | 11 |
| 17 | 17 | 30 | 5 | 12 |
| 18 | 18 | 12 | 7 | 13 |
| 19 | 19 | 12 | 9 | 14 |
| 20 | 20 | 9 | 11 | 15 |
| 21 | 21 | 5 | 13 | 16 |
| 22 | 22 | 15 | 15 | 17 |
| 23 | 23 | 23 | 17 | 18 |
| 24 | 24 | 22 | 19 | 19 |
| 25 | 25 | 13 | 21 | 20 |

Algoritm – bu algoritmik jarayon bilan ifodalanuvchi aniq koʻrsatmalar boʻlib, ixtiyoriy bеrilgan boshlangʻich maʻlumotdan boshlanadi (ushbu algoritm uchun mumkin boʻlgan bеrilganlar majmuasi) va ushbu bеrilganlar bilan ifodalanuvchi natija olishga qaratiladi.

Algoritmik jarayon – bu konstruktiv obʻеktlar (soʻzlar, sonlar, ifodalar)ning diskrеt qadamlar bilan amalga oshiriluvchi kеtma-kеt shakl oʻzgartirish jarayonidir.

Protsеdura (koʻrsatalar komplеksi) – alohida amallar barilishi qoidalarning knstruktiv aniqlanuvchi tizimidir.

Algoritm - algoritm bajaruvisi amalga oshiruvchi qaralayotgan masalalar sinfiga taalluqli boʻlgan ixtiyoriy masalaning еchimini topish uchun zarur boʻlgan chеkli sondagi amallar kеtma-kеtligi va mazmunini ifodalovchi formallashtirilgan va konstruktiv , aniq va toʻliq koʻrsatalar tizimi vositasida rеalizatsiya qilinadi.

Algoritm – alohida olingan olinga masalalar toʻplamini rchishga qaratilgan hamda cheklilik, aniqlik, kirish, chiqish va effektivlik xususiyatlariga ega boʻlgan qoidalarning chekli toʻplamidir (Dolald Knut)

Algoritm –qandaydir sondagi qadamdan keyin qoʻyilgan masalaning echimiga olib keluvchi qatʻiy qoidalar boʻyicha bajariluvchi ixttiyoriy hisoblash tizimidir (A.Kolmogorov)

Algoritm –tanlanuvchi boshlangʻich berilganlardan izlangan natijaga intiluvchi hisoblash jarayonini ifodalovchi aniq koʻrsatmadir (A.Markov)

Algoritmlashning vazifalari:

1. Yangi algoritm yaratish, protsеdurani formallashtiri yoki oldindan ishlab chiqilgan algoritmni modifikatsiyalash.
2. Algoritm toʻgʻriligini isbotlash (vеrifikatsiya, tеstlash).
3. Ishlab chiqilgan yoki modifikatsiya qilingan algoritmni rеalizatsiya qilish, uni boshqa bajaruvchilar buyruqlar tizimiga oʻgirish.
4. Algoritmni effеktivlik kritеriylari boʻyicha tahlil qilish va baholash.

Algoritmik jarayon xaraktеristikalari. Algoritmni xaraktеrlovchi paramеtrlar:

1. mumkin boʻlgan boshlangʻich bеrilganlarning majmuasi;
2. mumkin boʻlgan oraliq natijalar majmuasi;
3. natijalar majmuasi;
4. boshlash qoidasi;
5. axborotni bеvosita qayti ishlash qoidasi;
6. tugallash qoidasi;
7. Natijani olish qoidasi.

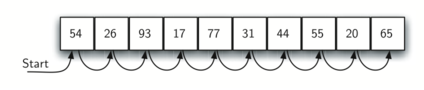
Qidirish algoritmlari ma'lum bir narsani topish uchun ishlatiladigan algoritmlardir. Bu narsa misol uchun, bir arrayda ko'p elementlar o'zaro bog'liq bo'lganda, yoki bir matn ichidagi ma'lum bir so'zni izlashda yoki fayl tizimida bir faylni topishda yoki boshqa bir ko'plab masalarda ishlatilishi mumkin.

Qidiruv masalasini shakllantirish quyidagicha bosqichda amalga oshiriladi:

* Ma’lumotlarni yig’ish, to’ldirish.
* Ma’lumotlarni tashkil etish (tartiblash va saralash);
* Ma’lumotlarni olish (hususiy qidiruv);
* Ketma ket qidiruv algoritmi

Ma'lumotlar ob'ektlari ro'yxat kabi to'plamda saqlanganida, biz ular chiziqli yoki ketma-ket munosabatga ega deb aytamiz. Har bir ma'lumot elementi boshqalarga nisbatan bir holatda saqlanadi. Python ro'yxatlarida bu nisbiy pozitsiyalar alohida elementlarning indeks qiymatlari hisoblanadi. Ushbu indeks qiymatlari tartiblanganligi sababli, biz ularga ketma-ket tashrif buyurishimiz mumkin. Bu jarayon bizning birinchi qidiruv texnikamiz, ketma-ket qidiruvni keltirib chiqaradi .

Quyidagi diagrammada ushbu qidiruv qanday ishlashi ko'rsatilgan. Roʻyxatning birinchi bandidan boshlab, biz izlayotgan narsani topmagunimizcha yoki elementlar tugamaguncha, asosiy ketma-ket tartibni kuzatib, shunchaki elementdan elementga oʻtamiz. Agar buyumlarimiz tugasa, biz qidirayotgan narsa yo'qligini aniqladik.



1-rasm.Butun sonlar ro'yxatini ketma-ket qidirish

Ushbu algoritm uchun Python ilovasi quyida ko'rsatilgan. Funksiya biz izlayotgan roʻyxat va elementga muhtoj va u mavjud yoki yoʻqligi haqida mantiqiy qiymatni qaytaradi. Esda tutingki, amalda biz Python operatoridan shu maqsadda foydalanardik , shuning uchun quyida keltirilgan algoritmni biz uchun taqdim etilmaganida innima qilishimiz haqida o'ylashingiz mumkin .

def sequential\_search(alist, item):

position = 0

while position < len(alist):

if alist[position] == item:

return True

position = position + 1

return False

testlist = [1, 2, 32, 8, 17, 19, 42, 13, 0]

sequential\_search(testlist, 3) # => False

sequential\_search(testlist, 13) # => True

Ketma-ket qidiruvni tahlil qilish

Qidiruv algoritmlarini tahlil qilish uchun biz asosiy hisoblash birligi haqida qaror qabul qilishimiz kerak. Eslatib o'tamiz, bu odatda muammoni hal qilish uchun takrorlanishi kerak bo'lgan umumiy qadamdir. Qidiruv uchun, amalga oshirilgan taqqoslashlar sonini hisoblash mantiqiy. Har bir taqqoslash biz izlayotgan narsani topishi yoki topmasligi mumkin. Bundan tashqari, biz bu erda yana bir taxmin qilamiz. Elementlar ro'yxati hech qanday tarzda tartiblanmagan. Elementlar tasodifiy ro'yxatga kiritilgan. Boshqacha qilib aytadigan bo'lsak, biz qidirayotgan elementning har qanday muayyan holatda bo'lish ehtimoli ro'yxatning har bir pozitsiyasi uchun aynan bir xil.

Agar element ro'yxatda bo'lmasa, uni bilishning yagona yo'li uni mavjud bo'lgan har bir element bilan solishtirishdir. Agar mavjud bo'lsa n elementlar, keyin ketma-ket qidiruv talab qiladi n buyum mavjud emasligini aniqlash uchun taqqoslash. Agar element ro'yxatda bo'lsa, tahlil unchalik oson emas. Aslida sodir bo'lishi mumkin bo'lgan uch xil stsenariy mavjud. Eng yaxshi holatda biz elementni birinchi navbatda, ro'yxatning boshida topamiz. Bizga faqat bitta taqqoslash kerak bo'ladi. Eng yomon holatda, biz oxirgi taqqoslashgacha, ya'ni n-chi taqqoslashgacha elementni aniqlay olmaymiz.

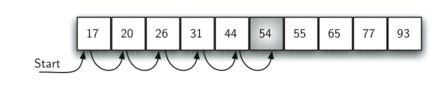
O'rtacha holat haqida nima deyish mumkin? O'rtacha, biz elementni ro'yxatning yarmida topamiz; ya'ni solishtiramiz .n\2.. buyumlar. Shuni esda tutingki, n kattalashgan sari, koeffitsientlar, ular qanday bo'lishidan qat'i nazar, bizning yaqinlashishimizda ahamiyatsiz bo'lib qoladi, shuning uchun ketma-ket qidiruvning murakkabligi,O ( n ):



2-rasm. Ketma-ket qidiruv algoritm holatlari

Ilgari biz kollektsiyamizdagi narsalar tasodifiy joylashtirilgan deb taxmin qilgan edik, shunda narsalar o'rtasida nisbiy tartib yo'q. Agar narsalar qandaydir tarzda buyurtma qilingan bo'lsa, ketma-ket qidiruv nima bo'ladi? Qidiruv texnikamizda samaradorlikka erisha olamizmi?

Ob'ektlar ro'yxati shunday tuzilgan deb faraz qilingki, elementlar pastdan yuqoriga o'sish tartibida joylashgan. Agar biz qidirayotgan element ro'yxatda bo'lsa, uning n pozitsiyadan birida bo'lish ehtimoli avvalgidek bo'ladi. Ob'ektni topish uchun bizda hali ham bir xil miqdordagi taqqoslashlar bo'ladi. Biroq, agar element mavjud bo'lmasa, ozgina afzallik bor. Quyidagi diagrammada algoritm 50-bandni qidirayotganda bu jarayon ko'rsatilgan. E'tibor bering, elementlar hali ham 54-gacha bo'lgan ketma-ketlikda taqqoslanadi. Biroq, bu nuqtada biz qo'shimcha narsani bilamiz. Nafaqat 54 biz izlayotgan element emas, balki 54 dan ortiq boshqa elementlar ham ishlay olmaydi, chunki roʻyxat tartiblangan.



3-rasm.Butun sonlarning tartiblangan ro'yxatini ketma-ket qidirish

Bunday holda, algoritm element topilmaganligi haqida xabar berish uchun barcha elementlarni ko'rib chiqishni davom ettirishi shart emas. U darhol to'xtashi mumkin. Quyidagi kod ketma-ket qidiruv funktsiyasining ushbu o'zgarishini ko'rsatadi.

def ordered\_sequential\_search(alist, item):

position = 0

while position < len(alist):

if alist[position] == item:

return True

if alist[position] > item:

return False

position = position + 1

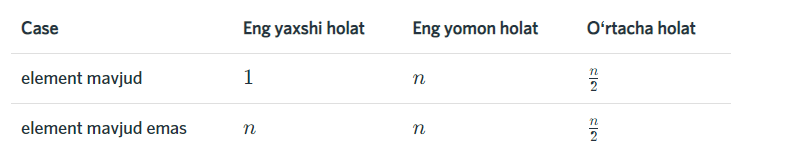
return False

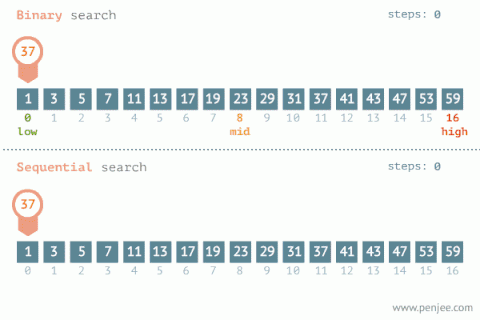
testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]

ordered\_sequential\_search(testlist, 3) # => False

ordered\_sequential\_search(testlist, 13) # => True

Quyidagi jadval ushbu natijalarni umumlashtiradi. E'tibor bering, eng yaxshi holatda biz faqat bitta elementga qarab, element ro'yxatda yo'qligini aniqlashimiz mumkin. O'rtacha, biz faqat ko'rib chiqqandan keyin bilib olamiz. n\2 buyumlar. Biroq, bu texnika hali ham O ( n ). Xulosa qilib aytadigan bo'lsak, ketma-ket qidiruv ro'yxatga buyurtma berish orqali yaxshilanadi, agar biz elementni topa olmasak.

Binar search (Ikkilik qidiruv)



Ta'rif

Ikkilik qidiruv (eng: Binary search — ikkilik qidiruv)- saralangan elementlar roʻyxatidan elementni topish uchun samarali algoritmlardan biri hisoblanadi. Ikkilik qidiruv algoritmi ishlash gʻoyasiga koʻra "boʻlib tashla va hukmronlik qil" paradigmasi asosida ishlaydi.

Ikkilik qidirish algoritmini ishlash prinsipini tushunish uchun keling do'stimiz bilan o'yin o'ynab ko'ramiz. O'yin shartlari quyidagicha:

* Do'stimizga 1 dan 50 gacha bo'lgan sonlar orasidan bitta sonni o'ylashini aytamiz. Siz esa do'stimiz o'ylagan sonni iloji boricha kamroq urinishda topishingiz kerak bo'ladi.
* Har bir taxminingizdan so'ng do'stimiz sizga o'ylagan soni siz taxmin qilgan sondan kichik yoki katta ekanligini aytadi va siz yana taxmin qilishda davom etasiz..
* Agar siz taxmin qilgan son do'stimiz o'ylagan son bilan bir xil bo'lsa o'yin tugaydi.

Do'stimiz 36ni o'yladi.Xo'sh bunday holatda nima qilamiz ?

Albatta

* birinchi bo'lib o'rtadagi sonni tanlaymiz ya'ni 25, do'stimiz bu son o'zi o'ylagan sondan kichik ekanligini aytadi va biz endi qidirilyotgan sonni [26 - 50] oraliqdan qidiramiz.
* Ikkinchi taxminimizda biz 38 ni tanlaymiz, do'stimiz esa bu son o'zi o'ylagan sondan katta ekanligini aytadi. Endi biz u o'ylagan sonni [26 - 37] oralig'idan qidiramiz.
* Uchinchi urinishimizda biz 31 ni tanlaymiz, u esa bu son o'zi o'ylagan sondan kichkina ekanligini aytadi. Biz oraliqni [32 - 37] qilib olamiz va qidirishda davom etamiz.
* To'rtinchi urinishimizda biz 35 sonini taxmin qilamiz va u bu son ham o'zi o'ylagan sondan kichkina deb aytadi endi u o'ylagan son 35 dan katta va 37dan kichik ekanligi malum bo'ladi.

Ushbu oraliqda bitta 36 soni borligi uchun biz unga sonini o'ylaganligini aytamiz va o'yinni yakunlaymiz.

Demak biz 1 dan 50 gacha oraliqda yashirin sonni topish uchun 5tagina urinish qildik, agar bu sonni biz chiziqli qidirganimizda 36ta urinish qilar edik.

Algoritmning murakkabligi Time (vaqt) miqdori asosida o'lchanadigan qiymat. Bunda Time dasturning ishlash jarayonida ketqizadigan vaqti. Bu faktor algoritmni effektivligini aniqlaydi. Yaxshi algoritmlarda Time kam miqdorda bo'ladi, bu esa usha algoritmda yozilgan kodning ishlash tezligini oshiradi.

Algoritmlarni tahlil qilishni yana Asymptotic analysis ham deyiladi. Bunda kiritilgan input kattaligiga qarab algoritmning ishlash vaqti hisoblanadi.

Asymptotic analysis ga misol sifatida tartiblangan array'dan berilgan X sonni topish uchun Linear Search va Binary Search algoritmlarini taqqoslaymiz.

Arrayning birinchi elementidan boshlab X ni topguncha array elementlarini bittalab tekshirib chiqadi. Array uzunligini N deb oladigan bo'lsak.  
Binary Search ro'yxatning o'rtasidagi element qiymatini – M ni oladi va uni X bilan solishtiradi:

X M ga teng bo'lsa, bingo! birinchi urinishdayoq kerakli element topildi.

Agar X M dan kichik bo'lsa, demak qidirishni ro'yxatning birinchi yarmidan davom ettirish kerak.

Agar X M dan katta bo'lsa, demak qidirishni ro'yxatning ikkinchi yarmidan davom ettiriladi.

Hudi shunday uslubda, avval qismning o'rtasi M topiladi, keyin X ni M bilan

solishtirib boriladi.  
Biz 100000 elementi bor katta ro'yxatni oladigan bo'lsak:  
Binary Search orqali minimal 1ta, maksimal Log(100000) bu 16ga teng urinish degani. Linear Searchda esa minimal 1ta maksimal esa 100 000 urinish qiladi.

Linear searchda maksimal 100,000 urinish Binary searchda maksimal 16 urinish!

Biz aytib o'tgan urinishlar sonini aniqlashda ko'pincha best case(eng yaxshi holat), average case(o'rtacha holat) va worst case(eng yomon holat)lar orqali hisoblanadi.  
Best case va worst case grekcha Θ harfi bilan yoziladi.

Linear search va binary search algoritmlari Θ da yozilganda:

Best case(yaxshi holat):

* linear search – Θ(1)
* binary search – Θ(1)

Worst case(yomon holat):

* linear search – Θ(n)
* binary search – Θ(log n)

Foydalanilgan adabiyotlar

1. Алгоритмы в современной мутематике и ее приложениях. Материалы международного симпозиума. Ургенч. 1979 г. Под редакцией А.П. Ершова и Д. Кнута. 8-с.
2. Paul E. Black "Postman's Sort" from Dictionary of Algorithms and Data Structures at NIST.
3. Robert Ramey '"The Postman's Sort" C Users Journal Aug. 1992
4. NIST's Dictionary of Algorithms and Data Structures: bucket sort

Foydalanilgan resurlar

1. <https://www.geeksforgeeks.org/lower-bound-on-comparison-based-sorting-algorithms/>
2. <https://www.programiz.com/dsa/bucket-sort>
3. <https://www.javatpoint.com/bucket-sort>
4. <https://xlinux.nist.gov/dads/HTML/bucketsort.html>

Mavzu: Bucket sort algoritmi

Reja

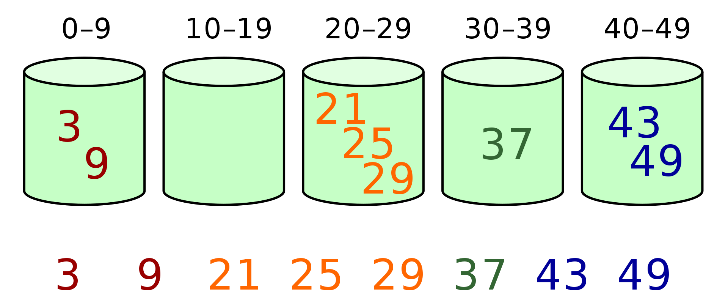
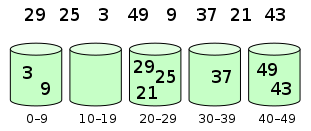
1.Bucket sort algoritmi dolzarbligi

2.Bucket sort algoritmi boʻyicha C# da tuzilgan dastur

3.Xulosa

4.Foydalanilgan adabiyotlar

**Bucket sort** , yoki **bin sort** , massiv elementlarini bir nechta chelaklarga taqsimlash orqali ishlaydigan tartiblash algoritmidir. Keyin har bir chelak alohida saralanadi, yoki boshqa tartiblash algoritmi yordamida yoki chelakni saralash algoritmini rekursiv qoʻllash orqali. Bu tarqatish turi, har bir chelakda bir nechta kalitlarga ruxsat beruvchi kaptar tuynuklari turining umumlashtirilishi va eng muhim raqam lazzatidagi radix turining qarindoshi. Chelaklarni saralash taqqoslash bilan amalga oshirilishi mumkin va shuning uchun ham taqqoslash tartibi algoritmi deb hisoblanishi mumkin. Hisoblashning murakkabligihar bir chelakni saralash uchun ishlatiladigan algoritmga, ishlatiladigan chelaklar soniga va kirish bir xil taqsimlanganligiga bogʻliq.



Paqirni saralash quyidagicha ishlaydi:

1. Dastlab boʻsh "chelaklar" qatorini oʻrnating.
2. **Tarqalish** : Har bir obʻektni chelakka solib, asl massivdan oʻting.
3. Har bir boʻsh boʻlmagan chelakni tartiblang.
4. **Yigʻish** : chelaklarga tartibda tashrif buyuring va barcha elementlarni asl massivga qaytaring.

Bucket sort (choʻntakli saralash) toʻplam elementlarini saralash algoritmi hisoblanadi va har bir elementni belgilangan saralash tartibidagi ʻʻtoʻplamʻʻlarga joylashtiradi.

Bu algoritm amaliyoti quyidagicha boʻladi:

1. Toʻplam elementlarning miqdorini aniqlash va boʻsh ʻʻtoʻplamʻʻlar yaratish.

2. Har bir elementni mos ʻʻtoʻplamʻʻga qoʻshish. Bu odatda elementlar qanday boʻlishi mumkinligiga qarab saralash tartibini bilishimizdan qatʻi nazar, misol uchun, sonlar toʻplami uchun sonlar boʻlgan holatda mos toʻplamga joylashadi.

3. Har bir ʻʻtoʻplamʻʻni oʻz ichiga saralash algoritmi bilan saralash, masalan, quyidagi algoritmlardan biri bilan tuzilishi mumkin.

• Insertion Sort

• Quick Sort

• Merge Sort

4. Har bir ʻʻtoʻplamʻʻdagi saralgan elementlarni olib birlashtirish. Bunda 1 dan n gacha boʻlgan toʻplamlarni bir-biri bilan bogʻlash kerak.

5. Natijadagi toʻplamlardan moslashgan elementlarni yigʻish va natijani qaytarish.

Bucket sort yuqori sanaqtarofli toʻplam elementlari uchun juda samarali, shuningdek, uning vaqti O(n) boʻlib, baʻzi yondashuvlarda esa O(nlogn) ga yaqinlashadi. Qoʻshimcha xotira xavfsizligi haqida unga ishlatilish mumkin.

ʻʻBucket sortʻʻ oʻz nomidan tushuntirib boʻlganidek, massivni chiqarish jarayonida chet elon qilingan ʻʻsaqqlarʻʻ yoki ʻʻqoʻshni saqqlarʻʻ yordamida ishlaydigan tartiblash algoritmasidir. Bu tartiblash algoritmasi amaliyotda juda foydali boʻlib, bir nechta tartiblash muammo va vaziyatlari uchun yaxshi ish bajaradi.

Algoritma xususiyatlariga koʻra, saqqlar boʻsh yoki oʻz ichiga boshqa elementlarga tegishli boʻlishi mumkin. Algoritma shu darajada ishlaydi: Har bir elementni mos saqqlarga joylashtirish orqali, keyin har bir saqqadagi elementlarni ochilish tartibida chiqarib oladi. Shu sababli bu tartiblash algoritmasi odatda elementlar doimiy tartibda boʻlmagan holatlar uchun ishlatiladi. Algoritmaning vaqti, muammoga qaranganda asosiy ravishda kiritilgan elementlar soniga vaqt va joyiga bogʻliq boʻladi, chunki algoritmaning samaradorligi undan kelib chiqqanidek elementlar sonining kichikligi va saqqlar sonining kattaligi bilan bogʻliq.

Paqirni saralash Qiyinchilik darajasi: oson

Chelaklarni saralash, asosan, kirish diapazonda bir xil taqsimlanganda foydalidir. Masalan, quyidagi muammoni koʻrib chiqing. 0,0 dan 1,0 gacha boʻlgan va diapazon boʻylab bir tekis taqsimlangan suzuvchi nuqtali raqamlarning katta toʻplamini tartiblang. Raqamlarni qanday qilib samarali saralaymiz? Oddiy usul - taqqoslashga asoslangan tartiblash algoritmini qoʻllashdir. Taqqoslashga asoslangan saralash algoritmining pastki chegarasi ( Birlashtirish, Uyma tartiblash, Tezkor saralash .. va hokazo) Ō(n Log n), yaʻni ular nLogn dan yaxshiroq ish qila olmaydi.   
Massivni chiziqli vaqtda tartiblashimiz mumkinmi? Bu yerda hisoblash tartibini qoʻllab boʻlmaydi, chunki biz kalitlarni sanashda indeks sifatida ishlatamiz. Bu erda kalitlar suzuvchi nuqta raqamlari.    
Gʻoya chelak saralashdan foydalanishdir. Quyida paqir algoritmi keltirilgan.

bucketSort(arr[], n)

1) Create n empty buckets (Or lists).

2) Do following for every array element arr[i].

.......a) Insert arr[i] into bucket[n\*array[i]]

3) Sort individual buckets using insertion sort.

4) Concatenate all sorted buckets.

BucketSort

Vaqtning murakkabligi: Agar chelakka kiritish O(1) vaqtni oladi deb faraz qilsak, yuqoridagi algoritmning 1 va 2-bosqichlari O(n) vaqtini oladi. Agar biz chelakni ifodalash uchun bogʻlangan roʻyxatni ishlatsak, O (1) osongina mumkin (Quyidagi kodda soddalik uchun C++ vektoridan foydalaniladi). 4-qadam ham O(n) vaqtni oladi, chunki barcha chelaklarda n ta element boʻladi. Tahlil qilish uchun asosiy qadam 3-bosqichdir. Agar barcha raqamlar bir xil taqsimlangan boʻlsa, bu bosqich ham oʻrtacha O(n) vaqtni oladi ( batafsilroq maʻlumot uchun CLRS kitobiga  
qarang) Quyida yuqoridagi algoritmni amalga oshirish koʻrsatilgan.

*using System;*

*using System.Linq;*

*using System.Collections.Generic;*

*namespace Bucket\_sort\_algarithm*

*{*

*class Program*

*{*

*static void BucketSort(float[] arr, int n)*

*{*

*if (n <= 0) return;*

*// Bucket yaratish*

*List<float>[] buckets = new List<float>[n];*

*// Har bir elementni mos bucketga joylashtirish*

*for (int i = 0; i < n; i++)*

*{*

*int bucketIndex = (int)(arr[i] \* n);*

*if (buckets[bucketIndex] == null)*

*{*

*buckets[bucketIndex] = new List<float>();*

*}*

*buckets[bucketIndex].Add(arr[i]);*

*}*

*// Har bir bucketni saralash va mos ravishda birlashtirish*

*for (int i = 0; i < n; i++)*

*{*

*if (buckets[i] != null)*

*{*

*buckets[i].Sort();*

*}*

*}*

*// Saralgan bucketlardagi ma'lumotlarni yana kiruvchi massivda joylashtirish*

*int index = 0;*

*for (int i = 0; i < n; i++)*

*{*

*if (buckets[i] != null)*

*{*

*foreach (float num in buckets[i])*

*{*

*arr[index++] = num;*

*}*

*}*

*}*

*}*

*static List<double>[] BucketSort(double[] array, int k)*

*{*

*List<double>[] buckets = new List<double>[k];*

*for (int i = 0; i < k; i++)*

*{*

*buckets[i] = new List<double>();*

*}*

*double M = array.Max() + 1;*

*for (int i = 0; i < array.Length; i++)*

*{*

*int bucketIndex = (int)Math.Floor(k \* array[i] / M);*

*buckets[bucketIndex].Add(array[i]);*

*}*

*for (int i = 0; i < k; i++)*

*{*

*buckets[i].Sort();*

*}*

*return buckets;*

*}*

*static void Main(string[] args)*

*{*

*////float tipidagi sonlar uchun*

*float[] arr = { 0.85f, 0.23f, 0.69f, 0.14f, 0.45f, 0.34f, 0.62f, 0.75f, 0.98f, 0.21f };*

*int n = arr.Length;*

*Console.WriteLine("Saralashdan avval:");*

*for (int i = 0; i < n; i++)*

*{*

*Console.Write(arr[i] + " ");*

*}*

*BucketSort(arr, n);*

*Console.WriteLine("\nSaralashdan keyin:");*

*for (int i = 0; i < n; i++)*

*{*

*Console.Write(arr[i] + " ");*

*}*

*////double tipidagi sonlar uchun*

*double[] array = new double[] { 5, 9, 2, 10, 1, 7, 12, 3, 20, 14};*

*int k = 5;*

*Console.WriteLine("\nSaralashdan avval:");*

*for (int i = 0; i < n; i++)*

*{*

*Console.Write(array[i] + " ");*

*}*

*List<double>[] sortedBuckets = BucketSort(array, k);*

*Console.WriteLine("\nSaralangan array:");*

*foreach (List<double> bucket in sortedBuckets)*

*{*

*foreach (double value in bucket)*

*{*

*Console.Write(value + " ");*

*}*

*}*

*}*

*}*

*}*

Agar qatʻiy ishlab chiqarish samaradorligi (time complexity) uchun aytilsa, Bucket sort O(n+k) vaqt talab qiladi, bu yerda ʻʻnʻʻ massivdagi elementlar soni va ʻʻkʻʻ saqqlar soni boʻladi. Bucket sort yordamida tartiblash mumkin boʻlgan elementlar kiritingan diapazondan qatʻiy chegaraga ega boʻlishi kerak, aks holda tartiblash natijasi yaxshi boʻlmasligi mumkin.

Bu kod natijada kiritilgan massivni saralab chiqaradi. Kod oʻzida oʻzini tushunishida bir ozgarishni kiritishingiz mumkin - ʻʻnumOfBucketsʻʻ saqqlarning sonini belgilaydi. Misol uchun, massivdagi elementlar diapazondagi elementlarni saralash uchun, yaʻni 0 va 1 oraligʻida boʻlsa, numOfBuckets 2 boʻlishi kerak. Natijada, Bucket sort algoritmasi katta massivlar uchun ishlatiladi, ammo bu misol kodni tushunish uchun muhimdir.

Paqirni saralash - tartiblanmagan massiv elementlarini chelaklar deb ataladigan bir necha guruhlarga ajratuvchi tartiblash algoritmi. Keyin har bir chelak mos keladigan tartiblash algoritmlaridan birini qoʻllash yoki bir xil chelak algoritmini rekursiv qoʻllash orqali tartiblanadi. Nihoyat, saralangan chelaklar yakuniy tartiblangan massivni hosil qilish uchun birlashtiriladi.

Scatter yigʻish yondashuvi-Paqirlarni saralash jarayonini scatter-toʻplash yondashuvi sifatida tushunish mumkin. Bu yerda elementlar avval chelaklarga sochiladi, keyin har bir chelakdagi elementlar tartiblanadi. Nihoyat, elementlar tartibda yigʻiladi. Scatter yigʻish yondashuvi Paqirlarni saralash jarayonini scatter-toʻplash yondashuvi sifatida tushunish mumkin. Bu yerda elementlar avval chelaklarga sochiladi, keyin har bir chelakdagi elementlar tartiblanadi. Nihoyat, elementlar tartibda yigʻiladi.



Paqirni saralash murakkabligi

Vaqtning murakkabligi

Eng yaxshi O(n+k)

Eng yomoni O(n2)

Oʻrtacha O(n)

*Kosmik murakkablik* O(n+k)

Eng yomon holatlarning murakkabligi: massivda yaqin diapazondagi elementlar mavjud boʻlsa, ular bir xil chelakka joylashtirilishi mumkin. Bu baʻzi chelaklarda boshqalarga qaraganda koʻproq elementlarga ega boʻlishiga olib kelishi mumkin. Bu murakkablikni chelak elementlarini saralash uchun ishlatiladigan tartiblash algoritmiga bogʻliq qiladi. Elementlar teskari tartibda boʻlsa, murakkablik yanada yomonlashadi. Agar qoʻshish tartiblash chelak elementlarini saralash uchun ishlatilsa, vaqt murakkabligi ga aylanadi .O(n2)

*Eng yaxshi holatning murakkabligi*: O(n+k)

elementlar har bir chelakda deyarli teng miqdordagi elementlar bilan chelaklarda bir tekis taqsimlanganda paydo boʻladi. Agar chelaklar ichidagi elementlar allaqachon tartiblangan boʻlsa, murakkablik yanada yaxshilanadi. Agar paqir elementlarini saralash uchun qoʻshish tartibi ishlatilsa, eng yaxshi holatda umumiy murakkablik chiziqli boʻladi, yaʻni. O(n+k). O(n)chelaklarni yasashning murakkabligi va O(k)eng yaxshi holatda chiziqli vaqt murakkabligiga ega boʻlgan algoritmlar yordamida chelak elementlarini saralashning murakkabligi.

*Oʻrtacha holatlar murakkabligi*: O(n)

elementlar massivda tasodifiy taqsimlanganda paydo boʻladi. Elementlar bir tekis taqsimlanmagan boʻlsa ham, chelaklarni saralash chiziqli vaqtda ishlaydi. Bu chelak oʻlchamlari kvadratlarining yigʻindisi elementlarning umumiy sonida chiziqli boʻlgunga qadar toʻgʻri boʻladi.

**Tarqatish - yigʻish usuli**

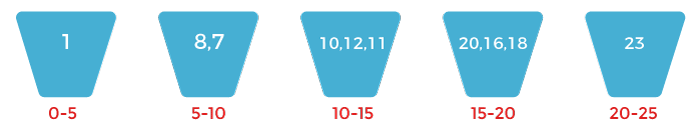
Biz paqirni tartiblash algoritmini scatter-toʻplash usuli orqali tushunishimiz mumkin. Bu erda berilgan elementlar birinchi navbatda chelaklarga sochiladi. Tarqalgandan soʻng, har bir chelakdagi elementlar barqaror tartiblash algoritmi yordamida tartiblanadi. Nihoyat, tartiblangan elementlar tartibda yigʻiladi. Paqir boʻyicha saralash jarayonini tushunish uchun tartiblanmagan massivni olaylik. Bir misol orqali chelak tartibini tushunish osonroq boʻladi.

Massivning elementlari - boʻlsin.

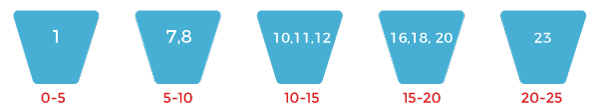
chelak turi

Endi 0 dan 25 gacha boʻlgan chelaklarni yarating. Chelaklar diapazoni 0-5, 5-10, 10-15, 15-20, 20-25. Elementlar chelaklar oraligʻiga muvofiq chelaklarga kiritiladi. Aytaylik, elementning qiymati 16 ga teng, shuning uchun u 15-20 oraligʻidagi chelakka kiritiladi. Xuddi shunday, massivning har bir elementi mos ravishda kiritiladi.

Bu faza massiv elementlarining sochilishi ekanligi maʻlum.



Endi har bir chelakni alohida tartiblang . Har bir chelakning elementlari har qanday barqaror tartiblash algoritmlaridan foydalangan holda saralanishi mumkin.



Nihoyat, har bir chelakdan tartiblangan elementlarni tartibda **yigʻing**

chelak turi

Endi massiv toʻliq tartiblangan!

*using System;*

*using System.Collections.Generic;*

*using System.ComponentModel;*

*using System.Data;*

*using System.Drawing;*

*using System.Linq;*

*using System.Text;*

*using System.Threading.Tasks;*

*using System.Windows.Forms;*

*namespace Bucket\_sort*

*{*

*public partial class Form1 : Form*

*{*

*public Form1()*

*{*

*InitializeComponent();*

*}*

*private void button1\_Click(object sender, EventArgs e)*

*{*

*Graphics g = this.CreateGraphics();*

*Pen pen = new Pen(Color.Red, 3);*

*g.DrawRectangle(pen, 29, 269, 200, 220);*

*g.DrawRectangle(pen, 299, 269, 200, 220);*

*g.DrawRectangle(pen, 579, 269, 200, 220);*

*timer1.Start();*

*timer2.Start();*

*int[] arr = { 3, 6, 2, 9, 1, 5, 4, 8, 7 }; // Buket saparlarimizni sinash uchun massivimiz*

*int maxVal = arr.Max(); // massivning eng katta qiymatini topamiz*

*List<int>[] bucket = new List<int>[maxVal + 1]; // Buketlarimizni sinash uchun List tipidagi ro'yxatimiz*

*for (int i = 0; i < bucket.Length; i++)*

*{*

*bucket[i] = new List<int>();*

*}*

*for (int i = 0; i < arr.Length; i++)*

*{*

*bucket[arr[i]].Add(arr[i]);*

*}*

*int index = 0;*

*foreach (List<int> bucketList in bucket)*

*{*

*if (bucketList.Count > 0)*

*{*

*bucketList.Sort(); // Buketlarimizni tartiblash*

*foreach (int num in bucketList)*

*{*

*arr[index] = num;*

*index++;*

*}*

*}*

*}*

*// Natijani chiqarish*

*string result = "";*

*foreach (int num in arr)*

*{*

*result += num.ToString() + " ";*

*}*

*MessageBox.Show(result);*

*}*

*string[] a;*

*private void textBox1\_TextChanged(object sender, EventArgs e)*

*{*

*a = textBox1.Text.Split();*

*if (a.Length == 7)*

*{*

*MessageBox.Show("Siz faqatgina 6ta element kiritishingiz mumkin", "Xabarnoma");*

*textBox1.Enabled = false;*

*button1\_Click(sender, e);*

*}*

*}*

*int aa = 10;*

*private void timer1\_Tick(object sender, EventArgs e)*

*{*

*buttonEllipse1.Location = new Point(buttonEllipse1.Location.X+2, buttonEllipse1.Location.Y+aa);*

*if (buttonEllipse1.Location.Y==100)*

*{*

*timer1.Stop();*

*aa = 0;*

*MessageBox.Show("tugadi", "tuxtamadimi");*

*}*

*}*

*private void timer2\_Tick(object sender, EventArgs e)*

*{*

*button2.Location = new Point(button2.Location.X + 2, button2.Location.Y + aa);*

*if (button2.Location.Y == 100)*

*{*

*timer2.Stop();*

*aa = 0;*

*MessageBox.Show("tugadi", "tuxtamadimi");*

*}*

*}*

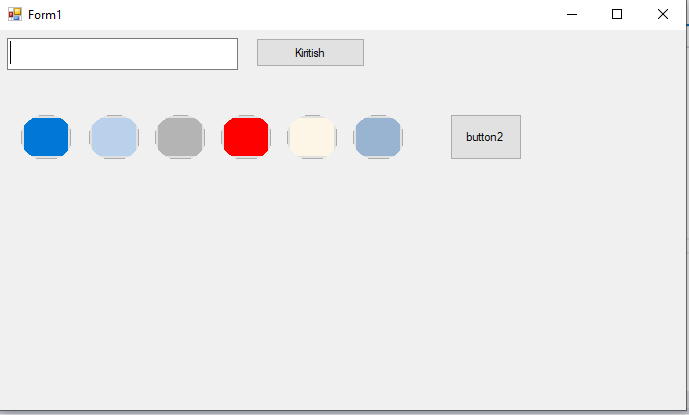
*private void Form1\_Load(object sender, EventArgs e)*

*{*

*}*

*}*

*}*



Foydalanilgan adabiyotlar

1. Алгоритмы в современной мутематике и ее приложениях. Материалы международного симпозиума. Ургенч. 1979 г. Под редакцией А.П. Ершова и Д. Кнута. 8-с.
2. Paul E. Black "Postman's Sort" from Dictionary of Algorithms and Data Structures at NIST.
3. Robert Ramey '"The Postman's Sort" C Users Journal Aug. 1992
4. NIST's Dictionary of Algorithms and Data Structures: bucket sort

Foydalanilgan resurlar

1. <https://www.geeksforgeeks.org/lower-bound-on-comparison-based-sorting-algorithms/>
2. <https://www.programiz.com/dsa/bucket-sort>
3. <https://www.javatpoint.com/bucket-sort>
4. <https://xlinux.nist.gov/dads/HTML/bucketsort.html>