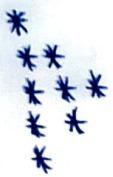
Recursive

Carpma işlemi

```
int carpma (int x, int y)
 { if( x==0 || y==0)
     { return 0;
     }
    else , x + carpma (x,(y-1)) ;
           return
 }
```

Yildiz işlemi



```
Void arton (int n)
{ if(n>0)
   { arton(n-1);
     for (int i=0; i<n ; i++)
      { Printf ("*");
      }
      Printf ("\n");
   }
}
Void yildiz(int baslama, int bitis)
{ arton (baslama);
  aztalon (bitis);
}
```

```
Void aztalon (int n)
 { if( n>0)
    { for(inti=0; i<n; i++)
      { printf ("*");
      }
      Printf ("\n");
     aztalon desen (n-1);
    }
}
```

Binary

```
Void binary (int sayi)
 { if (sayi<=1)
    { Printf ("%d", sayi); }
   else
    {
      binary (sayi/2);
      Printf ("%d", sayi%2);
    }
}
```

String Tersi Esitmi

```
int stringTersKontrol ( char * ifade , int baslama , int bitis)
  { if( baslangna >= bitis) return 1;

    else if ( str[baslama] != str [bitis])
       { return 0; }

    else {
      return stringTersKontrol ( ifade , baslama+1 , bitis-1)
      }
```

Bölme işlemini çıkarma ile yapan recursive fonk

```
int bolme (int bolunen, int bolen)
 {
    if (bolen ==0)
     {
       return -1;
     }
    else if (bolunen < bolen)
      { return 0;
      }
    else
      return 1 + bolme (bolunen-bolen, bolen);

     }
```

Metnin tersini yatma

```
Void terscumle (char *metin, int boy)
 {
    if (boy ==0)
     { printf ("%c", metin [boy]);
     }
   else
     { printf ("%c", metin [boy]);
       terscumle (metin, boy-1);
     }
 }
```

Bubble Sort

Komşu iki elemanın büyüklük küçüklük sıralamasına göre yer değiştirir.

```
Void bubble (int dizi[], int n)
{   int i, j, gecici;
    for (i=0; i=n-1; i++)
    {
        for (j=n-1; j>i; j--)
        {
            if (dizi[j] < dizi[j-1])
            {
                gecici = dizi[j];
                dizi[j] = dizi[j-1];
                dizi[j-1] = gecici;
            }
        }
}
```

12	10	4	5
0	1	2	3

i=0	j=2	12	4	10	5
i=0	j=1	4	12	10	5
i=1	j=3	4	12	5	10
i=1	j=2	4	5	12	10
i=2	j=3	4	5	10	12

Selection Sort

İlk elemandan başlanıyor. İlk elemanın sağından kendinden daha küçük bir eleman varsa yer değiştirir.

```
Void Selection (int dizi[], int n)
{   int i, j, min, gecici;
    for (i=0; i<n-1; i++)
    {
        min = i;
        for (j=i+1; j<n; j++)
        {
            if (dizi[j] < dizi[min]
            {   min = j;
            }
        }
    gecici = dizi[min];
    dizi[min] = dizi[i];
    dizi[i] = gecici;
    }
}
```

i=0	j=1	12	4	5	10	min 0
i=1	j=3	4	12	5	10	min 1
		4	5	12	10	
		4	5	10	12	

Insertion Sort

2. elemandan baslanip en soldaki elemana kadar kontrol ediliyor.

```
Void insertion (int dizi[], int n)
{   int i, j, mevcut;
    for (i=1; i<n; i++)
    {
        mevcut = dizi[i];
        j=i;
        while (j>0 && mevcut < dizi[j-1])
        {
            dizi[j] = dizi[j-1];
            j--;
        }
        dizi[j] = mevcut;
    }
}
```

20 10 15 17

i=1	j=1	mev. 10	20	20	15	17	j=0
i=1	j≠0	mev 10	10	20	15	17	
i=2	j=2	mev. 15	10	20	20	17	j=1
i=2	j=1	mev 15	10	15	20	17	
i=3	j=3	mev 17	10	15	20	20	j=2
i=3	j=2	mev 17	10	15	17	20	

Shell Siralama

```
Void shell (int dizi[], int n)
{   int artim, i, j, mevcut;
    for (artim=n/2; artim>0; artim/=2)
    {
        for (i=artim; i<n; i++)
        {
            mevcut = dizi[i];
            j=i;
            while (j>=artim && mevcut < dizi[j-artim])
            {   dizi[j] = dizi[j-artim];
                j-=artim;
            }
            dizi[j] = mevcut;
        }
    }
}
```

8 3 5 7

art.2	i=2	mev. 5	j=2	8	3	8	7		j=0
"	"	"	j=0	5	3	8	7		
art 2	i=3	mev. 7	j=3	5	3	8	7		
art 1	i=1	mev. 3	j=1	5	5	8	7		j=0
"	"	"	j=0	3	5	8	7		
art 1	i=2	mev. 8	j=2	3	5	8	7		
art 1	i=3	mev. 7	j=3	3	5	8	8		j=2
"	"	"	j=2	3	5	7	8		

Quick Sort

Dizi pivot değerine göre iki alt diziye ayrılır. Pivottan küçükler sola pivottan büyük elemanlar sağa yerleşir. Sonra sol ve sağ diziden pivot seçilir ve sıralanır.

```
Void Quick (int dizi[], int alt, int ust)
{   if (ust > alt)
    {   int pivot_indisi = Partition(dizi, alt, ust);
        Quick (dizi, alt, pivot_indisi-1);
        Quick (dizi, pivot_indisi+1, ust);
    }
}
```

```
int Partition (int dizi[], int alt_indis, int ust_indis)
{   int pivot_degeri = dizi[alt_indis];
    int i = alt_indis+1;
    int j = ust_indis;
    int gecici;

    while (i<=j)
    {   if( dizi[i] < pivot_degeri)
        {   i++;
        }
        else if( dizi[j] > pivot_degeri)
        {   j--;
        }
        else
        {   gecici = dizi[i];
            dizi[i] = dizi[j];
            dizi[j] = gecici;
            i++; j--;
        }
    }
    gecici = dizi[alt_indis];
    dizi[alt_indis] = dizi[j];
    dizi[j] = gecici;

    return j;
}
```

Merge Sort

Dizi 2 alt diziye ayrılır bir eleman kalıncaya kadar rekürsif çağırma geriye doğru dönmeye başlar sıralı olacak şekilde birleştirilir.

```
Void merge (int diti[], int ilk , int son)
{
    if (ilk < son)
    {   int orta = (ilk+son)/2 ;
        merge (dizi , ilk , orta);
        merge ( dizi , orta+1 , son) ;
        mergeS( dizi , ilk , orta, son);
    }
}
```

```
Void mergeS (int dizi[], int ilk , int orta , int son)
{
    int tempArray[1000] ;
    int ilk1 = ilk ; int son1 = orta ;
    int ilk2 = orta+1 ; int son2 = son;
    int index = ilk1 ;

    for ( ; (ilk1 <= son1) && (ilk2 <= son2); index++)
    {
        if( Dizi[ilk1] < Dizi[ilk2]
        {
            tempArray [index] = Dizi [ilk1];
            ilk1 ++;
        }

        else
        {
            tempArray [index ] = Dizi [ilk2];
        } ilk2 ++;
    }
    for ( ; ilk1 <= son1 ; ilk1 ++, index ++)
            tempArray [index] = dizi [ilk1];

    for ( ; ilk2 <= son2 ; ilk2 ++, index++)
            tempArray [index] = dizi [ilk2]

    for (index = ilk ; index <= son ; index++)
    dizi [index] = tempArray [index] ;
}
```