**№7-ma’ruza. Saralashning yaxshilangan usullari va algoritmlar samaradorligi**

**Reja:**

1. Daraxt yordamida saralash usuli

2. Piramidali saralash usuli

3. Tez saralash usuli

4. Birlashtirish orqali saralash usuli

5. Foydalanilgan manbalar

***Kalit so‘zlar:****binar saralash usuli, piramidali saralash usuli, tez saralash usuli, birlashtirish orqali saralash: ikki yo‘lli birlashtirish, yuqoriga yo‘naltirilgan, pastga yo‘naltirilgan.*

**1. Daraxt yordamida saralash**

Daraxt yordamida saralash asosini binar qidiruv daraxti tashkil etadi. Binar (ikkilik) qidiruv daraxti – quyidagi qo‘shimcha shartlar bajariladigan binary daraxt hisoblanadi. (qidiruv daraxti xususiyatlari):

- Ikkala shoxi ham – chap va o‘ng ikkilik qidiruv daraxti hisoblanadi.

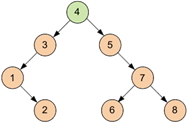
- Istalgan chap shox kaliti o‘zi chiqqan daraxtning kalitidan kichik.

- Istalgan o‘ng shox kaliti o‘zi chiqqan daraxtning kalitidan kichik emas.

Daraxt yordamida saralash uchun berilgan ketma-ketlik daraxt ko‘rinishidagi ma’lumotlar strukturasi bo‘lishi kerak. Masalan, berilgan ketma-ketlik quyidagi ko‘rinishga ega:

4, 3, 5, 1, 7, 8, 6, 2

Daraxrt ildizi ketma-ketlikning boshlang‘ich elementi hisoblanadi. Qolgan elementlar ildizdan kichik bo‘lsa chap shoxga joylashtiriladi, kattalari esa o‘ng shoxga joylashtiriladi. Holbuki, bu shart barcha shoxlarda bajarilishi shart. Shundan keyin daraxt tuzilishiga joylashtirilgan barcha elementlarni infiks ko‘rinishli aylanib chiqish yordamida chiqaramiz.



*6.4-rasm. Daraxt yordamida saralash*

Daraxt yordamida saralashning C++ dagi dasturi:

*#include <iostream>*

*using namespace std;*

*// Tuzlima  - daraxt tarmog‘i*

*struct tnode{*

*int field;           // ma'lumotlar maydoni*

*struct tnode \*left;  // chap avlod*

*struct tnode \*right; // o‘ng avlod*

*};*

*// daraxt tarmoqlarini chiqarish (infiks ko‘rinishli aylanib chiqish )*

*void treeprint(tnode \*tree)*

*{*

*if (tree != NULL) {      //toki bo‘sh tarmoq uchramaguncha*

*treeprint(tree->left);  //chap shoxning rekursiv chiqarish funksiyasi*

*cout << tree->field << " "; //daraxt ildizini chiqaramiz*

*treeprint(tree->right); // o‘ng shoxning rekursiv chiqarish funksiyasi*

*}*

*}*

*// daraxtga tarmoqlar qo‘shish*

*struct tnode \* addnode(int x, tnode \*tree){*

*if (tree == NULL)     //agar daraxt mavjud bo‘lmasa, ildizni shakllantiramiz*

*{*

*tree = new tnode; // tarmoq osti xotira*

*tree->field = x;   //ma'lumotlar maydoni*

*tree->left = NULL;*

*tree->right = NULL; //shoxlarni bo‘sh yaratamiz*

*}*

*else     // aks holda*

*if (x < tree->field)   //agar x element ildizdan kichik bo‘lsa chapga ketamiz*

*tree->left = addnode(x, tree->left); //elementni rekursiv qo‘shamiz*

*else  //aks holda o‘ngga ketamiz*

*tree->right = addnode(x, tree->right); // elementni rekursiv qo‘shamiz*

*return(tree);*

*}*

*//daraxt xotirasini bo‘shatish*

*void freemem(tnode \*tree)*

*{*

*if (tree != NULL)    // agar daraxt bo‘sh bo‘lmasa*

*{*

*freemem(tree->left);   // chap shoxni rekursiv o‘chiramiz*

*freemem(tree->right);  // o‘ng shoxni rekursiv o‘chiramiz*

*delete tree;           // ildizni o‘chiramiz*

*}*

*}*

*// Ishni testlash*

*int main(){*

*struct tnode \*root = 0;    // Daraxt tuzilishini e'lon qilamiz*

*system("chcp 1251");    // konsolda rus tiliga o‘tkazamiz*

*system("cls");*

*int a;            // joriy tarmoq qiymati*

*// Siklga daraxtning 8 tarmog‘ini kiritamiz*

*for (int i = 0; i< 8; i++)  {*

*cout << i + 1 << " - tugunni kiriting "<< ": ";*

*cin >> a;*

*root = addnode(a, root); // daraxtga kiritilgan tarmoqni joylashtiramiz*

*}*

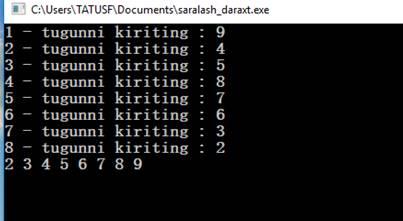
*treeprint(root);    // daraxt elementlarini chiqaramiz, saralangan massivga ega bo‘lamiz*

*freemem(root);      // ajratilgan xotirani o‘chiramiz*

*cin.get();  cin.get();*

*return 0;*

*}*

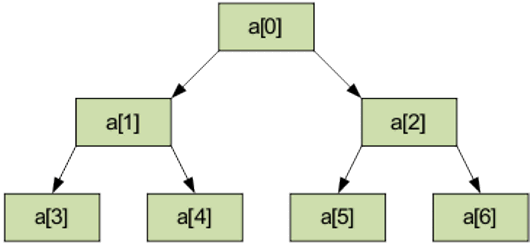
**Natija:**

**2. Piramidali saralash**

D. Villyams tomonidan yaratilgan piramidali saralash usuli daraxt yordamida saralashning yaxshilangan variantidir. Piramidali (top’lam) bu ikki tomonlama daraxtdir

a[i] ≤ a[2i+1];

a[i] ≤ a[2i+2].



*6.5-rasm. Piramidali saralash*

*a[0]* — piramidaning minimal elementi.

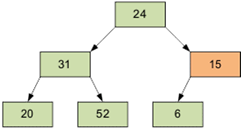
*Piramidal saralashning asl g‘oyasi*, umumiy arifmetik elementlardan olingan piramidaning oldindan yasalishi va elementlarning tartiblashidir. Algoritmning bajarilishi 2 etapga bo‘linadi.

***1 –bosqich.*** Piramidani qurish.  n/2-1 dan boshlab, daraxting o‘ng qismini aniqlab olamiz (daraxtning pastki darajasi). Massivning bu qismidan chaproqda turgan elementlarini olamiz va uni piramida bo‘ylab joylashtiramiz, undan kichik elementlar kelib qolsa, ularning kichigi bo‘ylab harakatlanamiz.

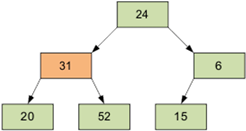
Masalan saralash uchun massiv: 24,   31,  15,   20,   52,   6

Elementlarni dastlabki piramida ko‘rinishida joylashtiramiz;

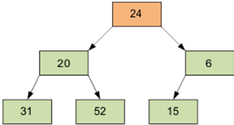
o‘ng qism elementining nomeri (6/2-1)=2 – bu element  15.



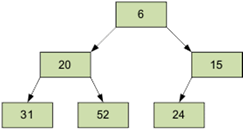
15 elementini piramida bo‘ylab joylashtirish natijasi:



Keyingi joylashtirilayotgan element – 1, 31 ga teng

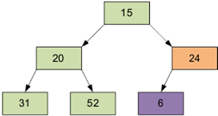
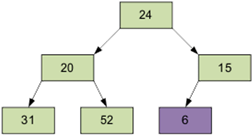


keyingi element –0,  24 ga teng.

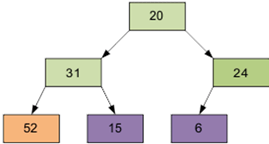
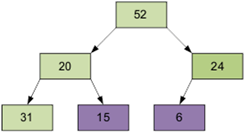


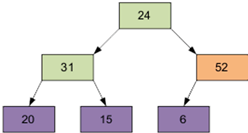
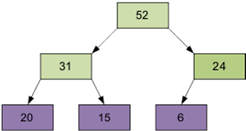
Albatta olingan massiv hali tartiblanmagan. Lekin joylashtirish jarayoni piramidali saralash asosi hisoblanadi. Joylashtirish natijasida eng kichik element piramida uchida bo‘lib qoladi.

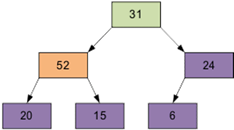
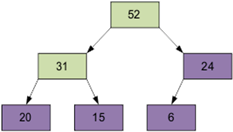
***2– bosqich*** qurilgan piramidada saralash. Massivning eng oxirgi elementini joriy qilamiz. Joriy elementni uchidagi element(eng kichik) bilan joylarini almashtiramiz. Joriy elementni (uchidagi) n-1 elementli piramida bo‘ylab joylashtiramiz. Keyin oxirgisidan bitta oldingi elementni tanlaymiz va h.k.



jarayonni davom ettiramiz. Natijada massiv kamayish tartibida saralangan bo‘ladi.







Piramidali saralashning C++ dagi dasturi:

*#include <stdio.h>*

*#include <stdlib.h>*

*// To‘plamni shakllantirish – to‘plam bo‘yicha “joylashtirish”*

*void siftDown(int \*numbers, int root, int bottom)*

*{*

*int maxChild; // maksimal avlod indeksi*

*int done = 0; // to‘plam shakllanganligi bayrog‘i*

*// Toki oxirgi qatorga bormaganimizcha*

*while ((root \* 2 <= bottom) && (!done))*

*{*

*if (root \* 2 == bottom)    // agar oxirgi qatorda bo‘lsak,*

*maxChild = root \* 2;    // chap avlodni eslab qolamiz*

*// aks holda, ulardan kattasini eslab qolamiz*

*else if (numbers[root \* 2] > numbers[root \* 2 + 1])*

*maxChild = root \* 2;*

*else*

*maxChild = root \* 2 + 1;*

*// agar uchidagi element maksimal avloddan kichik bo‘lsa*

*if (numbers[root] < numbers[maxChild])    {*

*int temp = numbers[root]; // ularning joylarini almashtiramiz*

*numbers[root] = numbers[maxChild];*

*numbers[maxChild] = temp;*

*root = maxChild;*

*}*

*else // aks holda*

*done = 1; // piramida hosil bo‘ladi  }*

*}*

*// to‘plamda saralash funksiyasi*

*void heapSort(int \*numbers, int array\_size){*

*// piramidaning pastki qatorini shakllantiramiz*

*for (int i = (array\_size / 2) - 1; i >= 0; i--)*

*siftDown(numbers, i, array\_size);*

*// qolgan elementlarni piramida bo‘ylab joylashtiramiz*

*for (int i = array\_size - 1; i >= 1; i--)   {*

*int temp = numbers[0];*

*numbers[0] = numbers[i];*

*numbers[i] = temp;*

*siftDown(numbers, 0, i - 1);  }}*

*int main() {*

*int a[10];*

*// Massivni tasodifiy sonlar bilan to‘ldiramiz*

*for (int i = 0; i<10; i++)*

*a[i] = rand() % 20 - 10;*

*// massivning saralashgacha bo‘lgan elementlarini chiqaramiz*

*for (int i = 0; i<10; i++)*

*printf("%d ", a[i]);*

*printf("\n");*

*heapSort(a, 10); // saralash funksiyasini chaqirish*

*// Saralashdan keying massiv elementlarini chiqarish*

*for (int i = 0; i<10; i++)*

*printf("%d ", a[i]);*

*printf("\n");*

*getchar();*

*return 0;*

*}*

**Natija**:



***Piramidali saralash algoritmining tahlili.*** Ba'zi tashqi murakkabliklarga qaramay, piramidali saralash eng samarali hisoblanadi. Saralash algoritmi katta n lar uchun samarali. Eng yomon holatda o‘zgaruvchan elementlar qadamlari soni  http://mt.samtuit.uz/pluginfile.php/13366/mod_page/content/2/image038.png. Almashtirishlarning o‘rtacha soni taxminan quyidagiga teng: http://mt.samtuit.uz/pluginfile.php/13366/mod_page/content/2/image040.png

**3. Tez saralash usuli**

*Tez saralash* - almashinish prinsipiga asoslangan mukammallashgan saralashning bir usuli. To‘g‘ridan-to‘g‘ri saralashlar ichida eng samarasizi pufakchali saralash hisoblanadi. Lekin mukammallashgani barcha ma’lum saralash algoritmlari ichida eng yaxshisi hisoblanadi. U shunday yorqin xususiyatlarga egaki, uning ixtirochisi Ch. Xoar uni tezkor saralash deb hisoblagan.

Katta samaraga erishish uchun katta masofadagi elementlarni almashtirish kerak. Massivda ruxsat beruvchi element tanalanadi. Undan keyin u shunday joyga joylashtiriladiki, saralangandan keyin o‘z joyida turgan hisoblanadi. Bu jarayonda ruxsat beradigan element shunday joylashadiki, undan chapda joylashgan element undan kichik, undan o‘ngda joylashgan element esa undan kata hisoblanadi. (massiv o‘sish tartibida saralanadi) shunday qilib massiv 2 qismga bo‘linadi:

* Ruxsat beruvchi elementdan chapda joylashgan saralanmagan elementlar;
* Ruxsat beruvchi elementdan o‘ngda joylashgan saralanmagan elementlar;

Bu ikkita kichik massivlarni saralash uchun funksiya o‘ziga rekursiv murojaat qiladi.

* Agar bitta elementdan ko‘proq elementni saralash kerak bo‘lsa massivdan ruxsat beruvchi element tanlaymiz;
* Elementni o‘zining natijaviy joyiga joylashtirib, massivni qayta tartiblaymiz;
* Ruxsat beruvchi elementdan chap tomondagi massivni rekursiv saralaymiz;
* Ruxsat beruvchi elementdan o‘ng tomondagi massivni rekursiv saralaymiz;
* Tezkor saralashning asosiy elementi qayta tartiblash algoritmi hisoblanadi. Saralashni massiv misolida ko‘rib chiqamiz:

*10, 4, 2, 14, 67, 2, 11, 33, 1, 15.*

Qayta tartiblash algoritmini amalga oshirish uchun massivning chap elementi ya’ni left ko‘rsatgichi olinadi. Ko‘rsatgich o‘ngga harakatlanadi toki, ruxsat beradigan elementdan kichik. Qayta tartiblash algoritmini amalga oshirish uchun massivning o‘ng elementi ya’ni left ko‘rsatgichi olinadi. Ko‘rsatgich chapga harakatlanadi toki, ruxsat beradigan elementdan katta.

Massivning chapdagi elementi ruxsat beruvchi pivot bo‘lsin. *Left* ko‘rsatgichini undan keying elementga o‘rnatamiz; *right* — eng oxirgi. Algoritm 10 ning to‘g‘ri joyini aniqlab va ish jarayonida noto‘g‘ri joylashgan elementlarni joylarini almashtiradi.

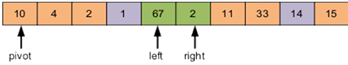
http://mt.samtuit.uz/pluginfile.php/13366/mod_page/content/2/image042.png

agar joylashishi ruxsat beruvci elementga nisbatan noto‘g‘ri bo‘lsa, ko‘rsatgichning harakati to‘xtaydi.

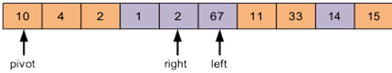
*left*  ko‘rsatgichi 10 dan kata element uchramaguncha harakatlanaveradi; *right* ko‘rsatgichi 10 dan kichik element uchramaguncha harakatni davom ettiradi.



bu elementlar joylari bilan almashishadi va ko‘rsatgichlar harakati davom etadi.



jarayon right  *left*  dan chapda bo‘lib qolmaguncha davom etadi.



 ruxsat beradigan elementning to‘g‘ri joyi topiladi.

Ruxsat beradigan element bilan  *right* ko‘rsatayotgan elementlar bilan joy almashadi.

http://mt.samtuit.uz/pluginfile.php/13366/mod_page/content/2/image050.png

Ruxsat beradigan element kerakli joyda joylashgan: undan chapda joylashgan elementlar kichik qiymatga ega, o‘ngdagilar- katta. Algoritm ruxsat beradigan elementning chap va o‘ng tomonidagi massivlar uchun rekursiv chaqiriladi.

Tez saralash algoritmining C dagi dasturi:

*#include <stdio.h>*

*#include <stdlib.h>*

*// Tez saralash funksiyasi*

*void quickSort(int \*numbers, int left, int right){*

*int pivot; // ruxsat beradigan element*

*int l\_hold = left; //chap chegara*

*int r\_hold = right; // o‘ng chegara*

*pivot = numbers[left];*

*while (left < right) // chegaralar yopiq bo‘lmaguncha{*

*while ((numbers[right] >= pivot) && (left < right))*

*right--; // [right] elementi [pivot] elementidan katta bo‘lguncha o‘ng chegara tomon yuramiz*

*if (left != right) // agar chegaralar yopilmasa    {*

*numbers[left] = numbers[right]; // ruzsat beradigan element joyiga [right] elementni joylashtiramiz*

*left++; // chap chegarani o‘ngga suramiz    }*

*while ((numbers[left] <= pivot) && (left < right))*

*left++; // [left] element [pivot] dan kichik bo‘lguncha chap chegarani suramiz*

*if (left != right) // chegaralar teng bo‘lmaguncha    {*

*numbers[right] = numbers[left]; //  [left] elemntni joyiga [right] elementni joylashtiramiz*

*right--; //o‘ng chegarani o‘ngga suramiz*

*}*

*}*

*numbers[left] = pivot; // ruxsat beradigan elementni joyiga qo‘yamiz*

*pivot = left;*

*left = l\_hold;*

*right = r\_hold;*

*if (left < pivot) // massivning chap va o‘ng qismlari uchun saralashni rekursiv chaqiramiz*

*quickSort(numbers, left, pivot - 1);*

*if (right > pivot)*

*quickSort(numbers, pivot + 1, right);*

*}*

*int main(){*

*int a[10];*

*// Massivni tasodifiy sonlar bilan to‘ldiramiz*

*for (int i = 0; i<10; i++)*

*a[i] = rand() % 20 - 10;*

*// saralashgacha bo‘lgan massiv elementlarini chiqaramiz*

*for (int i = 0; i<10; i++)*

*printf("%d ", a[i]);*

*printf("\n");*

*quickSort(a, 0, 9); // saralash funksiyasini chaqirish*

*// saralashdan keyingi massiv elementlarini*

*for (int i = 0; i<10; i++)*

*printf("%d ", a[i]);*

*printf("\n");*

*getchar();*

*return 0;*

*}*

**Natija**:

http://mt.samtuit.uz/pluginfile.php/13366/mod_page/content/2/image051.jpg

**4. Birlashtirish orqali saralash usuli**

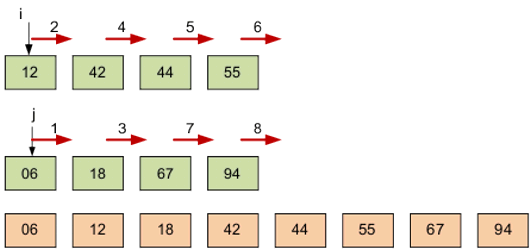
Saralash algoritmi har doim ham samarali bo‘lavermaydi, agar ma’lum bir vaqt ichida ma’lumotlarga faqat bittasiga ruxsat bo‘lgan ma’lumotlar tuzilmasi bo‘lsa.

*Fayllarni saralash asosiy metodi* – birlashtirishli saralash.  
Birlashtirishli saralash — ma’lum bir ketma-ketlikdagi tartiblangan ma’lumotlar ro‘yxatini (yoki boshqa tuzilma, elementlariga faqat ketma-ket murojaat qilsa bo‘ladigan) saralash algoritmi.Siklik elementlarni tanlash bilan bir nechta ketma-ketlikni bitta tartiblangan ketma-ketlik qilish birlashtirish deyiladi.

Birinchi asosiy vazifa bir nechta kichik vazifalarga bo‘linadi va agar ularning hajmi yetarlicha kichik bo‘lsa, vazifa rekursiv ravishda bajariladi. So‘ng ularning yechimi kombinatsiyalashtiriladi va natijada dastlabki vazifa yechimi topiladi. Bir nechta ma’lumotlar ustida amal faza deyiladi.

Takrorlanayotgan eng kichik jarayon o‘tish yoki etap deyiladi.

Birlashtirish jarayoni 2 ta n/2 o‘lchamli ketma-ketlikni birlashtirib bitta n o‘lchamli ketma-ketlik hosil qilishdir.bu ikkita tartiblangan ketma-ketlik boshlang‘ich elementlari tanlanadi va ulardan eng kichigi tanlanadi va o‘sha ketma-ketlikdan shu element olib tashlanadi. Bu jarayon birirta ketma-ketlik tugaguncha davom etadi. Qolgan elementlar asosiy ketma-ketlikning oxiriga qo‘yiladi.



*6.6-rasm. Birlashtirish orqali saralash*

Birlashtirish orqali saralash ko‘pincha tez saralash usuliga o‘xshaydi. Birlashtirishli saralashning unumdorligi piramidali va tez saralash usulining o‘rtasidadir. Lekin piramidali va tez saralashdan farqli ravishda birlashtirishli saralash, massivdagi elementlarni qayta joylashtirishga bog‘liq bo‘lmaganligi uchun birlashtirishli saralash bir xil ishlaydi.

Uning yana bir ustunligi shundan iboratki, u ketma-ket faqat bitta elementga ruxsat beriladigan ma’lumotlar tuzilmasi bilan ishlashga qulay hisoblanadi. Bularga tashqi qurilmadagi fayl va bog‘langan ro‘yxatlar misol bo‘la oladi. Bu usul asosan tashqi saralashda ishlatiladi.

Bu usulning kamchiligi shuki, u xotirada fayl hajmiga teng katta joy talab qiladi. Shuning uchun katta fayllarni saralashda birlashmali saralash tezkor xotira uchun muammo tug‘diradi. Agar saralash vaqti muhim bo‘lib saralanayotgan elementlar tezkor xotiraga sig‘ishi kafolatlansa, unda birlashtirishli saralash ishlatish ma’qulroq.

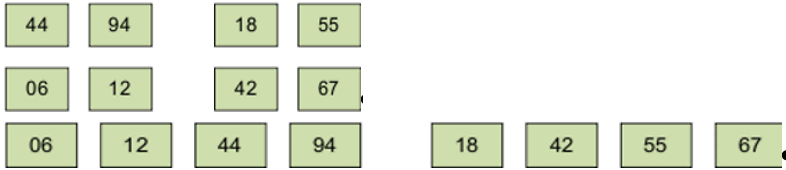
***Ikki yo‘lli birlashtirish algoritmi***

Chiquvchi ketma-ketlik ikkita ketma-ketlikka bo‘linadi:

http://mt.samtuit.uz/pluginfile.php/13366/mod_page/content/2/image054.png

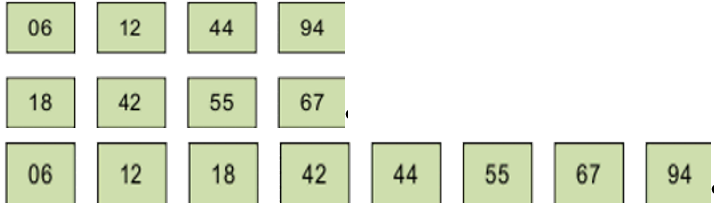
http://mt.samtuit.uz/pluginfile.php/13366/mod_page/content/2/image056.png

Bu ikki ketma-ketlik bittaga birlashtiriladi, tartiblangan juftliklardan iborat bo‘ladi:



Olingan ketma-ketlik qaytadan ikkiga bo‘linadi va tartiblangan to‘rttaliklar yig‘iladi.

Olingan ketma-ketlik qaytadan ikkiga bo‘linadi va tartiblangan sakkizliklar yig‘iladi.



Ushbu amal olingan tartiblangan ketma-ketlik saralangan ko‘rinishga kelmaguncha takrorlanadi.

Asosiy amal birlashtirish hisoblanadi. Birlashtirish orqali fayllarni joylashtirish uchun qo‘shimcha xotira ajratiladi. Chiziqli operatsiya e’lon qilingan massivdagi elementlarning soni bilan bog‘liq.

***Ikki yo‘lni birlashtirish usulining C dagi dasturi:***

*#include <stdio.h>*

*#include <stdlib.h>*

*// ikki yo‘lli saralash funksiyasi*

*void merge(int \*a, int n)*

*{*

*int mid = n / 2; // saralanayotgan ketma-ketlikning o‘rtasiga o‘tamiz*

*if (n % 2 == 1)*

*mid++;*

*int h = 1; // qadam*

*// shakllanayotgan ketma-ketlik uchun xotira ajratamiz*

*int \*c = (int\*)malloc(n \* sizeof(int));*

*int step;*

*while (h < n)*

*{*

*step = h;*

*int i = 0;   // birinchi yo‘l indeksi*

*int j = mid; // ikkinchi yo‘l indeksi*

*int k = 0;   // natijaviy ketma-ketlik element indeksi*

*while (step <= mid)*

*{*

*while ((i < step) && (j < n) && (j < (mid + step)))*

*{ // yo‘lni oxiriga bormagunimizcha*

*// shakllanayotgan ketma-ketlikning keying elementini to‘ldiramiz*

*// ikkalasidan eng kichigi ko‘riladi*

*if (a[i] < a[j])*

*{*

*c[k] = a[i];*

*i++; k++;*

*}*

*else {*

*c[k] = a[j];*

*j++; k++;*

*}*

*}*

*while (i < step)*

*{ // birinchi yo‘lning qolgan elementlarini ko‘chirib chiqamiz (agar 2-oldin tugagan bo‘lsa)*

*c[k] = a[i];*

*i++; k++;*

*}*

*while ((j < (mid + step)) && (j<n))*

*{  // ikkinchi yo‘lning qolgan elementlarini ko‘chirib chiqamiz (agar 1-oldin tugagan bo‘lsa)*

*c[k] = a[j];*

*j++; k++;*

*}*

*step = step + h; // keyingi bosqichga o‘tamiz*

*}*

*h = h \* 2;*

*// tartiblangan ketma-ketlikni dastlabki massivga ko‘chiramiz(oraliq variant)*

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

*a[i] = c[i];*

*}*

*}*

*int main()*

*{*

*int a[8];*

*// massivni tasodifiy sonlar bilan to‘ldiramiz*

*for (int i = 0; i<8; i++)*

*a[i] = rand() % 20 - 10;*

*// saralashgacha massiv elementlarini chiqarish*

*for (int i = 0; i<8; i++)*

*printf("%d ", a[i]);*

*printf("\n");*

*merge(a, 8); // saralash funksiyasini chaqirish*

*// saralashdan keyingi massiv elementlarini chiqarish*

*for (int i = 0; i<8; i++)*

*printf("%d ", a[i]);*

*printf("\n");*

*getchar();*

*return 0;*

*}*

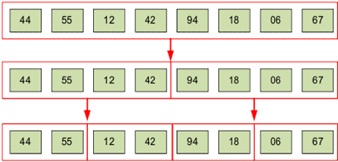
**Natija**:



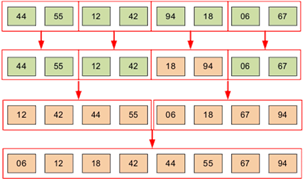
***Pastga yo‘nalgan birlashtirishli saralash.*** Chiquvchi ketma-ketlik 1 element bo‘ylab ketma-ketlikni olmagunimizcha o‘rtasidan rekursiv bo‘linadi. Olingan ketma-ketlikdan birlashtirish usulida tartiblangan juftliklarni, keyin choraklarni va h.k  hosil qilamiz.

Ketma-ketlikni ko‘rib chiqamiz.

Ketma-ketlikni 2 qismga bo‘lamiz.



Har bir ketma-ketlikni  birlashtirish usuli bilan tartiblaymiz va tayyor ketma-ketlikni olamiz.



Pastga yo‘nalgan birlashtirishli saralash usulining C dagi dasturi:

*#include <stdio.h>*

*#include <stdlib.h>*

*#define SIZE 16*

*// pastga yo‘nalgan birlashtirishli saralanuvchi funksiya*

*void mergeSort(int \*a, int l, int r){*

*if (l == r) return; // chegaralar yopilguncha*

*int mid = (l + r) / 2; // ketma-ketlik o‘rtasini aniqlaymiz*

*//va har bir qism uchun saralash funksiyasini rekursiv chaqiramiz*

*mergeSort(a, l, mid);*

*mergeSort(a, mid + 1, r);*

*int i = l;  // birinchi yo‘l boshi*

*int j = mid + 1; // ikkinchi yo‘l boshi*

*int \*tmp = (int\*)malloc(r \* sizeof(int)); // qo‘shimcha massiv*

*for (int step = 0; step < r - l + 1; step++) // qo‘shimcha massivning barcha elementlari uchun  {*

*// shakllanayotgan ketma-ketlikka ikkita yo‘lning kichigini yozib qo‘yamiz*

*// agar  j > r bo‘lsa, birinchining qoldig‘i*

*if ((j > r) || ((i <= mid) && (a[i] < a[j])))    {*

*tmp[step] = a[i];*

*i++;    }*

*else     {*

*tmp[step] = a[j];*

*j++;    }*

*}  // shakllangan ketma-ketlikni dastlabki massivga ko‘chiramiz*

*for (int step = 0; step < r - l + 1; step++)*

*a[l + step] = tmp[step];}*

*int main(){*

*int a[SIZE];*

*// Massiv elementlarini to‘ldirib chiqamiz*

*for (int i = 0; i<SIZE; i++)  {*

*a[i] = (rand() % 100);*

*printf(" %d ", a[i]);  }*

*mergeSort(a, 0, SIZE - 1); // saralash funksiyasini chaqiramiz*

*printf("\n");*

*// saralangan massivni chiqaramiz*

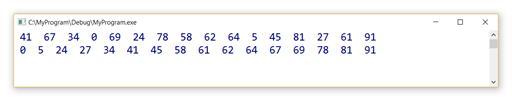
*for (int i = 0; i<SIZE; i++)*

*printf(" %d ", a[i]);*

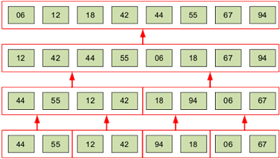
*getchar();*

*return 0;}*

**Natija**:



***Yuqoriga yo‘naltirib birlashtirishli saralash.*** Yuqoriga yo‘naltirib birlashtirishli saralash usuli ketma-ketlikni teng ikkiga bo‘lish protsedurasi orqali ifodalanadi. Chiquvchi ketma-ketlik xuddi ketma-ket olingan elementlar to‘plami ko‘rinishida bo‘ladi.



***Yuqoriga yo‘naltirib birlashtirishli saralash usulining C dagi dasturi:***

*#include <stdio.h>*

*#include <stdlib.h>*

*#define SIZE 15*

*// Yuqoriga yo‘naltirib birlashtirishli usuli*

*void mergeSort(int \*a, int n){*

*int step = 1;  //ketma-ketlikni bo‘lish qadami*

*int \*temp = (int\*)malloc(n \* sizeof(temp)); // qo‘shimcha massiv*

*while (step < n)  // qadamlar massiv uzunligidan kichik  {*

*int index = 0;    //natijaviy massiv indeksi*

*int l = 0;      // maydonning chap chegarasi*

*int m = l + step;  // maydonning o‘rtasi*

*int r = l + step \* 2;  // maydonning o‘ng chegarasi*

*do    {*

*m = m < n ? m : n;  // saralanayotgan maydon ketma-ketlik chegarasidan chiqmaydi*

*r = r < n ? r : n;*

*int i1 = l, i2 = m; // taqqoslanayotgan elementlar indeksi*

*for (; i1 < m && i2 < r; ) // i1 markazga yetmaguncha va  i2 oxiriga yetmaguncha      {*

*if (a[i1] < a[i2]) { temp[index++] = a[i1++]; } // natijaviy ketma-ketlik maydonini to‘ldiramiz*

*else { temp[index++] = a[i2++]; }      }*

*//yoki i1 < m yoki  i2 < r – while operatorlaridan faqat bittasi bajarilishi mumkin.*

*while (i1 < m) temp[index++] = a[i1++]; // saralanayotgan maydonning qolgan elementlarini kiritamiz*

*while (i2 < r) temp[index++] = a[i2++]; //natijaviy massivda*

*l += step \* 2; // keyingi saralanayotgan maydonga o‘tamiz*

*m += step \* 2;*

*r += step \* 2;*

*} while (l < n); // toki saralanayotgan maydonning chap chegarasi ketma-ketlikning ichida*

*for (int i = 0; i < n; i++) // shakllangan massivni a ga qaytaramiz*

*a[i] = temp[i];*

*step \*= 2; // Bo‘linish qadamini 2 barobar oshirish  }*

*}*

*int main(){*

*int a[SIZE];*

*//Massiv elementlarini to‘ldiramiz*

*for (int i = 0; i<SIZE; i++)  {*

*a[i] = (rand() % 100);*

*printf(" %d ", a[i]);  }*

*mergeSort(a, SIZE); // saralash funksiyasini chaqiramiz*

*printf("\n");*

*// saralangan massivni chiqaramiz*

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

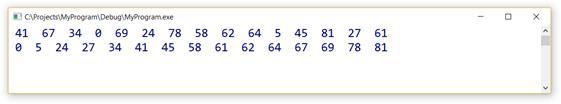
*printf(" %d ", a[i]);*

*getchar();*

*return 0;*

*}*

**Natija**:



**Nazorat uchun savollar:**

1.     Saralashning yaxshilangan usullarini sanab bering.

2.     Daraxt yordamida (binar) saralash usulida foydalaniladigan ma’lumotlar tuzilmasi.

3.     Piramidali saralash usuli nima uchun shunday nomlangan?

4.     Tez saralash usulining samaradorligini baholang?

5.     Birlashtirish orqali saralash usullarini tushuntirib bering va ularning samaradorligini alohida baholang.