ELİF ÖKSÜZALİ 180303010

SELMA GÜNDOĞDU 190303054

SIRALAMA ALGORİTMALARI VE ARAMA ALGORİTMALARI

Raporumuzda sıralama ve arama algoritmaları olmak üzere iki farklı konuyu ele aldık. Çok çeşitli sıralama algoritmaları bulunmaktadır. Biz bu algoritmalar içerisinden **Araya sokma Sıralaması (Insertion Sort), Seçmeli Sıralama (Selection Sort), Kümeleme Sıralaması (Heap Sort) ve Hızlı Sıralama (Quick Sort)** olmak üzere dört farklı çeşidini detaylıca inceleyip avantaj ve dezavantajlarına değinerek çalışma mantık ve prensiplerini hem teorik olarak hem de örnekler yardımıyla açıklayarak algoritmaların durumlarına göre zaman karmaşıklıklarını hesaplayacağız .Aynı şekilde arama algoritma çeşitleri olan **Doğrusal Arama (Linear Search) ve İkili Arama (Binary Search)** algoritmalarınıda raporumuz genelinde açıklayacağız .

SIRALAMA ALGORİTMALARI

Sıralama elimizde rastgele tutulmakta olan bilgilerin bir anahtara bağlı olarak düzenli bir şekilde erişmemizi sağlayan düzenlemelere denir. Bilginin sıralı olmasının bizim için önemi veriye erişmemiz gereken uygulamalar varsa ve bu erişimi çok fazla yapıyorsak verilerin sıralı olması ulaşımı hızlı ve kolay bir şekilde yapmak ve bu işlemler sırasında zaman maliyetinden kurtulmak demektir. Çok çeşitli sıralama algoritmaları vardır. Biz bu algoritmaları kullanırken avantaj ve dezavantajlarını göz önünde bulundurarak en uygun seçimi yapmamız gerekir. Uygun sıralama algoritmasının seçilmiş olması performansıda büyük ölçüde etkilemektedir . Bazı sıralama algoritmalarını detaylandırarak açıklayalım . Yapacak olduğumuz sıralama türlerinde genel olarak küçükten büyüğe sıralama işlemini yapacağız.

Araya sokma Sıralaması (Insertion Sort): Sıralama işlemi küçükte büyüğe veyahut büyükten küçüğe olarak yapılabilir. Biz giriş kısmında da belirttiğimiz gibi dizimizi küçükten büyüğe olacak şekilde adım adım sıralayacağız .

Genel olarak **Araya sokma Sıralamasının (Insertion Sort)** çalışma prensibi: Sıralanacak N elemanlı (N>=2) dizinin ilk iki elmanı alınır kendi aralarında karşılaştırılır. İlk elemanın ikinci elemandan büyük olması durumunda yer değiştirme işlemi uygulanır ancak büyük değilse herhangi bir yer değiştirme işlemi yapılmaz. Daha sonra üçüncü eleman alınır ve ilk üç eleman kendi arasında sıralanır. İlk önce ikinci ve üçüncü eleman karşılaştırılır. İkinci eleman üçüncü elemandan büyükse yer değiştirme işlemi yapılır yer değiştirmeyle ikinci elemanın yerine gelen yeni değer ile birinci eleman karşılaştırılır eğer birinci eleman daha büyük ise tekrar bir yer değiştirme işlemi yapılır. Ancak ikinci ve üçüncü eleman

karşılaştırıldığında ikinci eleman üçüncü elemandan büyük değilse elemanların yerleri değişmez ilk iki elemanı da ilk adımda sıralamıştık. Bu nedenle tekrar kontrol etmeye gerek kalmaz. Bu işlemleri gerçekleştirirken dizinin üçten önceki sıralı durumda olan tarafı ve üçten sonraki sıralanmamış olan taraf olmak üzere ikiye ayrılmış olur. Üçten sonraki sıralanmamış kısım sıralı görünse bile bütün adımları tek tek yaparak emin olmamız gerekir. Bu şekilde işlemleri dizideki sağ taraftaki karmaşık elmanlar sol tarafta sıralanana kadar devam eder. İşlemleri bitirdiğimizde dizinin sıfırıncı indisinde en küçük dizinin (N-1).indisinde en büyük eleman olacak şekilde küçükten büyüğe sıralanmış olur. Vereceğimiz bir örnek üzerinden bu anlattıklarımız adım adım gösterelim.

ÖRNEK:[6, 3, 5, 11, 2, 7, 20, 9, 12, 15] dizisini **Araya sokma Sıralama (Insertion Sort)** algoritmasını kullanarak sıralayınız?

	İşlem öncesi	İşlem sonrası
1.Adım	[6, 3][5, 11, 2, 7, 20, 9, 12, 15]	[3, 6][5, 11, 2, 7, 20, 9, 12, 15]
2.Adım	[3, 6, 5][11, 2, 7, 20, 9, 12, 15]	[3, 5, 6][11, 2, 7, 20, 9, 12, 15]
3.Adım	[3, 5, 6, 11][2, 7, 20, 9, 12, 15]	[3, 5, 6, 11][2, 7, 20, 9, 12, 15]
4.Adım	[3, 5, 6, 11, 2][7, 20, 9, 12, 15]	[2, 3, 5, 6, 11][7, 20, 9, 12, 15]
5.Adım	[2, 3, 5, 6, 11, 7][20, 9, 12, 15]	[2, 3, 5, 6, 7, 11][20, 9, 12, 15]
6.Adım	[2, 3, 5, 6, 7, 11, 20][9, 12, 15]	[2, 3, 5, 6, 7, 11, 20][9, 12, 15]
7.Adım	[2, 3, 5, 6, 7, 11, 20, 9][12, 15]	[2, 3, 5, 6, 7, 9, 11, 20][12, 15]
8.Adım	[2, 3, 5, 6, 7, 9, 11, 20, 12][15]	[2, 3, 5, 6, 7, 9, 11, 12, 20][15]
9.Adım	[2, 3, 5, 6, 7, 9, 11, 12, 20, 15][ ]	[2, 3, 5, 6, 7, 9, 11, 12, 15, 20][ ]

- 1.Adım
  - 2. Elemanı seç : [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]
  - 3 < 6 ? (doğru) => [3, 6, 5, 11, 2, 7, 20, 9, 12, 15]

İkinci(3) eleman seçildi birinci (6) ve ikinci(3) eleman karşılaştırıldı ikinci(3) elemanın birinci(6) elemandan küçük olduğu görüldü yer değiştirme işlemi yapıldı.

- 2.Adım
  - 3. Elemanı seç : [3, 6, 5, 11, 2, 7, 20, 9, 12, 15]
  - 5 < 6 ? (doğru) => [3, 5, 6, 11, 2, 7, 20, 9, 12, 15]
  - 5 < 3 ? (yanlış) => [3, 5, 6, 11, 2, 7, 20, 9, 12, 15]

Üçüncü(5) eleman seçildi ilk üç eleman kendi arasında sıralamaya alındı. Üçüncü(5) ve ikinci(6) eleman karşılaştırıldı üçüncü(5) elemanın ikinci(6)elemandan küçük olduğu görüldü ve yer değiştirme işlemi yapıldı. Yeni gelen ikinci(5) eleman ile ilk(3) eleman karşılaştırıldı ilk(3) elemanın yeni gelen ikinci(5) elemandan küçük olduğu görüldü yer değiştirme yapılmadı.

- 3.Adım
    - 4. Elemanı seç : [3, 5, 6, 11, 2, 7, 20, 9, 12, 15]
    - 11 < 6 ? (yanlış) => [3, 5, 6, 11, 2, 7, 20, 9, 12, 15]

Dördüncü(11) eleman seçildi ilk dört elemanın kendi arasında sıralamaya alındı. Dördüncü(11) eleman ile üçüncü(6) eleman karşılaştırıldı. Dördüncü(11) elemanın üçüncü(6) elemandan küçük olmadığı görüldü yer değiştirme işlemi yapılmadı. İlk üç eleman zaten sıralı durumdaydı işlem yapmaya gerek kalmadı .

- 4.Adım
    - 5. Elemanı seç : [3, 5, 6, 11, 2, 7, 20, 9, 12, 15]
    - 2 < 11 ? (doğru) =>[3, 5, 6, 2, 11, 7, 20, 9, 12, 15]
    - 2 < 6 ? (doğru)=> [3, 5, 2, 6, 11, 7, 20, 9, 12, 15]
    - 2 < 5 ? (doğru) =>[3, 2, 5, 6, 11, 7, 20, 9, 12, 15]
    - 2 < 3 ? (doğru) =>[2, 3, 5, 6, 11, 7, 20, 9, 12, 15]

Beşinci(2) eleman seçildi ilk beş elemen kendi arasında sıralamaya alındı1. Beşinci(2) eleman kendinden önceki alemanla(11) karşılaştırıldı kendinden önceki elemandan küçük olduğu için yer değiştirme işlemi yapıldı .Dördüncü elemen oldu .Geldiği konumda tekrar kendinden önceki elemanla(6) karşılaştırıldı karşılaştırıldığı elemnadan küçük olduğu görüldü ve tekrar yer değiştirme işlemi yapıldı. Bu işlem kendinden önce daha küçük bir elemen kalmayıncaya kadar devam ettirildi. Böylece eleman listenin en başına taşınmış oldu.

- 5.Adım
    - 6. Elamanı seç : [2, 3, 5, 6, 11, 7, 20, 9, 12, 15]
    - 7 < 11 ? (doğru) => [2, 3, 5, 6, 7, 11, 20, 9, 12, 15]
    - 7 < 6 ? (yanlış) => [2, 3, 5, 6, 7, 11, 20, 9, 12, 15]

Altıncı(7) eleman seçildi ilk altı eleman kendi arasında sıralamaya alındı. Altıncı(7) elemana kendinden önceki elemandan küçük olma şartını sağlandığı müddetçe yer değiştirme işlemi yapıldı. Şartı sağlamadığı yerde dizinin ilk altı elemanı sıralanmış duruma geldi.

- 6.Adım
    - 7.Elemanı seç : [2, 3, 5, 6, 7, 11, 20, 9, 12, 15]
    - 20 < 11 ? (yanlış) => [2, 3, 5, 6, 7, 11, 20, 9, 12, 15]

Yedinci(20) eleman seçildi ilk yedi eleman kendi arasında sıralamaya alındı. Yedinci eleman kendinden önceki elemandan küçük olmadığı için herhangi bir işlem yapmaya gerek kalmadı. İlk yedi eleman şu anki durumda kendi arasında sıralı konumdadır .

- 7.Adım
  - 8. Elemanı seç : [2, 3, 5, 6, 7, 11, 20, 9, 12, 15]
  - 9 < 20 ? (doğru) => [2, 3, 5, 6, 7, 11, 9, 20, 12, 15]
  - 9 < 11 ? (doğru) => [2, 3, 5, 6, 7, 9, 11, 20, 12, 15]
  - 9 < 7 ? (yanlış) => [2, 3, 5, 6, 7, 9, 11, 20, 12, 15]

Sekizinci(9) eleman seçildi ilk sekiz eleman kendi arasında sıralamaya alındı. Sekizinci(9) elemana kendinden önceki elemandan küçük olma şartını sağlandığı müddetçe yer değiştirme işlemi yapıldı. Şartı sağlamadığı yerde dizinin ilk sekiz elemanı sıralanmış duruma geldi.

- 8.Adım
  - 9. Elemanı seç : [2, 3, 5, 6, 7, 9, 11, 20, 12, 15]
  - 12 < 20 ? (doğru) => [2, 3, 5, 6, 7, 9, 11, 12, 20, 15]
  - 12 < 11 ? (yanlış) => [2, 3, 5, 6, 7, 9, 11, 12, 20, 15]

Dokuzuncu(12) eleman seçildi ilk dokuz eleman kendi arasında sıralamaya alındı. Dokuzuncu(12) elemana kendinden önceki elemandan küçük olma şartını sağlandığı müddetçe yer değiştirme işlemi yapıldı. Şartı sağlamadığı yerde dizinin ilk dokuz elemanı sıralanmış duruma geldi.

- 9. Adım
  - 10. Elemanı seç : [2, 3, 5, 6, 7, 9, 11, 12, 20, 15]
  - 15 < 20 ? (doğru) => [2, 3, 5, 6, 7, 9, 11, 12, 15, 20]

Onuncu(15) son eleman seçildi ilk on eleman kendi arasında sıralamaya alındı. Onuncu(15) son elemana kendinden önceki elemandan küçük olma şartını sağlandığı müddetçe yer değiştirme işlemi yapıldı. Şartı sağlamadığı yerde işlemlerimiz sonlanmış dizimiz **Araya sokma Sıralama (Insertion Sort)** algoritmasını kullanarak sıralanmış oldu.

Dizimizin son hali : [2, 3, 5, 6, 7, 9, 11, 12, 15, 20]

Sıralama algoritmamızın zaman karmaşıklığına değinecek olursak:

- En iyi durumda dizimiz sıralı halde gelebilir.
  [2, 3, 5, 6, 7, 9, 11, 12, 15, 20]
  Bu durumda elemanlar tek tek gezilerek sağındaki elemanla karşılaştırma yapılır. Herhangi bir yer değiştirme işlemi yapmadan ilerleriz. Buda O(n) lik zaman maliyetine neden olur
- En kötü durumda dizimiz ters sıralı halde gelebilir.
  [20, 15, 12, 11, 9, 7, 6, 5, 3, 2]
  Bu durumda karmaşıklık hesabı yaparken her adımda bütün elemanları gezmemiz gerekir bundan dolayı zaman karmaşıklığımız O(n^2) olur.

Seçmeli Sıralama (Selection Sort): Seçmeli sırlama isminden de anlaşıldığı gibi seçerek işlemler yapar. Küçükten büyüğe sıralamada sırayla dizi içerisindeki en küçük değerleri seçer ve ilk sıradan başlayarak sıralar aynı işlemi büyükten küçüğe sıralamada da yapar. Tek farkı bu sefer büyük olan elemanları seçer ve dizinin başından başlayarak sıralar. Biz giriş kısmında da belirttiğimiz gibi dizimizi küçükten büyüğe olacak şekilde adım adım sıralayacağız .

Genel olarak **Seçmeli Sıralama (Selection Sort)**çalışma prensibi: Sıralanacak N elemanlı (N>=2)dizide sıralama işlemi yapılırken ilk elemandan başlanır. Birinci eleman ile N. eleman arasındaki en küçük değer bulunur ve ilk elemanla yer değiştirilir. Ancak zaten birinci elemandaki değer dizideki en küçük elemansa herhangi bir işlem yapılmaz. Daha sonra ikinci eleman alınır ikinci eleman ile N. eleman arasındaki en küçük değer bulunur ve ikinci elemanla yer değiştirilir. Bu şekilde dizinin sonuna kadar ilerlenir en son (N-1). Eleman ile N. Eleman karşılaştırıp küçük olan değer yer değiştirilerek (N-1).eleman olur. Değerlerin eşit olması durumunda isteğe göre yer değiştirme işlemi yapılabilir ancak gerekli değildir. Bu işlemler doğru bir şekilde yapıldığı takdirde dizimiz sıralanmış olur.

ÖRNEK:[6, 3, 5, 11, 2, 7, 20, 9, 12, 15] dizisini **Seçmeli Sıralama (Selection Sort)** algoritmasını kullanarak sıralayınız?

- 1.Adım
  - En küçük elemanı ara: [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]
  - 2 < 6 ? (doğru) => [2, 3, 5, 11, 6, 7, 20, 9, 12, 15]

İlk eleman(6) seçildi. İlk eleman ile N. eleman arasındaki en küçük(2) değer bulundu iki değer karşılaştırıldı. Bulunan değer ilk elemandan daha küçük olma şartını sağladığı için yer değiştirme işlemi yapıldı.

- 2. Adım
  - En küçük elemanı ara: [2, 3, 5, 11, 6, 7, 20, 9, 12, 15]
  - 3 < 3 ? (yanlış) => [2, 3, 5, 11, 6, 7, 20, 9, 12, 15]

İkinci(3) eleman seçildi. İkinci eleman ile N. eleman arsındaki en küçük değer kendisi olduğu için herhangi bir yer değiştirme işlemi olmadı.

- 3. Adım
  - En küçük elemanı ara: [2, 3, 5, 11, 6, 7, 20, 9, 12, 15]
  - 5 < 5 ? (yanlış) => [2, 3, 5, 11, 6, 7, 20, 9, 12, 15]

Üçüncü(5) eleman seçildi. İkinci eleman ile N. eleman arsındaki en küçük değer kendisi olduğu için herhangi bir yer değiştirme işlemi olmadı.

- 4. Adım
    - En küçük elemanı ara: [2, 3, 5, 11, 6, 7, 20, 9, 12, 15]
    - 6 < 11 ? (doğru) => [2, 3, 5, 6, 11, 7, 20, 9, 12, 15]

Dördüncü(11) eleman seçildi. Dördüncü eleman ile N. Eleman arasındaki en küçük(6) değer bulundu .Bulunan değer ile dördüncü eleman karşılaştırıldı ve bulunan değerin dördüncü elemandan küçük olduğu görüldü yer değiştirme işlemi yapıldı.

- 5.Adım
    - En küçük elemanı ara: [2, 3, 5, 6, 11, 7, 20, 9, 12, 15]
    - 7 < 11 ? (doğru) => [2, 3, 5, 6, 7, 11, 20, 9, 12, 15]

Beşinci(11) eleman seçildi. Beşinci eleman ile N. Eleman arasındaki en küçük(7) değer bulundu .Bulunan değer ile beşinci eleman karşılaştırıldı ve bulunan değerin beşinci elemandan küçük olduğu görüldü yer değiştirme işlemi yapıldı.

- 6.Adım
    - En küçük elemanı ara: [2, 3, 5, 6, 7, 11, 20, 9, 12, 15]
    - 9 < 11 ? (doğru) => [2, 3, 5, 6, 7, 9, 20, 11, 12, 15]

Altıncı(11) eleman seçildi. Altıncı eleman ile N. Eleman arasındaki en küçük(9) değer bulundu. Bulunan değer ile altıncı eleman karşılaştırıldı ve bulunan değerin altıncı elemandan küçük olduğu görüldü yer değiştirme işlemi yapıldı.

- 7.Adım
    - En küçük elemanı ara: [2, 3, 5, 6, 7, 9, 20, 11, 12, 15]
    - 11 < 20 ? (doğru) => [2, 3, 5, 6, 7, 9, 11, 20, 12, 15]

Yedinci(20) eleman seçildi. Yedinci eleman ile N. Eleman arasındaki en küçük(11) değer bulundu .Bulunan değer ile yedinci eleman karşılaştırıldı ve bulunan değerin yedinci elemandan küçük olduğu görüldü yer değiştirme işlemi yapıldı.

- 8. Adım
    - En küçük elemanı ara: [2, 3, 5, 6, 7, 9, 11, 20, 12, 15]
    - 12 < 20 ? (doğru) => [2, 3, 5, 6, 7, 9, 11, 12, 20, 15]

Sekizinci(20) eleman seçildi. Sekizinci eleman ile N. Eleman arasındaki en küçük(12) değer bulundu .Bulunan değer ile sekizinci eleman karşılaştırıldı ve bulunan değerin sekizinci elemandan küçük olduğu görüldü yer değiştirme işlemi yapıldı.

- 9. Adım
    - En küçük elemanı ara: [2, 3, 5, 6, 7, 9, 11, 12, 20, 15]
    - 15 < 20 ? (doğru) => [2, 3, 5, 6, 7, 9, 11, 12, 15, 20]

Dokuzuncu(20) eleman seçildi. Dokuzuncu eleman ile N. Eleman arasındaki en küçük(15) değer bulundu. Bulunan değer ile sekizinci eleman karşılaştırıldı ve bulunan değerin dokuzuncu elemandan küçük olduğu görüldü yer değiştirme işlemi yapıldı. Dizinin sonuna geldik son elemanada dizinin en büyük elemanı yerleşti. Sıralama tamamlanmış oldu.

Sıralama algoritmamızın zaman karmaşıklığına değinecek olursak:

- En iyi durumda dizimiz sıralı halde gelebilir.
  [2, 3, 5, 6, 7, 9, 11, 12, 15, 20]
  Bu durumda yer değiştirme işlemi yapmamıza gerek yoktur. Ancak yine de bütün elemanları teker teker kontrol etmemiz gerekir. Bu nedenle zaman karmaşıklığı O(n^2) olur.
- En kötü durumda dizimiz ters sıralı halde gelebilir.
  [20, 15, 12, 11, 9, 7, 6, 5, 3, 2]
  Bu durumda en iyi durumdan farklı olarak yer değiştirme işlemi işlemi de yapılmaktadır. Bu da karmaşıklığı etkilemez. Zaman karmaşıklığımız yine O (n^2)olur.

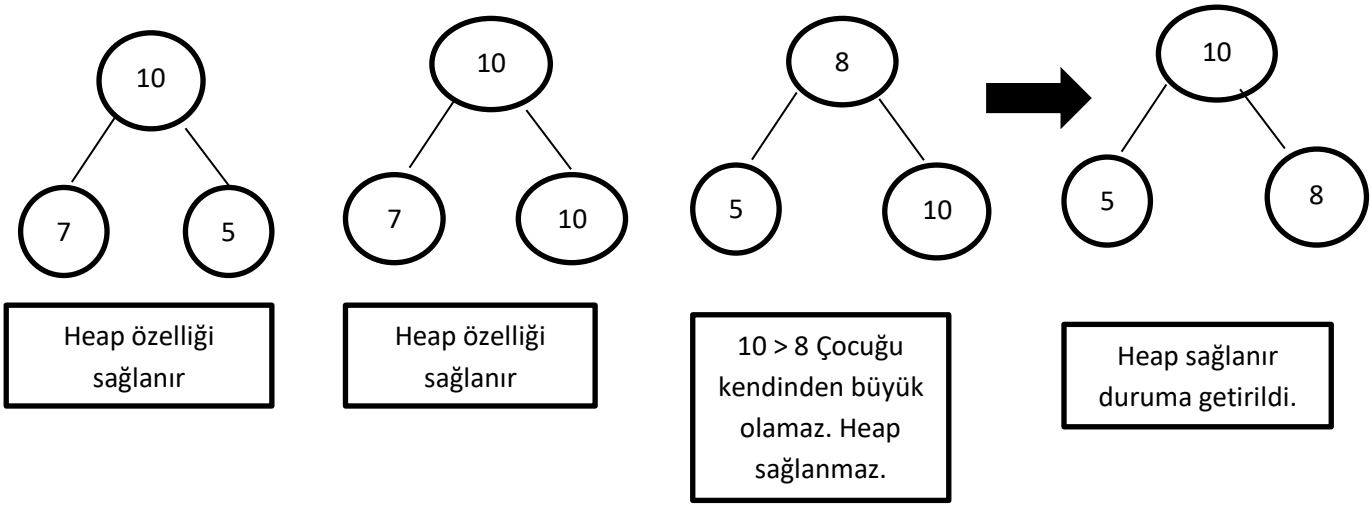
Kümeleme Sıralaması (Heap Sort): Bu sıralama türünde de diğer sıralama türlerindeki gibi asıl amaç elimizdeki verileri düşük zaman karmaşıklığı ile etkin bir şekilde sıralayabilmektir. Elimizdeki verileri ağacımıza düğüm olarak ekleyerek sıralama işlemini yapılmaya çalışılır.

Genel olarak **Kümeleme Sıralamasının (Heap Sort)**çalışma prensibi: Sıralamak istenilen dizinin elemanları teker teker ağaç yapımıza düğümler şekilde eklenir. Bu ekleme sırasında dikkat edilmesi gereken bazı hususlar vardır. Bu hususlar:

- Yerleştirme işlemi yapılırken bir düğümün çocukları kendisinden daha büyük değerler alamaz.(heapify işlemi) Bir düğümün çocukları kendisinden küçük değerlere sahip olacağı gibi düğümün kendi değerine eşit de olabilir. Bu özellik heap özelliği olarak da adlandırılır.
- Oluşturulan bir düğümün en fazla iki tane çocuğu olabilir.

Bu özellikler ağaca eklenen bütün düğümlerde sağlanmış olması gereklidir. Ağacımız kademelerden oluşmaktadır. Bu kademelere ekleme işlemi her zaman en soldaki boş yere olacak şekilde yapılır. Ekleme işlemi sırasında heap özelliği bozulmuş ise düzeltilerek devam edilir. Bu şartlar doğrultusunda dizideki bütün elemanlar ağaca eklendikten sonra eğer ağaç heap özelliğine uygun bir şekilde oluşturulmuş ise ağacın en başındaki eleman dizideki en büyük değere sahip eleman olur. Bu en büyük değer kökten koparılır ve ekleme işlemini

soldan yaptığımız için en sondaki kademenin en sağındaki eleman en son eklenen eleman olarak koparılır ve köke yerleştirilir. Heap özelliği kontrol edilir gerekli düzenlemeler yapılır. Ağaçtaki bütün elemanlar koparılıncaya kadar bu işlem devam eder. Bu işlemler sonucunda dizi sıralanmış olur.

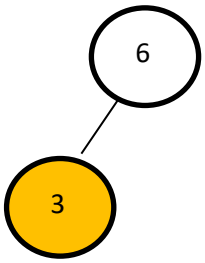


ÖRNEK:[6, 3, 5, 11, 2, 7, 20, 9, 12, 15] dizisini **Kümeleme Sıralama (Heap Sort)**algoritmasını kullanarak sıralayınız?

- 1. Adım
  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]

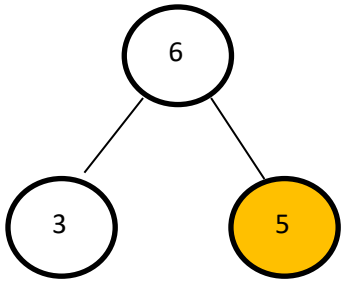
Dizimizin ilk elemanı ile ağacın ilk düğümünü oluşturuldu. Heap özelliğini bozacak herhangi bir durum söz konusu değil.

- 2. Adım
  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]



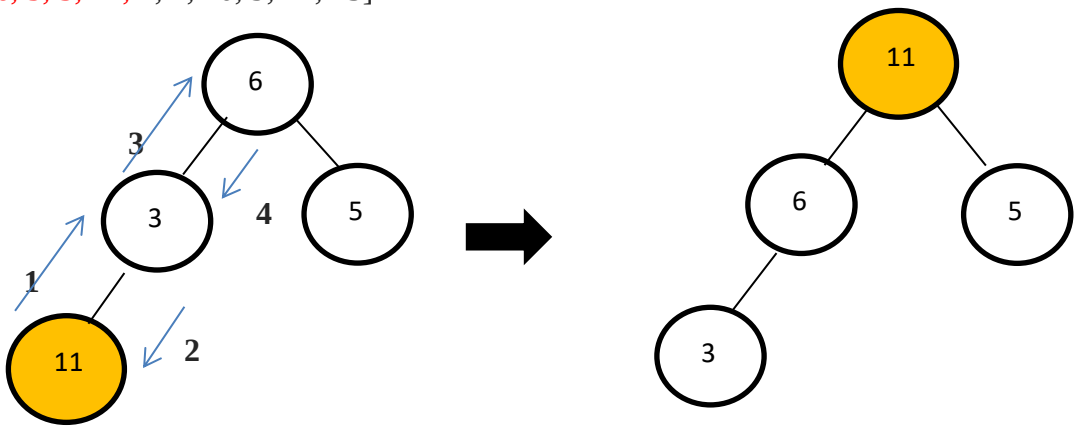
Dizinin ikinci elemanı ağaca eklendi. Heap özelliği kontrol edildi ve heap özelliğinin sağlandığı görüldü.

- 3. Adım
  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]



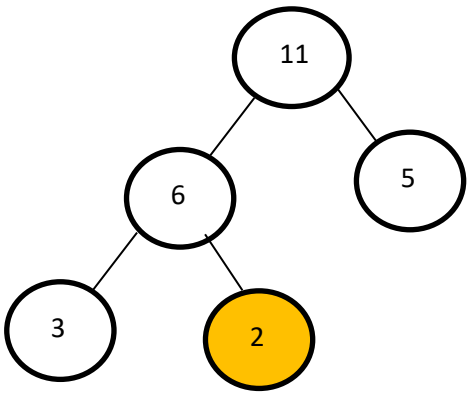
Dizinin üçüncü elemanı bir önceki adımda heap özelliğine uyan ağacımıza eklendi. Heap özelliği kontrol edildi ve heap özelliğinin sağlandığı görüldü.

- 4. Adım
  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]



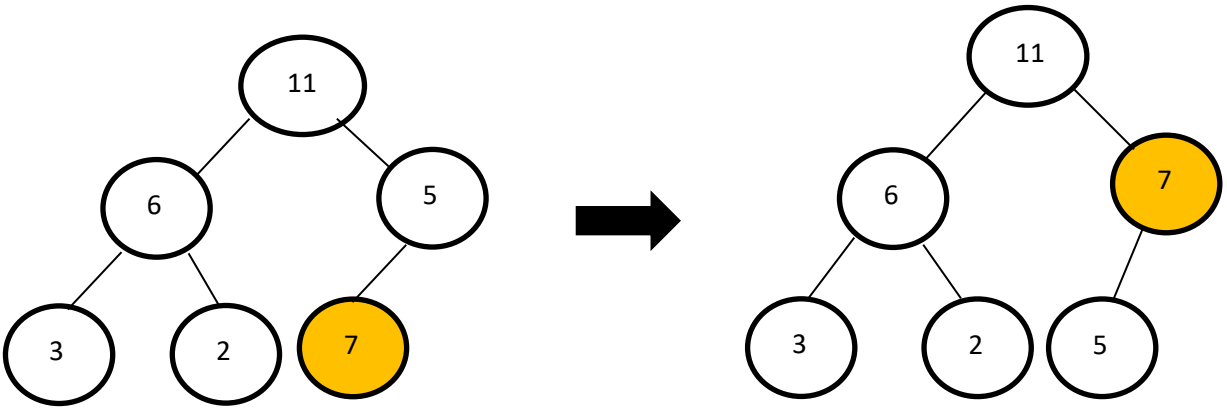
Dizinin dördüncü elemanı bir önceki adımda heap özelliğine uyan ağacımıza eklendi. Heap özelliği kontrol edildi ve son eklenen düğümün heap özelliğini bozduğu görüldü. Gerekli yer değiştirme işlemleri yapılarak ağaçta heap özelliği sağlandı.

- 5. Adım
  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]



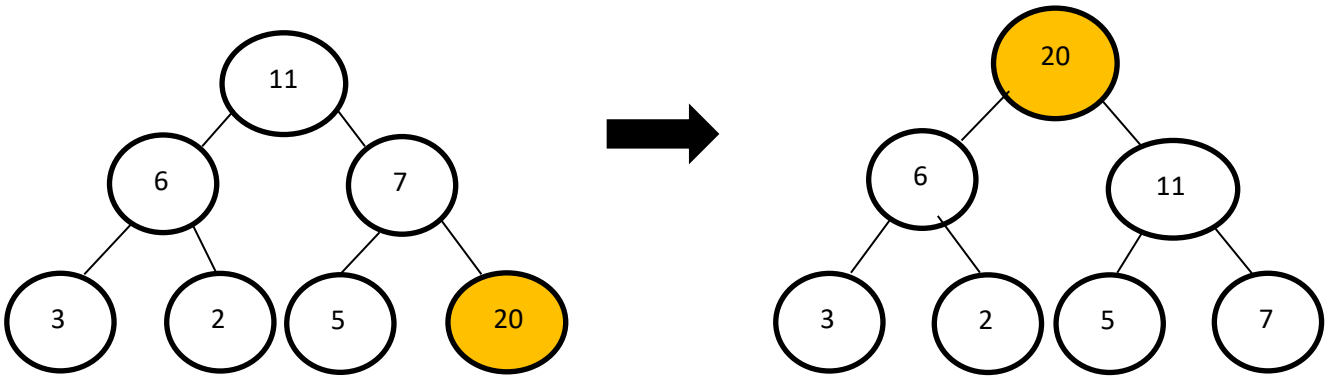
Dizinin beşinci elemanı bir önceki adımda heap özelliğine göre düzenlenmiş ağacımıza eklendi. Heap özelliği kontrol edildi ve heap özelliğinin sağlandığı görüldü.

- 6. Adım
  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]



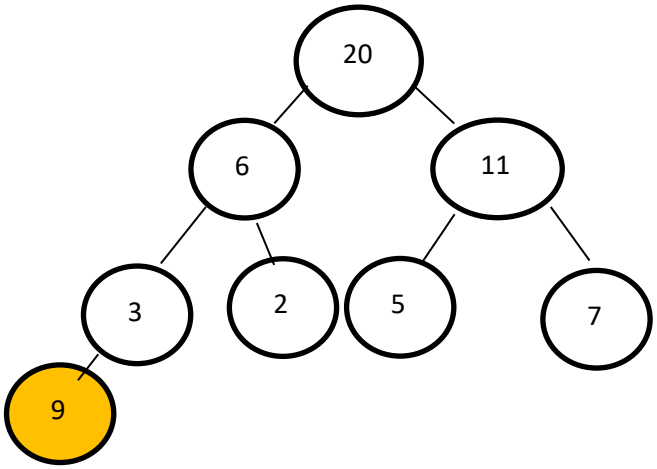
Dizinin altıncı elemanı bir önceki adımda heap özelliğine uyan ağacımıza eklendi. Heap özelliği kontrol edildi ve son eklenen düğümün heap özelliğini bozduğu görüldü. Gerekli yer değiştirme işlemleri yapılarak ağaçta heap özelliği sağlandı.

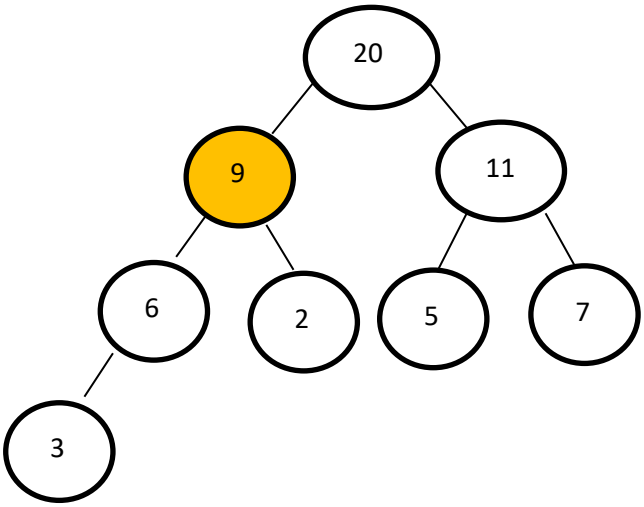
- 7. Adım
  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]



Dizinin yedinci elemanı bir önceki adımda heap özelliğine uyan ağacımıza eklendi. Heap özelliği kontrol edildi ve son eklenen düğümün heap özelliğini bozduğu görüldü. Gerekli yer değiştirme işlemleri yapılarak ağaçta heap özelliği sağlandı.

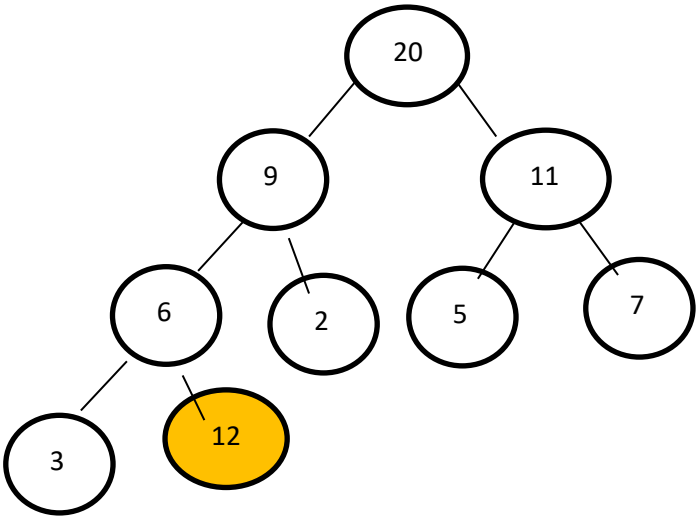
- 8. Adım
  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]

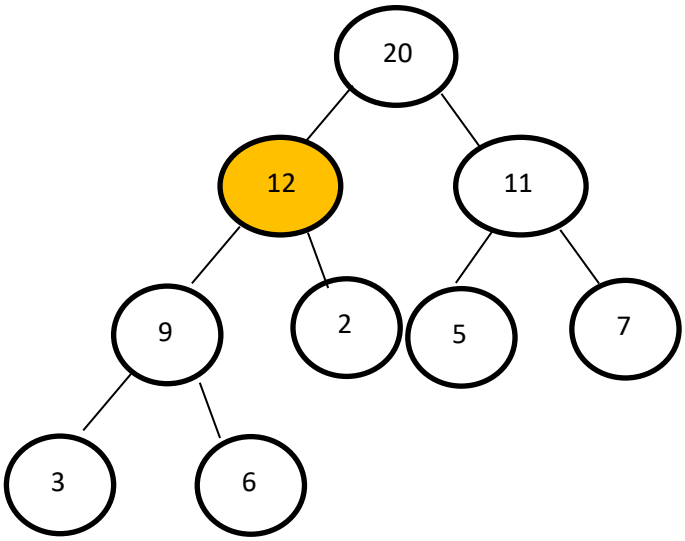




Dizinin sekizinci elemanı bir önceki adımda heap özelliğine uyan ağacımıza eklendi. Heap özelliği kontrol edildi ve son eklenen düğümün heap özelliğini bozduğu görüldü. Gerekli yer değiştirme işlemleri yapılarak ağaçta heap özelliği sağlandı.

- 9. Adım
- [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]

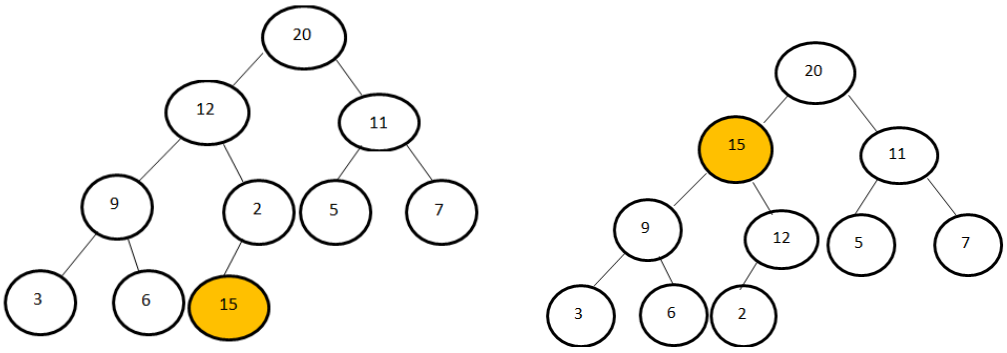




Dizinin dokuzuncu elemanı bir önceki adımda heap özelliğine uyan ağacımıza eklendi. Heap özelliği kontrol edildi ve son eklenen düğümün heap özelliğini bozduğu görüldü. Gerekli yer değiştirme işlemleri yapılarak ağaçta heap özelliği sağlandı.

- 10. Adım

  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]

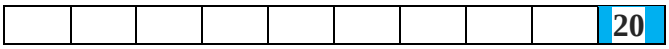


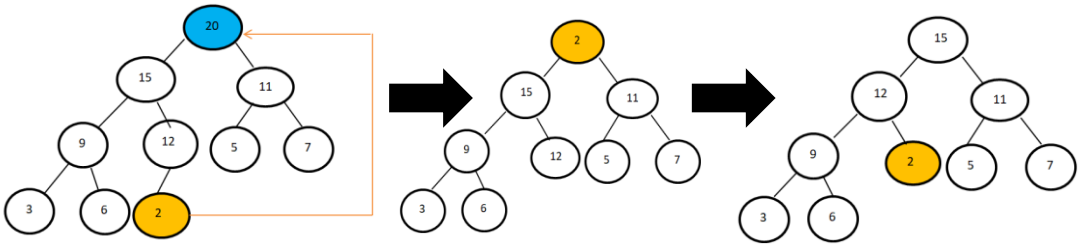
Dizinin onuncu son elemanı bir önceki adımda heap özelliğine uyan ağacımıza eklendi. Heap özelliği kontrol edildi ve son eklenen düğümün heap özelliğini bozduğu görüldü. Gerekli yer değiştirme işlemleri yapılarak ağaçta heap özelliği sağlandı.

Bu işlem sonrası dizideki bütün elemanları heap kuralına uygun olarak ağaca eklendi. Ağacın en tepesindeki değer ağaç genelinde heap sağlandığı için ağaç üzerinde yer alan düğümlerdeki verilerin en büyük olanıdır. Biz küçükten büyüğe sıralama yaptığımız için ağaçın en başındaki düğüm koparılır ve sıralı dizimizn son elemanı olarak yazılır. Koparılan elemanlar bu şekilde sondan başa doğru çıkarılma sırası ile yazılır. Koparılan elemanın yerine en son eklemiş olduğumuz değer yazılır ve Heap ın bozulup bozulmadığı kontrol edilir. Bozulan Heap gerekli düzenlemeler yapılarak tekrardann sağlanır. Bu şekilde ağaçtaki bütün düğümler koparılır ve sırallı listemize elkenir böylece dizimiz **Kümeleme Sıralama (Heap Sort)** algoritmasına göre sıralanmış olur.

- 11. Adım

  Sıralı dizi:



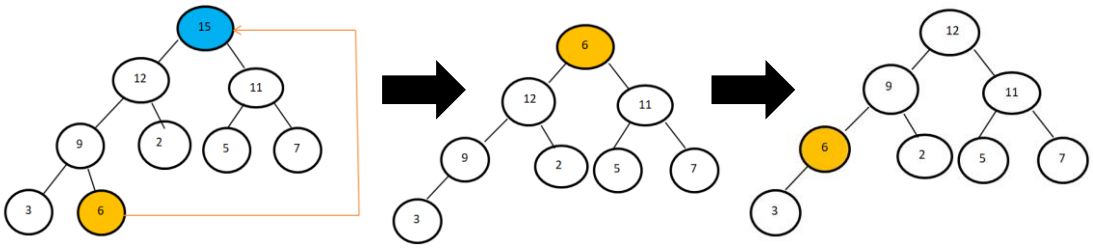


Ağacımızın en tepesinde bulunan düğüm kopartılır. Koparılan düğümün yerine en alt katmanın en sağındaki eleman(2) taşınır ve tekrar head sağlanıp sağlanmadığı kontrol edilir sağlanmadığı durumda sağlanana kadar gerekli yer değiştirme işlemleri yapılır.

- 12. Adım

  Sıralı dizi:

								15	20

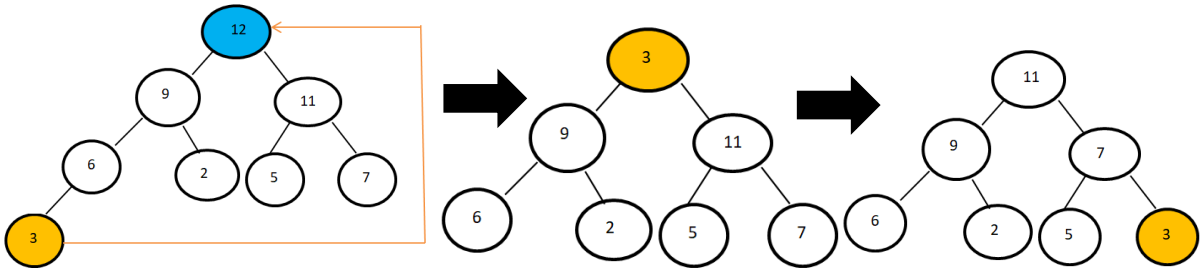


Ağacımızın en tepesinde bulunan düğüm kopartılır. Koparılan düğümün yerine en alt katmanın en sağındaki eleman(6) taşınır ve tekrar head sağlanıp sağlanmadığı kontrol edilir sağlanmadığı durumda sağlanana kadar gerekli yer değiştirme işlemleri yapılır.

- 13. Adım

  Sıralı dizi:

							12	15	20

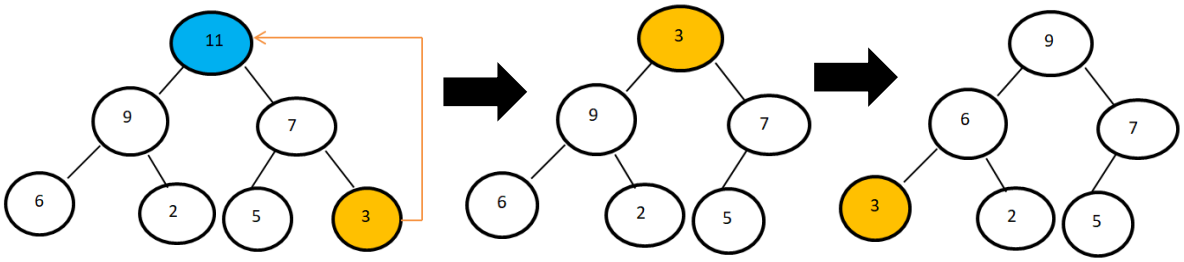


Ağacımızın en tepesinde bulunan düğüm kopartılır. Koparılan düğümün yerine en alt katmanın en sağındaki eleman(3) taşınır ve tekrar head sağlanıp sağlanmadığı kontrol edilir sağlanmadığı durumda sağlanana kadar gerekli yer değiştirme işlemleri yapılır.

- 14. Adım

  Sıralı dizi:

						11	12	15	20

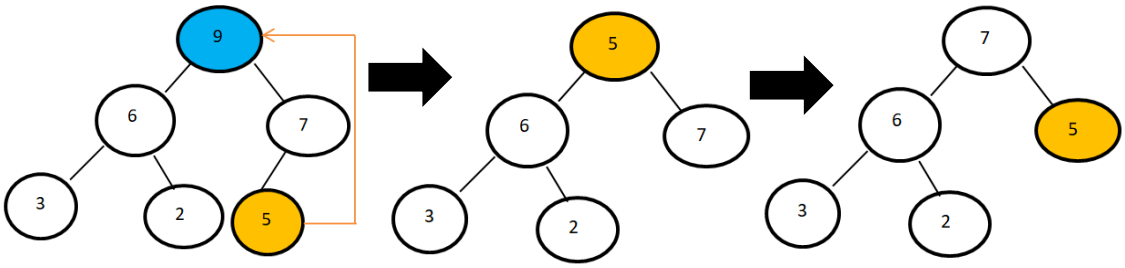


Ağacımızın en tepesinde bulunan düğüm kopartılır. Koparılan düğümün yerine en alt katmanın en sağındaki eleman(3) taşınır ve tekrar head sağlanıp sağlanmadığı kontrol edilir sağlanmadığı durumda sağlanana kadar gerekli yer değiştirme işlemleri yapılır.

- 15. Adım

  Sıralı dizi:

					9	11	12	15	20

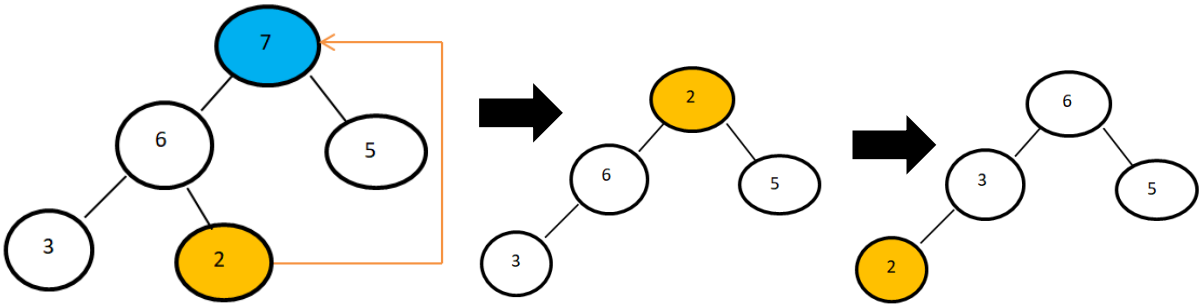


Ağacımızın en tepesinde bulunan düğüm kopartılır. Koparılan düğümün yerine en alt katmanın en sağındaki eleman(5) taşınır ve tekrar head sağlanıp sağlanmadığı kontrol edilir sağlanmadığı durumda sağlanana kadar gerekli yer değiştirme işlemleri yapılır.

- 16. Adım

  Sıralı dizi:

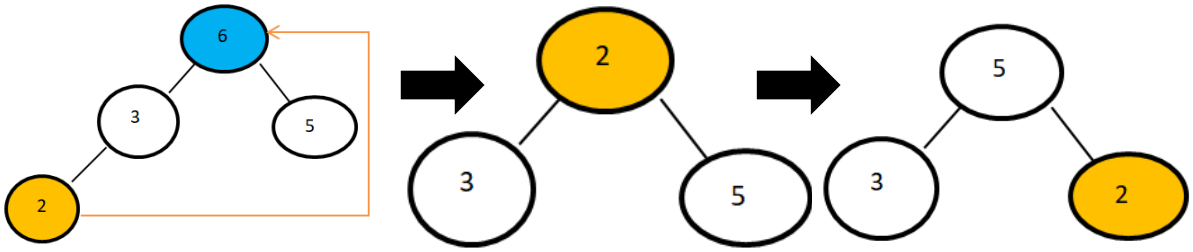
				7	9	11	12	15	20



Ağacımızın en tepesinde bulunan düğüm kopartılır. Koparılan düğümün yerine en alt katmanın en sağındaki eleman(2) taşınır ve tekrar head sağlanıp sağlanmadığı kontrol edilir sağlanmadığı durumda sağlanana kadar gerekli yer değiştirme işlemleri yapılır.

- 17. Adım
  Sıralı dizi:

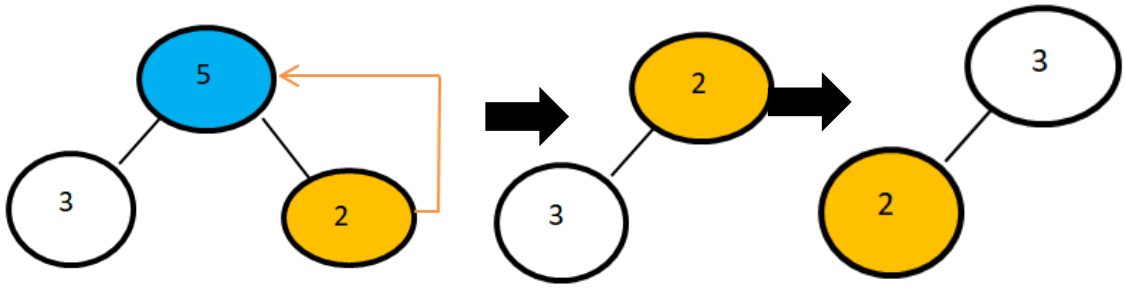
			6	**7**	**9**	**11**	**12**	**15**	**20**



Ağacımızın en tepesinde bulunan düğüm kopartılır. Koparılan düğümün yerine en alt katmanın en sağındaki eleman(2) taşınır ve tekrar head sağlanıp sağlanmadığı kontrol edilir sağlanmadığı durumda sağlanana kadar gerekli yer değiştirme işlemleri yapılır.

- 18. Adım
  Sıralı dizi:

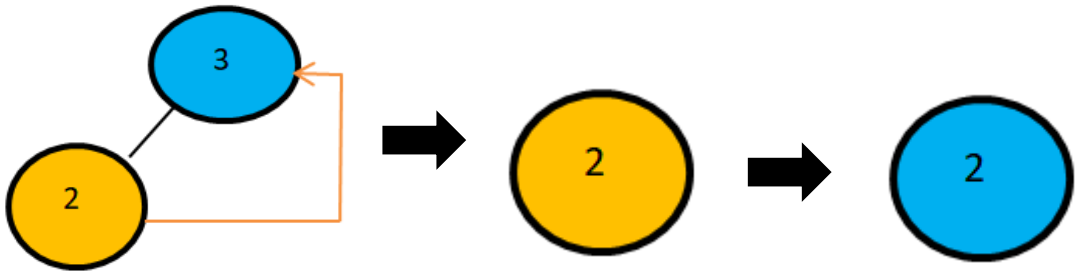
		5	**6**	**7**	**9**	**11**	**12**	**15**	**20**



Ağacımızın en tepesinde bulunan düğüm kopartılır. Koparılan düğümün yerine en alt katmanın en sağındaki eleman(2) taşınır ve tekrar head sağlanıp sağlanmadığı kontrol edilir sağlanmadığı durumda sağlanana kadar gerekli yer değiştirme işlemleri yapılır.

- 19. Adım
  Sıralı dizi:

2	**3**	**5**	**6**	**7**	**9**	**11**	**12**	**15**	**20**



Son adımda ağaçta kalan son düğümde koparılır ve elemanlarımız sıralı diziye eklenir böylece dizi sıralanmış olur.

Sıralama algoritmamızın zaman karmaşıklığına değinecek olursak:

- En iyi durumda dizinin bütün elemanları aynı değere sahip olabilir. Böyle bir durumda heapify işlemi yapmamız gerekmez. Zaman karmaşıklığımız O(n) olur.

- En kötü durumda ağaç üzerinde düğümün doğru yerde olup olmadığını kontrol etmek için heapify işlemi yapmamız gerekir. Bir heapify işleminin karmaşıklığı O(log n) değere sahiptir. Dizide N eleman var. Bu sıralamada zaman karmaşıklığımız O(n log n) olur.

Hızlı Sıralama (Quick Sort): Verilerin bellekte sıralı bir şekilde tutulması için geliştirilen algoritmaların sıklıkla kullanılanlarından sadece bir tanesidir. Basitçe sıralanacak olan dizimizin pivot değeri belirlenir ve belirlediğimiz bu pivot değerine göre bölünür. Bu bölünme işlemine veriler bölünemeyecek alt kümeler haline gelene kadar devam edilir. Anlaşıldığı üzere bu algoritma genelinde böl-fethet(divide-and-conquer)görüşüne uygun olarak çalışmaktadır.

Genel olarak **Hızlı Sıralamanın (Quick Sort)** çalışma prensibi: Sıralanacak N elmanlı bir diziyi tek elemanlı alt kümeler halinde kalacak şekilde bölerek sıralama işlemini uygulayacağız. Öncelikle yapmamız gereken pivot değerini belirlemektir.

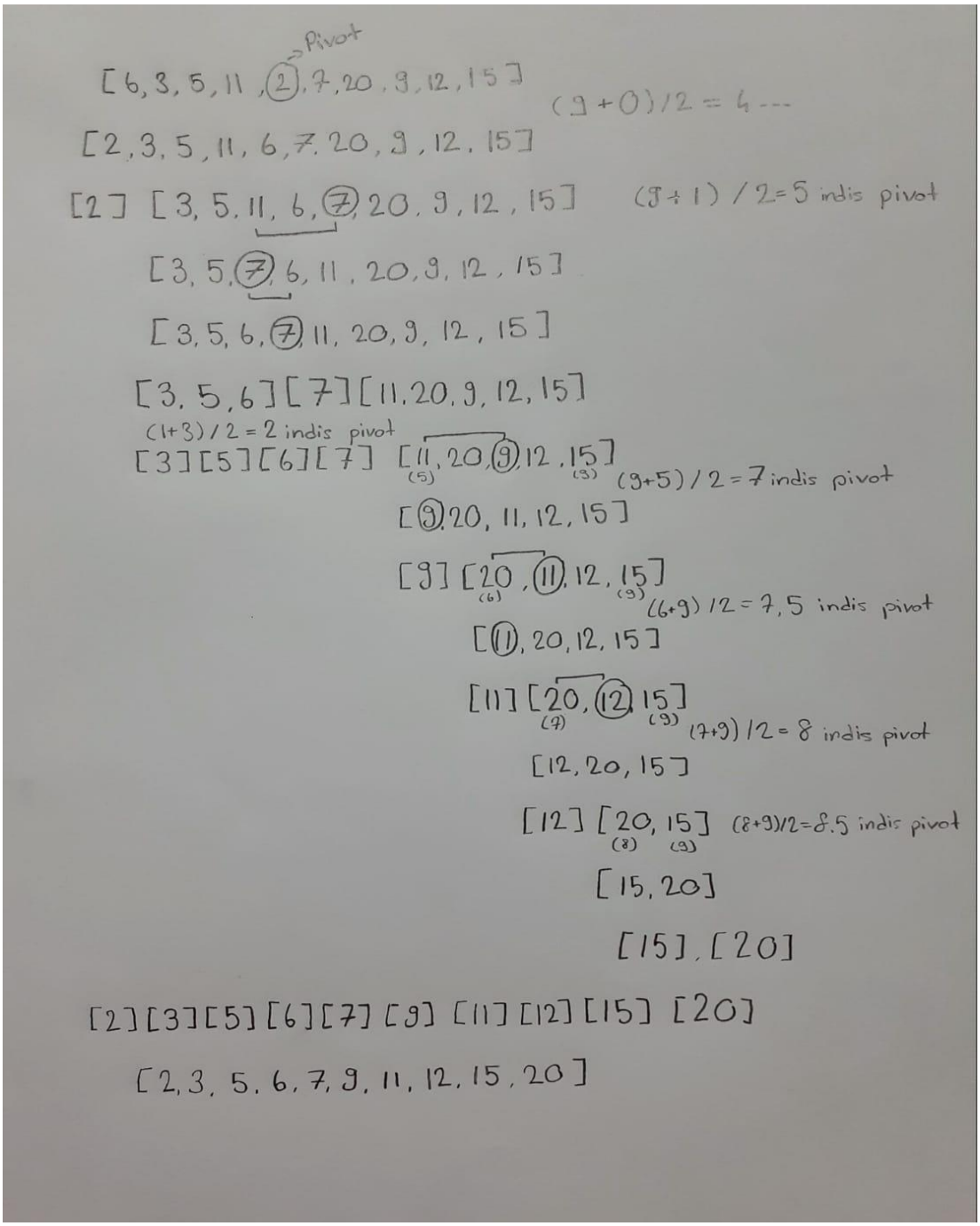
Pivot değerimizi isteğe göre:

- Her zaman ortadaki eleman(
- Her zaman ilk eleman
- Her zaman son eleman
- Rastgele eleman

Olarak farklı şekillerde seçebiliriz .

Pivot değerimizi seçtikten sonra elimizde ki elemanları pivot değerinden küçük olan değerleri pivotun soluna büyük olan değerleri pivotun sağına olacak şekilde gruplandırılır. Eğer dizide pivot değerine eşit değerler mevcut ise isteğe bağlı olarak ister sol alt kümeye ister sağ alt kümeye dahil edilebilir. Aynı işlemler yeni oluşan sağ ve sol alt kümelere de uygulanır. Bu işlemleri ta ki oluşan yeni alt kümelerde tek eleman kalıp da bölme işlemi yapılamayacak duruma gelene kadar tekrar ettirilir. Bu şekilde oluşan yeni dizimizin verileri sıralanmış olur.

ÖRNEK:[6, 3, 5, 11, 2, 7, 20, 9, 12, 15] dizisini **Hızlı Sıralamanın (Quick Sort)** algoritmasını kullanarak sıralayınız?



Sıralama algoritmamızın zaman karmaşıklığına değinecek olursak:

- En iyi durumda seçtiğimiz pivot değeri dizinin ortanca değeridir. Dolayısıyla böyle bir pivot sayesinde dizimizi hep ikiye bölerek ilerleriz. Yani bu durumda böl ve fethet prensibini kullanmış oluruz. Bu prensibin karmaşıklığı O (log n) kadardır. Dizmizde bulunan N tane eleman için bu sıralama işleminin karmaşıklığı O (n log n) olur.
- En kötü durumda ise pivot değerimiz her defasında en büyük sayı yada en küçük sayılar olarak seçilmesidir. Bu durumda ancak pivot değerini en baştan veya en sondan seçtiğimiz dizilerin sıralı olması durumunda meydana gelebilir.böyle bir durumda karmaşıklık değeri O (n^2) olur.

ARAMA ALGORİTMALARI

Arama işlemi bir anahtar sözcüğe bağlı olarak elimizde var olan veri kümesi içerisinde anahtar sözcükle eşleşen veri olup olmadığına bakıp eğer anahtar sözcükle eşleşen veri varsa bulup o verinin diğer bilgilerine erişim olayıdır. Arama işlemi yapılırken elimizdeki verilerin sıralı olup olmadığını kontrol etmemiz gerekir. Aynı şekilde bu bilgilerin nasıl tutulduğu ve saklamak için hangi ver yapısını kullanılmış olduğu da oldukça önemelidir. Bu bilgileri göz önünde bulundurarak hangi arama algoritmasının ne tür veri grupları için daha uygun ve verimli olacağı hakkında yorumlar yapılabilir. Şimdi bazı arama algoritmalarının çalışma şekil ve prensipleri açıklayalım.

Doğrusal Arama (Linear Search): En basit arama algoritmasıdır. Bir dizi veya liste ilk elemanından başlanarak son elemana kadar ilerlenir. İlerlerken elemanlar ile aranan sözcük karşılaştırılır. Basit bir yaklaşım olmakla birlikte etkili ve verimli bir arama lagoritması değildir. Elemanlar sıralı ise daha verimli olabilir. Ancak eleman sayısı fazla ise verimsiz bir algoritmadır. En kötü durumda aranan eleman dizinin en sonunda yada hiç olmayabilir. Bu durumda dizi boyutunca karşılaştırma işlemi yapılmak durumda kalınır. Eleman sayısı az ise bu sorun olmayabilir ancak fazla ise zaman kaybına neden olur dolayısıyla eleman sayısının fazla olduğu durumda oldukça verimsiz bir algoritmadır. Eğer liste veya dizimizin boyutu az ve sıralı değil ise tercih edilebilir.

Genel olarak **Doğrusal Aramanın (Linear Search)** çalışma prensibi: Bir anahtar sözcük belirlenir. Daha sonra dizinin en başından başlanarak verilerle anahtar sözcük karşılaştırılır. Anahtar sözcükle uyuşan veri bulunana kadar ilerlenir. Eğer dizi içirisinde anahtar sözcükle eşleşen veri yoksa -1 değeri dödürülür. Anahtar sözcükle eşleşen veri bulunursa da o verinin indisini adresini gönderir.

ÖRNEK:[6, 3, 5, 11, 2, 7, 20, 9, 12, 15] dizisinde **Doğrusal Arama (Linear Search)** algoritmasını kullanarak 7 değerini dizide arayınız?

0	1	2	3	4	5	6	7	8	9
6	3	5	11	2	7	20	9	12	15
6	3	5	11	2	7	20	9	12	15
6	3	5	11	2	7	20	9	12	15
6	3	5	11	2	7	20	9	12	15
6	3	5	11	2	7	20	9	12	15
6	3	5	11	2	7	20	9	12	15

- 1.Adım 6 = 7 (yanlış)
- 2.Adım 3 = 7 (yanlış)
- 3. Adım 5 = 7 (yanlış)
- 4.Adım 11 = 7 (yanlış)
- 5.Adım 2 = 7 (yanlış)
- 6.Adım 7 = 7 (doğru)

Dizinin başından başlayarak anahtar veri ile uyuşan elemanı bulana kadar karşılaştırma işlemini yaparak ilerlenir. 5. İndekste bulunan altıncı elemanın anahtar veri ile uyuştuğunu görülür. Dizinin 5. İndeksinde tutuluyor diyerek 5 değerini geri döndürülür.

Arama algoritmamızın zaman karmaşıklığına değinecek olursak:

- En iyi durumda aranan eleman 6 olsun
  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]
  Bu durumda aranan eleman dizinin ilk elemanıdır. İlk karşılaştırma işleminde sonuca ulaşmış oluruz. Bu durumun zaman karmaşıklığı da O(1) olur.

- En kötü durumda
  [6, 3, 5, 11, 2, 7, 20, 9, 12, 15]
  Aranan eleman dizinin son elemanı olan 15 ya da listede olmayan 17 elemanı olsun. Bu iki durumda dizideki bütün elemanlar karşılaştırma işlemine girmek durumunda kalır. Bu durumda da zaman karmaşıklığı O(n) olur.

İkili Arama (Binary Search): Sıralı durumda olan bir veri yapısında arama işlemi yapmak için kullanılır. Genel olarak bakıldığında böl-fethet(divide and conquere)mantığı hakimdir. Eğer veri sayımız fazla ise oldukça kullanışlı ve verimli bir arama algoritmasıdır. Arama işlemi yapmak istediğimiz dizi sıralı değilse öncelikle uygun bir sıralama algoritması kullanarak verileri sıralamak gerekir. Ancak sıralama işleminden sonra arama işlemi gerçekleştirilebilir.

Genel olarak **İkili Aramanın (Binary Search)** çalışma prensibi: İlk olarak elimizdeki sıralı dizinin en ortasındaki elemanı alırız. Eğer dizimiz tek eleman sayısına sahip ise tam ortadaki eleman, çift eleman sayısına sahip ise de ortadaki elemanlardan istenilen seçilir. Seçilen eleman ile aranan eleman karşılaştırılır. Aranan elemen ile seçilen eleman aynı ise seçilen elemanın indisi döndürülür. Şayet aynı değil ise büyüklük küçüklük durumu kontrol edilir. Seçtiğimiz sayı aranan sayıdan küçük ise aradığımız değer seçtiğimiz değerden önceki değerler arasında olmaz. Ancak seçtiğimiz sayı arana sayıdan büyük ise aranan sayı seçtiğimiz değerden sonraki değerler arasında olamaz. Bu koşul kontrol edilerek seçilen sayının bir tarafı devre dışı kalır. Aynı işlemler geriye kalan tarafın ortasındaki değer seçilerek devam ettirilir. Ta ki sonuca ulaşana kadar. Bu işlemler bittiğinde hala bir sonuca ulaşılmamış ise -1 gibi bir değer döner.

ÖRNEK 1 : [2, 3, 5, 6, 7, 9, 11, 12, 15, 20]dizisinde **İkili Arama (Binary Search):** algoritmasını kullanarak 7 değerini dizide arayınız?

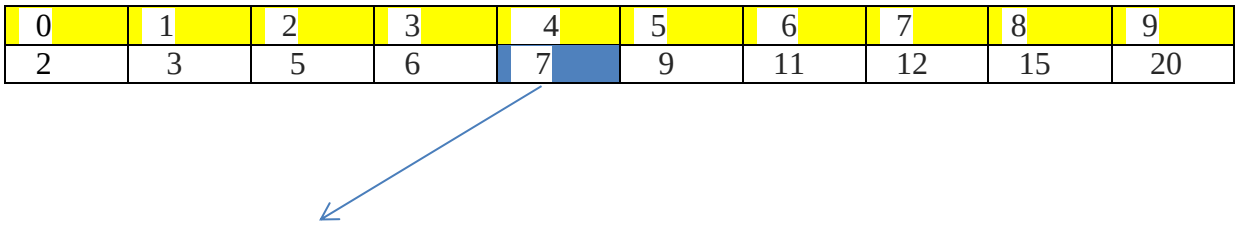
0	1	2	3	4	5	6	7	8	9
2	3	5	6	7	9	11	12	15	20

Seçilen ortanca değer ile istenilen eleman karşılaştırılır.

- 1.Adım
  - 7 = 7 ? (doğru)

İlk adımda istenilen değer bulundu. Seçilen değerin indeksi olan 4 döndürülür.

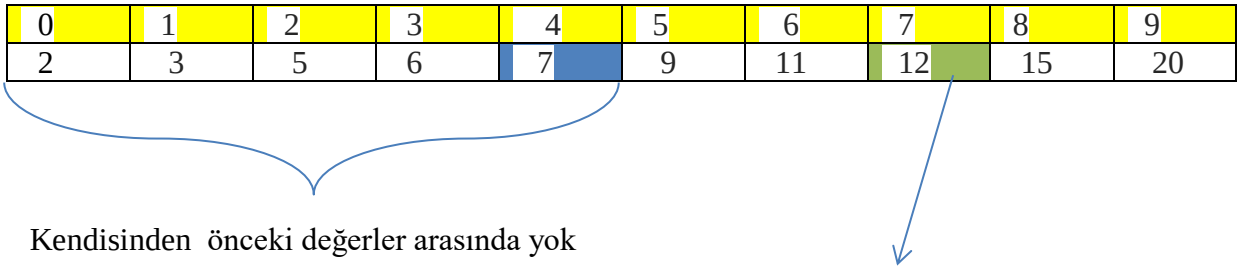
ÖRNEK 2 : [2, 3, 5, 6, 7, 9, 11, 12, 15, 20]dizisinde **İkili Arama (Binary Search):** algoritmasını kullanarak 11 değerini dizide arayınız?



Seçilen ortanca değer ile istenilen eleman karşılaştırılır

- 1. Adım
  - 7 = 11 ? (yanlış)
  - 7 > 11 ? (yanlış)
  - 7 < 11 ? (doğru)

Görüldüğü üzere aranan eleman seçtiğimiz elemandan büyük değere sahiptir. Yani aranan elemen seçtiğimiz değerden önceki elemanlar arasında olamaz .

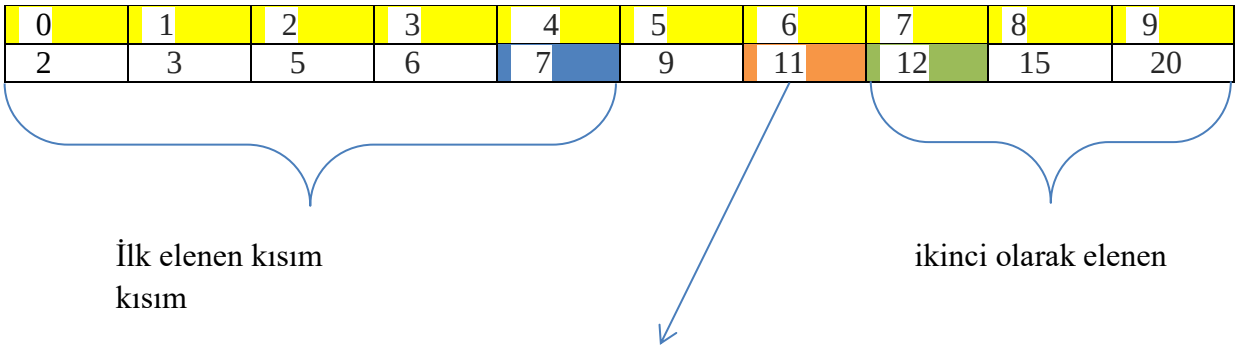


Kendisinden önceki değerler arasında yok

Kendinden sonraki elmanlar arasında yeni bir ortanca değer seçildi.

- 2. Adım
  - 12 = 11 ? (yanlış)
  - 12 < 11 (yanlış)
  - 12 > 11 (doğru)

Seçilen yeni ortanca değer aranan elemandan büyük olduğu için aranan eleman seçilen elemandan sonraki veriler arasında olamaz.



- 3.Adım
  - 11 = 11 ? (doğru)

İstenilen eleman bulundu. İstenilen elemanın bulunduğu indeks değeri döndürülür.

Arama algoritmamızın zaman karmaşıklığına değinecek olursak:

- En iyi durumda aranan eleman 7 ve ya 9 olsun
  [2, 3, 5, 6, 7, 9, 11, 12, 15, 20]
  Bu durumda aranan eleman dizinin ortasındaki elemana eşittir .İlk arama işleminde sonuca ulaşmış oluruz. Bu durumun zaman karmaşıklığı da O(1) olur.

- En kötü durumda aranan eleman dizimizde bulunmaz

  [2, 3, 5, 6, 7, 9, 11, 12, 15, 20]

Bu durumda arana elemen 13 olduğunu düşünebiliriz. Her seferinde dizimiz aranan sayıya göre ortadan ikiye bölünmektedir. Bu durumda aranan eleman 2 tabanında log(n) adımda bulunması beklenir. Aranan elemanın bu dizide olmaması durumunda da bu durum değişmez. Zaman karmaşıklığı O (log (n)) olur.