code academy

Sort and Search

Sıralama və axtarış alqoritmləri

“Bahtsız Dörtlü”

təqdim edir...

Nicat Məcidov

“ Slayt bizden puan sizden”

Rəşid Babazadə

“ İshlayın ıshlayın”

Əfsanə Rəcili

“flowchartda yazacıq?”

Fatimə Kərimli

Burada sizin reklamınız ola bilərdi

Təqdim olunacaq alqoritimlər:

- Sorts
- **Selection Sort**
- **Merge Sort**
- **Radix Sort**
- Searchs
- **Ternary Search**
- **Interpolation Search**
- **Fibonacci Search**

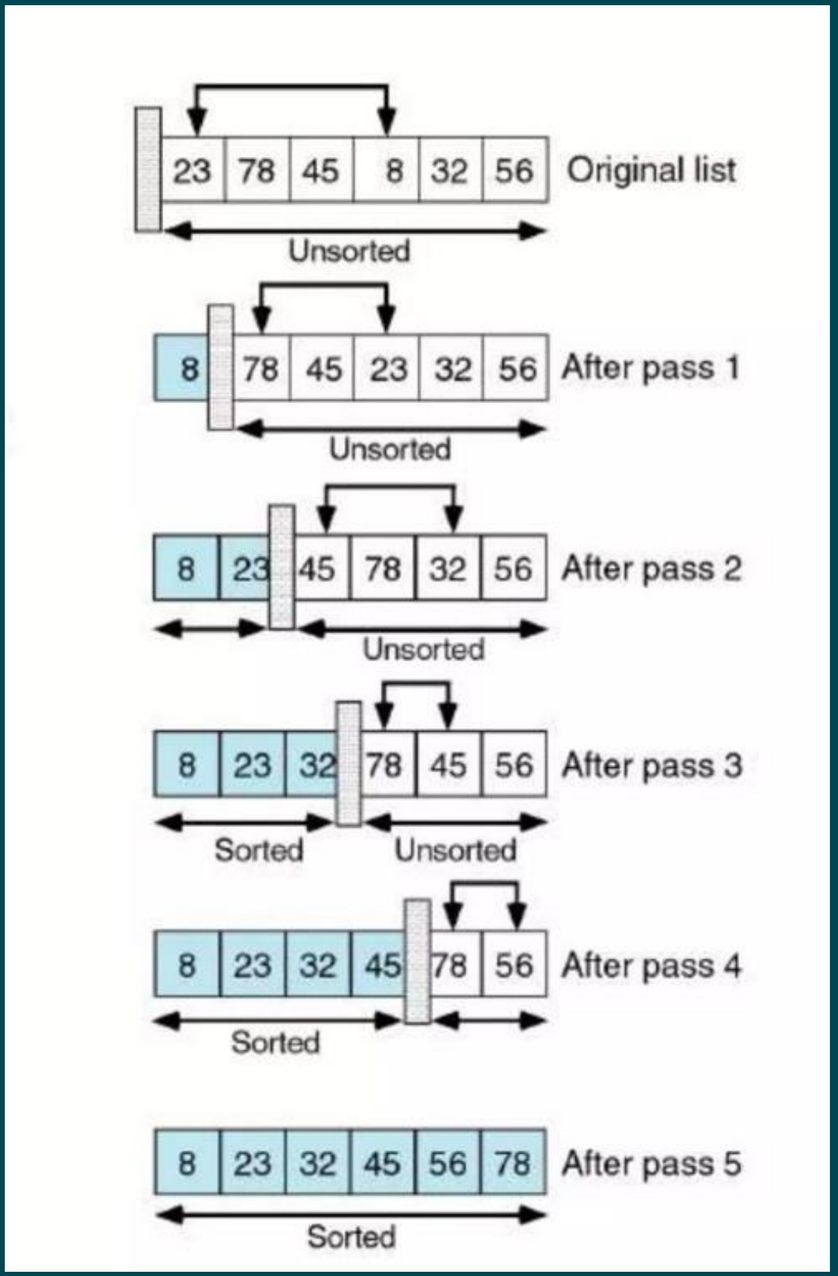
Selection Sort

COMPLEXITY

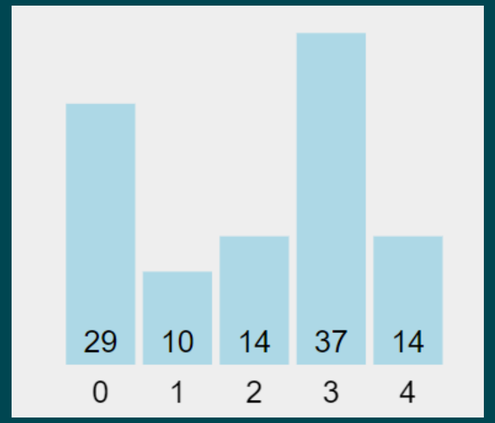
Average Complexity	$O(n^2)$
Best Case	$O(n^2)$
Worst Case	$O(n^2)$
Space Complexity	O(1)

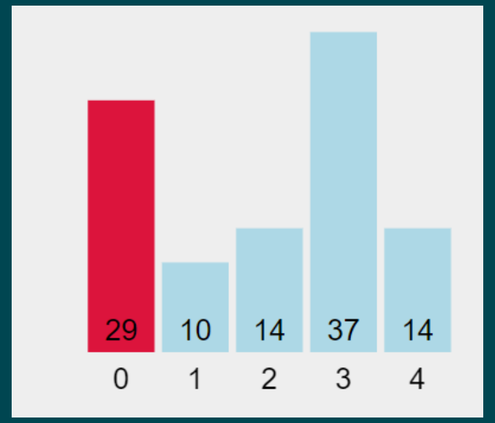
Selection Sort- verilmiş məlumat strukturu sıralanmış və sıralanmamış iki alt siyahıya bölünür və siyahının sıralanmamış hissəsindən ən kiçik elementi dəfələrlə seçərək onu siyahının sıralanmış hissəsinə köçürməklə işləyən sadə və səmərəli çeşidləmə alqoritmidir.

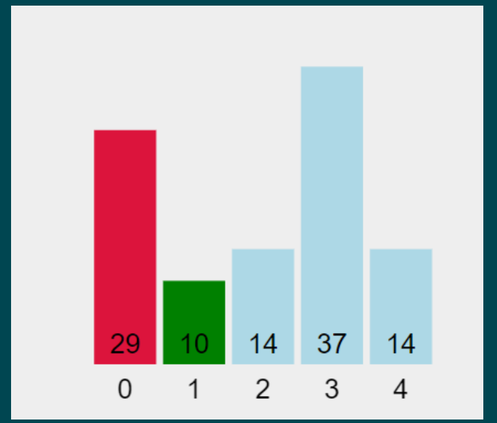
Selection sort əlavə yaddaş tələb etməyən həqiqətən sadə və intuitiv alqoritmdir, lakin kvadratik (O(N2)) zaman mürəkkəbliyinə görə böyük məlumat strukturlarında həqiqətən effektiv deyil.

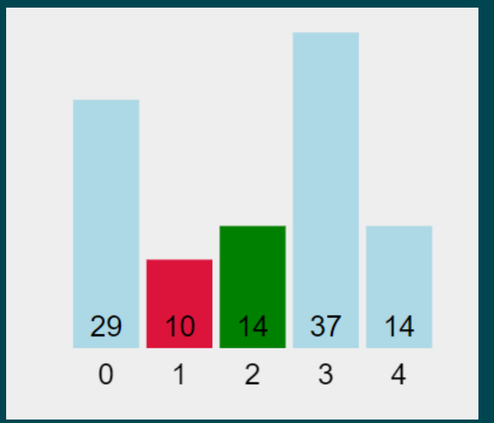


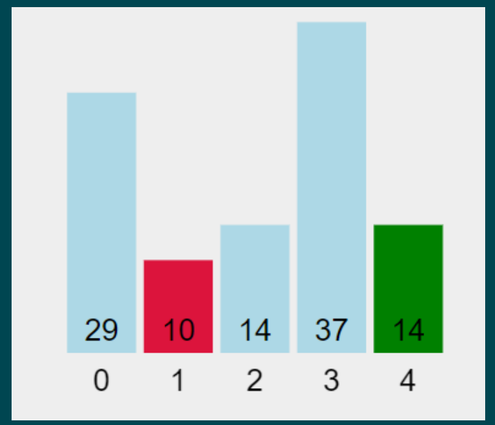
Algoritimin necə işlədiyinə baxaq

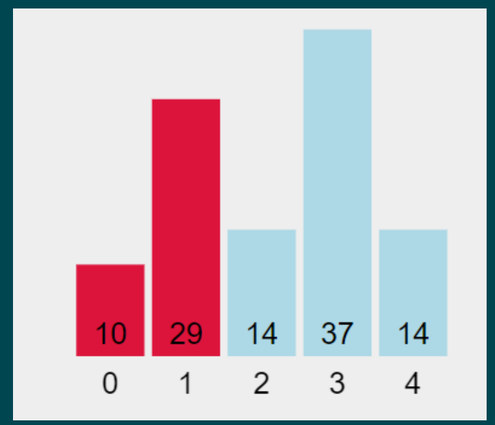


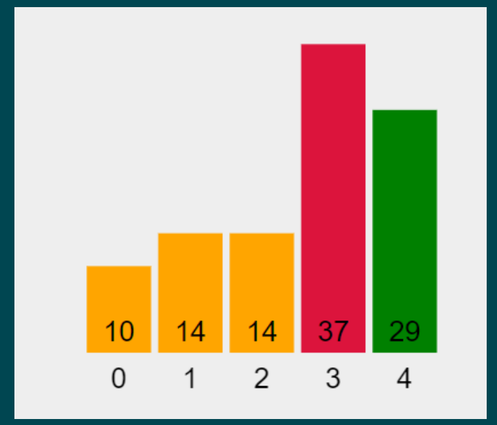


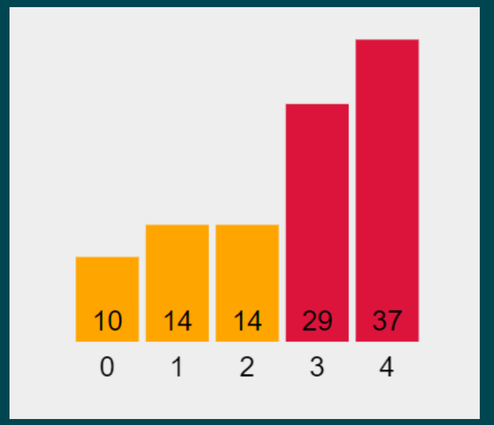












Merge Sort

COMPLEXITY

Average Complexity	O(n × log n)
Best Case	O(n × log n)
Worst Case	O(n × log n)
Space Complexity	O(n)

Merge Sort- Merge sıralamasının əsas müddası “parçala və hökm et” prinsipinə dayanır. Beləki, bizə verilmiş massiv əvəlcə ortadan yarıya bölünür sonra ayrılmış hər iki qrup da vahid elementli massiv olana qeder bu proses davam edir. Parçalama bitdikdən sonra elementlər müqayisə edilərək geri sıralı şəkildə birləşdirilir.

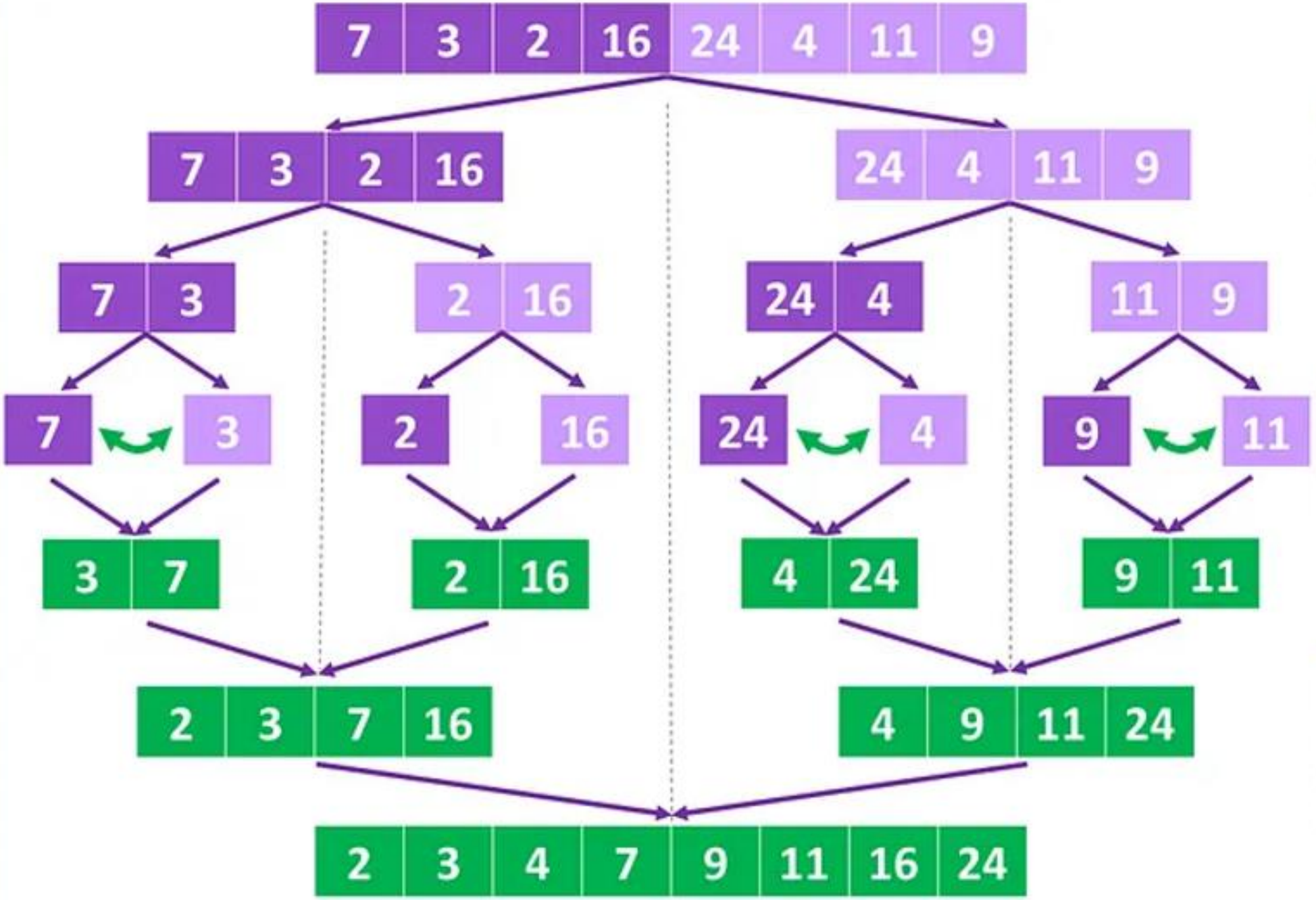
Üstünlüklər:

- Əsas effektivliyi: Ən yaxşı vaxt mürəkkəblik sinifinin O(n log n) olduğu hesablanır.
- Böyük sıralar üçün əlverişlidir: Merge sort, böyük nizalar üçün effektivdir, çünki sıranı iki bərabər hissəyə bölərək və hər hissəni ayrı-ayrı sıralayaraq işləyir.

Mənfi cəhətlər:

- Əlavə yaddaş tələb edir: Merge sort, əlavə massiv (array) tələb edir və bu da yaddaşda xərclərinin artmasına səbəb ola bilər.
- Bazarda işləyə bilməz: Merge sort, yerdəyişmə ilə işləmir və bu səbəbdən bazarda işləyə bilməz. Bu, bazarda böyük bir maneçilikdir.

Merge Sort



Step 1:
Split sub-lists in two until you reach pair of values.

Step 3:
Sort/swap pair of values if needed.

Step 4:
Merge and sort sub-lists and repeat process till you merge to the full list.

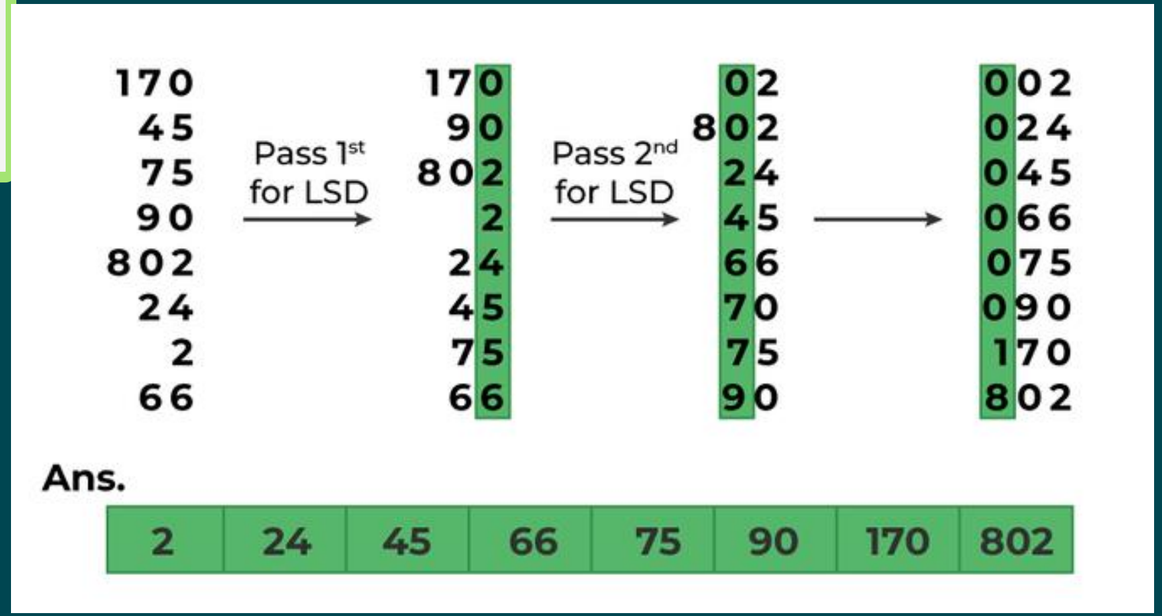
Radix Sort

COMPLEXITY

Average Complexity	O(d × (n + b))
Best Case	O(d × (n + b))
Worst Case	O(d × (n + b))
Space Complexity	O(n + 2^d)

Radix sort- ədədlərin mərtəbələrinə görə sıralama prosesi istifadə edən bir sırlama alqoritmasıdır.

Time complexity O(d*(n+k))-dır. Burada d mərtəbə sayını, n-ədəd sayını, k isə istifadə edilər say sisteminin əsasıdır(10luq saysistemində 10). Bu isə təqribi O(n) ə bərabərdir.



Algoritimin necə işlədiyinə baxaq

1. İlk öncə təkliklərə görə sıralama edilir

170	45	75	90	802	24	2	66

2. Sonra onluqlara görə sıralama edilir

170	90	802	2	24	45	75	66

3. Sonra yüzlüklərə görə sıralama edilir

802	2	24	45	66	170	75	90

2	24	45	66	75	90	170	802

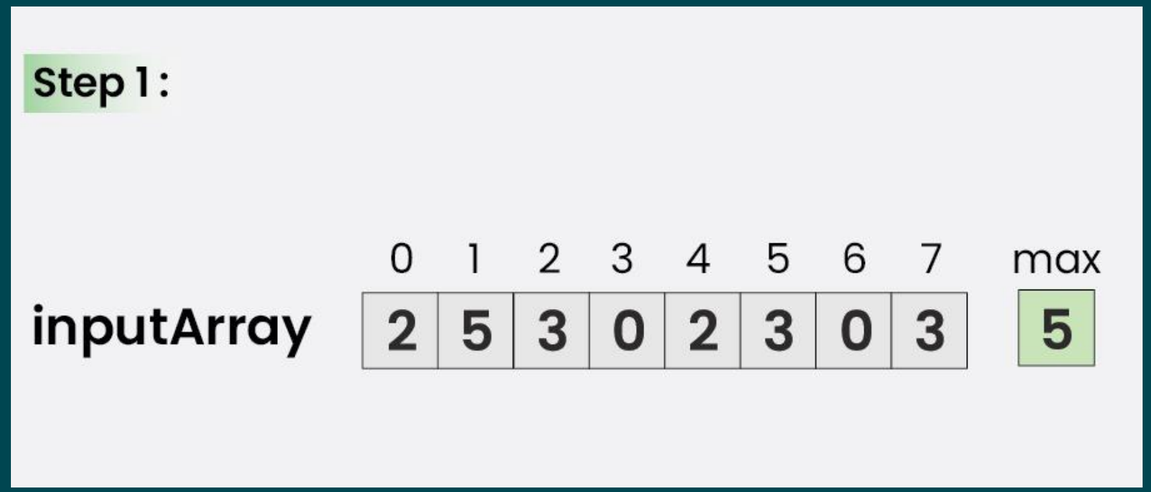
Maksimum olan ədədin son mərtəbəsinə qədər bu proses təkrarlanır və dayanır

Mərtəbələrdəki sıralama hansı alqoritimlə edilir?

Bu proses 2 alqoritmlə daha çox edilir.

- Bucket Sort
- Counting Sort

Bu prosesi couting sortla edək :



Step 3 :

countArray

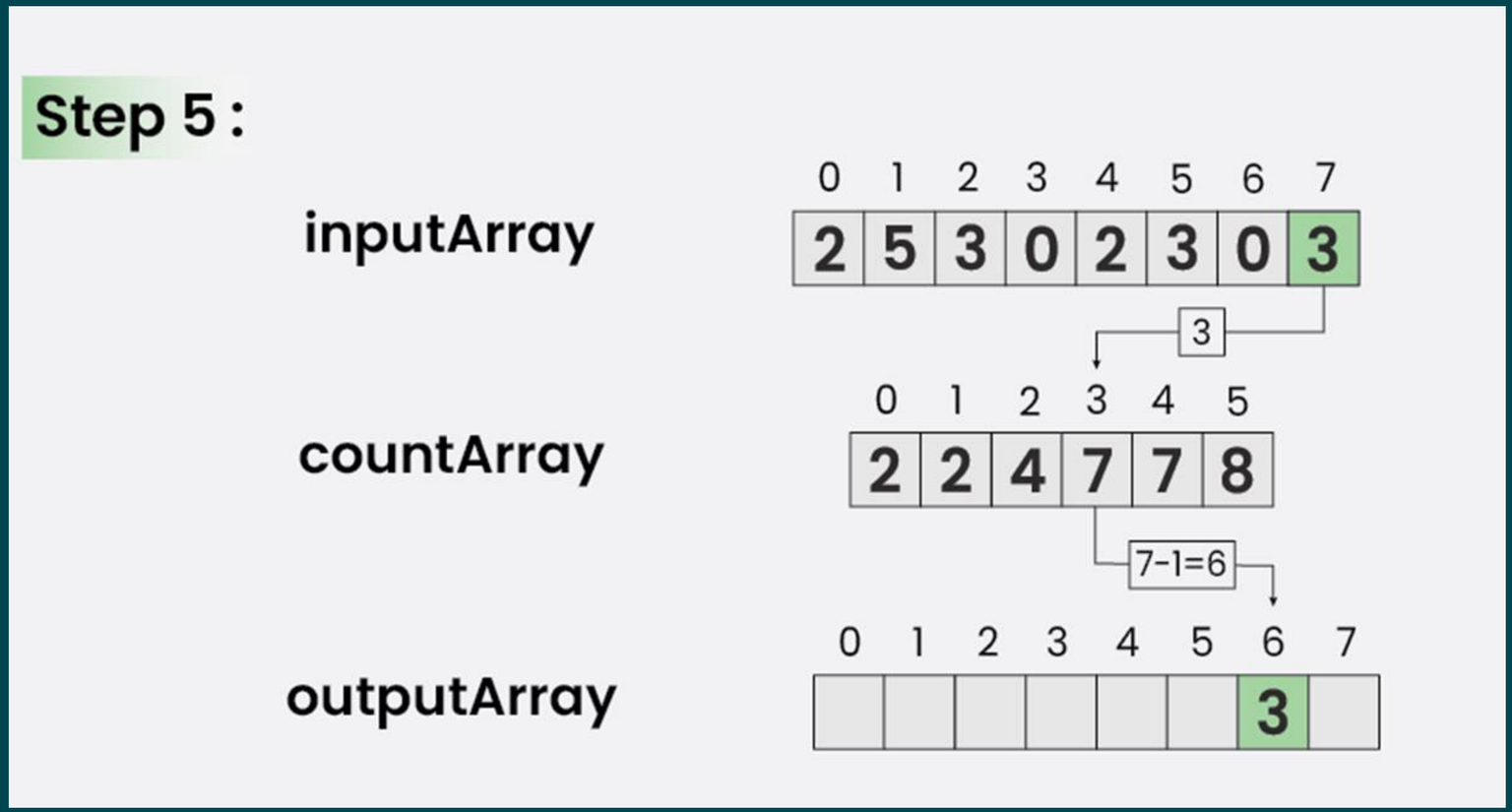
0	1	2	3	4	5
2	0	2	3	0	1

Step 4 : coutArrayın perfix sum(məcmu cəmi) tapılır:
countArray[i] = countArray[i – 1] + countArray[i]

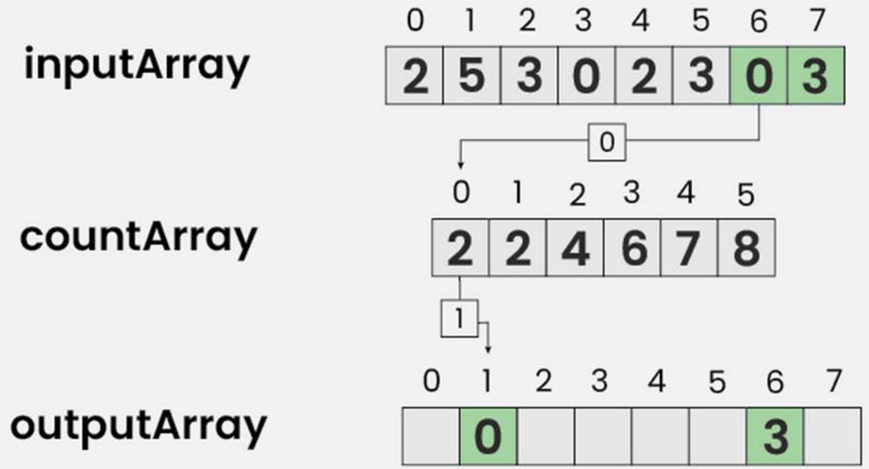
countArray

0	1	2	3	4	5
2	2	4	7	7	8

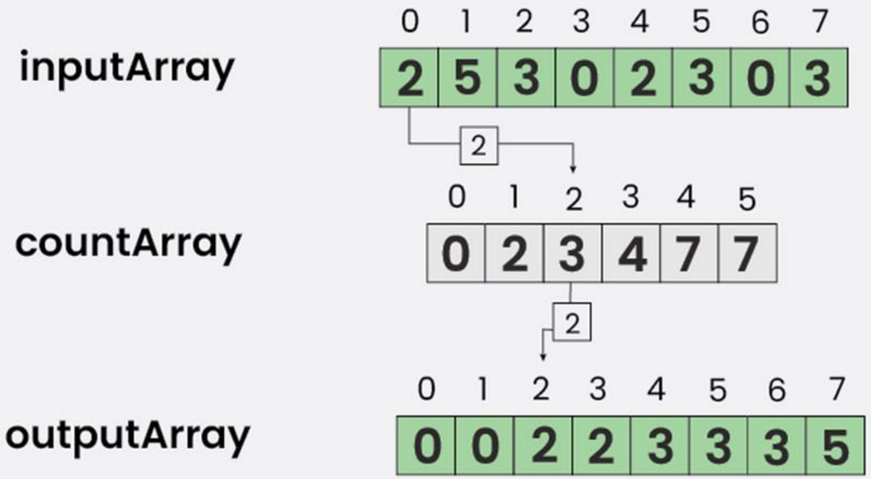
outputArray[countArray[inputArray[i]] – 1] = inputArray[i]
countArray[inputArray[i]] = countArray[inputArray[i]]--
Yuxarıdakı düsturlara uyğun olaraq output array yığılır və countArray yenilənir



Step 6 :



Step 12 : Sonda veriliş nizamsız massiv nizamlanmış olur



Ternary Search

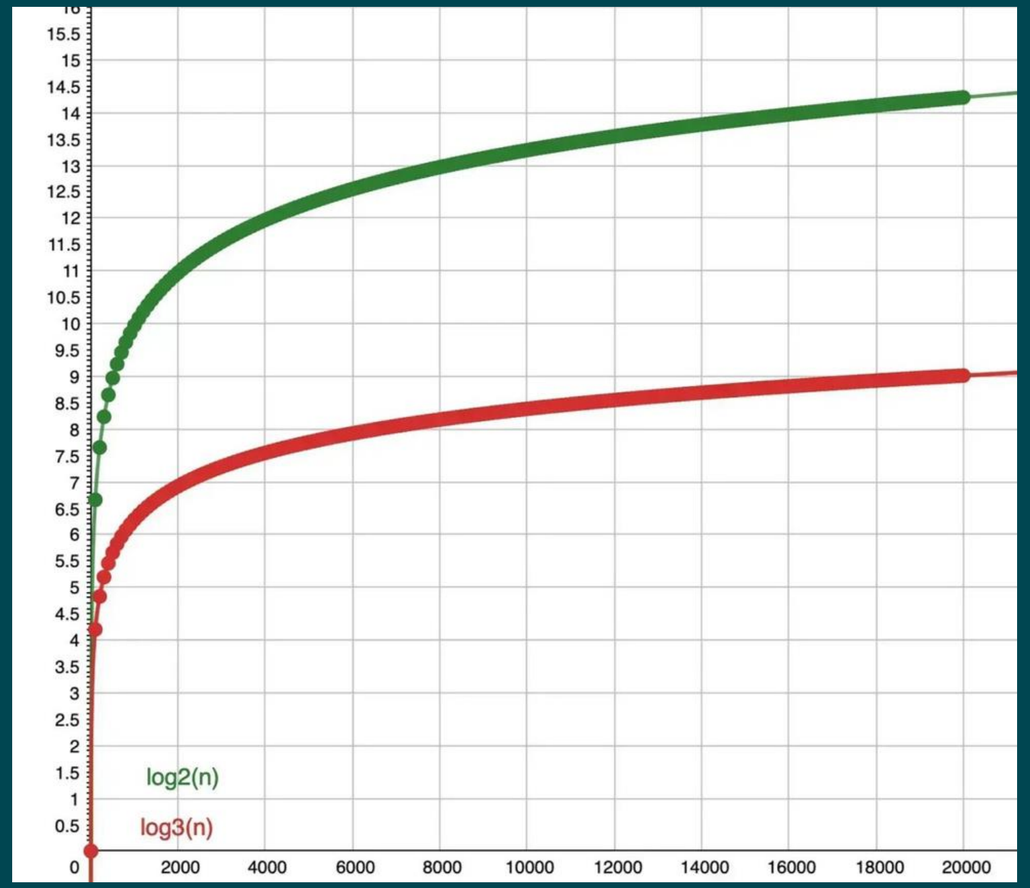
COMPLEXITY

Average case: $\Theta(\log_3 N)$

Best case: $\Omega(1)$

Worst case: $O(\log_3 N)$

Kompüter sistemləri xüsusi məlumatları tapmaq üçün müxtəlif üsullardan istifadə edir. Müxtəlif axtarış alqoritmləri var, hər biri müəyyən vəziyyətlərə daha uyğundur. Məsələn, binar axtarış informasiyanı iki hissəyə, üçlü axtarış isə eyni şeyi üç bərabər hissəyə bölür. Qeyd etmək lazımdır ki, üçlü axtarış yalnız çeşidlənmiş məlumatlar üçün effektivdir.Üçlü axtarış nizamlanmis massivdə hədəf dəyərin yerini tapmaq üçün istifadə edilən axtarış alqoritmidir. O, ikili axtarışda olduğu kimi massivi iki deyil, üç hissəyə bölmək prinsipi ilə işləyir. Əsas ideya, hədəf dəyərini massivi üç bərabər hissəyə bölən iki nöqtədəki elementlərlə müqayisə edərək axtarış sahəsini daraltmaqdır.



Aşağıda Üçlü Axtarışın necə işlədiyinə dair addım-addım izahat verilmişdir:

Baslangic:

•Sıralanmış massivlə başlayın.

•Əvvəlcə massivin birinci və sonuncu elementlərinə işarə edən sol və sağa iki göstərici təyin edin.

Sirani bolun:

•Cari axtarış sahəsini təxminən üç bərabər hissəyə bölməklə iki orta nöqtəni, orta 1 və orta2 **hesablayın:**

- mid1 = left + (right – left) / 3
- mid2 = right – (right – left) / 3
- [left, mid1] , (mid1, mid2) ve [mid2, right] olaraq bölünmüşdür

Axtarilan hedefle qarsilasdirma:

•Hedef mid1 veya mid2'deki elemente beraberdirse axtaris tamamlanir.

•Hedef mid1'deki elementden kicikdirse sağ işaretçiyi mid1 – 1 ile tezeleyin .

•Hedef mid2'deki elementden büyükdurse sol işaretçiyi mid2 + 1 ile tezeleyin .

•Hedef mid1 ve mid2 elementleri arasındadirsa , sol işaretçiyi mid1 + 1'e ve sağ işaretçiyi mid2 – 1 kimi tezeleyin.

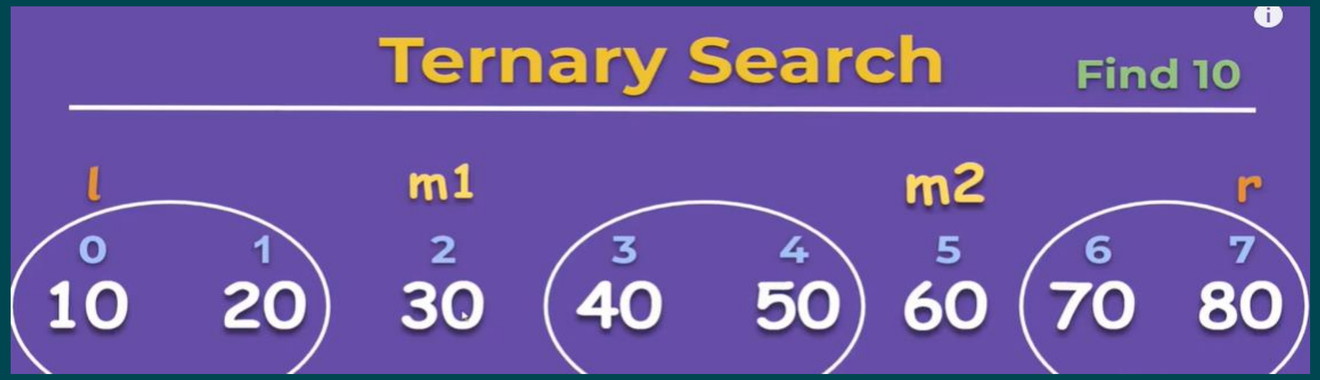
Tekrarlayın veya Sonuçlandırın :

• Hədəf tapılana və ya axtarış sahəsi boş olana qədər prosesi azaldılmışaxtarış sahəsi ilə təkrarlayın

• Axtarış sahəsi boşdursa və hədəf tapılmırsa, hədəfin massivdə tapılmadığın ıgöstərən dəyər qaytarın.

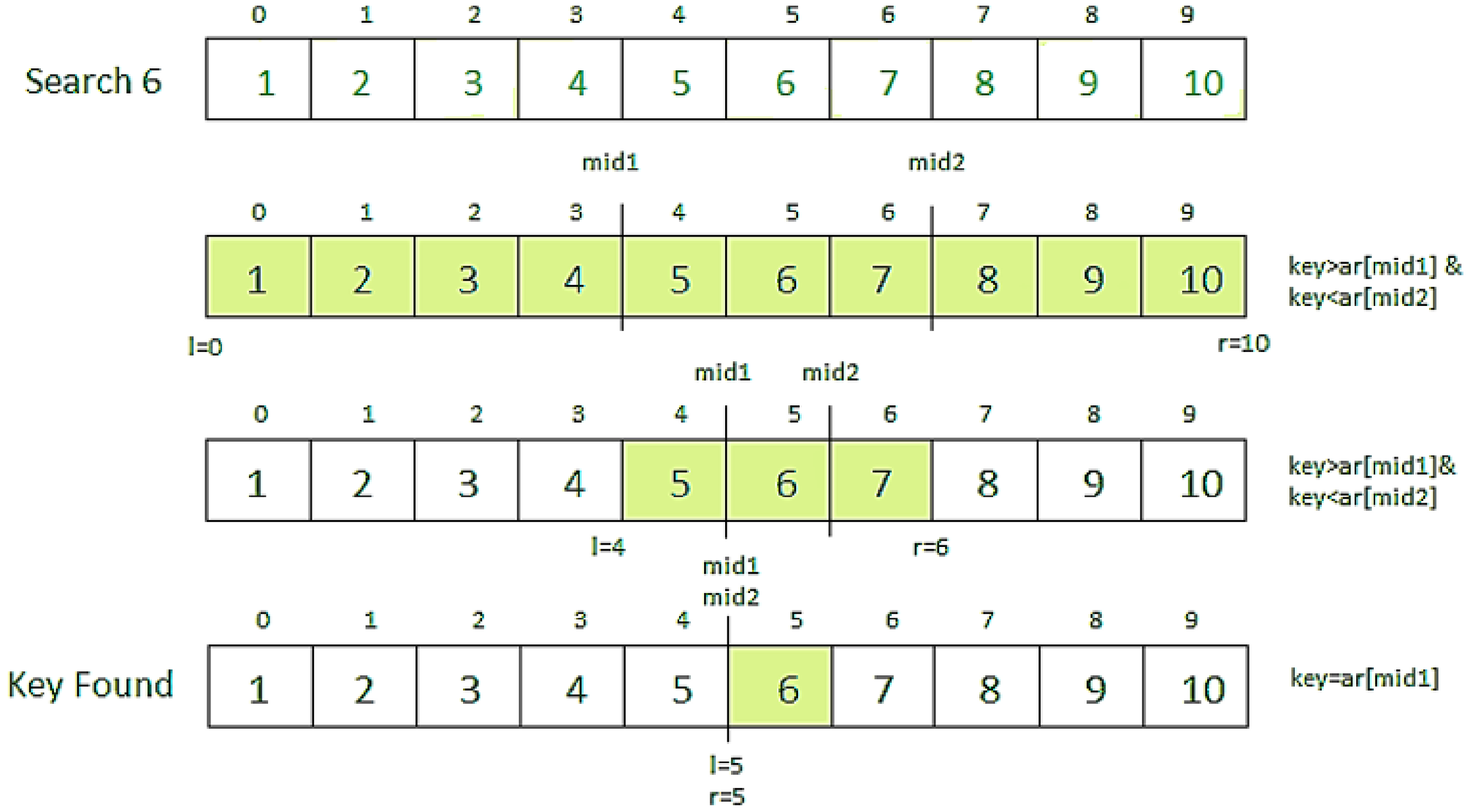
Ilkin olaraq 0-ci index l ,sonuncu 7-ci index ise r olaraq secilir ve mid1 , mid2 hesablanir:

- Mid1=l+(r-l)/3=2
- Mid2=r-(r-l)/3=5



mid1 ve mid2 m1 , m2 olaraq muvafik indekslerde movqelenir ve hemin noqtelere gore nizamli sira 3 hisseye bolunur axtarilan 10 m1-den yeni 30dan kicik oldugu ucun right=m1-1; sekline dusecek.

r ve l movqeleri hazirki veziyyeti alir ve netice artiq axtarilan deyer m1-dedir element tapildi ve netice olaraq m1-n indeksi yeni 0 qayidacaq


Search 6
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
mid1
mid2
key>ar[mid1] &
key<ar[mid2]
l=0
r=10
mid1
mid2
key>ar[mid1]&
key<ar[mid2]
l=4
r=6
mid1
mid2
Key Found
key=ar[mid1]
l=5
r=5

Interpolation Search

COMPLEXITY

Average case: Θ(log(log(N))

Best case: Ω(1)

Worst case: O(N)

Interpolation Search nədir?
Binary search-ın daha inkişaf etdirilmiş forması kimi düşünə bilərik. Bu seach alqoritması nizamlı(sorted) massivlərdə istifadə edilir.

Binary seach dan nə fərqi var?
Bu alqoritm binary seach alqoritmindən fərqli olaraq mid pointeri (left+right)/2 yerinə pos = low + ((target - arr[low]) * (high - low)) / (arr[high] - arr[low]) düsturu ilə tapır. Bu isə xətti olaraq artan və ya azalan çoxluqlarda Binary seachdan daha sürətli axtarış imkanı yaradır. Lakin ədədlər nizamsız artan sıra ilə çoxalır və ya azalırsa bu zaman Binary search daha əlverişli olur.

Algoritimin necə işlədiyinə baxaq

0	1	2	3	4	5	6	7	8	9
10	14	19	26	27	31	33	35	42	44

pos = low + ((target - arr[low]) * (high - low)) / (arr[high] - arr[low]) – düsturla hesablasaq:

low = 0, arr[low] = 10, high = 9, arr[high] = 44
target = 19 -- olarsa

pos = 0 + ((19 – 10)*(9 – 0) / (44 – 10)) = (9 * 9) / 34 = 2.38~ 2

arr[2] = 19 – beləliklə axardığımız ədəd tapılmış oldu

Fibonacci Search

COMPLEXITY

Average case: $\Theta(\log N)$

Best case: $\Omega(1)$

Worst case: $O(\log N)$

Fibonacci Search - sıralanmış massivdə elementi axtarmaq üçün Fibonaççi nömrələrindən istifadə edən müqayisəyə əsaslanan bir texnikadır.

Binary Search ilə oxşarlıqlar:

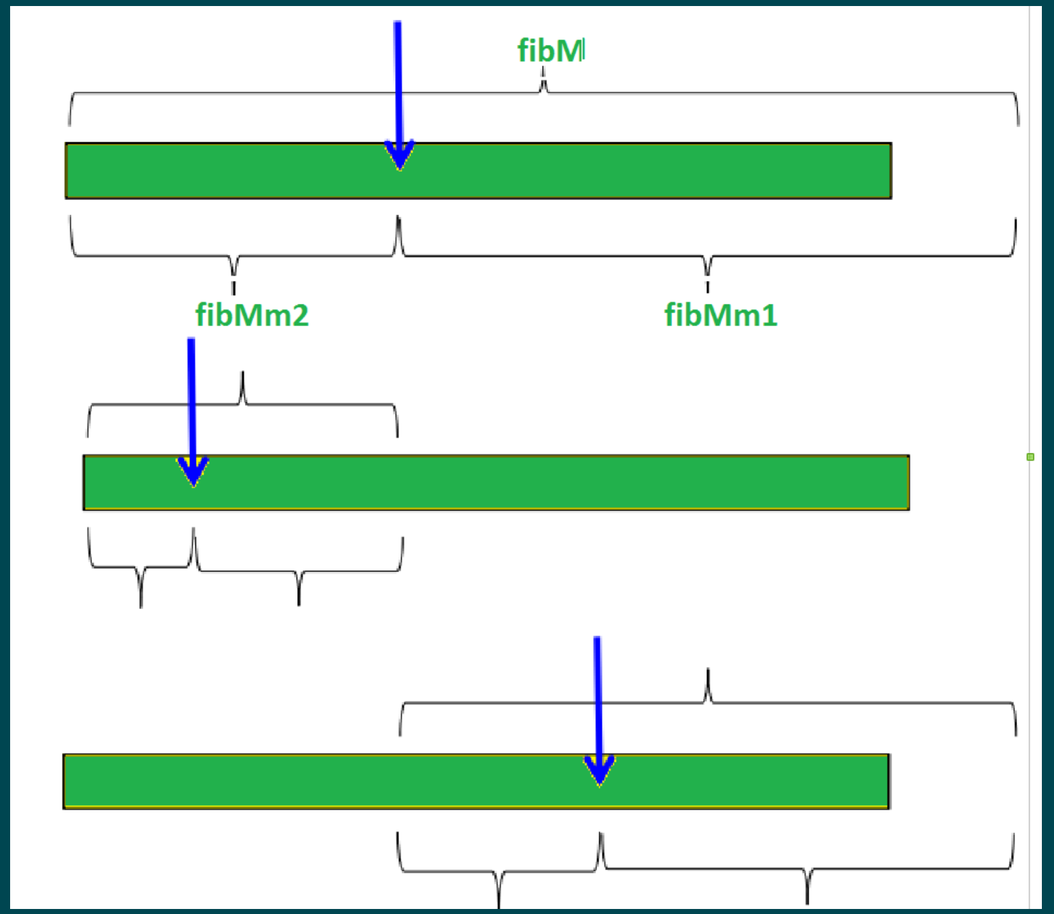
- Sıralanmış massivlər üçün işləyir
- Parçala və hökm et Alqoritmi.
- Log n zaman mürəkkəbliyinə malikdir.

Binary Search ilə fərqlər:

- Fibonacci Search verilmiş massivi qeyri-bərabər hissələrə bölür
- Binary Search diapazonu bölmək üçün bölmə operatorundan istifadə edir. Fibonacci Search “/” istifadə etmir, lakin + və - istifadə edir. Bölmə operatoru bəzi CPU-larda baha başa gələ bilər.
- Fibonacci Search sonrakı addımlarda nisbətən daha yaxın elementləri araşdırır. Beləliklə, giriş massivi CPU keşinə və ya hətta RAM-a sığmayan böyük olduqda, Fibonacci Axtarış faydalı ola bilər.

Verilmiş sıralanmış massivdə axtarılan element x olsun.
İdeya əvvəlcə verilmiş massivin uzunluğundan böyük və ya ona bərabər olan ən kiçik Fibonaççi ədədini tapmaqdır. Tapılan Fibonaççi nömrəsi m'ci Fibonacci nömrəsi olsun. İndeks olaraq (m-2)'ci Fibonaççi nömrəsindən istifadə edirik . (m-2)'-ci Fibonaççi ədədi i olsun, arr[i] ilə x ilə müqayisə edirik, əgər x eynidirsə, i qaytarırıq. Əks halda, x böyükdürsə, biz i-dən sonra alt massiv üçün təkrar edirik, əks halda i-dən əvvəl alt massiv üçün təkrar edirik. Aşağıda tam alqoritm verilmişdir:

- n-dən böyük və ya ona bərabər olan ən kiçik Fibonaççi ədədini tapın. Bu ədəd fibM [m'th Fibonacci Number] olsun. Ondan əvvəlki iki Fibonaççi rəqəmi fibMm1 [(m-1)'th Fibonacci Number] və fibMm2 [(m-2)'th Fibonacci Number] olsun.
- Massivin yoxlanılması lazım olan elementləri olsa da:
  - x-i fibMm2 ilə əhatə olunan aralığın sonuncu elementi ilə müqayisə edin
  - Əgər x uyğun gəlirsə, indeksi qaytarın
  - Əks təqdirdə , x elementdən kiçikdirsə, üç Fibonacci dəyişənini iki Fibonacci aşağı hərəkət etdirin, qalan massivin təxminən arxa üçdə ikisinin aradan qaldırılmasını göstərir.
  - Əks halda x elementdən böyükdür, üç Fibonacci dəyişənini bir Fibonacci aşağı hərəkət etdirin. Ofseti indeksə sıfırlayın. Bunlar birlikdə qalan massivin təxminən üçdə birinin ön hissəsinin aradan qaldırılmasını göstərir.
- Müqayisə üçün bir element qala biləcəyi üçün fibMm1-in 1 olub-olmadığını yoxlayın. Bəli olarsa, qalan elementlə x-i müqayisə edin. Uyğundursa, indeksi qaytarın.


fibM
fibMm2
fibMm1

İstifadə etdiyimiz mənbələr

https://www.geeksforgeeks.org/	https://chat.openai.com/
https://www.sortvisualizer.com/	https://bilgisayarkavramlari.com/
https://visualgo.net/	https://www.canva.com/

Başqa sualınız?

Mence olmasınn :)

thanks for listening

THE END