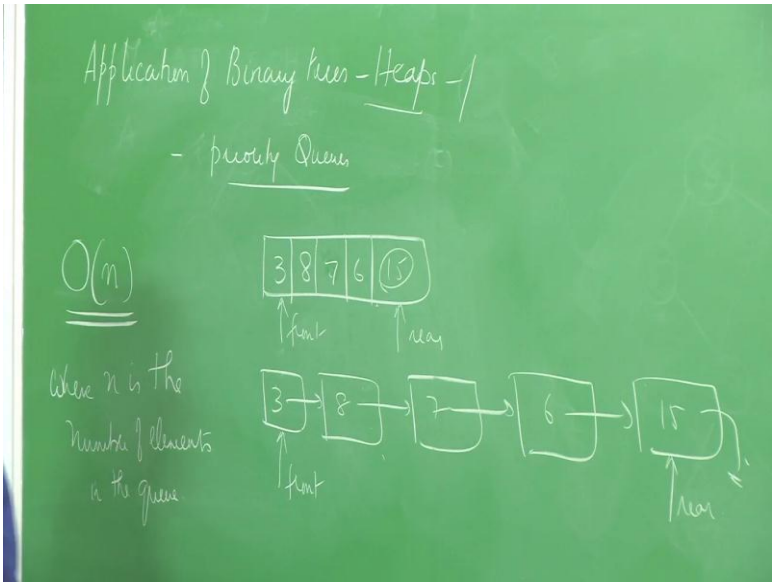
Programming, Data Structures and Algorithms
Prof. Hema Murthy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 13
Lecture – 51
Priority queues – heaps
Max heap: Complete binary tree, which is also a max tree
Operation on heaps: create, insert new element, delete largest element
Implementation of heap using

We talk about different types of binary trees, we look at the construction of Huffman code using the binary tree ADT. And then we also look at another special type of binary tree call the binary search tree, and, then we also look at how to construct a binary tree from we preorder and in order traverses given the preorder and in order traverses.

(Refer Slide Time: 00:40)



Now, I am going to give you one more application of binary trees, which is very important called the heaps or priority queue. Now, in the first few classes when we did list for example, we talked about queue as a special type of list, where we set insertion happen at one end and deletions happen at the other end. And it was the first in first out queue, you got it to the queue first, you got serve first that was the ideal. But, now sometimes is possible that you want have queue is where you give some priority tree.

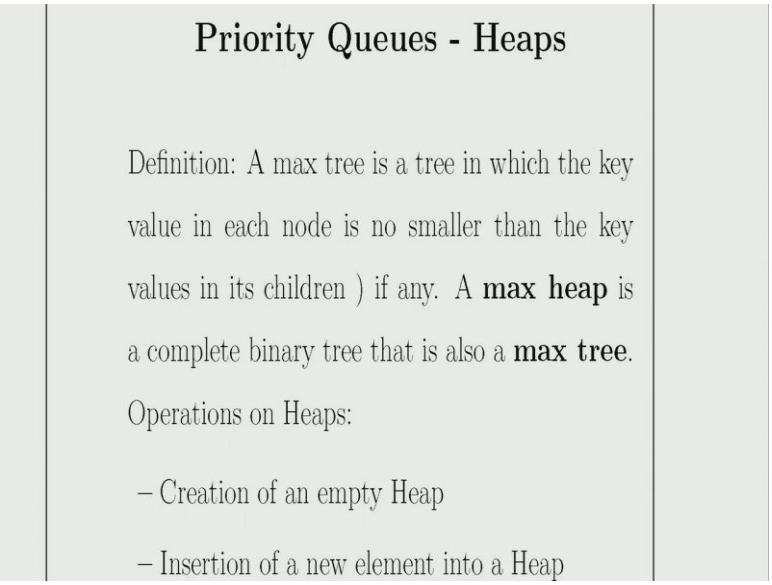
For example, if I am computer center in charge and there are large number of programs that are running, I like to ensure that those programs or those process is which take very less time should be executed first, rather than process is take a lot of time. So, I would like gives some kind of a priority, are you know if you have they your internet service provider, if you request is very small, you get higher priority compare to request where your downloading huge videos.

So, now how do we implement priority queues, that is the idea, suppose we implemented priority queue using an array like this or link list like this. And we also stored at every node the priority, let us say this was 3, 8, 7, 6, 15 and something like this and same numbers over here 6 of 15 remember in a queue, you always dedicated and this pointed to the front and this pointed to the rear and this pointed to the front and this pointed to the rear.

Now, if you want to serve the jobs based on priority then what do you have to do, it will keep on moving down front, until you come to the rear end for example. Because, that much the where the highest priorities and then serve that job. Similarly, in this link list example you start from the front, keep moving down till you reach the priority with the largest value and serve that. So, basically what happens is whether it is the array implementation or the link list implementation, remember this is an abstract data type.

So, everything is hidden from the user you have to use only the front keep moving down, until you come find out which is the largest priority. And you will know that only if you have end reach the end of the queue. Therefore, the time complexity of implementing a priority queue can be as largest order n, where n is number of elements in the queue number of elements in the queue, the question is can we do better and that is what going to answer using what a call priority queues, let see how you can use priority queue is to do this.

(Refer Slide Time: 03:57)



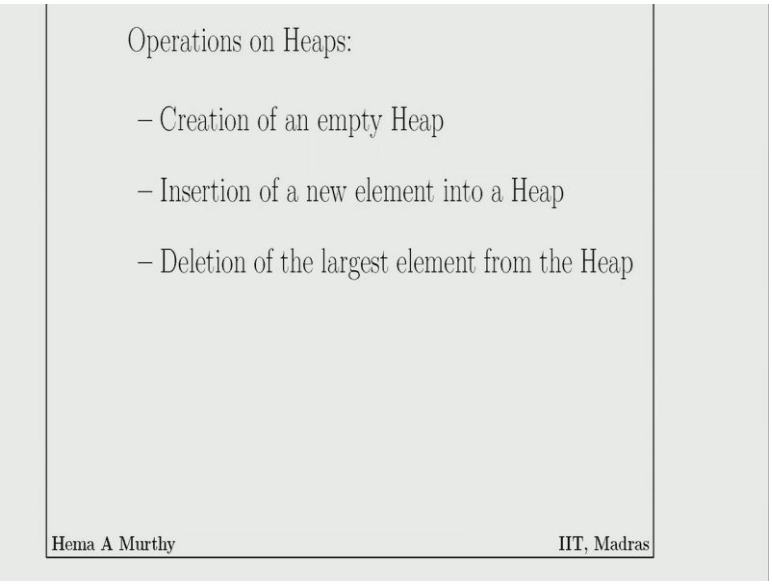
The definition of a priority queue comes from the definition of the maximum tree. What is the maximum tree? Maximum tree is a tree, in which the key value in each node is no smaller than the key value in it is children if any. A max heap, look at the definition over here is a complete binary tree that is also max tree, what it be remember sometimes back we saw the definition of a complete binary tree and we said that complete binary tree is look like this remember.

(Refer slide Time: 04:40)

We said this is complete binary tree, while this is not complete binary tree. So, what we are saying is we not at the have to have this max tree satisfied, it is no smaller than this children. That means, this can be 8, 7, 6, 4 and this is a heap, this is what is the definition of the max heap and it is a complete binary tree, not a full binary tree it is a complete binary tree now the questions.

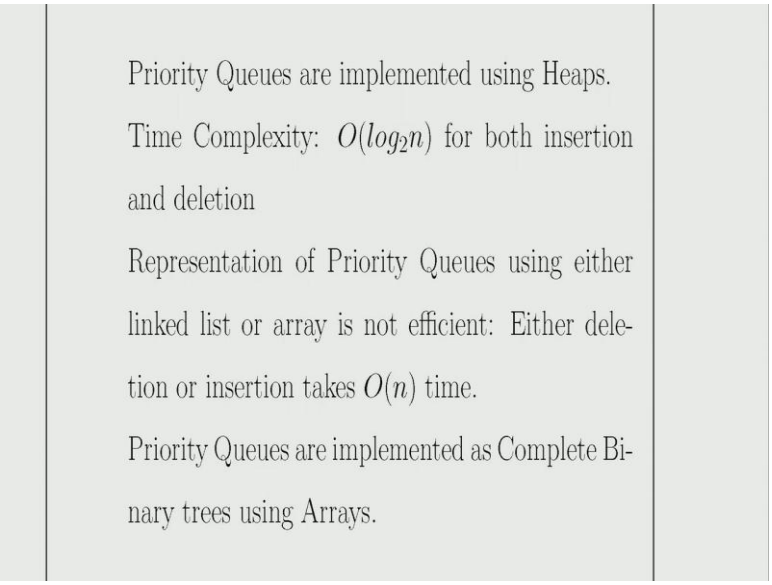
So, what our this complete binary tree is a binary the structure of the tree being a complete binary tree is very important for the implementation of operations on heaps. We already saw in the context of queues, we inserted in to the queue we deleted from the queue.

(Refer Slide Time: 05:39)



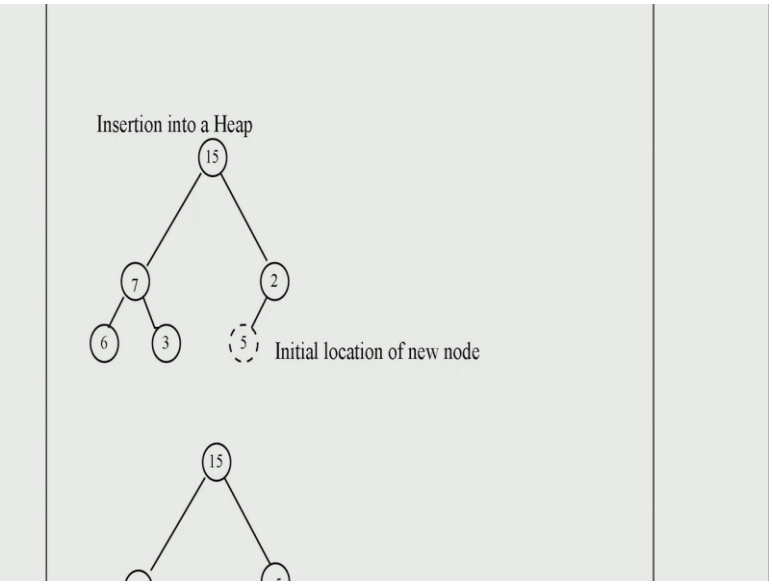
So, in much the same way by look at a operations on heaps is a priority queue, what a created empty priority queue, you want a insert it new element in to a priority queue and you want to delete the largest element from the priority queue. But, at the same thing can be done for minimum also, does not mater can use a min tree or max tree. The difference between max tree and max heaps is that, the max heap is a max tree which is also complete binary tree. So, now, let see how we can perform these operations of insertion into a max heap.

(Refer Slide Time: 06:15)



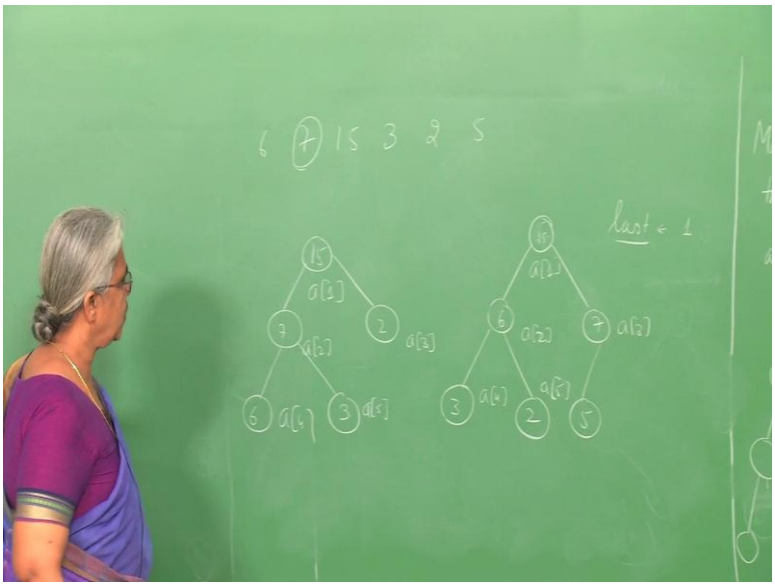
So, what is cleaned is that when you implement priority queues using heaps, the time complexity is order log n for both insertion and deletion. And we already talk about this that be, if we used normal queue it would be very expensive. But, priority queue is after be implemented has complete binary tree using arrays, that is what gives it this property of log n let us take an example.

(Refer Slide Time: 06:45)



Here is an example, let us say I want to I will go through this example completely.

(Refer Slide Time: 07:01)



Let us say I want to insert the elements, let say 6, 7 and 15, 3, 2 and 5 into a priority queue. Let us see how this is done, it is implemented is an array in this example here, the root node corresponds to the index of the array 1 and the next level for example, corresponds to the index of the first left child, corresponds to the index of the array 2. So, for example, in this particular tree what we are saying is, 15 has index suppose if this an array implementation a is the array that we are using, then 7 is at index 2 and 2 is at index 3, this is a very, very important.

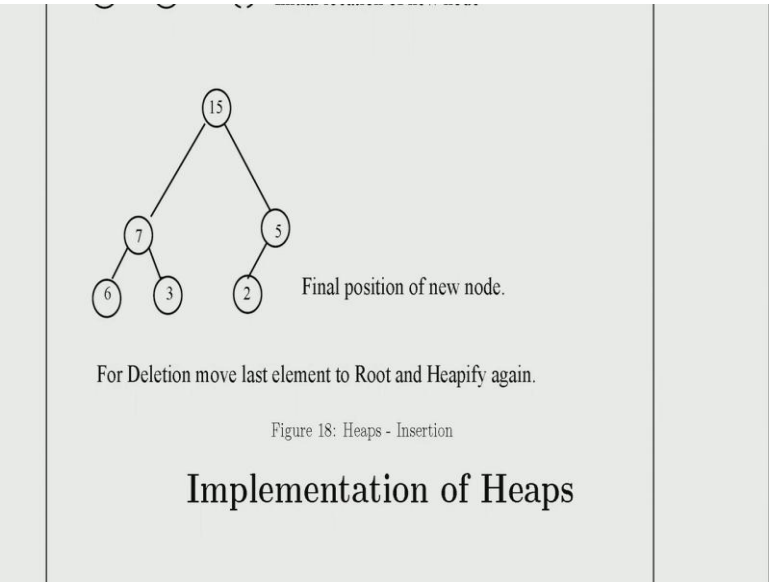
And so you would go essentially, if you look at it the array index is go through something like a level order, this is first level, second level, third level and so on and 3 is at a 5 and so on. Let us see now, how we insert in this elements into the heap, let say this is what was give to this, see is start with initially there is an empty heap. So, you create a node with the number 6, there is a store now this is store at a of 1.

How many elements they are in the heap? There is only one element in the heap. So, you keep track of variable call last and say last is pointing to 1, last is being pointing to the newest element data has been inserted into the heap, next say I want to insert call. So, what do I do, I insert it at a of 2 initial then I find that, because is now what is happening now, if I inserted here 7 and 6 when I compare what is happening, the heap property is valid.

So, what I do is I exchange this numbers I bring here 6 and 7, next what do you have, we have 15. So, now, I insert initially 15 at the position I compare 15 and 7 I find that 15 is larger than 7. So, I exchange 15 and 7 next I want to insert 3, so where will I insert 3 is initially, this is a 1, a 2, a 3 suppose is the array implementation and a at 4 I will first insert the element, then what do I do I compare this node with it is parent and you find this is correct order, so I leave it as it is.

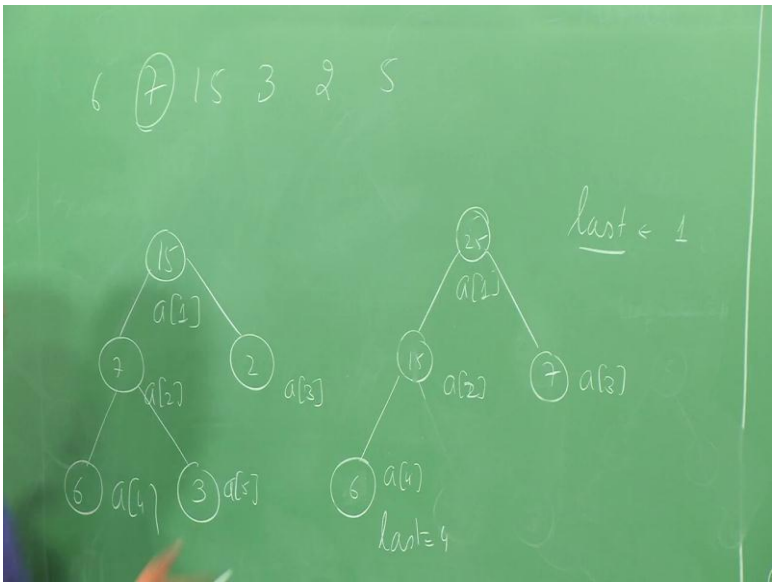
Next I insert 2, now 2 is also in the correct place with respective it is parent. And next I insert 5 which will be in the correct place. In the example let us given in the slide, again the slightly different example, suppose I had inserted 2 earlier suppose it was their at this position in the tree, then if I was inserting 5, then what will happen I will compare 5 it to would pair it, I find that 2 is smaller than 5, therefore I will move them and get this new tree.

(Refer Slide Time: 10:46)



So, basically this is the idea of insertion into a heap.

(Refer Slide Time: 11:34)



Now, let us see how long we took to insert a new element into the heap, first when we inserted element 6 there was only one element to be inserted. Therefore, constant time we set, then when we came to the second element we have to go one level down and so on. How many elements where there are in the heap at that type? Their only two elements in the heap and we are to go height of one.

Now, when I came to inserting the let us say I want to insert 3 for example, then what happen, now clearly what do I do, I am initially putting it over here and comparing it. Suppose, let say instant of 3 I have to insert 25 or something like that, let say this where not. So, I put 5 in the original position which is at the position of last, so this is last, this was last, this was last, this was last and finally now last is equal to 4.

Then what I do is compare it with it is pair it, then what do I do, I move it to it is position depending up on I keep comparing with this. For example, this is clearly larger than this therefore, this should be about this then I compare with it spared it keep going doing this, until I reach the root. In this particular case what will happen, will find that 6 will move here, 15 will move here and 25 will move here.

So, this is the idea of insertion do it, how long it will take now there are 4 nodes and we have to go only 2 levels down or we have to basically do log in operations 2 insert and element into a heap. It really makes lot of difference when you have huge heaps, especially jobs in a priority queue.

(Refer Slide Time: 12:38)

```
/*  The class Heap defines an ADT Heap. */
/* begin {Definition of class Heap} */
#define HEAP_SIZE 256
typedef int ElemType;
typedef int Position;
class Heap {
  ElemType priority[HEAP_SIZE];        // A Heap of integers
  Position last;
public:
  void insertHeap(ElemType x);       // Insertes an element on to the Heap
  ElemType deleteRoot();             // DeleteRoot of the Heap
  int empty();                    // returns 1 if Heap is empty
  void createHeap();              // creates an empty Heap
  void printHeap();                // Prints the heap
 };
/* end{Definition of class Heap} */
```

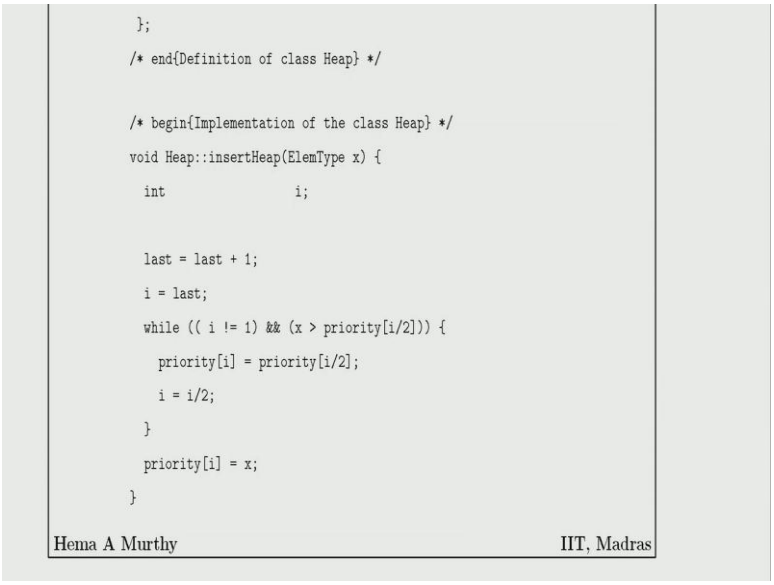
Here is an implementation of the heap have defined a class heap and I have given insertion into the heap, notice that I defined a priority heap size basically an array of heap size. The heap size can be fix, let us say reallocated if you want, that is a insertion of element x as usual element type is defined by the user and delete to you always only in a priority queue, you only have to delete the root.

(Refer Slide Time: 13:11)

So, now, let us see what will you do if you have to delete this root, because highest priority element has to be serve first, then what will happen it will root let us start. So,

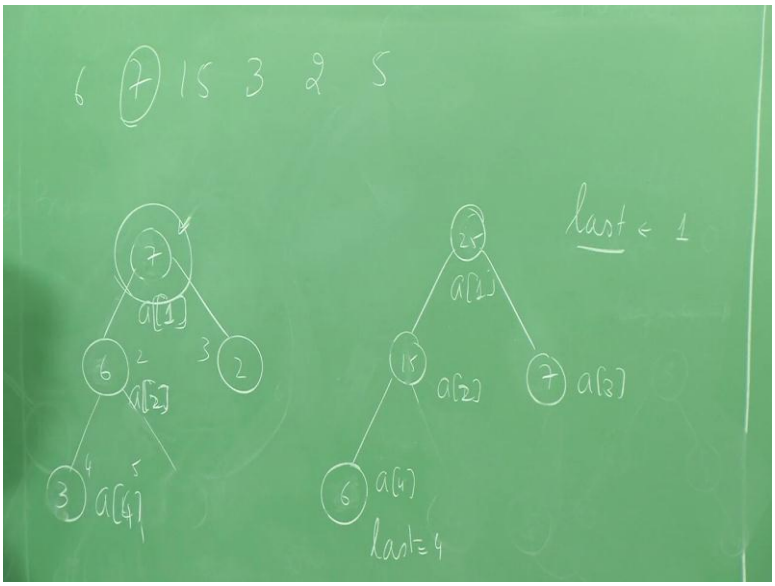
what we do is be take this put it over here, so 3 will come here once 15 has been deleted. And then what we do, it compare with it is children, clearly 3 is greater than 7 and it is also greater than 6. So, 3 will come here, 6 will come here and 7 will go here and this will be the new structure of the heap. So, when you delete a particular element, again how many element should we do, it took the last element moved it here move to steps down. So, again the complexity of delete is also only log n.

(Refer Slide Time: 13:52)



So, here is a C plus plus implementation and insertion what have doing, we incrementing the position last by one and what have we doing here, they checking whether the priority of the given node is greater than priority of i by 2. So, greater than i by 2 where ever it is for example, it putting in that particular position and moving all the other elements appropriate.

(Refer Slide Time: 14:21)



Our it finding the in this is, if this is i clearly if this is the notice that, the index of the parent is 2. So, index of the root is 1, index of this child is either 2 or 3 and index of 2 children is what now 4 or 5 and so on. So, given the index of the parent I can find the index of the child children and given the index of the children, we can finding index of the parent, this is possible only because I am implementing the priority queue as an array, otherwise it will not be possible.

So, what is this piece of code is simply doing is, while i dot equal to 1 while looking at we are starting from the last element and we are contemptuously checking trying to move up the tree depending up on where that particular node has to be insert, notice that we are starting from last. So, we are assigning to i priority i by 2; that means, the eliminated i by 2 a is replaced.

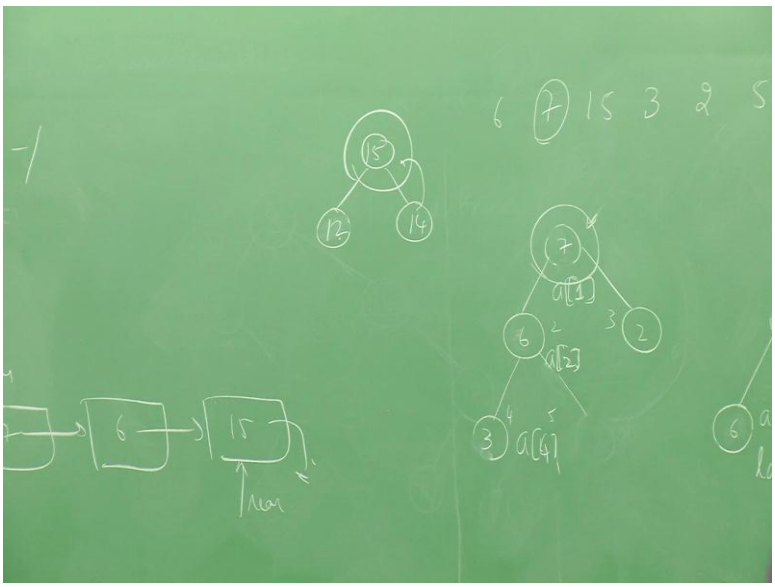
So, basically if you here if this one is smaller than what is being recently inserted, you may me move it one down, the example I already talk about in the context of insertion and this is I need repeat that until you have reach the root node.

(Refer Slide Time: 15:36)

```
Position          parent, child;
int               flag;
item = priority[1];
parent = 1;
child = 2;
temp = priority[last];
last = last - 1;
flag = 0;
while ((child <= last) && (!flag)){
  if ((child < last) && (priority[child] < priority[child+1]))
    child = child+1;
  if (temp >= priority[child])
    flag = 1;
  else {
    priority[parent] = priority[child];
    parent = child;
```

Now, this is the deletion again what are we doing, we are looking at basically first what we do is, here we once again deletion means what the number of elements is going to reduce by one. First of course, what do we do is, we start with the parent as the root, child as the two the left child of the root and we take the last element from the node and in percolated down, that is what I say go back to the root and the percolated down. So, what I doing over here, we first check which of the two children has higher priority and switch them, because you want to higher priority node to move up to the root.

(Refer Slide Time: 16:13)



Let us what we are saying is, if I have 15 here and I have deleted 15 and this was 17, this was 14, let us say this was 12 and this was 14, then what I do is, I want to make sure that 14 goes on tops. So, that the first thing that I do this is being deleted, so the next element that was become root is 14 and that is what this step is doing over here it is comparing which of the two children have higher priority.

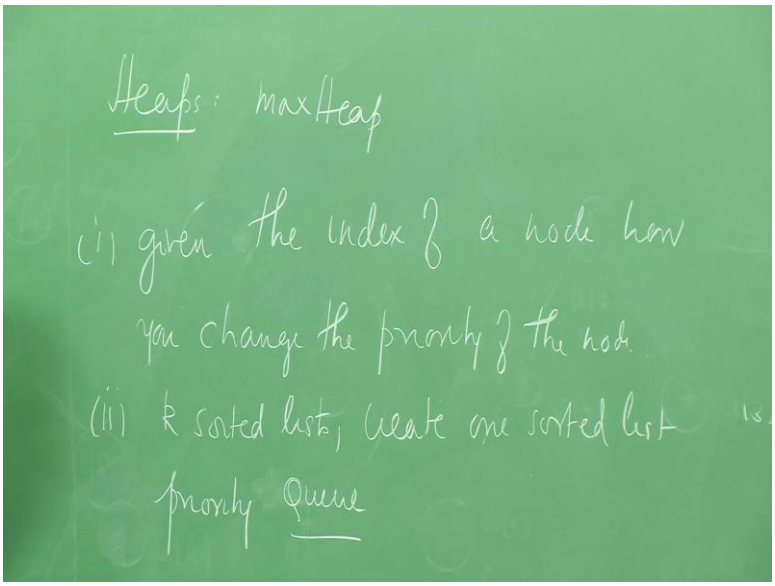
(Refer Slide Time: 16:44)

```
    while ((child <= last) && (!flag)){
      if ((child < last) && (priority[child] < priority[child+1]))
        child = child+1;
      if (temp >= priority[child])
        flag = 1;
      else {
        priority[parent] = priority[child];
        parent = child;
        child = parent*2;
      }
    }
    priority[parent] = temp;
    return(item);
  }

  void Heap::createHeap() {
     last = 0;
```

And then after words what is do, it keeps on finding out the child by taking parent times 2. Because, if the parent is i by 2 which we saw in the insertion case, in the deletion case parent is that child is act parents star to. So, what we do is be essentially, what are we doing over here, we have essentially repeating the same process over here. And we do the deletion of the given node, again percolated let us take the last element from the heap put it to the root and percolated down. So, that is all that is true heaps and what is the interesting is the complexity the of the heap, whether insertion or deletion is only a log n operation.

(Refer Slide Time: 17:50)



Now, I what you to think of some nice problems, let us see I am giving you a heap and let us look at some problems on heaps which I would like you to do thing about this let given a heap that is already, let us call it a max heap. Now, what I want you to do is, I given the index of a node problem 1, how do I change the priority of the node. I want to give you one more problem, let say I am given k sorted list, I want to create one sorted list, and I want use a priority queue to do this, let see how you can do these two problems.