# Veri Yapıları ve Algoritmalar

DR. ÖGR. ÜYESİ MEHMET AKİF BÜLBÜL

2023-2024 GÜZ YARIYILI

YIĞIN (STACK)

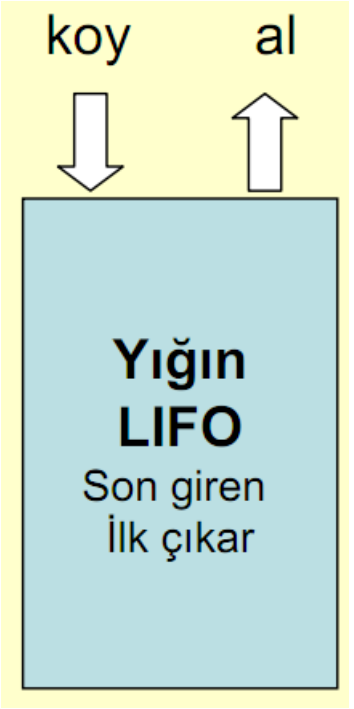
# Yığın (Stack)

Son giren ilk çıkar (Last In First Out-LIFO) mantığıyla çalışır.

Eleman ekleme çıkarmaların en üstten (top) yapıldığı veri yapısına yığın (stack) adı verilir.

Bir eleman ekleneceğinde yığının en üstüne konulur. Bir eleman çıkarılacağı zaman yığının en üstündeki eleman çıkarılır.

Bu eleman da yığındaki elemanlar içindeki en son eklenen elemandır. Bu nedenle yığınlara LIFO (Last In First Out Son giren ilk çıkar) listesi de denilir.



# Yığın (Stack)

Yığın yapısını gerçekleştirmek için 2 yol vardır.

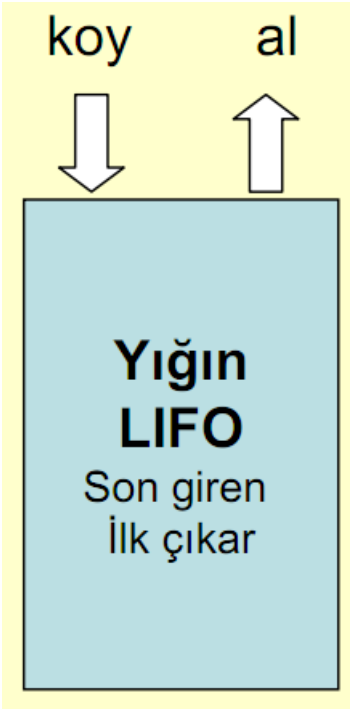
- Dizi kullanmak
- Bağlantılı liste kullanmak

# Yığın (Stack)

empty stack: Boş yığıt

push (koy):Yığıta eleman ekleme

pop (al):Yığıttan eleman çıkarma



# Yığın (Stack)

Ana yığın işlemleri:

push(nesne): yeni bir nesne ekler

Girdi: Nesne Çıktı: Yok

pop(): en son eklenen nesneyi çıkarıp geri döndürür.

Girdi: Yok Çıktı: Nesne

Yardımcı yığın işlemleri:

top(): en son eklenen nesneyi çıkarmadan geri döndürür.

Girdi: Yok Çıktı: Nesne

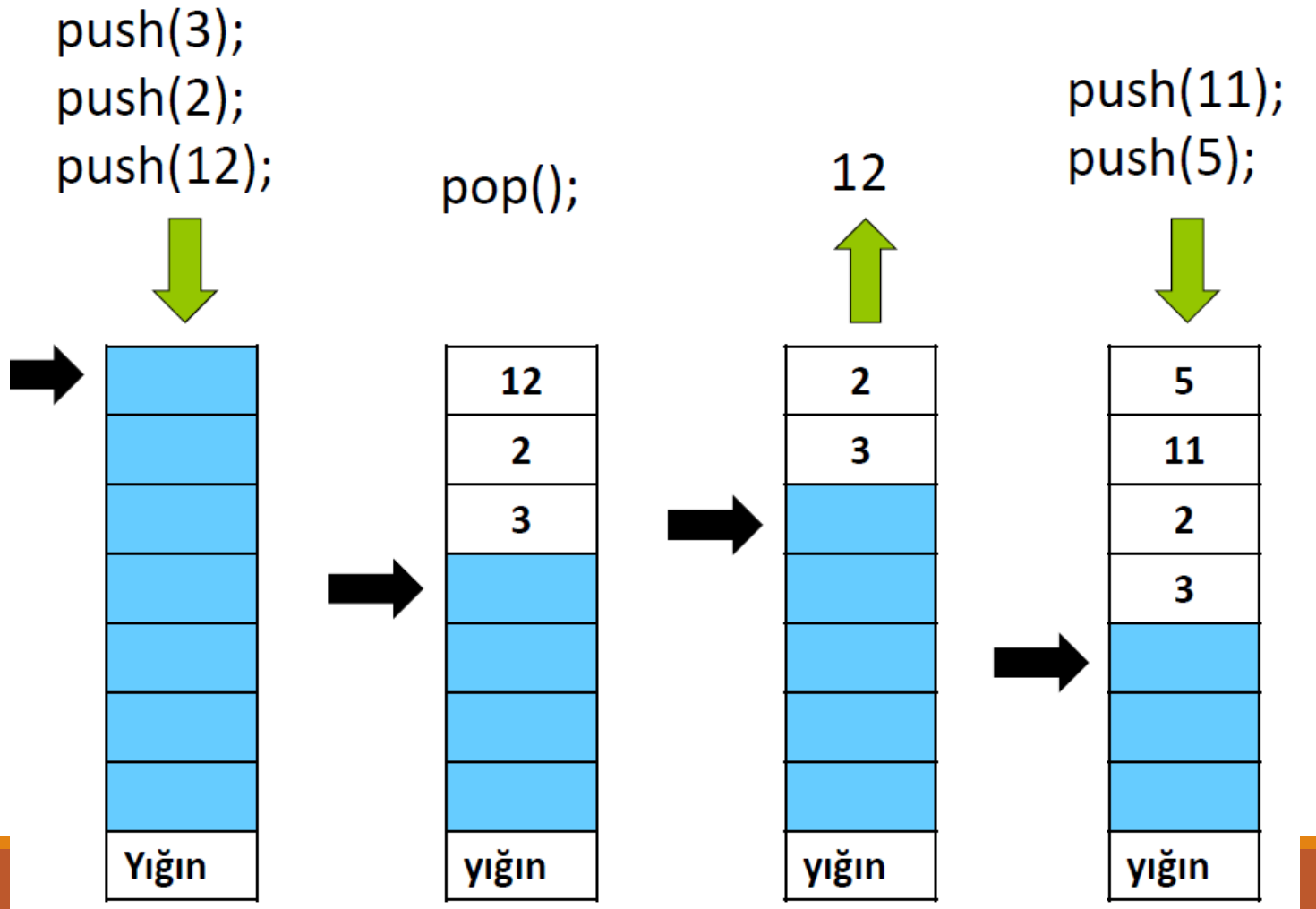
size(): depolanan nesne sayısını geri döndürür.

Girdi: Yok Çıktı: Tamsayı

isEmpty(): yığında nesne bulunup bulunmadığı bilgisi geri döner.

Girdi: Yok Çıktı: Boolean

# Yığın (Stack)


push(3);
push(2);
push(12);
pop();
12
push(11);
push(5);
12
2
3
Yığın
yığın
2
3
yığın
5
11
2
3
yığın

# Yığın (Stack)

**NERELERDE KULLANABİLİRİZ ?**

# Yığın (Stack)

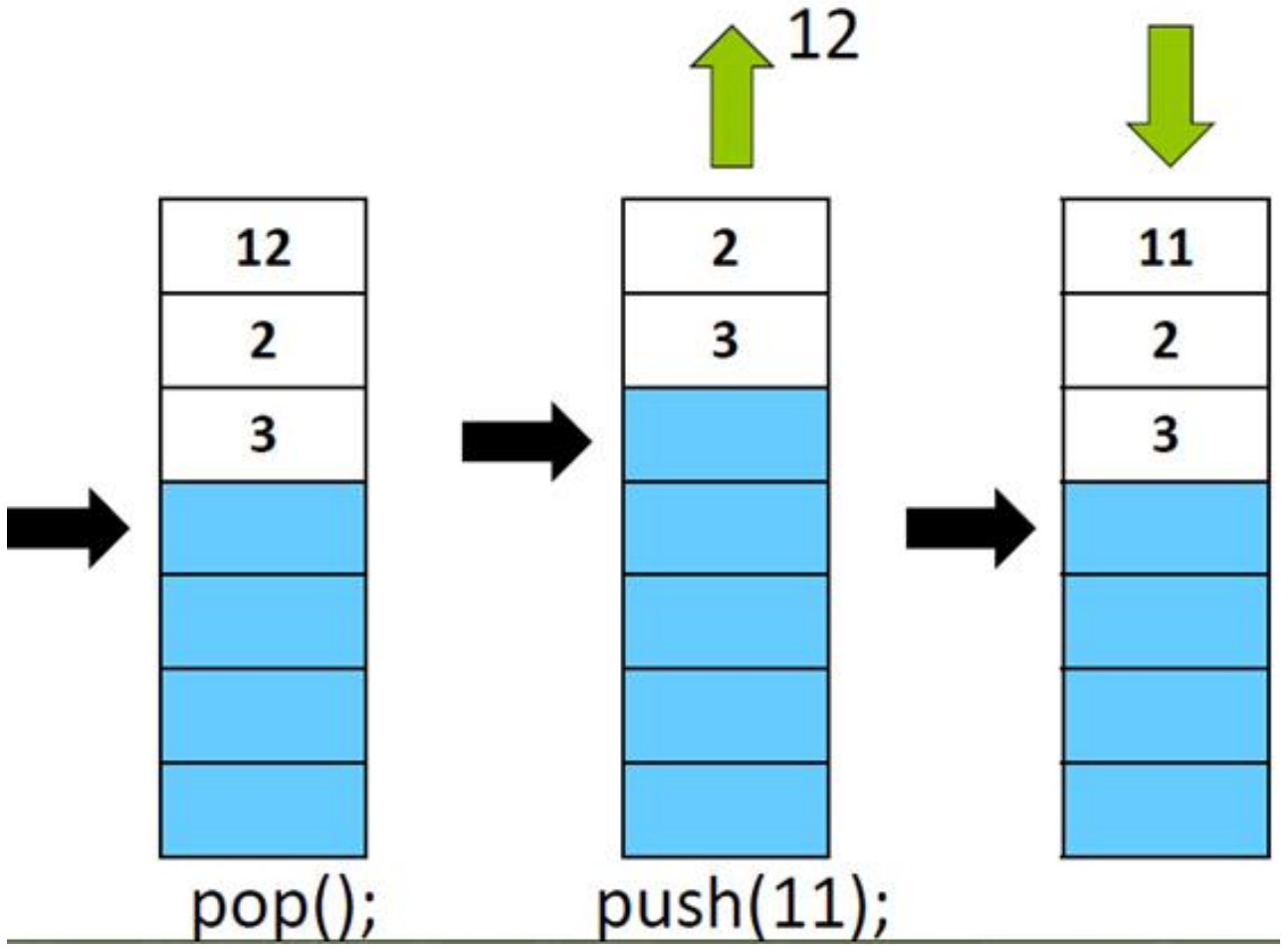
- Web browser'lardaki Back butonu (önceki sayfaya) stack kullanır. Buradada LIFO yapısı kullanılır.
- Matematiksel işlemlerdeki operatörler (+,*,/,- gibi) ve operandlar için stack kullanılabilir.
- Yazım kontrolündeki parantezlerin kontrolünde stack kullanılabilir.

# Yığın (Stack)

| İşlem | Yığıt (tepe) | Çıktı |
|---|---|---|
| push(“M”); | M | |
| push(“A”); | MA | |
| push(“L”); | MAL | |
| push(“T”); | MALT | |
| pop(); | MAL | T |
| push(“E”); | MALE | T |
| pop(); | MAL | TE |
| push(“P”); | MALP | TE |
| pop(); | MAL | TEP |
| push(“E”); | MALE | TEP |
| pop(); | MAL | TEPE |

# Dizi Tabanlı Yığın

Bir yığını gerçeklemenin en gerçeklemenin en kolay yolu dizi kullanmaktır. Yığın yapısı dizi üzerinde en fazla N tane eleman tutacak şekilde yapılabilir.



# Dizi Tabanlı Yığın

Nesneleri soldan sağa doğru ekleriz. Bir değişken en üstteki nesnenin index bilgisini izler. Eleman çıkarılırken bu index değeri alınır.


```
Algorithm size()
    return t + 1

Algorithm pop()
    if isEmpty() then
        throw EmptyStackException
    else
        t ← t − 1
        return S[t + 1]
```

# Dizi Tabanlı Yığın

Yığın nesnelerinin saklandığı dizi dolabilir. Bu durumda push işlemi aşağıdaki mesajı verir.

**FullStackException (DoluYığınİstinası)**


```
Algorithm push(o)
    if t = S.length − 1 then
        throw FullStackException
    else
        t ← t + 1
        S[t] ← o
```

# Dizi Tabanlı Yığında Başarım

Başarım

- n yığındaki nesne sayısı olsun
- Kullanılan alan O(n)
- Her bir işlem O(1) zamanda gerçekle

# Dizi Tabanlı Yığında Sınırlamalar

Sınırlamalar

- Yığının en üst sayısı önceden tanımlanmalıdır ve değiştirilemez.
- Dolu bir yığına yeni bir nesne eklemeye çalışmak istisnai durumlara sebep olabilir.

# Büyüyebilir Dizi Tabanlı Yığın Yaklaşımı

**push işlemi esnasında dizi dolu ise ne yapabiliriz ????!!!!*****-----☹**

# Büyüyebilir Dizi Tabanlı Yığın Yaklaşımı

**Yeni dizi ne kadar büyüklükte olmalı?**

# Büyüyebilir Dizi Tabanlı Yığın Yaklaşımı

**Artımlı strateji:** yığın büyüklüğü sabit bir c değeri kadar arttırılır.

**İkiye katlama stratejisi:** önceki dizi boyutu iki kat arttırılır


**Algorithm** *push*$(o)$
  **if** $t = S.length - 1$ **then**
    $A \leftarrow$ new array of size …
    **for** $i \leftarrow 0$ **to** $t$ **do**
      $A[i] \leftarrow S[i]$
    $S \leftarrow A$
  $t \leftarrow t + 1$
  $S[t] \leftarrow o$

# Yığın ve Operasyonları

```
public class Yigin {

  int kapasite=100;  // maksimum eleman sayısı
  int S[]; //Yığın elemanları – pozitif tam sayı
  int p;  // eleman sayısı

  public Yigin(){   // yapıcı yordam
    s[] = new int[kapasite];
    p = 0;
  }

  int koy(int item);
  int al();
  int ust();
  boolean bosmu();
  boolean dolumu();
}
```

# Yığın ve Operasyonları

```
// yığın boşsa true döndür
public boolean bosmu() {
  if (p < 1) return true;
  else return false;
} //bitti-bosmu


// Yığın doluysa true döndür
public boolean dolumu(){
  if (p == kapasite-1) return true;
  else return false;
} // bitti-dolumu
```

# Yığın ve Operasyonları

```
// Yığının üstüne yine bir eleman koy
// Başarılı ise 0 başarısız ise -1 döndürür.
int koy(int yeni){

  if (dolumu()){
    // Yığın dolu. Yeni eleman eklenemez.
    return -1;
  }

  S[p] = yeni;
  p++;

  return 0;
} /bitti-koy
```

# Yığın ve Operasyonları

```
// Yığının en üstündeki sayıyı döndürür
// Yığın boşsa, -1 döndürür
public int ust(){
   if (bosmu()){
      // Yığın başsa hata dönder
      System.out.println("Stack underflow");
      return -1;
   }

   return S[p-1];
}
```

# Yığın ve Operasyonları

```
// En üsteki elemanı dönder.
// Yığın boşsa -1 dönder.
public int al(){
  if (bosmu()){
    // Yığın boşsa hata dönder
    System.out.println("Stack underflow");
    return -1;
  }

  int id = p-1; // en üsteki elemanın yeri
  p--;          // elemanı sil

  return S[id];
}
```

# Yığın ve Operasyonları

```
public static void main(String[] args){
  Yigin y = new Yigin();

  if (y.bosmu())
    System.out.println("Yığın boş");

  y.koy(49);   y.koy(23);

  System.out.println("Yığının ilk elemanı: "+ y.al());

  y.koy(44);   y.koy(22);

  System.out.println("Yığının ilk elemanı: "+ y.al());
  System.out.println("Yığının ilk elemanı: "+ y.al());
  System.out.println("Yığının ilk elemanı: "+ y.ust());
  System.out.println("Yığının ilk elemanı: "+ y.al());.

  if (y.bosmu()) System.out.println("Yığın boş");
}
```