HEAP

1. BINARY HEAP IMPLEMENTATION

```
#include <bits/stdc++.h>
using namespace std;

class MinHeap{
    int *arr;
    int size;
    int capacity;

    public:

    MinHeap(int c){
    size = 0;
    capacity = c;
    arr = new int[c];
    }

    int left(int i) { return (2*i + 1); }
    int right(int i) { return (2*i + 2); }
    int parent(int i) { return (i-1)/2; }
};

int main()
{
    MinHeap h(11);
    return 0;
}
```

2. INSERT

T:O(logn)

```
void insert(int x)
   {
   if (size == capacity)return;
   size++;
   arr[size-1]=x;

   for (int i=size-1;i!=0 && arr[parent(i)]>arr[i];)
   {
      swap(arr[i], arr[parent(i)]);
      i = parent(i);
   }
   }
```

3. HEAPIFY

T:O(logn) S:O(h)

```
void minHeapify(int i)
   {
   int lt = left(i);
   int rt = right(i);
   int smallest = i;
   if (lt < size && arr[lt] < arr[i])
       smallest = lt;
   if (rt < size && arr[rt] < arr[smallest])
       smallest = rt;
   if (smallest != i)
   {
       swap(arr[i],arr[smallest]);
       minHeapify(smallest);
   }  }
```

4. EXTRACT MIN

T:O(logn)

```
int extractMin()
   {
   if (size <= 0)
      return INT_MAX;
   if (size == 1)
   {
      size--;
      return arr[0];
   }
   swap(arr[0],arr[size-1]);
   size--;
   minHeapify(0);

   return arr[size];
   }
```

5. DECREASE KEY

T:O(logn)

```
void decreaseKey(int i, int x)
   {
   arr[i] = x;
   while (i != 0 && arr[parent(i)] > arr[i])
   {
     swap(arr[i], arr[parent(i)]);
     i = parent(i);
   }
   }
```

6. **DELETE KEY**

T:O(logn)

```
void deleteKey(int i)
   {
      decreaseKey(i, INT_MIN);
      extractMin();
   }
```

7. **BUILD HEAP**

T:O(n)

```
void buildHeap(){
      for(int i=(size-2)/2;i>=0;i--)
         minHeapify(i);
   }
```

8. **HEAP SORT**

T:O(nlogn)

```
#include <iostream>
using namespace std;


void heapify(int arr[], int n, int i)
{
        int largest = i;
        int l = 2*i + 1;
        int r = 2*i + 2;
        if (l < n && arr[l] > arr[largest])
                largest = l;

        if (r < n && arr[r] > arr[largest])
                largest = r;

        if (largest != i)
        {
                swap(arr[i], arr[largest]);
                heapify(arr, n, largest);
        }
}

void buildheap(int arr[],int n){
    for (int i = n / 2 - 1; i >= 0; i--)
                heapify(arr, n, i);
}
void heapSort(int arr[], int n)
{
        buildheap(arr,n);

        for (int i=n-1; i>0; i--)
        {
                swap(arr[0], arr[i]);
                heapify(arr, i, 0);
        }
}

void printArray(int arr[], int n)
{
        for (int i=0; i<n; ++i)
                cout << arr[i] << " ";
        cout << "\n";
}
```

```
int main()
{
        int arr[] = {12, 11, 13, 5, 6, 7};
        int n = sizeof(arr)/sizeof(arr[0]);

        heapSort(arr, n);

        cout << "Sorted array is \n";
        printArray(arr, n);
}
```

9. MAX HEAP PRIORITY QUEUE

```
#include <iostream>
#include<queue>
using namespace std;

int main(){
    priority_queue <int> pq;
    pq.push(10);
    pq.push(15);
    pq.push(5);
    cout<<pq.size()<<" ";
    cout<<pq.top()<<" ";
    while(pq.empty()==false){
        cout<<pq.top()<<" ";
        pq.pop();
    }

    return 0;
}
```

10. MIN HEAP PRIORITY QUEUE

```
#include <iostream>
#include<queue>
using namespace std;

int main(){
    priority_queue <int,vector<int>,greater<int>> pq;
    pq.push(10);
    pq.push(15);
    pq.push(5);
    cout<<pq.size()<<" ";
    cout<<pq.top()<<" ";
    while(pq.empty()==false){
        cout<<pq.top()<<" ";
        pq.pop();
    }

    return 0;
}
```

11. SORT K SORTED ARRAY

T:O(nlogk)

```
#include <bits/stdc++.h>
using namespace std;

int sortK(int arr[], int n, int k)
{
     priority_queue<int, vector<int>, greater<int> > pq;

    for(int i=0;i<=k;i++)
        pq.push(arr[i]);
     int index = 0;
     for (int i = k + 1; i < n; i++) {
             arr[index++] = pq.top();
             pq.pop();
             pq.push(arr[i]);
     }

     while (pq.empty() == false) {
             arr[index++] = pq.top();
             pq.pop();
     }
}

void printArray(int arr[], int size)
{
     for (int i = 0; i < size; i++)
             cout << arr[i] << " ";
     cout << endl;
}

int main()
{
     int k = 3;
     int arr[] = { 2, 6, 3, 12, 56, 8 };
     int n = sizeof(arr) / sizeof(arr[0]);
     sortK(arr, n, k);

     cout << "Following is sorted array" << endl;
     printArray(arr, n);

     return 0;
}
```

12. BUY MAXIMUM ITEMS WITH GIVEN SUM

T:O(nlogn)

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n=5;
    int cost[n]={1,12,5,111,200};
    int sum=10;

    priority_queue<int,vector<int>,greater<int>> pq;

    int res=0;
    for(int i=0;i<n;i++)
        pq.push(cost[i]);

    for(int i=0;i<n;i++){
        if(pq.top()<=sum){
            sum-=pq.top();
            pq.pop();
            res++;
        }else{
            break;
        }
    }
    cout<<res;
    return 0;
}
```

13. K LARGEST ELEMENTS

T:O((n-k)logk)

```
   #include <bits/stdc++.h>
using namespace std;

void firstKElements(int arr[], int n, int k)
{

        priority_queue <int,vector<int>,greater<int>> minHeap;
        for(int i = 0; i < k; i++)
        {
                minHeap.push(arr[i]);
        }
        for(int i = k; i < n; i++)
        {
                if (minHeap.top() > arr[i])
                        continue;
                else
                {
                        minHeap.pop();
                        minHeap.push(arr[i]);
                }
        }

        while(minHeap.empty()==false){
            cout<<minHeap.top()<<" ";
            minHeap.pop();
        }
}

int main()
{
        int arr[] = { 11, 3, 2, 1, 15, 5, 4, 45, 88, 96, 50, 45 };

        int size = sizeof(arr) / sizeof(arr[0]);

        int k = 3;

        firstKElements(arr,size,k);

        return 0;
}
```

14. K CLOSEST ELEMENTS

T:O(nlogk)

```
#include <bits/stdc++.h>
using namespace std;

void printKClosest(int arr[], int n, int k, int x)
{
     priority_queue<pair<int, int> > pq;
    for (int i = 0; i < k; i++)
        pq.push({ abs(arr[i] - x), i });

    for (int i = k; i < n; i++) {
        int diff = abs(arr[i] - x);
        if ( pq.top().first>diff ){
        pq.pop();
        pq.push({ diff, i });
        }
    }

    while (pq.empty() == false) {
        cout << arr[pq.top().second] << " ";
        pq.pop();
    }
}

int main()
{

         int arr[] = { 10,30,5,40,38,80,70 };

         int size = sizeof(arr) / sizeof(arr[0]);

         int x=35; int k = 3;

         printKClosest(arr,size,k,x);

         return 0;
}
```

15. MERGE K SORTED ARRAYS

T:O(nklogk)

```
#include <bits/stdc++.h>
using namespace std;

struct Triplet{
    int val,aPos,vPos;
    Triplet(int v,int ap, int vp){
        val=v;aPos=ap;vPos=vp;
    }
};

struct MyComp{
    bool operator()(Triplet &t1,Triplet &t2){
        return t1.val>t2.val;
    }
};

vector<int> mergeArr(vector<vector<int> > &arr)
{
    vector<int> res;

    priority_queue <Triplet, vector<Triplet>,MyComp> pq;

    for(int i=0;i<arr.size();i++){
        Triplet t(arr[i][0],i,0);
        pq.push(t);
    }

    while(pq.empty()==false){
        Triplet curr=pq.top();pq.pop();
        res.push_back(curr.val);
        int ap=curr.aPos;int vp=curr.vPos;
        if(vp+1<arr[ap].size()){
            Triplet t(arr[ap][vp+1],ap,vp+1);
            pq.push(t);
        }
    }

    return res;
}

int main()
{

         vector<vector<int> > arr{ { 10, 20, 30 },
                              { 5, 15 },
                              { 1, 9, 11, 18 } };
    vector<int> res=mergeArr(arr);
    cout << "Merged array is " << endl;
    for (auto x : res)
        cout << x << " ";

    return 0;
}
```

16. MEDIAN OF A STREAM

T:O(nlogn)

```
#include <bits/stdc++.h>
using namespace std;

void printMedians(int arr[],int n){
    priority_queue<int> s;
    priority_queue<int, vector<int>,greater<int>> g;
    s.push(arr[0]);
    cout<<arr[0]<<" ";
    for(int i=1;i<n;i++){
        int x=arr[i];
        if(s.size()>g.size())
        {
            if(s.top()>x){
                g.push(s.top());
                s.pop();
                s.push(x);
            }else g.push(x);
            cout<<(s.top()+g.top())/2.0<<" ";
        }else{
            if(x<=s.top()){
                s.push(x);
            }else{
                g.push(x);
                s.push(g.top());
                g.pop();
            }
            cout<<s.top()<<" ";
        }
    }
}

int main()
{
         int keys[] = { 12, 15, 10, 5, 8, 7, 16};

    printMedians(keys,7);

         return 0;
}
```