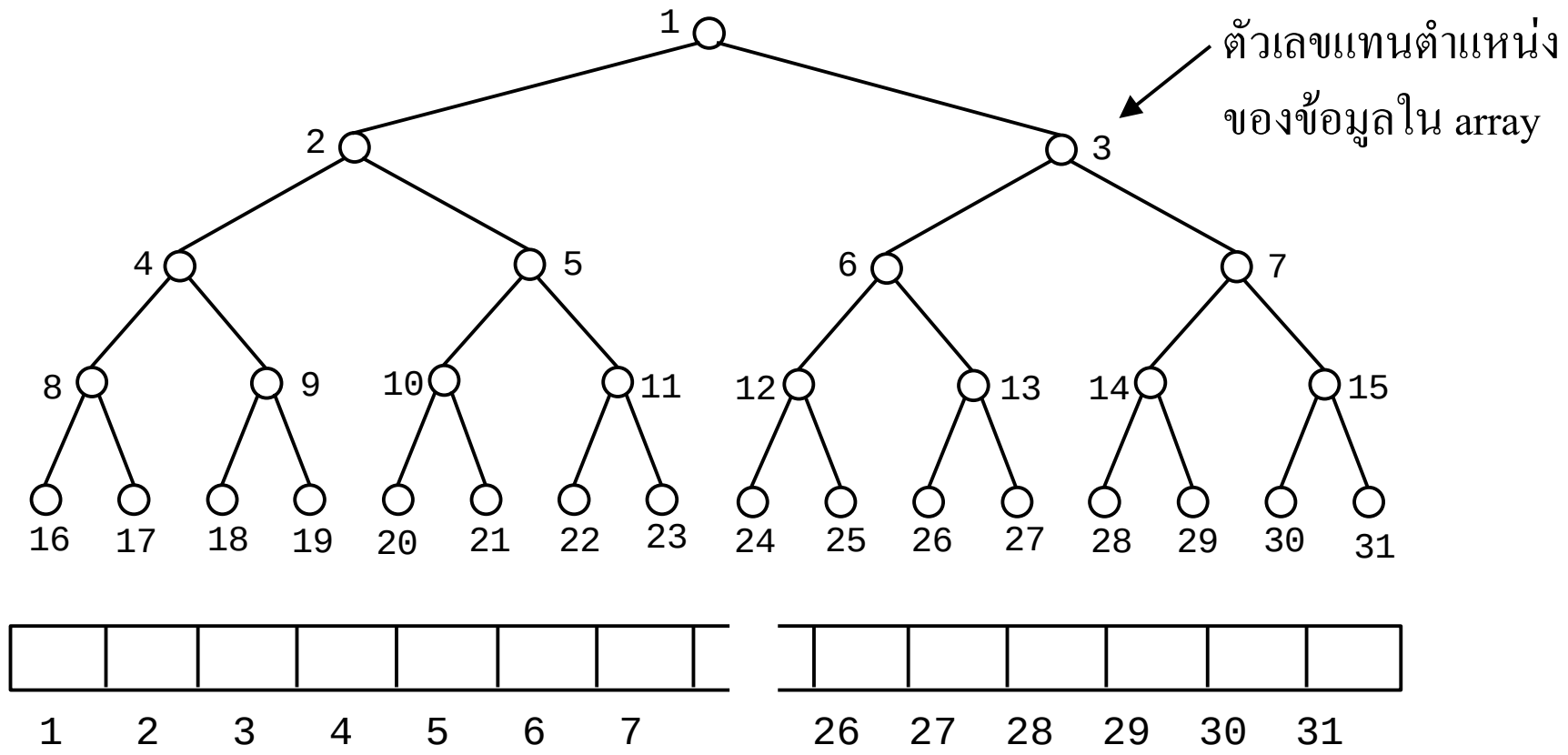
BINARY HEAPS

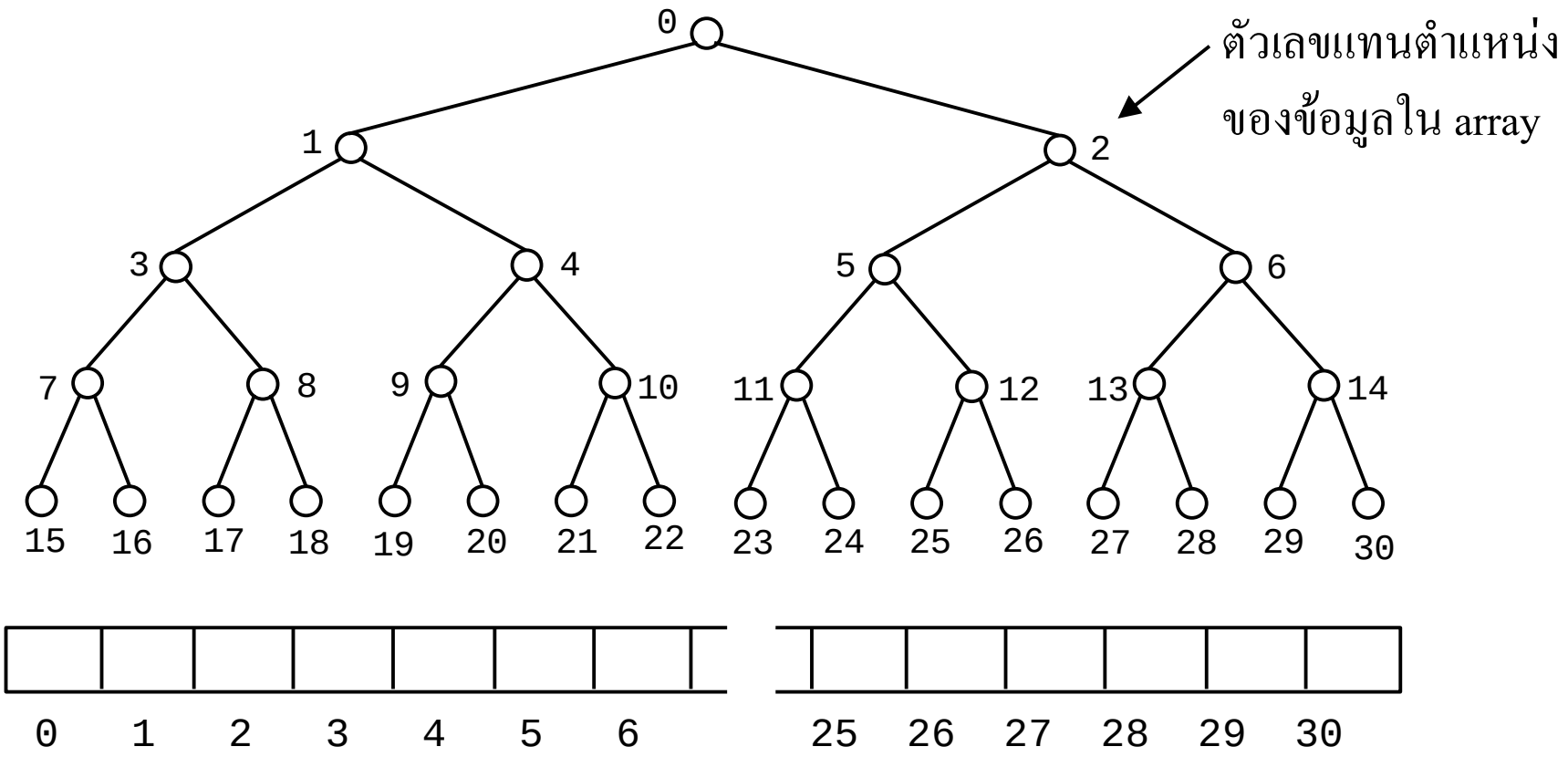
Binary Trees in Contiguous Storage



ลูกทางซ้ายของจุดที่ตำแหน่ง k จะอยู่ที่ตำแหน่งที่ $2k$

" ขวา " " " " $2k+1$

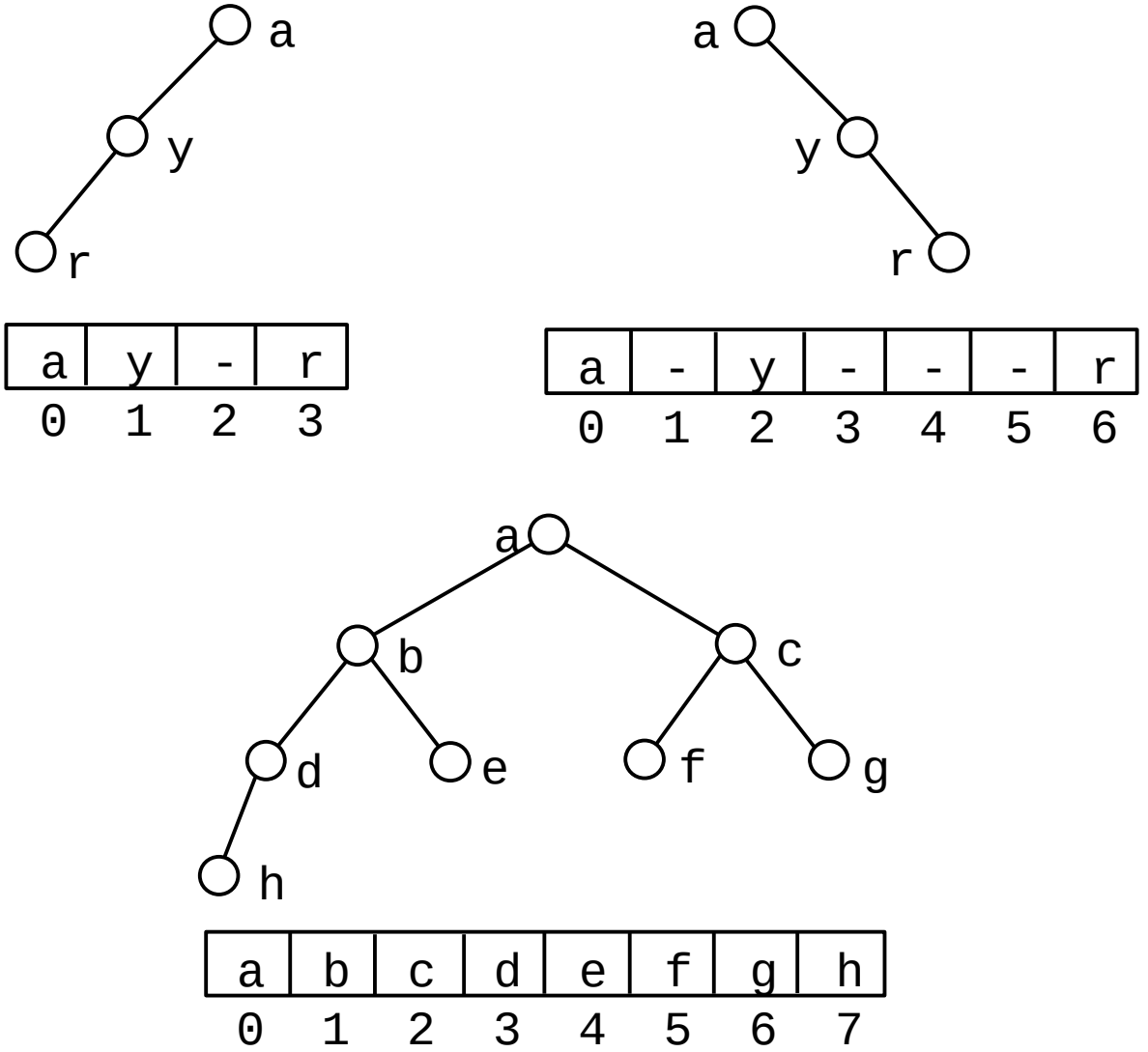
Binary Trees in Contiguous Storage



ลูกทางซ้ายของจุดที่ตำแหน่ง k จะอยู่ที่ตำแหน่งที่ $2k+1$

" ขวา " " " " $2k+2$

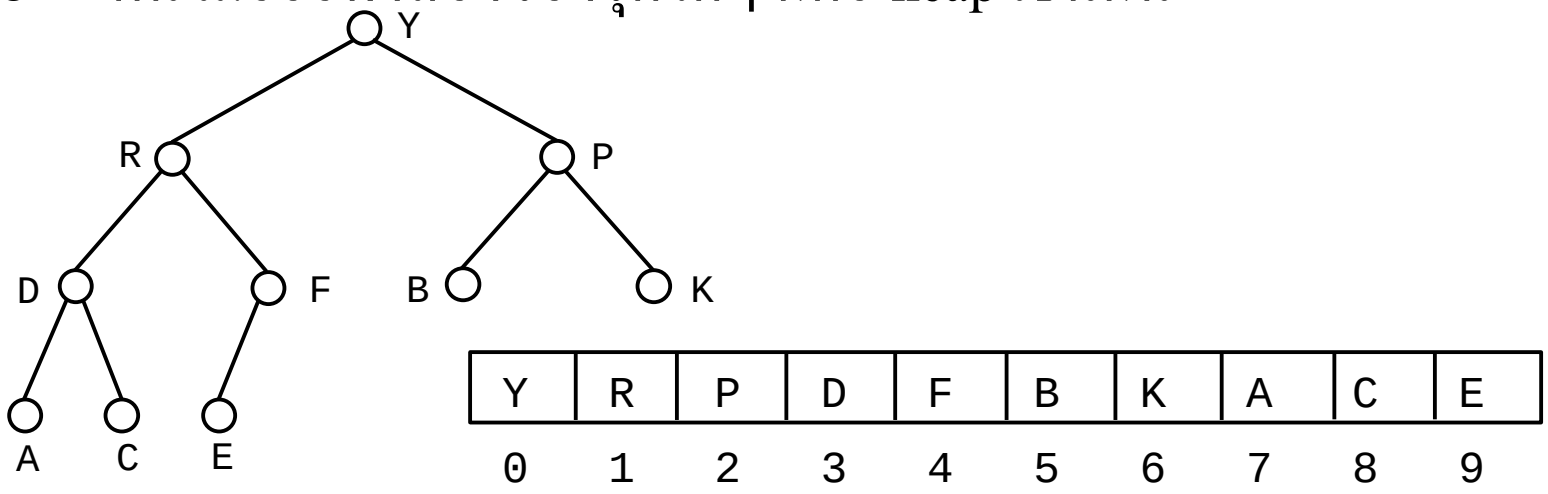
Binary Trees in Contiguous Storage



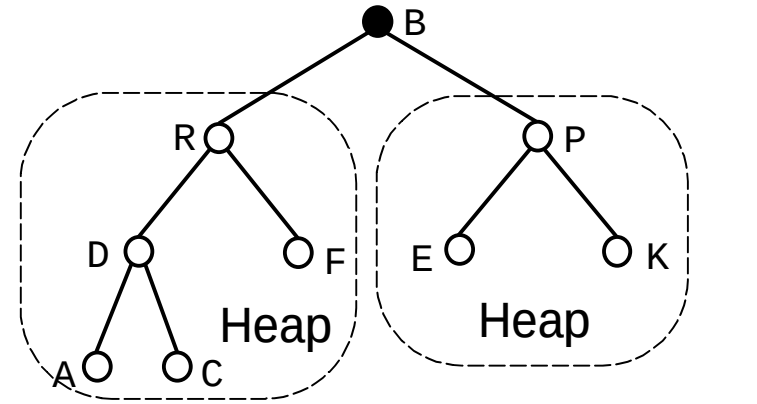
Heaps

Heap คือต้นไม้แบบทวิภาคที่มีคีย์ของข้อมูลกำกับจุด โดยที่

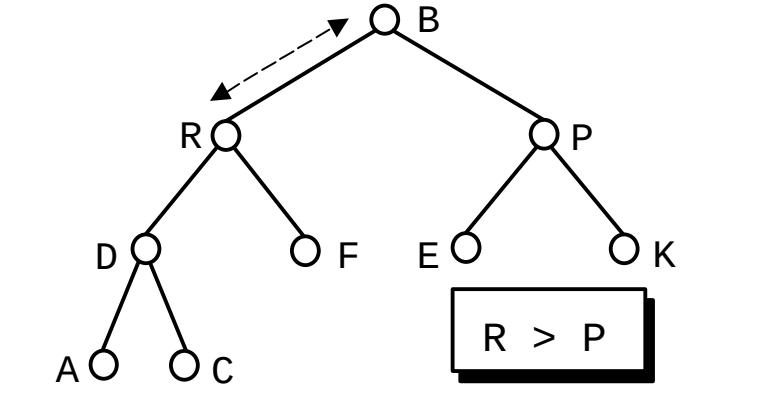
- ทุกๆใบในต้นไม้จะอยู่ในระดับเดียวกัน หรือระดับที่ติดกัน
- จุดในทุกๆระดับจะเต็ม ยกเว้นระดับล่างสุด
- ใบในระดับล่างสุด จะอยู่ชิดทางซ้าย
- คีย์ของข้อมูลที่จุดระดับพ่อ/แม่ จะมีค่ามากกว่าคีย์ของลูกหลาน
- ต้นไม้ย่อยทั้งสองของจุดใดๆ ก็คือ heap เช่นกัน



Heapify

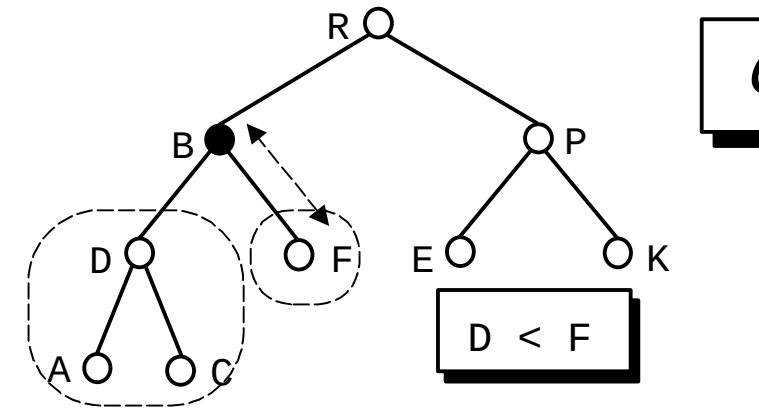


B	R	P	D	F	E	K	A	C
0	1	2	3	4	5	6	7	8

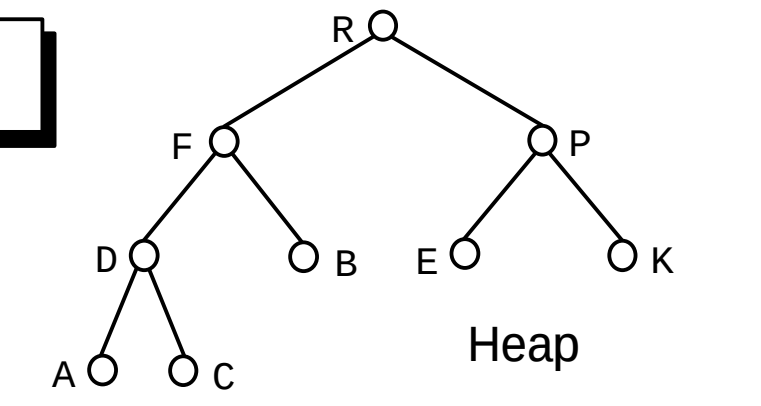


B	R	P	D	F	E	K	A	C
0	1	2	3	4	5	6	7	8

$O(\log n)$



R	B	P	D	F	E	K	A	C
0	1	2	3	4	5	6	7	8

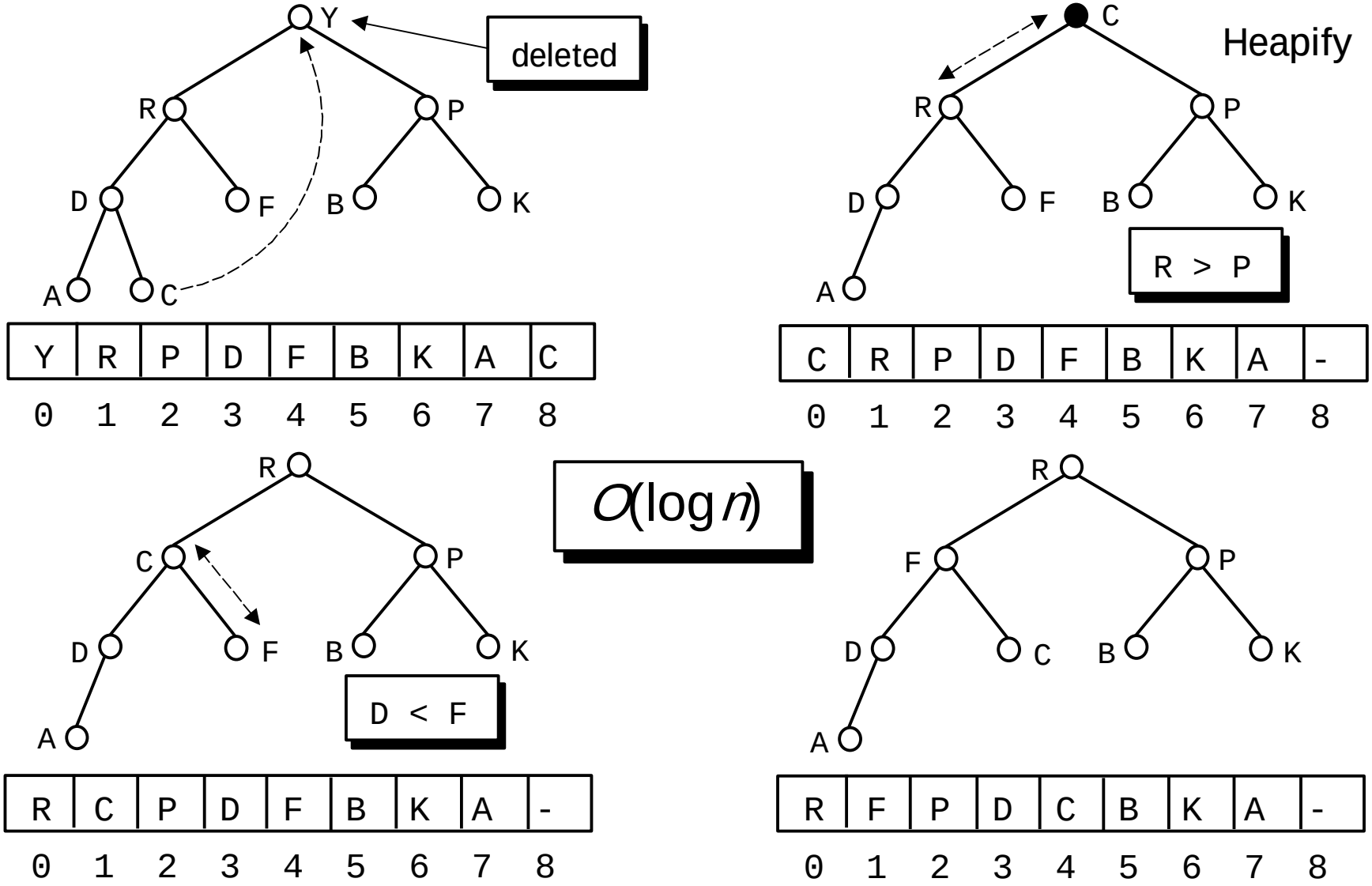


R	F	P	D	B	E	K	A	C
0	1	2	3	4	5	6	7	8

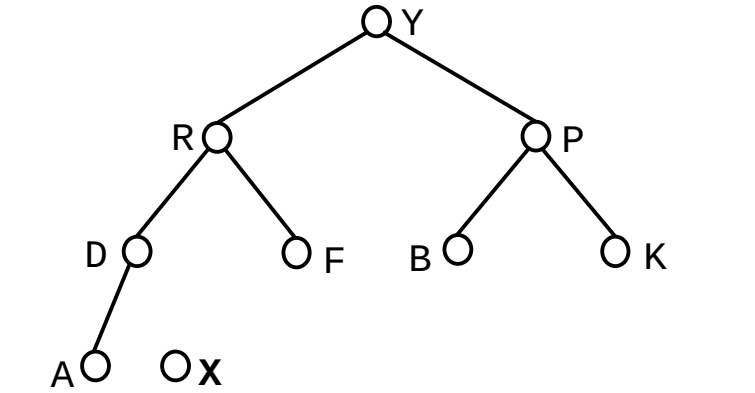
Heapify

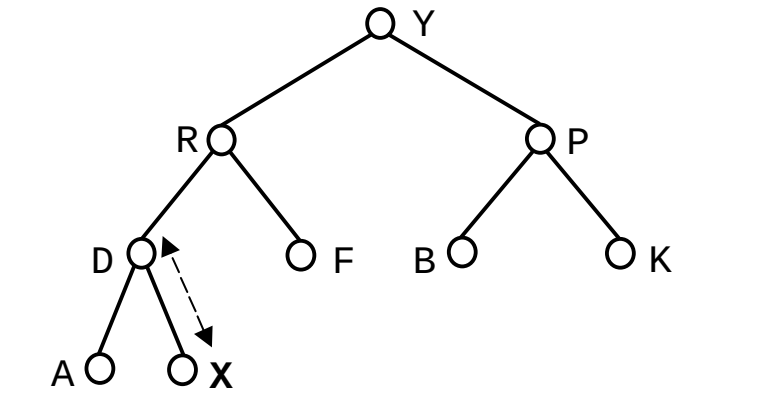
```
void  Heapify( ListType *pHeap, int i )
{
  int    largest;

  left  = LeftHeap( i );
  right = RightHeap( i );
  largest = i;
  if ( left  <= pHeap->count-1 ) &&
       GT( pHeap->entry[left].key, pHeap->entry[largest].key ) )
    largest = left;
  if ( right <= pHeap->count-1 ) &&
       GT( pHeap->entry[right].key, pHeap->entry[largest].key ) )
    largest = right;
  if ( largest != i ) {
    Swap( pHeap, i, largest );
    Heapify( pHeap, largest );
  }
}
```

Heaps : Deletion

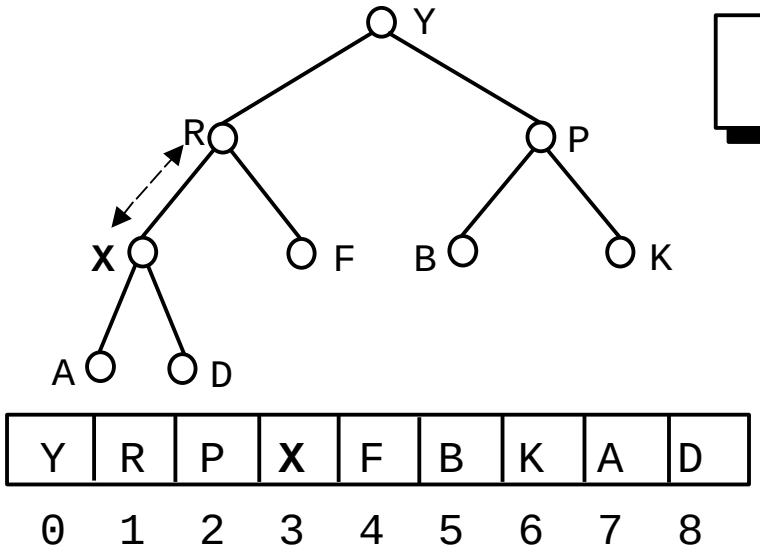


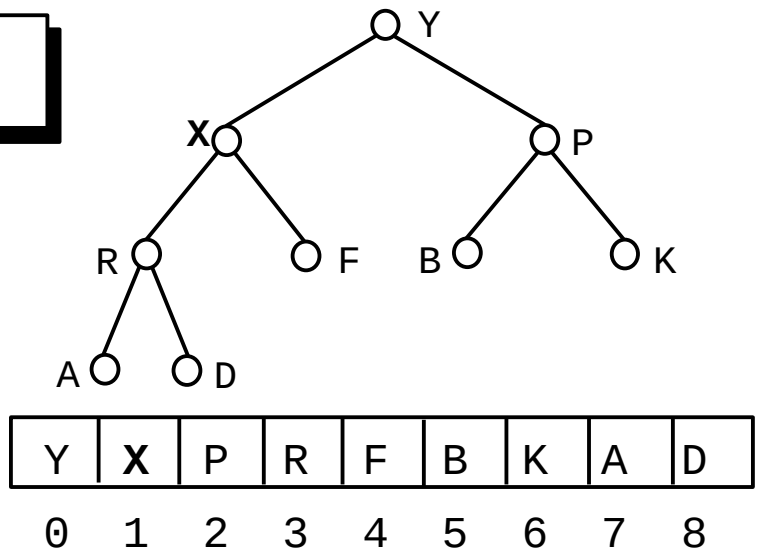
Heaps : Insertion



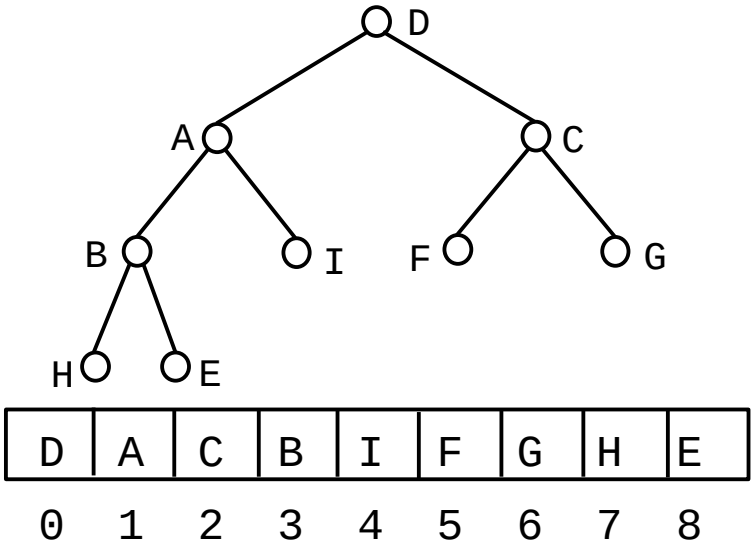


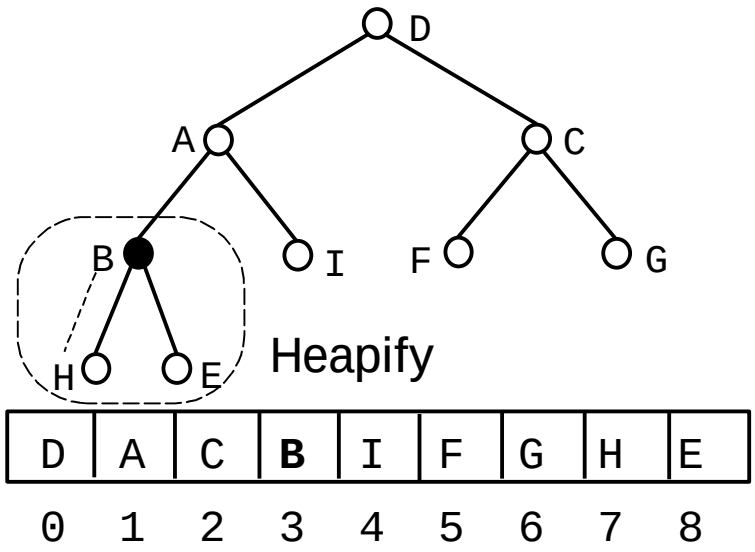
$O(\log n)$

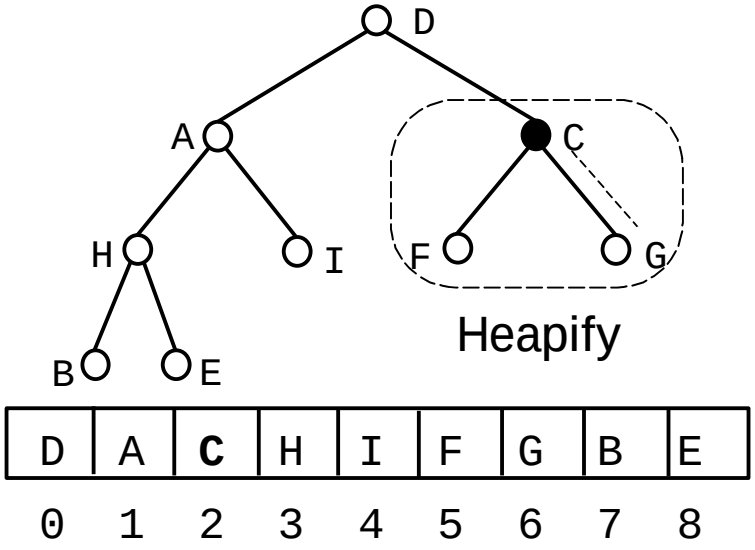


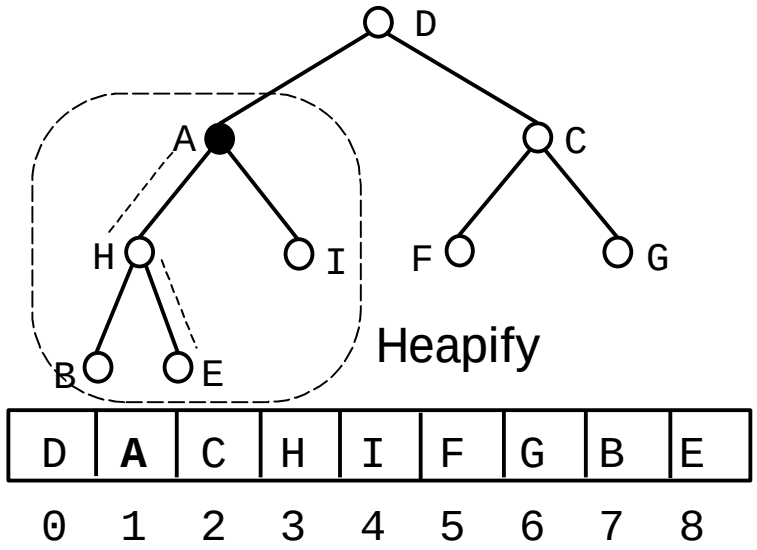


Building a Heap

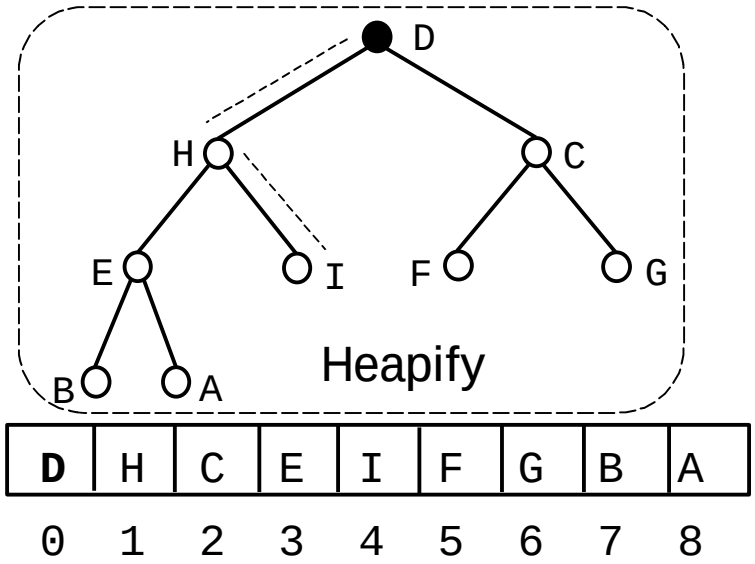


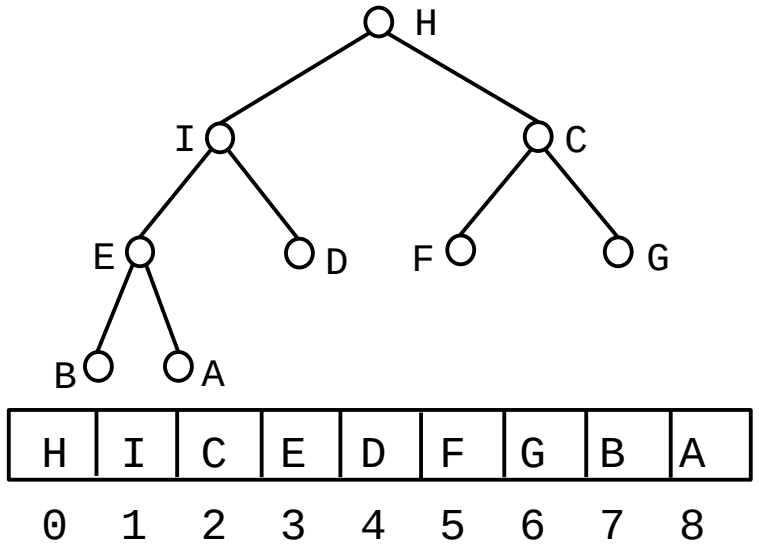






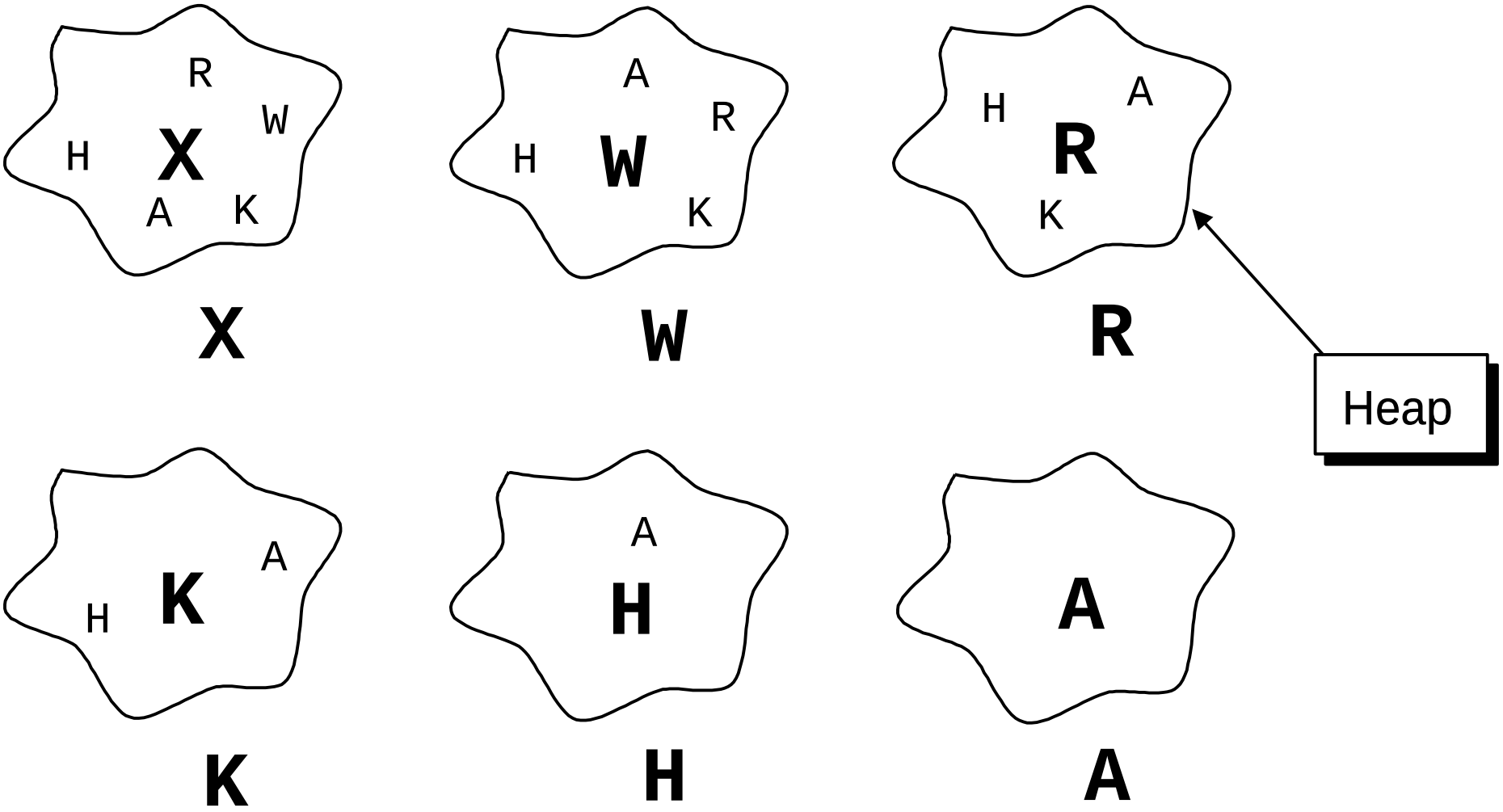
Building a Heap



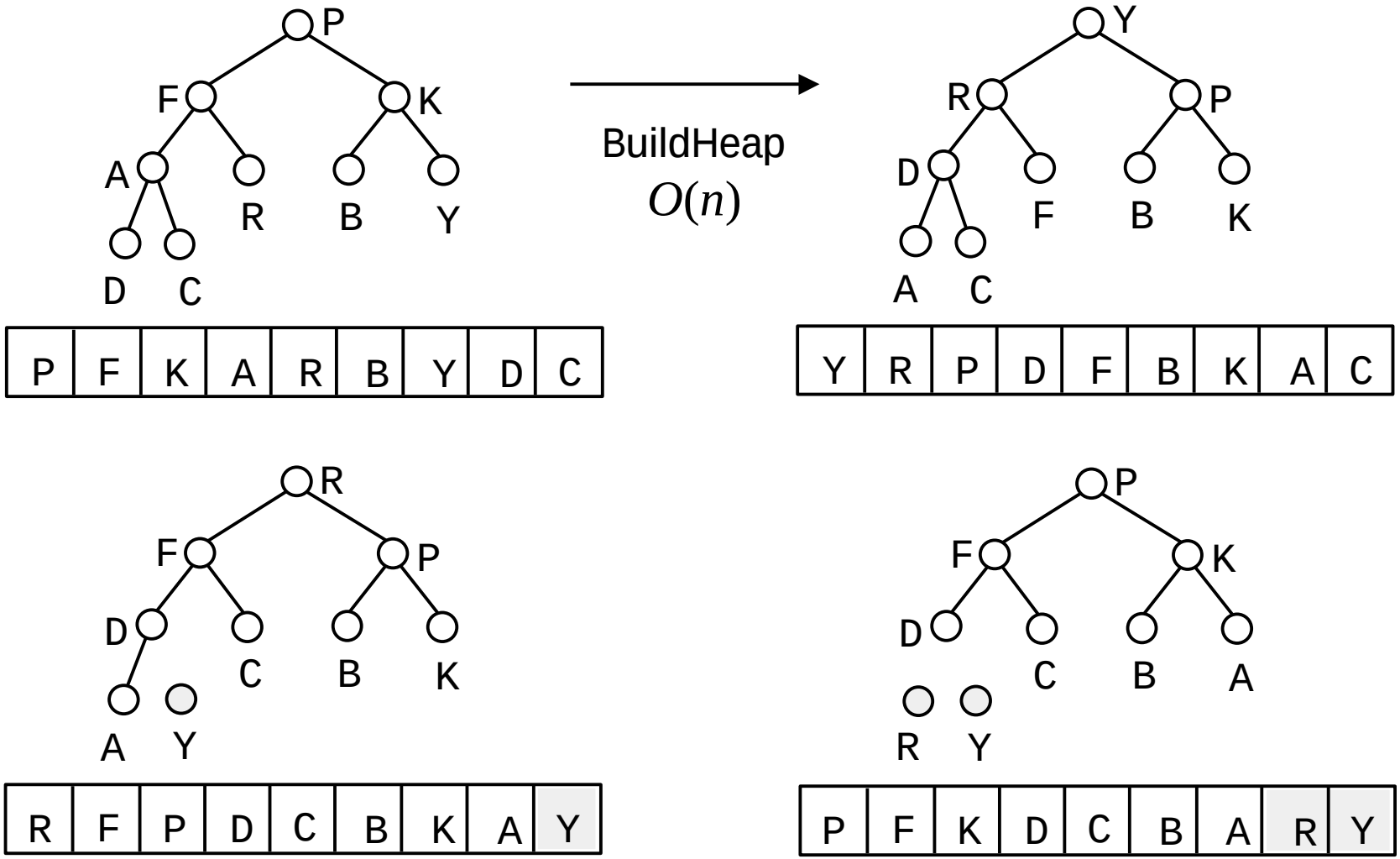


$$O(\log n)$$

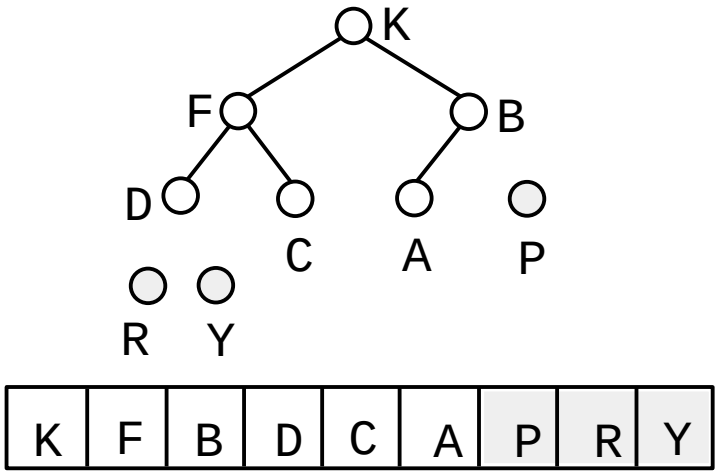
Heapsort

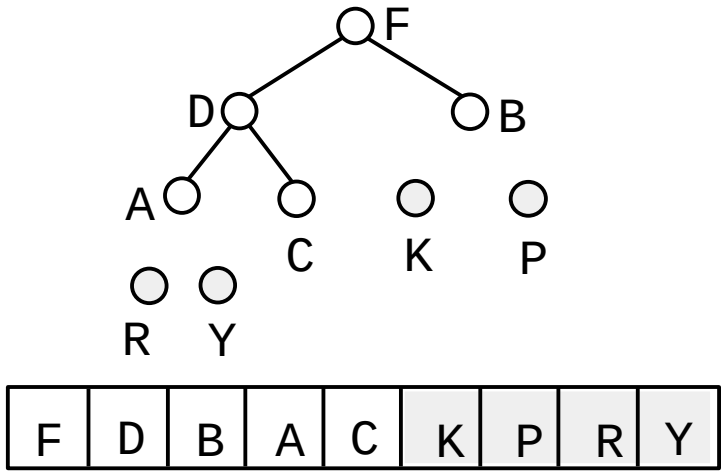


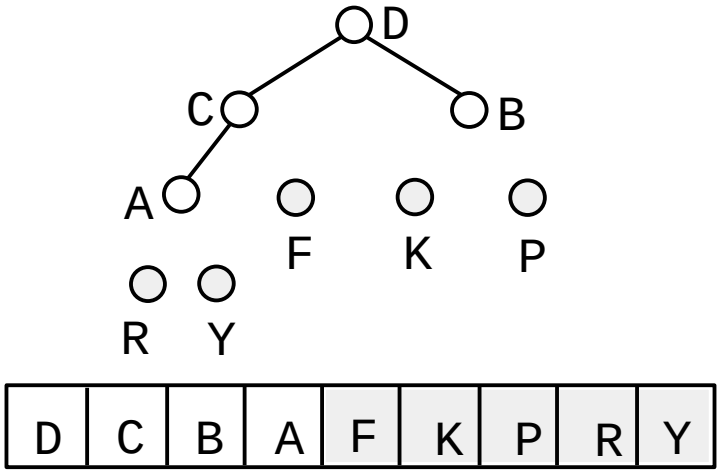
Heapsort

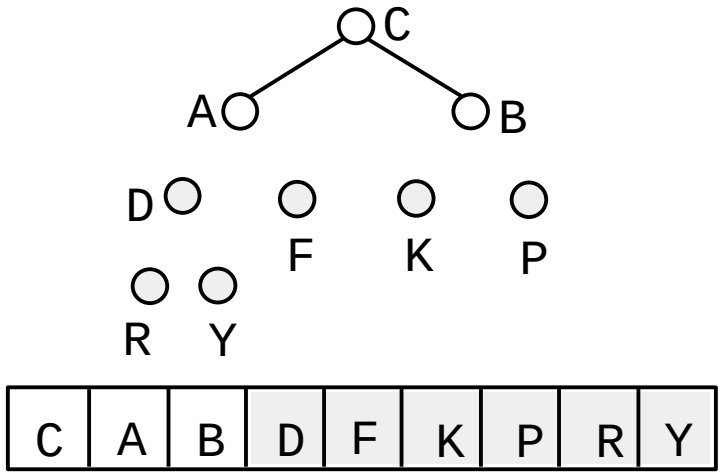


Heapsort

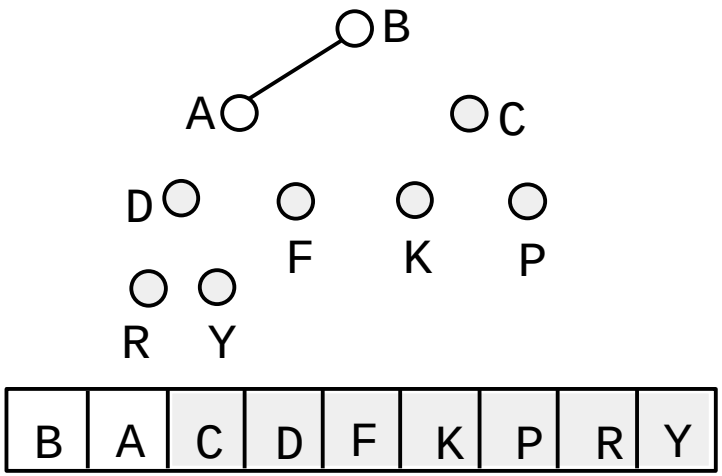


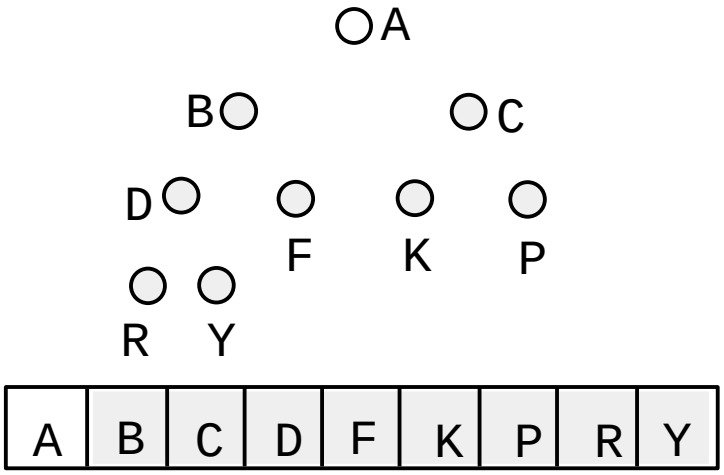






Heapsort





The total number of comparisons

$$2\sum_{i=1}^{n}\log_2 i = 2(\log_2 1+\log_2 2+\ldots+\log_2 n)$$

$$= 2\log_2(1\cdot 2\cdot\ldots\cdot n) = 2\log_2 n!$$

$$\approx 2n\log_2 n - 3n$$

$$= O(n\log n)$$

Heapsort

```
void  Heapsort( ListType *pList )
{
  int         i;

  BuildHeap( pList );
  for ( i=pList->count-1; i>=1; i-- ) {
    Swap( pList, 0, i );
    Heapify( pList, i );
  }
}

BuildHeap( ListType *pList )
{
  int         i;

  for (i=pList->count/2; i>=0; i--)
     Heapify( pList, i );
}
```

Priority Queues

Two operations

- Insert : Insert an item
- ExtractMax : Remove the item having the largest key

```
Insert     :  23  17  31   *   *  99  87   *   *   *
ExtractMax :              31  23          99  87  23
```

Implementations :

	Insert	ExtractMax
Sorted contiguous list	$O(n)$	$O(1)$
Binary search tree	$O(\log n)$ avg.	$O(\log n)$ avg.
Heap	$O(\log n)$	$O(\log n)$