O’ZBEKISTON RESPUBLIKASI

OLIY VA O’RTA-MAXSUS TA’LIM VAZIRLIGI

ALISHER NAVOIY NOMIDAGI SAMARQAND DAVLAT UNIVERSITETI

MEXANIKA-MATEMATIKA FAKULTETI

AMALIY MATEMATIKA VA INFORMATIKA BO’LIMI

5110700-INFORMATIKA O’QITISH METODIKASI YO’NALISHI

„C++ TILIDA MASSIVLARNI TEZKOR SARALASH USULLARI “

MAVZUSIDAN

KURS ISHI

206 guruh talabasi: IsomovJ

Kurs ishi rahbari: Nazarov F

SAMARQAND-2016

Mundarija

Kirish

Dastur so‘zi ham komandalarning alohida blokini (berilgan kodini) aniqlovchi so‘z, ham yaxlit holdagi bajariluvchi dasturiy mahsulotni belgilovchi so‘z sifatida ishlatiladi. Dasturlashga talabni o‘zgarishi nafaqat tillarning o‘zgarishiga balki uni yozish texnologiyasini ham o‘zgarishiga olib keldi. Dasturlash evolyusiyasi tarixida ko‘pgina bosqichlar bo‘lishiga qaramay biz bu kursimizda protsedurali dasturlashdan ob’ektlarga mo‘ljallangan dasturlashga o‘tishni qaraymiz.

Keyingi yillarda amaliy dasturchilarga juda ko‘p integratsion dastur tuzish muhitlari taklif etilayapti. Bu muhitlar u yoki bu imkoniyatlari bilan bir–biridan farq qiladi. Aksariyat dasturlashtirish muhitlarining fundamental asosi C++ tiliga borib taqaladi.

Vaqt o‘tishi bilan dasturchilar oldiga quyilgan masalalar o‘zgarib boryapti. Bundan yigirma yil oldin dasturlar katta hajmdagi ma’lumotlarni qayta ishlash uchun tuzilar edi. Bunda dasturni yozuvchi ham, uning foydalanuvchisi ham kompyuter sohasidagi bilimlar bo‘yicha professional bo‘lishi talab etilardi. Hozirda esa ko‘pgina o‘zgarishlar ro‘y berdi. Kompyuter bilan ko‘proq uning apparat va dasturiy ta’minoti, haqida tushunchalarga ega bo‘lmagan kishilar ishlashyapti. Kompyuter odamlar tomonidan uni, chuqur o‘rganish vositasi emas, ko‘proq o‘zlarining oldilariga qo‘yilgan, o‘zlarining ishlariga tegishli bo‘lgan muammolarini echish instrumenti bo‘lib qoldi.

Foydalanuvchilarning ushbu yangi avlodini dasturlar bilan ishlashlarini osonlashtirilishi bilan bu dasturlarning o‘zini murakkabligi darajasi oshadi. Zamonaviy dasturlar - foydalanuvchi bilan do‘stona munosabatni yuqori darajada tashkil qiladigan ko‘p sondagi oynalar, menyu, muloqot oynalari va vizual grafikaviy muhitlardan tarkib topgan interfeysga ega bo‘lishi lozim.

I . C++ tilida massivlarni tezkor saralash usullari .

1.1)Massivlar va ularning berilish usullari.

Ko’p hollarda, bir necha sonlarni yoki o’zgaruvchilarni bitta nom ostida tizib chiqishga to’g’ri keladi. Buning uchun quyidagicha tartibda ish bajariladi. Bu sonlar bitta to’plamga (massivga) tizib chiqiladi va unga biror nom beriladi. Bu to’plam (massiv)ga kirgan har bir o’zgaruvchi massivning elementi deb yuritiladi. Massivning elementlari massivda tizilgan o’rniga qarab nomerlab chiqiladi va bu nomerlar massiv elementlarining indeksi deb yuritiladi. Massivga murojaat qilinayotganda, massiv nomi ko’rsatiladi va uning yoniga [] qavslar ichiga bu massivning jami elementlar soni ham yozib qo’yiladi. Massivga kirgan har bir o’zgaruvchi ham massiv nomi bilan yuritiladi va nom yoniga uning bu massivdagi tartib raqami ham yozib qo’yiladi. Massivlardan dastur tuzishda foydalanish uchun ularni dastlab e'lon qilish va kerak bo'lsa massiv elementlarini initsalizatsiya qilish, ya’ni tavsiflash kerak bo’ladi. Massiv e'lon qilinganda, kompilyator elementlar soniga teng miqdorda xotiradan joy ajratadi. Masalan, char tipidagi o’zgaruvchili, m nomli massiv quyidagicha e'lon qilinadi: Massivlar dasturlashda eng ko'p qo'laniladigan ma'lumot turlari hisoblanadi. Bundan tashqari ma’lumot tuzilmalari turlicha o'zgaruvchilardan tashkil topgan bo'lishi mumkin. Bunday ma’lumot tuzilmasi klass deb nomlanadi. Masalan bunday tuzilmada odam ismi va yoshi bo'lishi mumkin.
Massivlar xotirada ketma-ket joylashgan, bir tipdagi o'zgaruvchilar guruhidir. Alohida bir o'zgaruvchini ko'rsatish uchun massiv nomi va kerakli o'zgaruvchi indeksini yoziladi. C++ dasturlash tilida massivlardagi elementlar indeksi har doim noldan boshlanadi. [] qavslar ichidagi indeks butun son yoki butun songa olib keluvchi ifoda bo'lmog'i kerak. Oxirgi element indeksi n-1 bo'ladi (n - massiv elementlari soni). Bir vaqtning o’zida bir necha massivni e'lon qilish ham mumkin: Massiv bu bir tipli nomerlangan ma’lumotlar jamlanmasidir. Massiv indeksli o’zgaruvchi tushunchasiga mos keladi. Massiv ta’riflanganda tipi, nomi va indeksi chegarasi ko’rsatiladi. Misol uchun long int a[5]; char w[200]; double f[4][5][7]; char [7][200]. Massiv indekslar har doim 0 dan boshlanadi. C++ tilida standarti bo’yicha

indekslar soni 31 tagacha bo’lishi mumkin, lekin amalda bir o’lchovli massivlar qo’llaniladi. Bir o’lchovli massivlarga matematikada vector tushunchasi mos keladi. Massivning int z[3] shaklidagi ta’rifi, int tipiga tegishli z[0], z[1], z[2] elementlaridan iborat massivni aniqlaydi. Massivlar ta’riflanganda inistilizatsiya qilinishi, ya’ni boshlang’ich qiymatlari ko’rsatilishi mumkin. Misol uchun: float C[]={1,-1,2,10,-12.5};Bu misolda massiv chegarasi avtomatik aniqlanadi. Agar massiv inistilizatsiya qilinganda elementlar chegarasi ko’rsatilgan bo’lsa, ro’yhatdagi elementlar soni bu chegaradan kam bo’lishi mumkin emas. C++ tilida massivlar quyidagicha berilishi mumkin so’zlar massivlari, ko’rsatgichlar massivlari, funksiya va massivlar, simvolli massivlar, satrli massivlar, ko’rsatgich va massivlar.C++ tilida so’zlar massivlari ikki o’lchovda simvolli massivlar sifatida ta’riflaadi. Misol uchun :Char name [4][5].Bu ta;rif yordamida har biri5 ta harfdan iborat bo’lgan 4 ta so’zli massiv kiritiladi.So’zlar massivlari quyidagicha inistilizatsiya qilinishi mumkin:Chat Name [3][8]={“anvar”,”mirkomil”,”yusuf”}. Bu ta’rifda har bir so’z uchun hotiradan 8 bayt joy ajratiladi va har bir so’z ohiriga ‘\0’ belgisi qo’yiladi so’zlar massivlari inistilizatsiya qilinganda so’zlar soni ko’rsatilmasligi mumkin

1.2.Massivni tezkor saralash usullari.

Ichki saralash modeli. Ma’lumotlar EHM xotirasida muayyan tartibda saqlanadigan bo‘lsa, axborotga ishlov berish va uni izlash bilan bog‘liq ko‘p masalalar oddiyroq, tezroq va samaraliroq hal qilinadi. Bir qator hollarda ma’lumotlarning tartibga solinganligidan foyda aniq bo‘lib, maxsus isbotlashlarni talab etmaydi. Agar lug‘at yoki telefon ma’lumotnomasida so‘zlar va familiyalar alifbo tartibida joylashtirilmaganda ulardan foydalanish qanchalik qiyin bo‘lishini tasavvur etish mumkin. Lekin ma’lumotlarni tartiblash zaruriyati masalasi har safar muayyan vazifaga nisbatan hal qilinishi zarur. Bunda tashqi xotira qurilmalari imkoniyatlari, operativ xotira hajmi, ma’lumotlarga murojaat qilish tezligi, ularni yangilab turish tezligi va ishlov berish xarakteri kabilarni tahlil qilish zarur.

Turli ilovalarda tartibga solishning turli mezonlaridan foydalaniladi. Ma’lumotlar ularga murojaat qilish ehtimolining qiymati, qancha tez-tez murojaat

etib turilishiga ko‘ra tartibga solinishi mumkin. Odatda, tartibga solish kalit bo‘yicha amalga oshiriladi.

Axborot tizimlari bilan ishlov beradigan ma’lumotlar birligi bir qator axborot maydonlaridan iborat bo‘lgan yozuv hisoblanadi. Kalit bitta yozuv maydoni ichidagi yoki muayyan maydonlar majmuidan iborat bo‘lishi mumkin. Keyingi holda kalit tarkibiy deb ataladi. YOzuv faqat bittagina maydondan iborat bo‘lishi mumkin va bu holda u kalitli hisoblanadi. Tartibga solish natijasida yozuvlar kalitlarning qiymati ortib borishi yoki kamayib borish bo‘yicha joylashadi. Bunday tartibga solish jarayoni tartiblash deb ataladi. Masalan, fakultet talabalari to‘g‘risidagi ma’lumotlardan iborat bo‘lgan yozuvlar talabalarning reyting daftarchalari nomerlari bo‘yicha tartibga solingan bo‘lishi mumkin.

Ichki tartiblashning ko‘plab usullari mavjud va ularning har biri o‘z afzalli texnikalari va texnik kamchiliklariga ega bo‘lib, ma’lumotlar va apparaturaning muayyan konfigurasiyalarida boshqalaridan samaraliroq bo‘lishi mumkin ekan. Tartiblash usullarining tavsiflarini baholash har bir muayyan holatda bu usullardan birini to‘g‘ri tanlash texnik imkonini beradi.

Saralashning sodda sxemalari. Eng sodda tartiblash usullaridan biri bo‘lib «pufakcha» usuli hisoblanadi. Bu algoritmning asosiy g‘oyasini yozish uchun tartiblanishi kerak bo‘lgan yozuvlar vertikal joylashgan massivda saqlanadi deb faraz qilamiz. Kalit maydonning kichik qiymatli yozuvlari «yengil» va shuning uchun pufakcha kabi ular yuqoriga «suzib chiqadi». Massiv bo‘ylab birinchi o‘tishda massivning birinchi yozuvi olinadi va uning kaliti navbatma-navbat keyingi yozuvlarning kalitlari bilan solishtirib boriladi. Agar nisbatan «og‘ir» kalitli yozuvlar uchrasa, u holda bu yozuvlar joyini almashtiradi. Nisbatan «yengil» yozuvlar uchraganda bu yozuv taqqoslash uchun etolon bo‘ladi va keyingi barcha yozuvlar shu kalit bilan solishtiriladi. Natijada eng kichik qiymatli kalit massivning eng yuqorisiga chiqadi. Massiv bo‘ylab ikkinchi o‘tishda massivning massivni birinchi o‘tishda topilgan yozuvdan keyin joylashgan og‘irligi bo‘yicha ikkinchi kalit olinadi. Massiv bo‘ylab ikkinchi va keyingi o‘tishlarda oldingi o‘tishlarda topilgan yozuvlarni ko‘rib chiqish shart emas, chunki ular qolgan yozuvlarga qaraganda kichik kalitlarga ega.

Boshqacha qilib aytganda, t – o‘tishda i pozitsiya yuqorida turgan elementlar tekshirilmaydi. 8.1-dasturda ushbu algoritm keltirilgan.

«pufakcha» algoritmi

```
(1) for i:= I to n - 1 do
(2)      for j:= 1 downto i + 1 do
(3)       if A[j].key < A[j - 1].key then
(4)            swap(A[j], A[j - 1])
```

swap protsedurasi yozuvlarning o‘rnini almashtirish uchun ko‘plab tartiblash algoritmlarida ishlatiladi, uning kodi quyidagi dasturda keltirilgan.

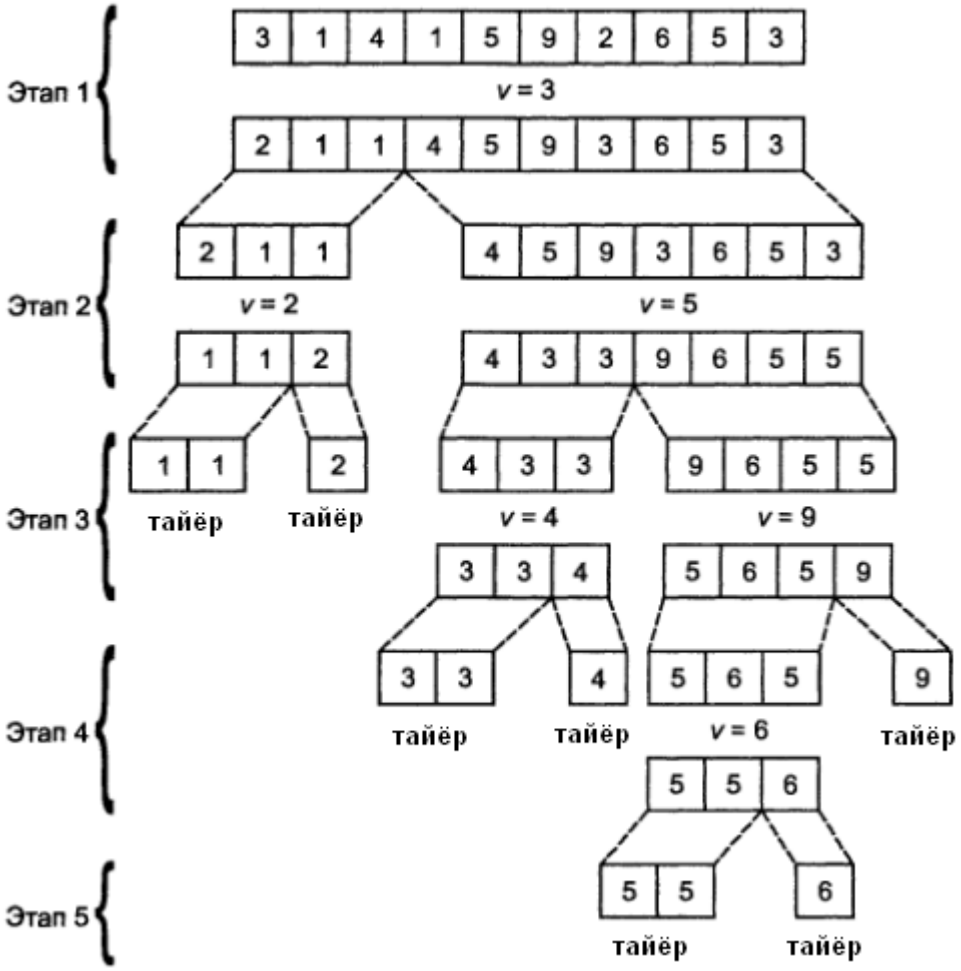
8.1-dastur. swap protsedurasi

```
procedure swap ( var x, u: recordtype )
{swap x va u yozuvlarning o‘rnini almashtiradi}
var
temp: recordtype;
begin
temp:= x;
x:= y;
y:= temp;
end; { swap }
```

Tez saralash. O(n log n) vaqtda bajariladigan, ichki saralash usullarining eng samaradori bo‘lib hisoblangan tez tartiblashni ko‘rib chiqamiz. Bu algoritmda massivning A[1],...,A[n] elementlarini tartiblash uchun bu elementlardan massiv elementlari unga nisbatan tartiblanadigan tayanch element sifatida v kalitning qandaydir qiymati tanlanadi. Qulaylik uchun, tayanch element sifatida kalit qiymatlari taqsimotining medianaga eng yaqin bo‘lganini tanlab olish zarur. Chunki, tayanch element kalit qiymatlarini deyarli teng ikki qismga ajratadi.

Keyin massiv elementlari shunday joylashtiriladiki, qandaydir j indeks uchun barcha A[1], ..., A[j] keltirilgan elementlar υ dan kichik kalit qiymatlarga, A[j+1], ..., A[n] barcha elementlari υ ga teng yoki katta kalit qiymatlarga ega bo‘ladi. Keyin tez tartiblash protsedurasi A[1], ... A[j] va A[j+1], ..., A[n] elementlar to‘plamiga bu

to‘plamlarni alohida tartiblash uchun rekursiv ishlatiladi. Birinchi to‘plamning kalit qiymatlari ikkinchi to‘plamning kalit qiymatlaridan kichik bo‘lgani uchun boshlang‘ich massiv to‘g‘ri tartiblanadi.



8.1-rasm. Tez tartiblash algoritmi etaplari.

Misol. 8.1-rasmda 3, 1, 4, 1, 5, 9, 2, 6, 5, 3 butun sonlar ketma-ketligi ustida bajariladigan tez tartiblash algoritmining bajarilish qadamlari keltirilgan. Har bir qadamda υ ning qiymati eng chapdagi turli xil ikkita element-sonning kattasi sifatida tanlanadi. Tartiblash protsedurani rekursiv chaqirish jarayonida alohida qism to‘plamlar bir xil kalitdan tashkil topganda tugatiladi. 8.1-rasmda har bir etap ikki qadamda ko‘rsatilgan: avval har bir alohida qism to‘plam uchun o‘zining qiymati tanlanadi va keyin bu qism to‘plamning elementlari tanlangan qiymatga mos ravishda joylashtiriladi, va xuddi shunday tartiblash protsedurasi rekursiv ishlatiladigan yana ikkita qism to‘plamga bo‘linadi.

Endi quicksort protsedurasidan tashqarida aniqlangan A massiv elementlari bilan ishlaydigan quicksort rekursiv protsedurasini ishlab chiqishni boshlaymiz. Quicksort(i,j) protsedurasi A[1], ..., A[n] elementlarni tartiblashi kerak. Bu protseduraning algoritmi 8.2-dasturda kiritilgan.

8.2-dastur. Tez tartiblash usuli protsedurasi

(1) if A[i], ..., A[j] ikkita turli xil kalitlarga ega bo‘lsa then begin

(2) v — topilgan ikkita turli xil kalitlarning kattasi bo‘lsin;

(3) A[x], ..., A[j] elementlar shunday joylashtiriladiki, qandaydir k, i+l<k<j uchun A[i], ..., A[k-1] lar v dan kichik, A[k], ..., A[j] esa v dan katta yoki teng kalitga ega bo‘lsin;

(4) quicksort{i, k-1) ,-

(5) quicksort{k, j)

end

Endi tez tartiblash algoritmining eskizini (8.2-dastur) haqiqiy quicksort dasturiga aylantiramiz. Bu dasturning kodi 8.3-dasturda keltirilgan. aggau[1..n] of recordtype turidagi A massivni saralash uchun quicksort(l, n) protsedurasini chaqirish kerak.

8.3-dastur. Tez tartiblash usuli protsedurasi

```
procedure quicksort ( i, j: integer );
{A tashqi massivning A[i],...,A[j] elementlarini tartiblaydi}
var
pivot: keytype;
pivotindex: integer;
{kaliti pivot ga teng bo‘lgan A massiv elementi}
k: integer;
{kalit>pivot bo‘lgan elementlar guruhining boshlang‘ich indeksi}
begin
(1)          pivotindex:= findpivot{i, j) ;
(2)          if pivotindex <> 0 then begin
{agar barcha kalitlar teng bo‘lsa, hech narsa bajarish kerak emas}
(3)                  pivot:= Alpivotindex] .key;
```

```
(4)                 k:= portition(i, j, pivot);
(5)                 quicksortd, k-1) ;
(6)                 quicksort{k, j)
end
end; {quicksort}
```

“CHo‘ntak” saralash. Ko‘p hollarda O(n log n) dan kam bo‘lgan tartiblash vaqtini olish mumkin, lekin tartiblanayotgan kalitlar haqida qo‘shimcha ma’lumot kerak bo‘ladi.

8.5-misol. Faraz qilamiz, kalit qiymatlar 1 dan to n gacha bo‘lgan butun sonlar bo‘lib hisoblansin, ular takrorlanmaydi va tartiblanayotgan elementlar soni ham n ga teng. Agar A va V orqali aggau[1..n] of recordtype massivni ifodalansa, tartiblanishi kerak bo‘lgan n ta element birinchi A massivda joylashadi, u holda V massivga kalitlarni quyidagi qiymatlarda navbatma-navbat joylashtirib borishni tashkillashtirish mumkin:

```
for i:= 1 to n do
V[A[i].key]:= A[i];          (8.4)
```

Bu kod A[i] element V massivning qaerida joylashishini hisoblab beradi. Butun bir sikl O(n) vaqt tartibiga ega va barcha kalitlar qiymatlari turli xil bo‘lganda va 1 dan to n intervaldagi butun sonlar bo‘lganda yaxshi ishlaydi.

O(n) vaqtda A massiv elementlarini, lekin ikkinchi V massivni ishlatmasdan tartiblaydigan boshqa usullar ham mavjud. Navbatma-navbat A[1], ..., A[n] elementlarga murojaat qilamiz. Agar A[i] yacheyka j kalitga ega bo‘lsa va j≠i bo‘lsa, u holda A[i] va A[j] yacheykalardagi yozuvlar o‘rnini almashtiradi. Agar bu almashtirishdan keyin A[i] yacheyka k kalitga ega bo‘lsa va k≠i bo‘lsa, u holda A[i] va A[k] orasida o‘rin almashtirishlar bo‘ladi va h.k. Har bir o‘rin almashtirish hech bo‘lmaganda bitta yozuvni kerakli tartibda joylashtiradi. SHuning uchun A massiv elementlarini tartiblaydigan ushbu algoritm O(n) bajarilish vaqtiga ega bo‘ladi.

```
for i:= 1 to n do
while A[i].key <> i do
swap(A[i], A[A[i].key]);
```

(8.4) dastur — bu «cho‘ntak» saralashga oddiy misol, bu erda aniqlangan qiymatli kalitlarni saqlash uchun «cho‘ntak» lar ishlatadi. Berilgan yozuv r qiymatli kalitga ega bo‘lsa, u holda bu yozuvni yozuvlar uchun «cho‘ntak»ga joylashtiramiz. Dasturda (8.4) «cho‘ntak» bo‘lib V massivning elementlari hisoblanadi, bu erda B[i] – i qiymatli kalitga ega bo‘lgan yozuvlar uchun «cho‘ntak». «Cho‘ntak» sifatidagi massivlarin faqat oddiy hollarda ishlatish mumkin (bitta «cho‘ntak»da bittadan ortiq yozuv bo‘lmagan taqdirda). Bundan tashqari, massivlarni «cho‘ntak» lar sifatida ishlatishda «cho‘ntak» ichida elementlarni tartiblash zaruriyati paydo bo‘lmaydi, chunki bitta «cho‘ntak» bittadan ko‘p bo‘lmagan yozuvdan iborat, algoritm shunday yaratilganki, massivdagi elementlar to‘g‘ri tartibda joylashadi.

Lekin umumiy holda bitta «cho‘ntak» da bir nechta yozuvlarni saqlash, shuningdek bir nechta «cho‘ntak»larni bittaga birlashtirish mumkinligini e’tiborga olish kerak. Aniqlik uchun A massiv elementlari recordtype ma’lumotlar turiga, yozuv kalitlari esa – keytype turiga ega deb hisoblaymiz. recordtype turidagi ro‘yxat elementlarini listtype (ro‘yxat turi) ma’lumotlar turi orqali ifodalab olamiz. Listtype turi ro‘yxatning ixtiyoriy turi bo‘lishi mumkin, lekin hozirgi holatda bizga ko‘proq bog‘langan ro‘yxatlar to‘g‘ri keladi. Ro‘yxatning umumiy uzunligi fiksirlangan va n ga teng deb faraq qilishimiz mumkin.

Shuningdek array[keytype] of listtype turidagi V massiv ham kerak. Bu ro‘yxatlarni saqlovchi massiv «cho‘ntak»i bo‘lib hisoblanadi. V massivning indeksi keytype ma’lumotlar turiga ega, chunki bu massivning har bir elementi kalitning bitta qiymatiga mos keladi. SHunday qilib, 8.9-dasturni umumlashtirish mumkin, chunki endi «cho‘ntak»lar keraklicha hajmga ega. «Cho‘ntak»larni qanday birlashtirish mumkinligini ko‘rib chiqamiz. a_1, a_2, ... a_i va b_1, b_2, … b_j ro‘yxatlar ustida ro‘yxatlar konkatenatsiyasi operatsiyasini bajarish kerak, natijada a_1, a_2, ... a_i i b_1, b_2, … b_j ro‘yxatga ega bo‘lamiz. L_1L_2, ro‘yxatlarning birlashmasidan hosil bo‘lgan L_1 ro‘yxatni egallovchi konkatenatsiya CONCATENATED(L_1, L_2) operatsiyasini tadbiq etish uchun ro‘yxatlarning ixtiyoriy ifodalanishidan foydalanish mumkin.

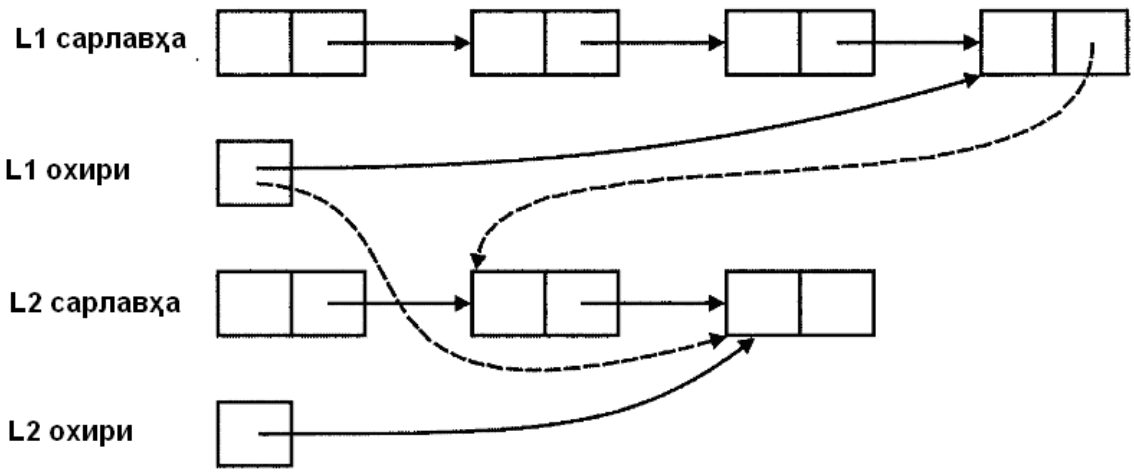
Ro‘yxat sarlavhalariga qo‘shishda konkatenatsiya operatori yanada samarali bajarilishi uchun ro‘yxatning oxirgi elementlariga ko‘rsatkichlarni ishlatish mumkin.

8.4-rasmda punktir chiziqlar bilan L_1 i L_2 ro‘yxatlarni bitta L_1 ro‘yxatga konkatenatsiyasida o‘zgargan ko‘rsatkichlar ko‘rsatilgan.

Endi yozuvlarning «cho‘ntak» saralashining dasturin tuzsak bo‘ladi. Bu dastur 8.5-dasturda ko‘rsatilgan, dastur ro‘yxatlar ustida bajariladigan bazaviy operatorlarni ishlatib tuzilgan.

8.5-dastur. binsort dasturi (cho‘ntak saralash)

```
procedure binsort;
{A massiv elementlarini tartiblaydi, tartiblangan ro‘yxatni V[lowkey] «cho‘ntak»ga yozadi}
var
i: integer;
v: keytype;
begin
{"cho‘ntak"ka yozuvlarni kiritish boshlanadi}
(1)          for i:= 1 to n do
{ A[i] elementni «cho‘ntak» boshiga joylashtirish}
(2)          INSERT(A[i], FIRST(B[A[i].key]), B[A[i].key]);
(3)          for v:= succ(lowkey) to highkey do
(4)          CONCATENATE(B[lowkey] , B[v])
end; { binsort }
```



8.4-rasm. Bog‘langan ro‘yxatlar konkatenatsiyasi

Taqqoslanma saralashlarning bajarilish vaqtlari. Qo‘yish usuli. Agar ichki siklga qarasak yozuvlarning tartiblangan qismiga qo‘shilgan element qolgan elementlardan kichik bo‘lsa operasiyalar eng ko‘p bajariladi. Bu holda location o‘zgaruvchisi 0 ga teng bo‘lganda sikl o‘z ishini tugatadi. SHuning uchun yangi element massiv boshiga qo‘shilganda algoritm eng ko‘p bajariladi. Bunday holat joriy massivning elementlari kamayish tartibida joylashgan bo‘lsa bo‘lishi mumkin. Bu yomon holatlardan biridir.

Bunday massivni qayta ishlash jarayoni qanday bo‘lishini ko‘rib chiqamiz. Birinchi massivning ikkinchi elementi qo‘yiladi. U faqat bitta element bilan solishtiriladi. Ikkinchi qo‘yiladigan element (tartib buyicha uchinchi) oldingi ikkita element bilan, uchinchi qo‘yilgan element oldingi uchta element bilan solishtiriladi. Umuman olganda i - qo‘yiladigan element oldingi i ta element bilan solishtiriladi va bu jarayon N-1 marta takrorlanadi. SHunday qilib, qo‘yish usulida tartiblashning qiyinligi yomon holatda quyidagicha bo‘ladi.

$$W(N) = \sum_{i=1}^{N-1} i = \frac{(N-1)N}{2} = \frac{N^2 - N}{2}$$

$$W(N) = O(N^2)$$

O‘rta holatni ikkita etapga bo‘lamiz. Avval navbatdagi elementning joyini aniqlash uchun kerak bo‘ladigan tenglashtirishlarning o‘rtacha sonini hisoblaymiz. Keyin birinchi qadamdan foydalanib barcha kerakli operasiyalarning o‘rtacha sonini hisoblaymiz. ***i*** elementning joyini aniqlash uchun kerak bo‘ladigan tenglashtirishlarning o‘rtacha sonini hisoblashdan boshlaymiz. Avval aytib o‘tilganidek, ***i*** - elementning massivga qo‘shilishi kerakli joyda turgan bo‘lsa ham kamida bir marta tenglashtirishlarni talab qiladi. Tengliklar quyidagilarni beradi.

$$A(N) \approx \frac{(N-1)N}{2} + (N-1) - (\ln N - 1)$$

Almashtirish usuli tahlili. Biz almashtirish usuli algoritmining qisqacha tahlilini ko‘rib chiqamiz. Qaysi holatda bajariladigan ishning hajmi minimal darajada bo‘ladi. ***For*** siklining birinchi aylanishida to‘liq bajarilishi kerak, shuning uchun algoritmda kamida N-1 ta tenglashtirish ko‘rib chiqiladi.

Yaxshi holatda N-1 ta tenglashtirishlar bajariladi. Bu degani massivdagi elementlar to‘g‘ri tartibda joylashgan. Massiv elementlari teskari tartibda joylashgan bo‘lsa, tenglashtirishlar necha marta bajariladi. Birinchi aylanishda N-1 ta, ikkinchi aylanishda N-2 ta tenglashtirishlar bajariladi. Keyingi jarayonlar shuni ko‘rsatadiki, navbatdagi har bir aylanishda tenglashtirishlar soni birga kamayib boradi. SHuning uchun bu holatda qiyinlik quyidagi formula bilan hisoblanadi.

$$W(N) = \sum_{i=N-1}^{1} i = \sum_{i=1}^{N-1} i = \frac{(N-1)N}{2} = \frac{N^2 - N}{2}$$

$$\approx \frac{1}{2} N^2 = O(N^2).$$

SHell usuli. Tartiblash algoritmlarini tahlil qilishda ayrim hollarda inversiya miqdorini hisoblaymiz. Inversiya – bu noto‘g‘ri tartibda keladigan massivning ikkita elementi. Masalan, (3,2,4,1) massivda 4 ta inversiya mavjud, ular (3,4), (3,1), (2,1), (4,1). Massiv elementlari teskari tartibda joylashgan bo‘lsa, inversiya eng ko‘p bo‘ladi, uning soni $(N^2-N)/2$.

Tartibli statistika. Tartibli statistikani hisoblash masalasi quyidagidan iborat: n ta elementdan iborat ro‘yxat va k butun son berilgan, o‘sish tartibida tartiblangan ro‘yxatda k-pozitsiyada turgan yozuv kalitini topish kerak. Qisqartirib bu masalani «k-tartibli statistikani topish masalasi» deb ataymiz. Bu masalaning maxsus hollari k=1 (eng kichik elementni topish), k=n (eng katta elementni topish) va agar n toq bo‘lsa, k=(n+1)/2 (medianani topish) da paydo bo‘ladi.

Ayrim hollarda tartibli statistikani hisoblash masalasi chiziqli vaqtda topiladi. Masalan, minimal (maksimal) elementni topish O(n) tartibli vaqtni talab etadi. Agar k≤n/logn bo‘lsa, u holda k- tartibli statistikani topish uchun **kucha** ni qurish va keyin undan O(n+k logn)=O(n) vaqtda k ta eng kichik elementlarni tanlash kerak. Xuddi shunday k≥n-n/logn bo‘lganda O(n) vaqt ichida k-tartibli statistikani topish mumkin.

Tez tartiblash variantlari. O‘rta holatda k-tartibli statistikani topish uchun tez tartiblash usuliga asoslangan protsedurani rekursiv ishlatiladi. Bu protsedurani select (tanlash) deb nomlaymiz. select(l, j, k) protsedurasi qandaydir katta A[1], ..., A[n] massivdan olingan A[i], ..., A[j] elementlar orasidan k – elementni topadi. select protsedurasi quyidagi harakatlarni bajaradi.

1. Tayanch elementni tanlash, masalan υ.

2. partition protsedurasini ishlatib, A[i], ..., A[j] elementlarni ikki guruhga bo‘lish. Birinchi A[i], ..., A[m-1] guruhda barcha yozuvlarning kalit υ dan kichik, ikkinchi A[m], ..., A[j] guruhda—υ ga teng yoki katta.

3. Agar k≤m-i bo‘lsa, u holda k-element birinchi guruhda joylashadi va unga yana select(i, m-1, k) protsedurasi qo‘llaniladi. Agar k>m-i bo‘lsa, u holda select(m, j, k-m+ i) protsedurasi chaqiriladi.

Bir xil qiymatlardagi kalitlarga ega bo‘lgan A[i], ..., A[j] elementlar uchun select(i, j, k) protsedurasi ertami-kech chaqiriladi (va shuning uchun i=j bo‘ladi). Bu elementlarning kalitlari izlanayotgan k-tartibli statistika qiymati bo‘ladi.

Tez tartiblash usulidagidek, select protsedurasi eng yomon holatda $\Omega(n^2)$ dan kam bo‘lmagan vaqt talab qiladi. Masalan, agar eng kichik elementni topishda tayanch element sifatida har doim kalit qiymatlaridan eng kattasini olish kerak, u holda bu protsedura uchun bajarilish vaqti $O(n^2)$ tartibda bo‘ladi. Lekin o‘rta holatda select protsedurasi tez tartiblash algoritmiga nisbatan tez ishlaydi, o‘rtacha u O(n) vaqtda bajariladi. Bu algoritmlar orasidagi farq shundan iboratki, tez tartiblash protsedurasi ikki marta chaqirilganda, select protsedurasi bir marta chaqiriladi. Select protsedurasining tahlili tez tartiblash protsedurasining tahlilidek bo‘ladi. Bu protsedura avvalgi qadamda chaqirilgan to‘plamning bir qismi bo‘lib hisoblangan qism to‘plamda o‘zini takroran chaqiradi. Bu protseduraning har bir chaqirilishi protseduraning avvalgi chaqirilishi amalga oshirilgan to‘plamning 9/10 qismidan iborat bo‘lgan elementlar to‘plamida amalga oshiriladi. U holda, agar T(n) orqali n ta elementdan iborat to‘plamda select protsedurasiga ketgan vaqtni belgilasak, qandaydir o‘zgarmaslar uchun quyidagilarga ega bo‘lamiz:

$$T(n)\leq T(9/10^* n) + sn \qquad (8.14)$$

(8.14) T(n) = O(n) teng bo‘lishini ko‘rsatish qiyin emas.

Tartibli statistikani topishning chiziqli usullari. Select protsedurasida yomon holatda bajarilish vaqti O(n) tartida bo‘lishini ko‘rsatish uchun chiziqli vaqtda shunday tayanch elementni tanlash kerakki, o‘lchami boshlang‘ich to‘plamning fiksirlangan qismidan katta bo‘lmagan ikkita qism to‘plamga bo‘linsin. Masalan,

(8.14) tengsizlikni echish shuni ko‘rsatadiki, agar tayanch element (n/10)- tartibdagi elementdan kichik bo‘lmasa va (9n/10)-tartibdagi elementdan katta bo‘lmasa 9/10 boshlang‘ich to‘plamdan oshmaydigan qism to‘plamlarga bo‘ladi, bu esa eng yomon holatda algoritm chiziqli vaqtda bajarilishini kafolatlaydi.

«YAxshi» tayanch elementni quyidagi ikki qadam vositasida topish mumkin.

1. n ta elementni boshqa guruhga kirmagan 1-4 elementlar chetga qo‘yib, 5 ta elementdan iborat guruhlarga bo‘lish. 5 ta elementdan iborat har bir guruh o‘sish tartibida ixtiyoriy algoritm bilan tartiblanadi va har bir guruhdan o‘rtacha element olinadi. Bunday o‘rtacha elementlar jami bo‘lib [n/5] bo‘ladi.

2. select protsedurasi ishlatilib, o‘rtacha elementlarning medianasi topiladi. Agar [n/5] juft bo‘lsa, o‘rtachaga yaqin bo‘lgan element olinadi. Ixtiyoriy holatda o‘rtacha elementlarning tartiblangan ro‘yxatidagi [(n+5)/10] pozitsiyada turgan element topiladi.

8.6-dastur. k- tartibli statistikani topishning chiziqli algoritmi

```
function select (i, j, k: integer ): keytype;
{ Funksiya A[i], ..., A[j] dan k- elementning kalit qiymatini qaytaradi}
var
m: integer; {indeks sifatida ishlatiladi }
begin
(1)                    if j-i<74 then begin
 (2)                   A[i], ..., A[j] ni tartiblash oddiy algoritm bilan;
(3)                    return(A[i+k-1].key)
end
else begin { select rekursiv ishlatiladi}
(4)                 for m:= 0 to (j-i-4) div 5 do
{5-ta elementdan iborat guruhda o‘rtacha elementlarni topish}
(5)                    guruhda kattaligi bo‘yicha 3-bo‘lgan elementni topish
A[i+5*m], ..., A[i+5*m+4] va
uni s A[i+m] bilan o‘rnini almashtirish;
(6)                    pivot:= select(i, i+(j-i-4) div 5,(j-i-4) div 10);
```

```
  {o‘rtacha elementlarning medianasini topish}
  (7)                    m:= partition(i, j, pivot);
  (8)                    if k<=m-i then
  (9)                        return(select(i, m-1, k) )
  else
  (10)                       return(select(m, j, k-(m-i)))
  end
    end; { select }
procedure sort (l,r: integer);
begin
| if (l = r) then begin
| | nichego delat ne nado - uchastok pust
| end else begin
| | vыbrat sluchaynoe chislo s v poluintervale (l,r]
| | b := a[s]
| | perestavit elementы sortiruemogo uchastka tak, chtobы
| |   snachala shli elementы, menshie b - uchastok (l,ll]
| |   zatem elementы, ravnыe b        - uchastok (ll,rr]
| |   zatem elementы, bolshie b       - uchastok (rr,r]
| | sort (l,ll);
| | sort (rr,r);
| end;
end;
```

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
 int  A[100];
 int i, j, n, b, k, ll, rr;
void Sort(int l, int r)
{  if ((l==r) || (l==0) || (r==0))   {}
 else
  { b=A[l+1]; ll=0;
   for (i=1; i<=r; i++)
    if (b>A[i])
     { ll=ll+1;
      k=A[ll];
      A[ll]=A[i];
      A[i]=k;
     }
    rr=ll;
   for (i=ll; i<=r; i++)
    if (b==A[i])
     { rr=rr+1;
      k=A[rr];
      A[rr]=A[i];
      A[i]=k;
     }
    Sort(l, ll);
   Sort(rr, r);
  }
}
```

```
int main()
{  clrscr();
 cout << "A (NxN) massiv o`lchamini kiriting: N = ";
  cin >> n;
 for (i=1; i<=n; i++)
  { cout << " A ["<< i << "] = ";
   cin >> A[i];
  }
 cout << "Dastlabki massiv elementlari: \n";
  for (i=1; i<=n; i++) cout << A[i] << " ";
   cout << endl;
 Sort(1, n);
 cout << "Saralangan massiv elementlari: \n";
   for (i=1; i<=n; i++) cout << A[i] << " ";
  return 0;
}
```

Test (Ishlashi):

(NxN) massiv o`lchamini kiriting: N = 5

A [1] = 5

A [2] = 2

A [3] = 1

A [4] = 3

A [5] = 4

Dastlabki massiv elementlari:

5 2 1 3 4

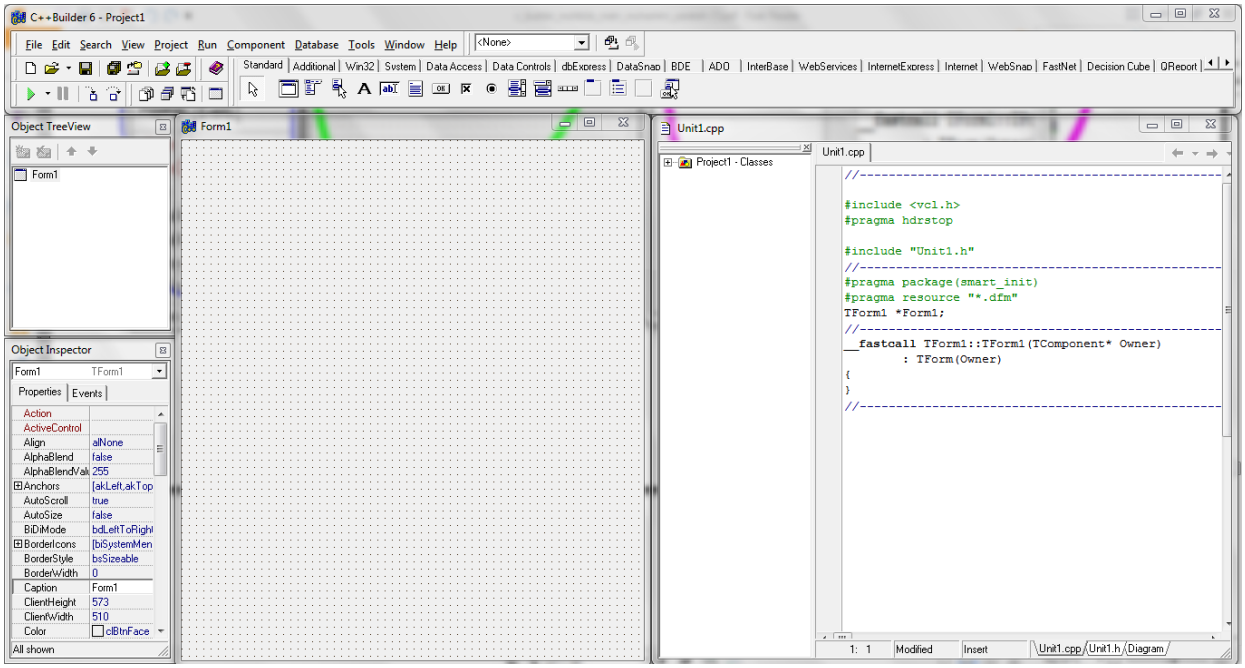
Saralangan massiv elementlari:

1 2 3 4 5

II . C++ tilida massivlarni saralash usullarini dastruy taminoti.

2.1)C++tilida massivlarni tez saralash usullari dasturiy ta’minoti

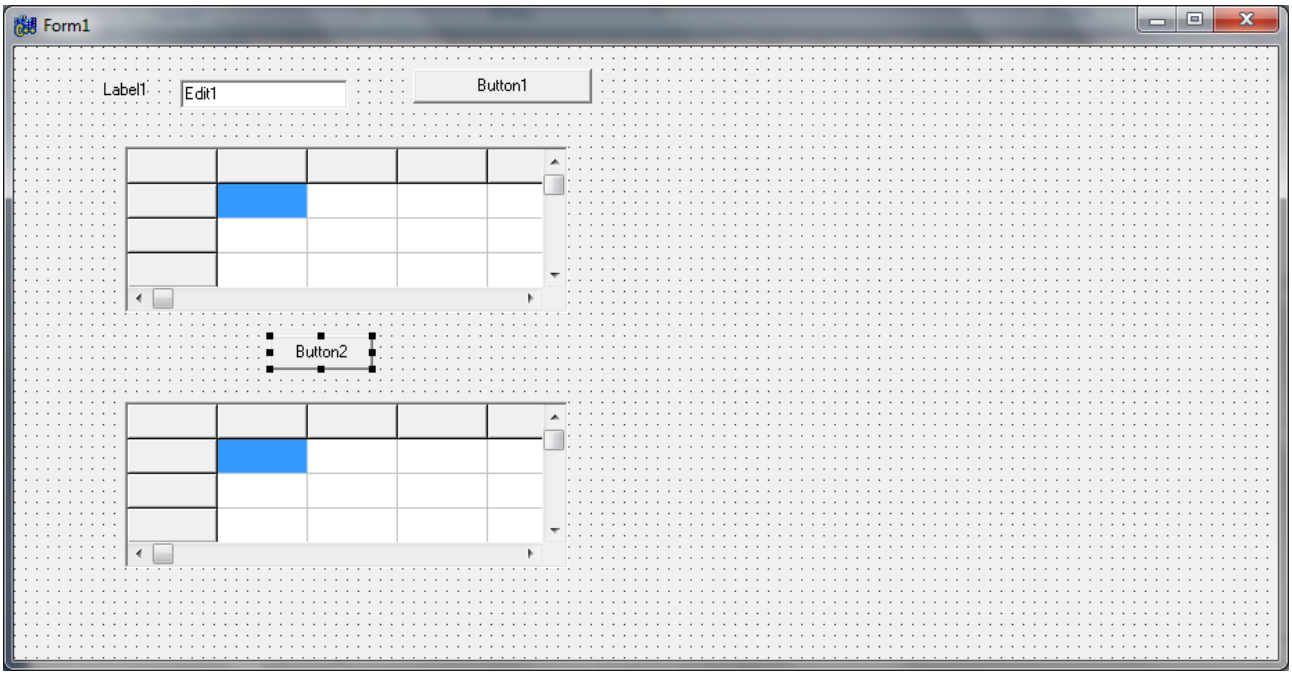
Dastur tuzish muhiti: Dasturni C++ Builder muhitida tuzishni ko’rib chiqamiz. Biz ishlaydigan muhitning boshlang’ich ko’rinishida bosh menyu, Forma, Unit qismi, obyektlar sxemasini, ko’rsatish qismi va obyekt tahrilagich bo’limchalaridaniborat bo’ladi. Yani quyidagi oynalar bo’ladi:



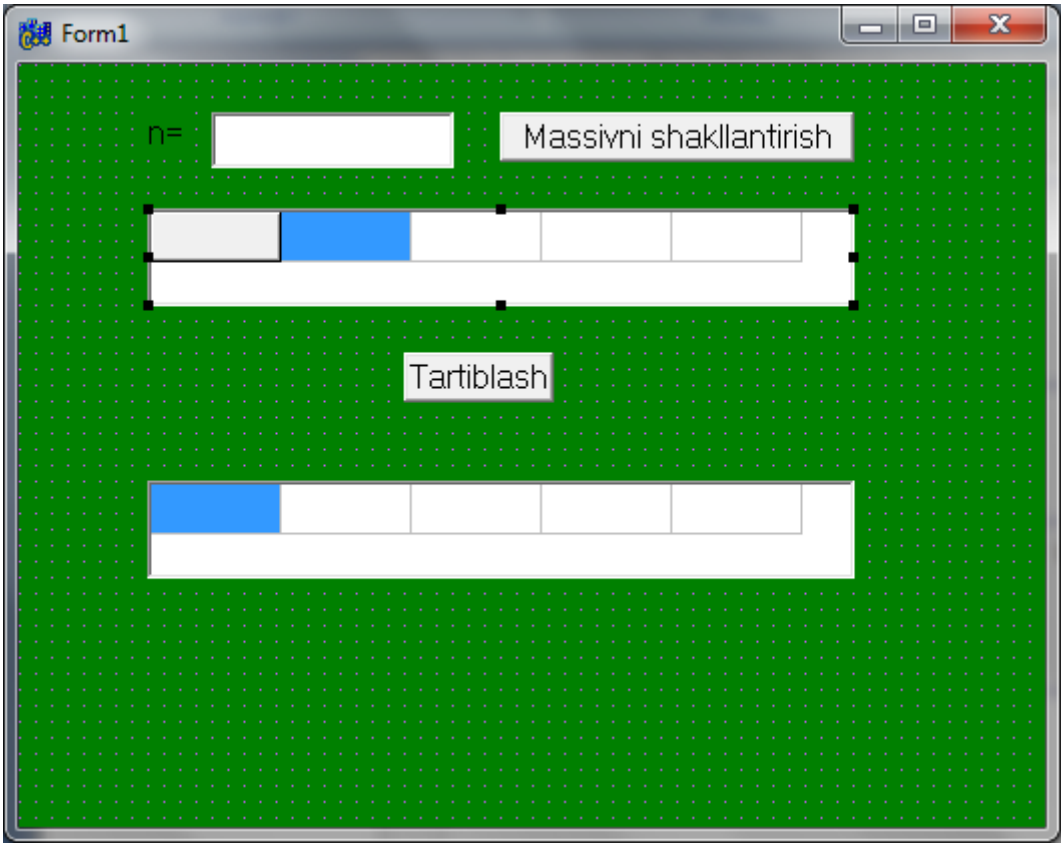
Dasturni tuzish jarayoni.

Dastlab C++ Builder dasturlash tilini ishga tayyorlab olamiz. Bu dasturni uchun quyidagi ketma- ketlikda C++ Builder 6 dasturiga kiramiz: **Пуск ->Всепрограмми -> Borland C++ Builder 6 -> C++ Builder 6 .**

Forma ochib Formaga Standart komponentalar palitrasidan Edit , 2 ta Button, Label va Additional komponentalar palitrasidan 2 ta StringGrid olamiz. Shunda formamiz quyidagicha ko’rinishga keladi.



Formada joylashgan Label1 ning “**Caption**” xossasiga “**n=**”, Button1 va Button2 larning “**Caption**” xossasiga mos ravishda “**massivni shakllantirish**”, “**Tartiblash**” deb kiritamiz. Edit 1 ning **Text** xossasini tozalaymiz. StringGrid1 va StringGrid2 larning “**RowCount**” xossasiga 1 qiymatni yozamiz. StringGrid1 ning **goEditing** xossasiga **true** yozamiz. Bundan tashqari Formga ham mos o’zgarishlar kiritsak Form oynasi quyidagicha ko’rinishga keladi



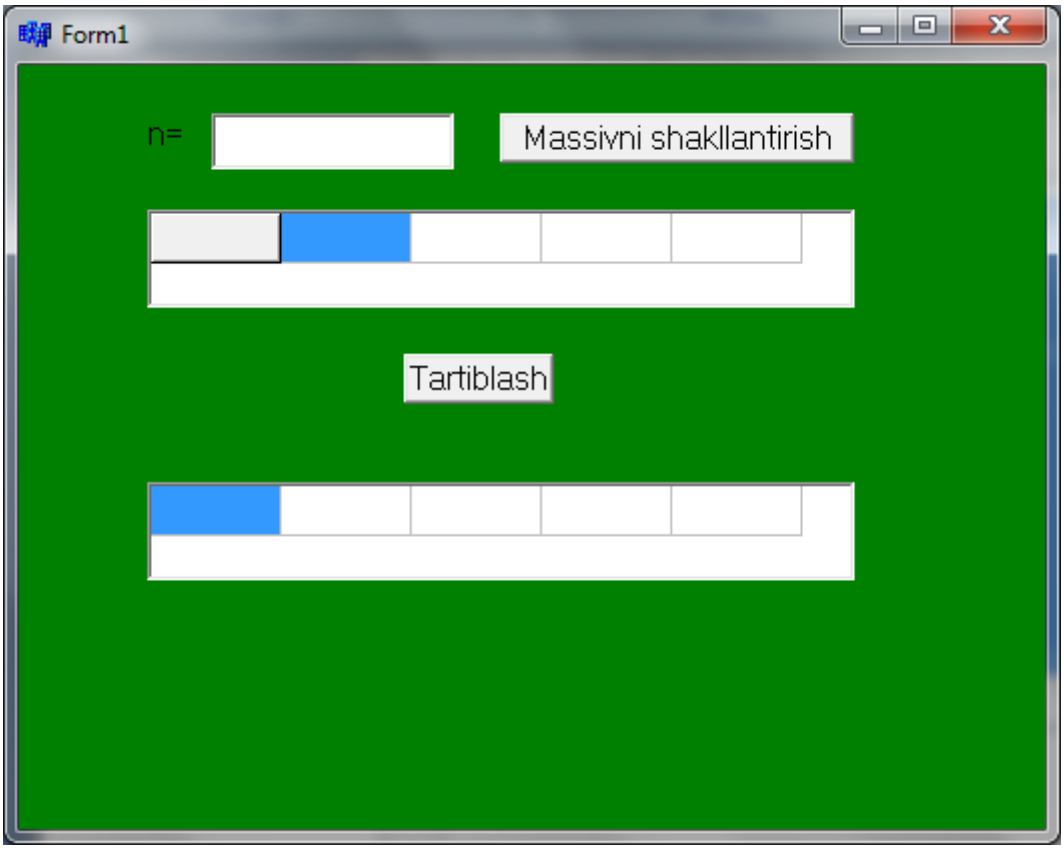
Endi **Button1** ning **onClick** xossasiga quyidagi kodni kiritamiz.

```
int n;
 n=StrToInt(Edit1->Text );
 StringGrid1->Cells[0][0]="A massiv" ;
 StringGrid2->Cells[0][0]="B massiv" ;
 StringGrid1->ColCount =n+1;
 StringGrid2->ColCount =n+1;
```

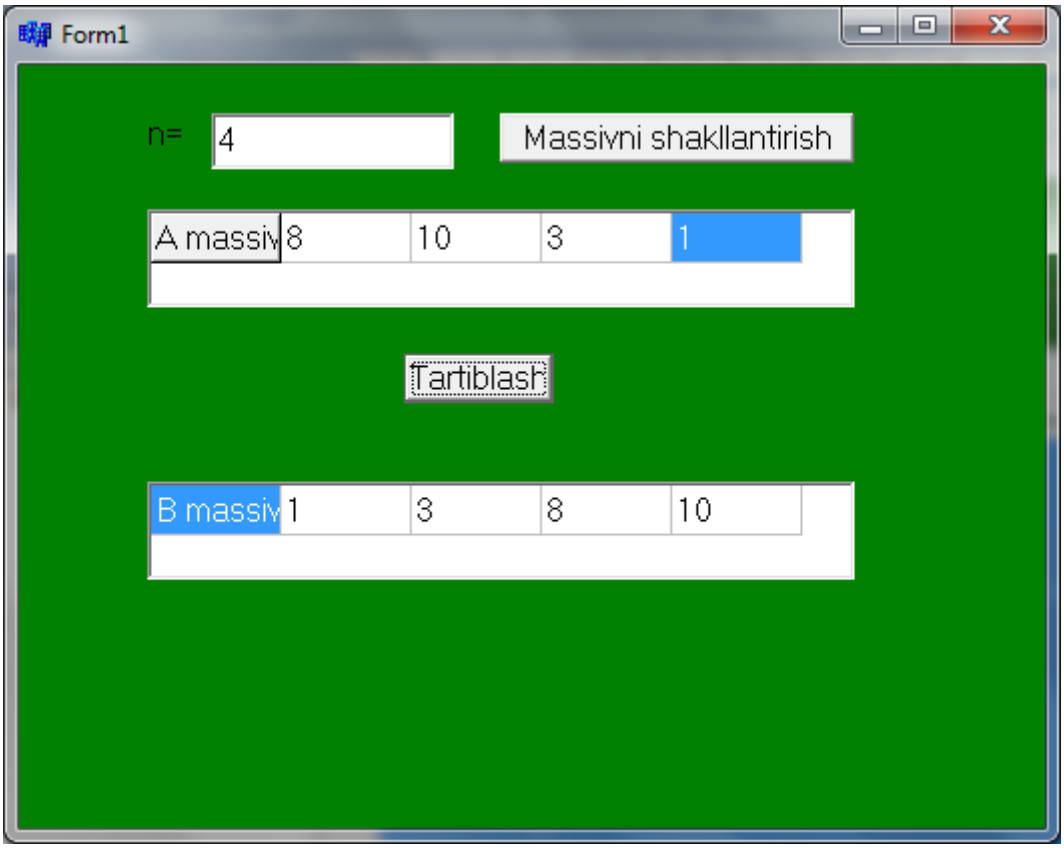
Endi **Button2** ning **onClick** xossasiga quyidagi kodni kiritamiz.

```
int n,c;
 n=StrToInt(Edit1->Text );
int a[100];
for (int i=1;i<=n;i++)
  a[i]=StrToInt(StringGrid1->Cells [i][0]);
for (int k=1;k<=n;k++)
       for(int i=1;i<=n-1;i++)
       if (a[i]>a[i+1])
       { c=a[i];
         a[i]=a[i+1];
         a[i+1]=c ;
        };
for (int i=1;i<=n;i++)
 StringGrid2->Cells [i][0]=IntToStr(a[i]) ;
```

Endi dasturni ishlatish uchun “**Run**” menyusining “Run” buyrig’ini yoki klaviaturadan **F9** klavishini bosib ishgatushirishimiz mumkin. Shunda bizga quyidagicha oyna xosil bo’ladi.



Bu dasturni ishlatish uchun avval **n** ga qiymat berib “Massivni shakllantirish“ buyrug’i orqali massivni shakllantirib so’ngra massivni qiymatini berib “**tartiblash**” buytug’ini tanlaymiz. Masalan:



Xulosa.

Xulosa qilib shuni aytish keraki bu kurs ishida, C++ dasturlash tilidan foydalanib uning dasturiy ta’minotini yaratish tug’risida qisqacha ma’lumotni izoxladim.

Birinchi bo’limda “C++” tilida massivlarni tezkor saralsh usullarini ko’rib chiqdim va ushbu mavzuga oid nazariy ma’lumotarga ega bo’ldim hamda tezkor saralash algoritminig (pufakcha, tez saralash, o’rin almashish) usullari bilan tanishib chiqdim.

Ikkinchi bo’limda C++ dasturlash tili xaqida va unda ishni tashkil etish jarayoni, qo’yilgan masalani yechish davomida qo’llaniladigan operatorlar va massivlar xaqidagi ma’lumotlarni izoxladim.

Ushbu kurs ishi “C++” kursining mavzusida misol va masalalar yechishda qo’llaniladi.

Foydalanilgan adabiyotlar.

1. Jess Liberti, “особая сомостоятелно” C++ za 21 den”, Sankt Peterburg 2000, 815 s.
2. Liberti D. Osvoy samostoyatelno C++: 10 minut na urok. Per s angl. Vilyams, 374 str,2004 g.
3. SHmidskiy YA.K. Prorammirovanie na yazыke C++: Samouchitel. Uchebnoe posobie. Dialektika. 361 str, 2004 g.
4. Kimmel P., «Borland C++5» . – SPb.: BHV, 1997.
5. Sayfiev J. F., «C++ tiliga kirish», Buxoro 2004 y.
6. Nazirov SH. A., Qobulov R. V., «Ob’ektga mo‘ljallangan dasturlash», Toshkent 2006 y.
7. Boltaev SH. J., Elov B.B., «Zamonaviy dasturlash tillari», Buxoro 2004 y.
8. Internetdan (ziyonet, dastirum.uz, google.uz, edu.uz)