```
/////////////// Heaps
#include <stdio.h>
#include <stdlib.h>
//min-heap

void printArray(int *arr,int n){
    int i=0;
    while(n--) printf("[%d] ",arr[i++]);
    printf("\n");
}

///////////// Sentinel array[0]=number of element
////////////   One  based indexing  ////////////
void Insert(int hp[], int value){
   int i;hp[0]++;
   for( i = hp[0]; hp[ i / 2 ] > value && i>1; i /= 2 )
                    hp[ i ] = hp[ i / 2 ];
   hp[ i ] = value;
   for( i = 0; i<=hp[0] ; i++ )
         printf("%2d ",hp[i]);printf("<<((%2d))\n",value);
}

int Delete(int hp[]) {
   int last=hp[hp[0]--],i, child, retValue=hp[1];
   for( i = 1; i * 2 <= hp[0]; i = child )
       {
       child = i * 2;
       if( child != hp[0] && hp[ child + 1 ] < hp[ child ]) child++;
       /* Percolate one level */
       if( last > hp[ child ] ) hp[ i ] = hp[ child ];else  break;
       }
    hp[ i ] = last;
   for( i = 0; i<=hp[0] ; i++ )
       printf("%2d ",hp[i]);printf("(>>(%2d))\n",retValue);
  return retValue;
}
///////////// No Sentinel, Zero based indexing /////////////
void siftUP(int hp[], int i)
{
  if (i>0) {
    int parent = (i-1)/2;
    if (hp[parent]>hp[i] ) {
        int t=hp[i];hp[i]=hp[parent];hp[parent]=t;//swap
        siftUP(hp, parent);
         }
   }
}

void heapifyUp(int *hp,int n){ // O(n log(n))
  for (int i=1;i<n;i++)
           {siftUP(hp,i);printArray(hp,n);}
}
```

```
void siftDown(int hp[], int i,int n)
{
   if (i<=n/2 ) {
    int child = 2*i+1;
    if( child != n && hp[ child + 1 ] < hp[ child ])child++;
    if (hp[i]>hp[child] ) {
        int t=hp[i];hp[i]=hp[child];hp[child]=t;//swap
        siftDown(hp, child,n);
         }
  }
}

void heapifyDown(int *hp,int n){ //O(n)
  for (i=n/2;i>1;i--)
           {siftDown(hp,i,n);printArray(hp,n);}
}


//////////// Miscellaneous ///////////////
void copArray(int *dest,int *source,int n){
   *dest++=0;while (n--) *dest++=*source++;
}

void showHeap(int arr[],int n){
 char format[]=" %3d/%1d";int y=6,i,j,jj;
 printf("\n--------\n");
    int levels=(int)log2(n)+1;
    int ls=1,k=0;int st=1<<(levels-1);
    printf("Number of levels %d\n",levels);
    for ( i=0;i<levels;i++){printf("%d>>\t",i);
        for ( j=0;j<(st-1)*y/2;j++) printf(" ");
        for ( j=0;j<ls;j++) {printf(format,arr[k],k);k++;
                                for ( jj=0;jj<(st-1)*y;jj++) printf(" ");
                                if (k==n)break;}
        printf("\n");
        ls<<=1; st>>=1;
    }
 printf("\n--------\n");
}
```

```c
int main()
{
    int heap[100];heap[0]=0;// Sentinel : #number of elements
    //////////////// One based Counting ///////////
    int x[]={700,10,30,500,400,300,200,2,3,1};
    int n=sizeof(x)/sizeof(int);
    int *y=copArray(x,n);
    printf("Inserting (one based Counting: [0]=sentinel):\n");
    for (int i=0;i<n;i++) Insert(heap,x[i]);printf("\n");
    printArray(heap,n+1);showHeap(&heap[1],n);
    printf("Deleting:\n");
    for (int i=0;i<n;i++) Delete(heap);

    // WE CAN DO IT IN PLACE
    printf("\ninPlace (zero based Counting, No Sentinel:\n");
    printf("\nHeapify (SiftUp) in Place (not Sort) O(n log(n))\n");
    printArray(x,n);
    heapifyUp(x,n);
    printArray(x,n);
    //int l=(int)log2(n);
    //printf("%d %d\n",l,(int)pow(2,l)-1);
    showHeap(x,n);
    printArray(y,n);
    printf("\nHeapify (SiftUp) in Place (not Sort) O(n)\n");
    heapifyDown(y,n);
    printArray(x,n);
    showHeap(x,n);
}
```

```
/* Output
Inserting (one based Counting: [0]=sentinel):
 1 700 <<((700))
 2 10 700 <<((10))
 3 10 700 30 <<((30))
 4 10 500 30 700 <<((500))
 5 10 400 30 700 500 <<((400))
 6 10 400 30 700 500 300 <<((300))
 7 10 400 30 700 500 300 200 <<((200))
 8  2 10 30 400 500 300 200 700 <<(( 2))
 9  2  3 30 10 500 300 200 700 400 <<(( 3))
10  1  2 30 10  3 300 200 700 400 500 <<(( 1))

[10] [1] [2] [30] [10] [3] [300] [200] [700] [400] [500]

--------
Number of levels 4
0>>                              1/0
1>>                 2/1                     30/2
2>>          10/3          3/4        300/5        200/6
3>>      700/7 400/8 500/9

--------
```

```
Deleting:
 9  2  3 30 10 500 300 200 700 400 (>>( 1))
 8  3 10 30 400 500 300 200 700 (>>( 2))
 7 10 400 30 700 500 300 200 (>>( 3))
 6 30 400 200 700 500 300 (>>(10))
 5 200 400 300 700 500 (>>(30))
 4 300 400 500 700 (>>(200))
 3 400 700 500 (>>(300))
 2 500 700 (>>(400))
 1 700 (>>(500))
 0 (>>(700))

inPlace (zero based Counting, No Sentinel:

Heapify (SiftUp) in Place (not Sort) O(n log(n))
[700] [10] [30] [500] [400] [300] [200] [2] [3] [1]
[10] [700] [30] [500] [400] [300] [200] [2] [3] [1]
[10] [700] [30] [500] [400] [300] [200] [2] [3] [1]
[10] [500] [30] [700] [400] [300] [200] [2] [3] [1]
[10] [400] [30] [700] [500] [300] [200] [2] [3] [1]
[10] [400] [30] [700] [500] [300] [200] [2] [3] [1]
[10] [400] [30] [700] [500] [300] [200] [2] [3] [1]
[2] [10] [30] [400] [500] [300] [200] [700] [3] [1]
[2] [3] [30] [10] [500] [300] [200] [700] [400] [1]
[1] [2] [30] [10] [3] [300] [200] [700] [400] [500]
[1] [2] [30] [10] [3] [300] [200] [700] [400] [500]

--------
Number of levels 4
0>>                            1/0
1>>               2/1                   30/2
2>>         10/3        3/4       300/5       200/6
3>>      700/7 400/8 500/9

--------
[700] [10] [30] [500] [400] [300] [200] [2] [3] [1]

Heapify (SiftUp) in Place (not Sort) O(n)
[700] [10] [30] [500] [400] [300] [200] [2] [3] [1]
[700] [10] [30] [500] [1] [300] [200] [2] [3] [400]
[700] [10] [30] [2] [1] [300] [200] [500] [3] [400]
[700] [10] [30] [2] [1] [300] [200] [500] [3] [400]
[1] [2] [30] [10] [3] [300] [200] [700] [400] [500]

--------
Number of levels 4
0>>                            1/0
1>>               2/1                   30/2
2>>         10/3        3/4       300/5       200/6
3>>      700/7 400/8 500/9

--------*/
```

```
//Heapify / Sort

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void printArray(int *arr,int n){
    int i=0;
    while(n--) printf("[%d] ",arr[i++]);
    printf("\n");
}
void showHeap(int arr[],int n){
 char format[]=" %3d/%d";int y=6,i,j,jj;
 printf("\n--------\n");
    int levels=(int)log2(n)+1;
    int ls=1,k=0;int st=1<<(levels-1);
    printf("Number of levels %d\n",levels);
    for ( i=0;i<levels;i++){printf("level %d",i);
        for ( j=0;j<(st-1)*y/2;j++) printf(" ");
        for ( j=0;j<ls;j++) {printf(format,arr[k],k);k++;
                                for ( jj=0;jj<(st-1)*y;jj++) printf(" ");
                                if (k==n)break;}
        printf("\n");
        ls<<=1; st>>=1;
    }
 printf("\n--------\n");
}
#define swap(x,y) {int temp=x;x=y;y=temp;}

void heapifyDown(int *arr,int n){
  int i;
  for ( i = 1; i < n; i++)
        // if child is bigger than parent
        if (arr[i] > arr[(i-1)/2])
        {   int j = i;
            // swap child and parent until parent is smaller
            while (arr[j] > arr[(j-1)/2])
            {
                swap(arr[j], arr[(j-1)/2]);
                j = (j-1)/2;
            }
        }
}


void heapSort(int arr[],int n){ // 0 base indexing
    int i;heapifyDown(arr, n);
    printArray(arr,n);
    showHeap(arr,n);
    for (i = n - 1; i > 0; i--)
    {
        // swap value of first indexed with last indexed
        swap(arr[0], arr[i]);
        // maintaining heap property after each swapping
```

```
            int j = 0, index;
            do
            {
                index = (2 * j + 1);
                // if left child is smaller than right child point to right
                if (arr[index]<arr[index + 1] && index < (i - 1))index++;
                // if parent is smaller than child then swap parent with child
                if (arr[j]<arr[index] && index < i) swap(arr[j], arr[index]);
                j = index;
            } while (index < i);
        }
    }

    int main()
    {
        int yyy[]={0, 10,20,3,17,88,100,1,200,7};
        printArray(yyy,sizeof(yyy)/4);
        heapSort(yyy,sizeof(yyy)/4);
        printArray(yyy,sizeof(yyy)/4);
        return 0;
    }

         /*Output
         [0] [10] [20] [3] [17] [88] [100] [1] [200] [7]
         [200] [100] [88] [17] [7] [10] [20] [0] [1] [3]

         --------
         Number of levels 4
         level 0                      200/0
         level 1            100/1                   88/2
         level 2      17/3         7/4         10/5         20/6
         level 3    0/7    1/8    3/9

         --------
         [0] [1] [3] [7] [10] [17] [20] [88] [100] [200] */
```

```cpp
/// Same in cpp ! The Caller:
#include <iostream>
#include "heap.h"
#include <math.h>
#include <stdio.h>
using namespace std;
int main()
{
    Heap hp(3);//hp.verbose=0;
    hp.push(10); hp.push(12);
    hp.push(9); hp.push(8);
    hp.push(100);hp.push(7);
    hp.push(2);hp.push(300);
    hp.push(1);
 while (!hp.isEmpty()) hp.pop();
 int array[]=
  {10,20,3,17,88,100,200,9,1,700};
 Heap h(array,sizeof(array)/4);
 h.verbose=0;h.show();
 int *p=h.heapSort();
 for (int i=0; i<sizeof(array)/4;i++)
   cout << p[i] << " "; cout << endl;
 return 0;
}
////////////// The Header heap.h
#ifndef HEAP_H_INCLUDED
#define HEAP_H_INCLUDED

class Heap{
private:
    int n; // number of elements
    int capacity;
    int * arr; // 1 based indexing

public:
    // constructors
    int verbose;
    Heap();
    Heap(int);
    Heap(int*,int);
    //methods
    void siftDown(int ,int );
    void heapifyDown();
    int* heapSort();
    void push(int value);
    int peek();
    int pop();
    int isEmpty() {return n==0;}
    int isFull();
    void show();
};
#endif // HEAP_H_INCLUDED
```

```
/*Required Output:
capacity 3,allocating 16 bytes
10 <<((10))
10 12 <<((12))
 9 12 10 <<(( 9))
capacity 7,re-allocating 32 bytes
 8  9 10 12 <<(( 8))
 8  9 10 12 100 <<((100))
 7  9  8 12 100 10 <<(( 7))
 2  9  7 12 100 10  8 <<(( 2))
capacity 15,re-allocating 64 bytes
 2  9  7 12 100 10  8 300 <<((300))
 1  2  7  9 100 10  8 300 12 <<(( 1))
 2  9  7 12 100 10  8 300 (>>( 1))
 7  9  8 12 100 10 300 (>>( 2))
 8  9 10 12 100 300 (>>( 7))
 9 12 10 300 100 (>>( 8))
10 12 100 300 (>>( 9))
12 300 100 (>>(10))
100 300 (>>(12))
300 (>>(100))
(>>(300))
capacity 15,allocating 64 bytes
Number of levels 4
                   1
          9                 3
    10          88     100     200
20      17   700

-------------
700 200 100 88 20 17 10 9 3 1
*/
```