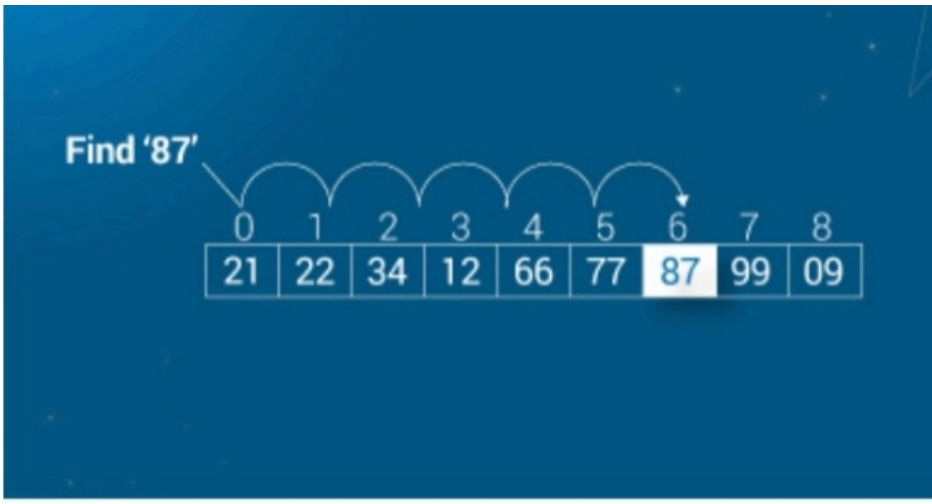
Search Algorithms

Kompüterdə hansısa data structurelər üzərində bir məlumatın axtarışı zamanı istifadə olunan alqoritmlərə search algorithms(axtarış alqoritmləri) deyilir.
Search algorithms 2 yerə bölmək olar:
I. Uninformed search (Bilmədən axtarma)
2. İnformed Search (Bilərək axtarma)

Hər vəziyyətdə eyni şəkildə işləyən alqoritmlər birinci kategoriyadaki algoritmlərdir. Bunların bəzilərinə nəzər yetirək

Linear Search

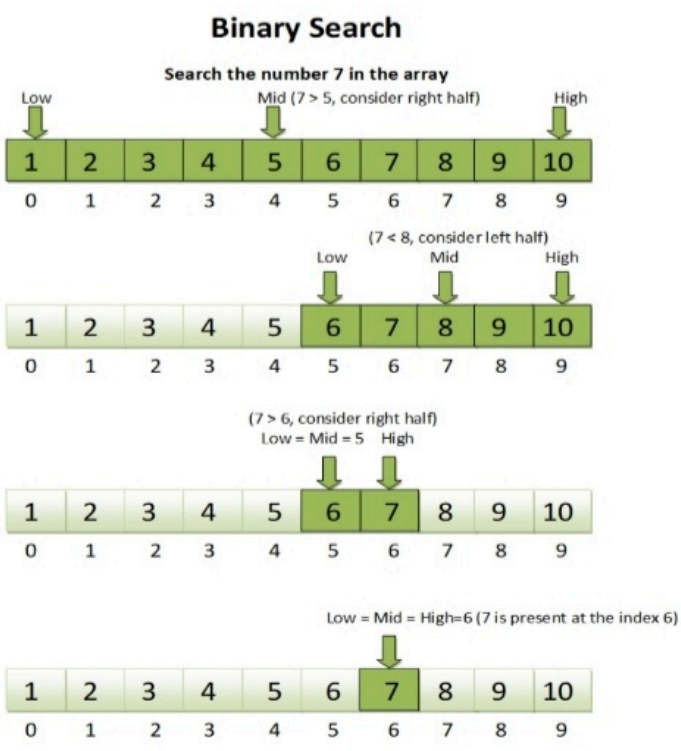


Tutaq ki bizə n elementli bir massiv verilmişdir. Biz massivin içərisindəki X elementini axtarmaq üçün müəyyən bir funksiya yazmalıyıq. Buradaki ən sadə yanaşma linear search-dır. Linear searchin işləmə prinsipi belədir: Massivin ən soldaki "0"cı indexindən başlayır və X elementini massivin hər bir elementi ilə sırayla qarşı qarşıya qoyur. Əgər X hansısa element ilə uyğunlaşarsa onda indexi return edir. Əgər X heçbir elementlə uyğun gəlmirsə -1 return edir. Bu alqoritmin Time complexity O(n). Linear Searchda massivin sort olmasına ehtiyyac yoxdur. Çünki ilk elementdən başlayır və sıra ilə uyğunlaşdırmanı aparır. Linear search ən təməl alqoritmlərdəndir. O hər hansı bir arrayda olan istenilen elementi istifadə etmək üçün tətbiq edilə bilər

Binary Search

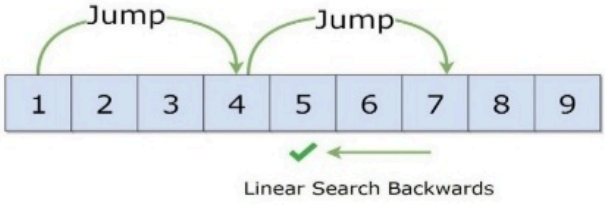
Binary search(ikili axtarış) alqoritmi parçala və fəthet metodu ilə işləyir. İşləmə prinsipi belədir: Əvvəlcə verilmiş massivin ortancıl elementi tapılır. Əgər axtarılan element ortancıl element ilə eynidirsə axtarış sona çatır. Əgər eyni deyilsə və ortancıl elementdən böyükdürsə, həmin ele sağ tərəfinə -yəni böyük elementlərinə baxılır. Əgər Axtarılan elementimiz ortancıl elementdən kiçikdirsə, onda ortancıl elementin sol tərəfinə- yəni kiçik elementinə baxılır.

Beləliklə daha tez və effektiv bir alqoritm həyata keçmiş olur. Burda vacib bir məqama fikir vermək lazımdır,bizim üzərində işlədiyimiz array mütləq sort olunmuş(yəni sıralanmış) olmalıdır. Əks halda alqoritmi düzgün yerinə yetirmək mümkün olmayacaq. Binary Search-alqoritminin Big O Notationu O(log n) dir



Jump Search

Eyni Binary Search kimi, sort olunmuş bir massiv üzərində aparıla bilən alqoritmdir. Jump Search alqoritmində istənilən bir X elementini massivdə blok metodu ilə axtara bilərik. Adından da göründüyü kimi burada axtarış məntiqi müəyyən aralıqlarla jump edərək linear search ilə axtarılan elementi tapmağı hədəfləyir.
İşləmə prinsipi belədir: Massivdə Cəmi neçə element varsa əvvəlcə onun vasitəsilə optimum blok ölçüsü tapılır. Tutaq ki bir array verilib və 16 elementi var. 16 nı kök altına saldıqda 4 çıxır, onda addımlarımız 4 blok şəkilində olacaqdır.
Arraydə axtarılan müəyyən bir element seçilir və optimum blok ölçüsü qədər jump etməyə başlanır. Həmin elementdən böyük elementə çatana qədər bu proses dəvam edir. Çatdığımız nöqtə həmin elementdən böyük olarsa onda proses dayanır və ən başdan linear search ilə element axtarılır. Bu alqoritmin Time complexity si $O(\sqrt{n})$ dir



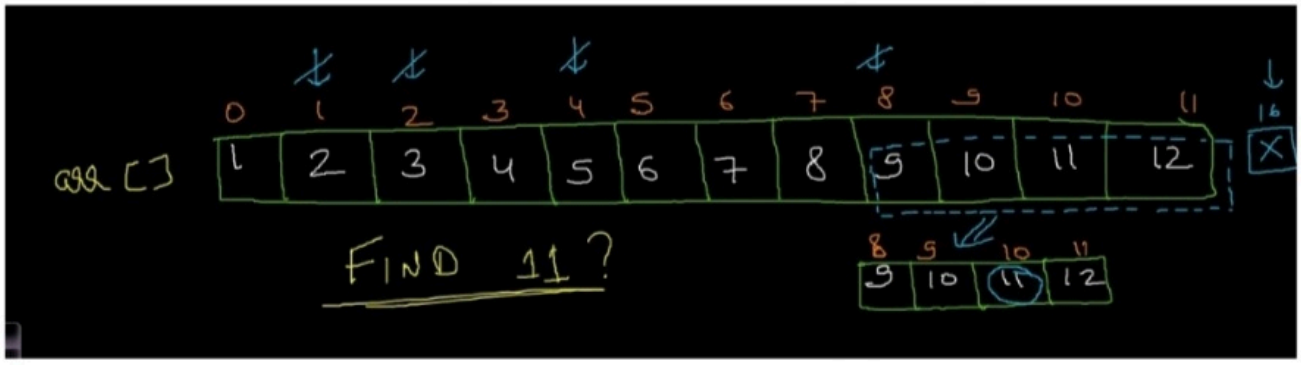
İnterpolation Search

Bu axtarış alqoritmi digər alqoritmlərə nisbətən mürəkkəblik təşkil edir. Əvvəlcə array mütləq sort olunmuş olmalıdır. Həmdə bu alqoritmin Big O Notationu O(n) dir.
İndi bir nümunə ilə bu alqoritmi başa düşməyə çalışaq:
a=[1 3 4 5 6 8 9 11 12]
Tutaq ki bu massivdə 6 elementi axtarılır. Axtarışımıza əsasən sol :0 (index) və sağ:8(index) axtarışa başlayırıq. Əvvəla biz orta dəyəri tapmalıyıq. Sol=0 Sağ=8 a[sol]=1 a[sağ]=12 (elementlər) və x=6 axtarılan dəyər. Orta dəyəri tapmaq üçün bu formulanı tətbiq edək:
orta= sol+((x-a[sol])*(sağ-a[sol]))/(a[sağ-a[sol])
massivə tətviq etdikdə orta= 0+((6-1)*(8-1))/(12-1)
orta=35/11 ~ 3(nəticə təqribi alınırsa təqribi götürürük). Deməli a[3] bizim orta dəyərimizdir və a[3]=5. Axtardığımız element 6 olduğuna görə, 5<6. Buna görə də yeni sol dəyərimiz olacaq və formulanı eynisiylə təkrarlıyacayıq. sol=3 sağ=8 a[sol]=5 a[sağ]=12 orta=4 deməli a[4]=6 axtardığımız element 6 olduğuna görə alqoritm tamamlandı

$$probe = lowEnd + \frac{(highEnd - lowEnd) \times (item - data[lowEnd])}{data[highEnd] - data[lowEnd]}$$

Exponential Search

Exponential Search digərlərinə nisbətən biraz mürəkkəb alqoritmlərdən biridir. Exponential searchda axtarış üçün massiv sort olunmuş olmalıdı, içərisində istənilən qədər element ola bilər. Bu alqoritmin Big O Notationu O(log n) dir. Biz bu alqoritm üçün deyə bilərik ki o linear və binary searchın kombinasiyasıdır. İşləmə prinsipini öyrənmək üçün biz aşağıdaki massivi göstərək:
arr=[21,27,32,41,52,68,71,89,95]
Tutaq ki bu massivdə axtardığımız element 71 olsun. key=71
Əvvəlcə arr[0] == key yoxlanır. 21!=71 olduğuna görə biz i=1 veririk və hər addımda i*2 indexi taparaq yoxlayırıq. Yəni
i=1 arr[i]>key 27>71 false, onda göre də
i=i*2 arr[2]>key 32>71 false,
i=4 arr[4]>key 52>71 false,
i=8 arr[8]>key 95>71 true Onda indexi 4 olan element ilə 8 olan element arasında binary search aparmağa başlanılır.



Fibonacci Search

Bu axtarış alqoritmi sort olunmuş bir massivdə elementin tapılması üçün istifadə olunur. Parçala və fəthet yanaşmasına biraz bənzəsə də, binary search kimi bərabər parçalara yox, 1:1,618 nisbətibdə bölür. Fibonacci alqoritmi çox sürətli işləyən bir alqoritmdir. Big O notationu O(log n) olduğuna görə sərfəli bir alqoritmdir. İndi isə alqoritmin işləmə prinsipinə baxaq: Daha yaxşı anlamaq üçün bir array götürək
array=[10 15 30 60 90 100 200]
Tutaq ki biz 100 ü axtarırıq
İndi isə massivin uzunluğundan daha böyük və ya ona bərabər olan ən kiçik fibonaççi ədədini götürürük. Bu massivdə 7 element olduğuna görə 8 i götürürük. Fib=8 Bu fibonaççi ədədindən daha əvvəlki 2 sayı da bizə lazım olacaq. Yəni 5 və 3. Fib1=5 Fib2=3. Offset adlanan bir dəyişkənimiz var. əvvəlcə onu -1 götürürük. Baxacağımız indexin formuluna göz gəzdirək: i= min((offset+Fib2),n-1) i=min(2,6) minimum 2 olduğuna görə əvvəlcə 2ci indexə baxılır. Bizim arrayımızdə 2 ci indexdəki say 30 idi, lakin biz 100ü axtarırdıq. 30<100 olduğuna görə Fibonaççi ədədlərimizi özlərindən əvvəlki fibonaççi ədədlərinə çevirlir yəni Fib=5 Fib1=3 Fib2=2 offset dəyişkəni isə ən son alınan indexə çevirlir yəni offset=2 və sayımızı tapana qədər bu dəvam edir

Fibonacci Search'in Binary searchdan fərqləri

1.Fibonacci search massivi bərabər olmayan parçalara bölür

2.Binary Search ortancıl ədədi tapmaq üçün bölmə operatorunu istifadə edir. Fibonacci searchda isə bölmə yox + və - istifadə edilir.

3.Fibonacci searchda sonraki addımlarda bir birinə yaxın elementlər axtarışa verilir. Əgər alqoritm üçün RAMda yer yoxdursa Fibonacci search daha sərfəli ola bilər