Heapsort:

Unsorted List ➔ Heap ➔ Sorted List

```
#define SIZE 10
typedef struct {
   int  elem[SIZE];
   int lastNdx;
}HeapList;
```

A] Unsorted List

HeapList HL;   //Unsorted List

elem	5	1	8	2	9	1	7	2	3
	0	1	2	3	4	5	6	7	8

lastNdx	8

Binary Tree (not a POT)

```
          5
        /   \
      1       8
    /   \   /  \
   2     9 1    7
  / \
 2   3
```

B] Heap [Insert all elements in an initially empty Heap

HeapList HL; //Unsorted List

elem	5	1	8	2	9	1	7	2	3
	0	1	2	3	4	5	6	7	8

lastNdx 8

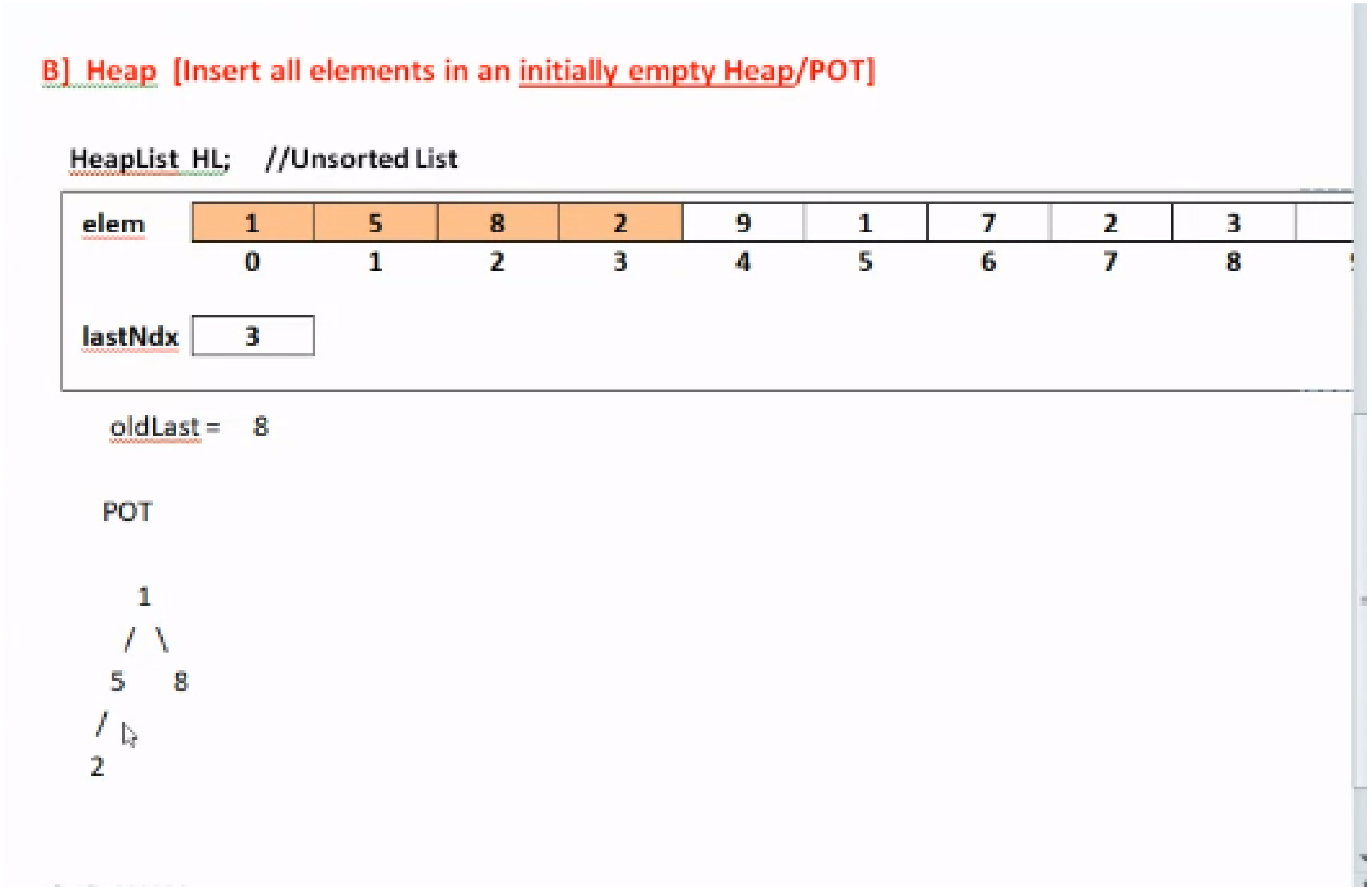
B] Heap [Insert all elements in an initially empty Heap/POT]

HeapList HL; //Unsorted List

elem	5	1	8	2	9	1	7	2	3
	0	1	2	3	4	5	6	7	8

lastNdx -1

oldLast = 8



B] Heap [Insert all elements in an initially empty Heap/POT]

HeapList HL; //Unsorted List

elem	1	5	8	2	9	1	7	2	3	
	0	1	2	3	4	5	6	7	8	

lastNdx	3

oldLast = 8

POT

```
    1
   / \
  5    8
 /
2
```

B] Heap [Insert all elements in an initially empty Heap/POT]

HeapList HL; //Unsorted List

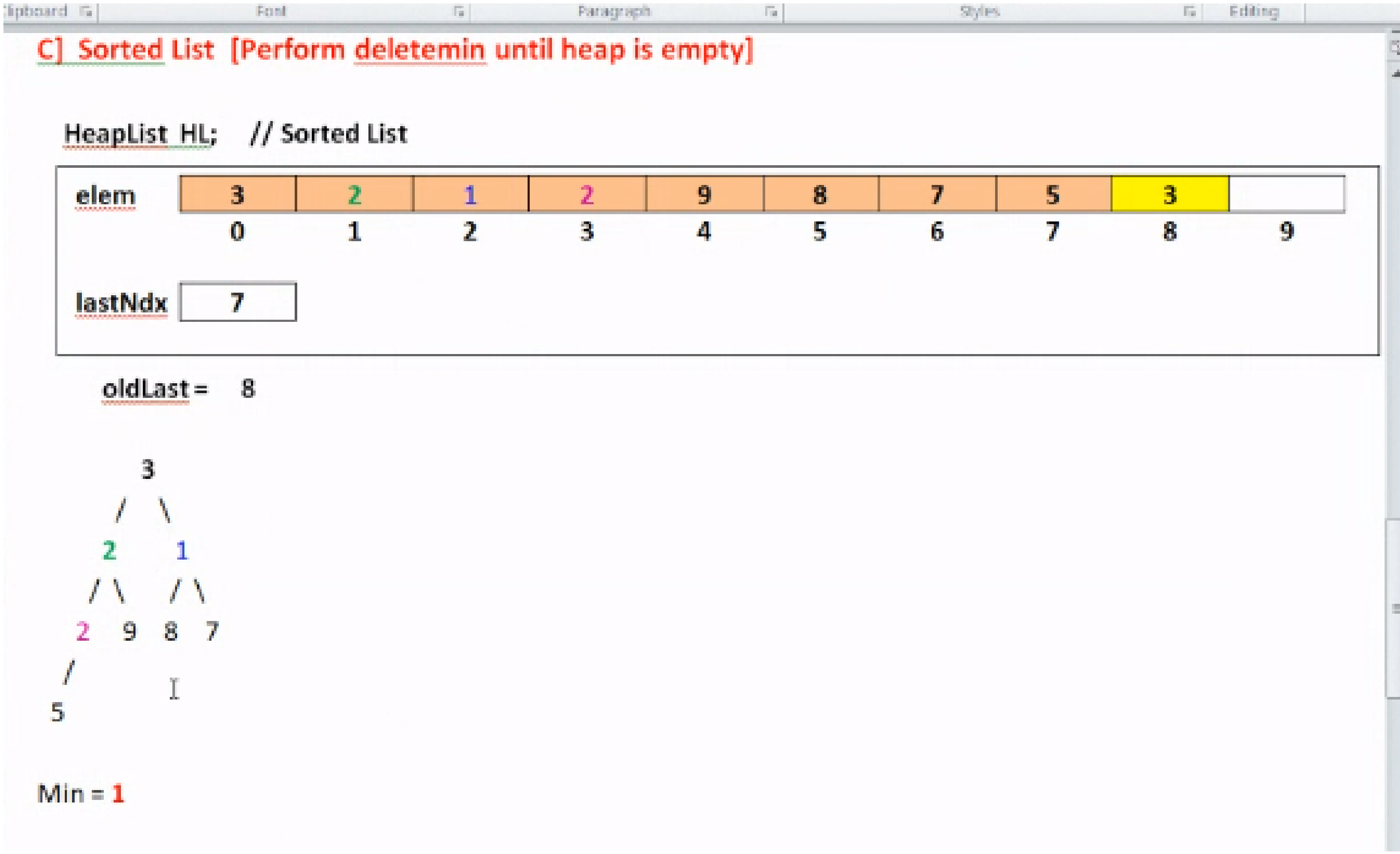
elem	1	2	8	5	9	1	7	2	3
	0	1	2	3	4	5	6	7	8

lastNdx 8

oldLast = 8

```
      POT
        1
       / \
     2     1
    / \   / \
   2   9 8   7
  / \
 5   3
```

cfpeña082821



C] Sorted List [Perform deletemin until heap is empty]

HeapList HL; // Sorted List

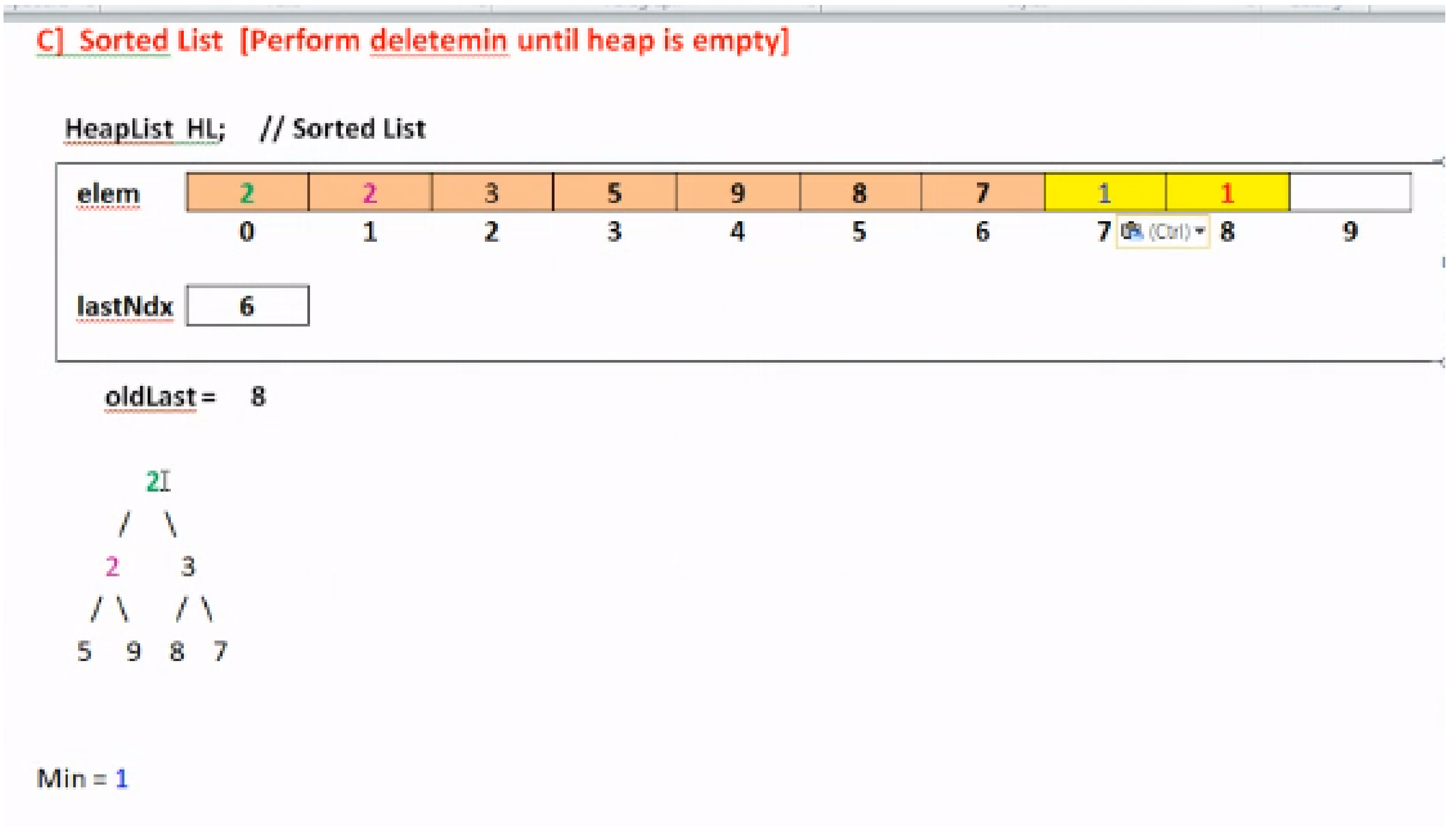
	0	1	2	3	4	5	6	7	8	9
elem	3	2	1	2	9	8	7	5	3	

lastNdx 7

oldLast = 8

```
        3
      /   \
    2       1
   / \     / \
  2   9   8   7
 /
5
```

Min = 1



C] Sorted List [Perform deletemin until heap is empty]

HeapList HL; // Sorted List

	0	1	2	3	4	5	6	7	8	9
elem	2	2	3	5	9	8	7	1	1	

lastNdx 6

oldLast = 8

```
        2
      /   \
    2       3
   / \     / \
  5   9   8   7
```

Min = 1

C] Sorted List [Perform deletemin until heap is empty]

HeapList HL; // Sorted List

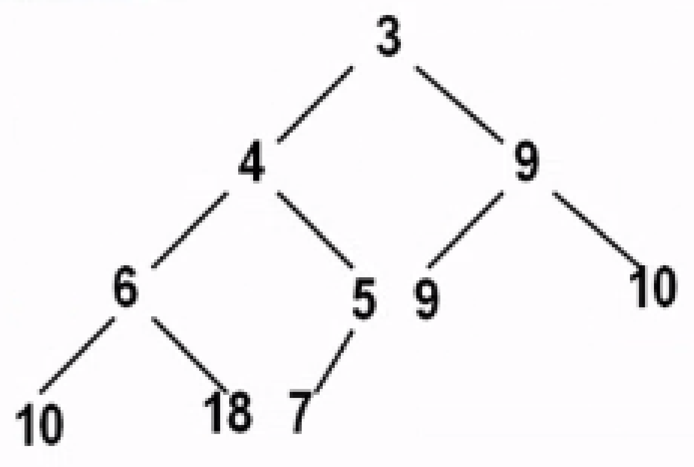
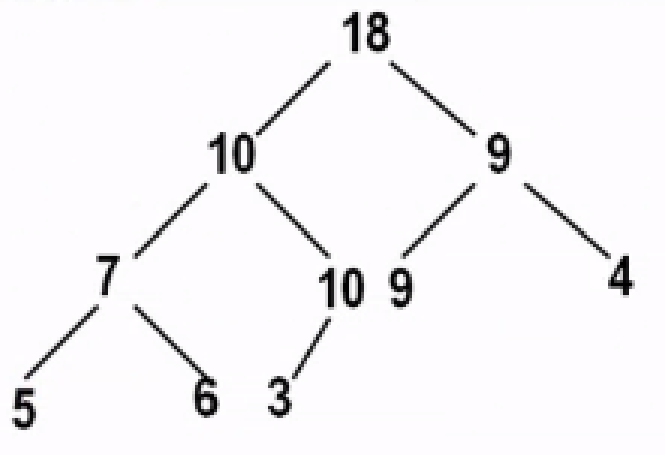
elem	2	5	3	7	9	8	2	1	1	
	0	1	2	3	4	5	6	7	8	9

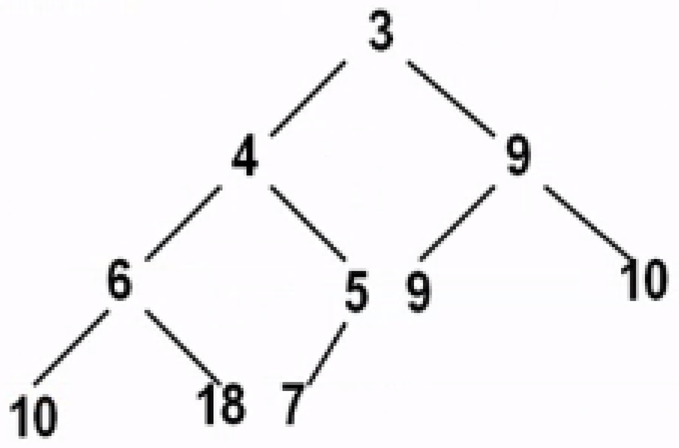
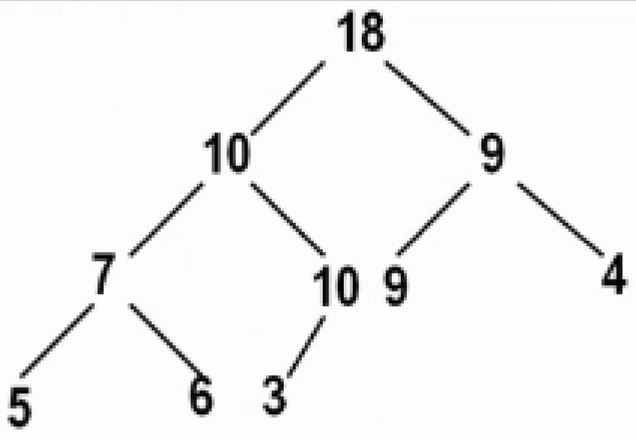
lastNdx	5

oldLast = 8

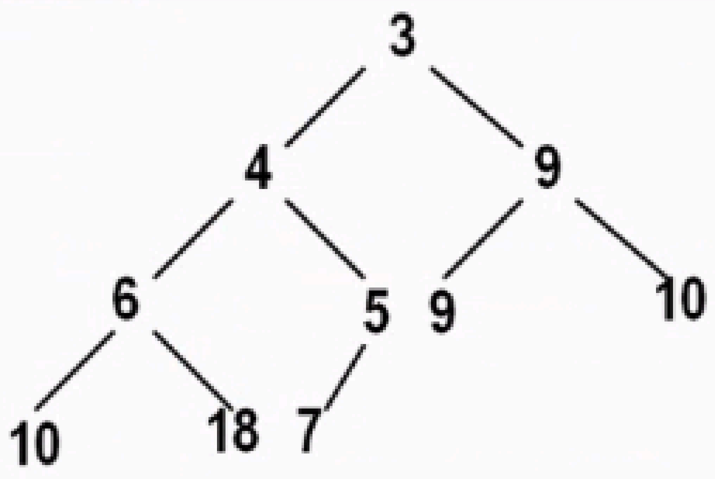
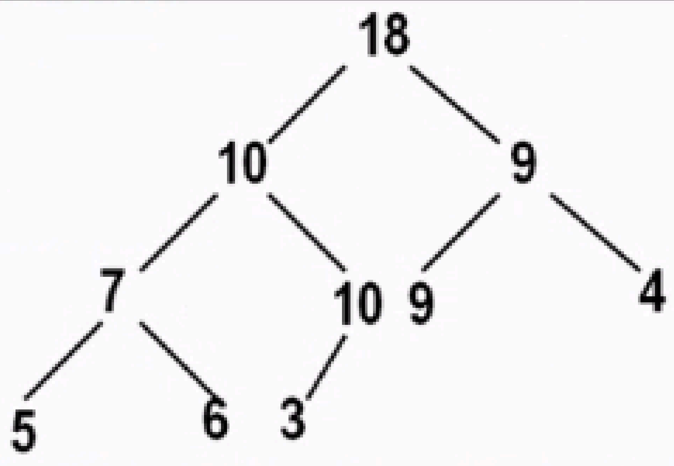
```
        2
      /   \
    5       3
   / \     /
  7   9   8
```

Min = 2

MinHeap	MaxHeap
	
POT Property: Priority(Parent) <= Priority(child)	POT Property: Priority(Parent) >= Priority(child)

MinHeap	MaxHeap
	
POT Property: Priority(Parent) <= Priority(child)	POT Property: Priority(Parent) >= Priority(child)

Type of Heap	POT Property	Root Value			
MinHeap	Priority(Parent) <= Priority(child)	Minimum			
MaxHeap	Priority(Parent) >= Priority(child)	Maximun			

MinHeap	MaxHeap
	
POT Property: Priority(Parent) <= Priority(child)	POT Property: Priority(Parent) >= Priority(child)

Type of Heap	POT Property	Root Value	Operations	Heapsort in place
MinHeap	Priority(Parent) <= Priority(child)	Minimum	Insert() and deletemin	Descending order
MaxHeap	Priority(Parent) >= Priority(child)	Maximun	Insert() and deletemax	Ascending order

2 ways in making the unsorted list into a heap

1) Insert each element into an initially empty POT (heap)
2) Heapify Process

2 ways in making the unsorted list into a heap

1) Insert each element into an initially empty POT (heap)
2) Heapify Process

A] Unsorted List

HeapList HL; //Unsorted List

elem	5	1	8	2	9	1	7	2	3	
	0	1	2	3	4	5	6	7	8	9
lastNdx	8									

Binary Tree (not a POT)

```
        5
       / \
     1     8
    / \   / \
   2   9 1   7
  / \
 2   3
```

```
      /   \
    1       5
   /  \   /  \
  2     9 8    7
 /  \
2    3
```

POT using heapify() procedure	POT by inserting all elements in an initially empty PO

POT using heapify() procedure

```
          1
        /   \
      1       5
     /  \   /  \
    2     9 8    7
   /  \
  2    3
```

POT by inserting all elements in an initially empty PO

```
       1
     /   \
    2      1
   / \    / \
  2   9  8   7
 / \
5   3
```

(Ctrl)