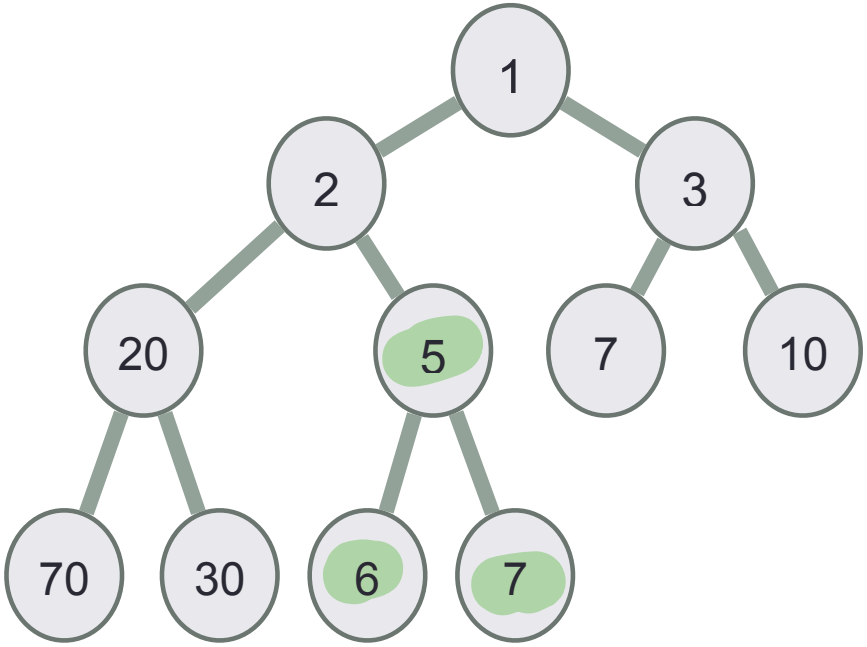
HEAPS &HEAP SORT

Problem Solving with Computers-II

Make a copy of the handout for today's lecture:
https://bit.ly/cs24-lect15-handout

Two important properties of a (min) heap



Shape property:

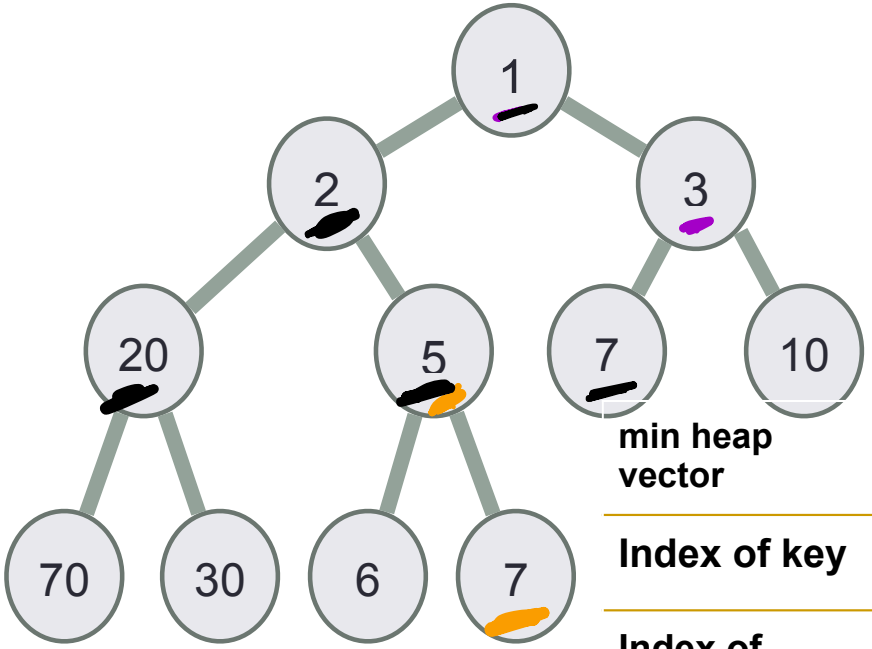
Complete binary tree

Heap property :

For every node x

min heap: key(x) ≤ key(children of x)

Internally the "heap binary tree" is just a vector!



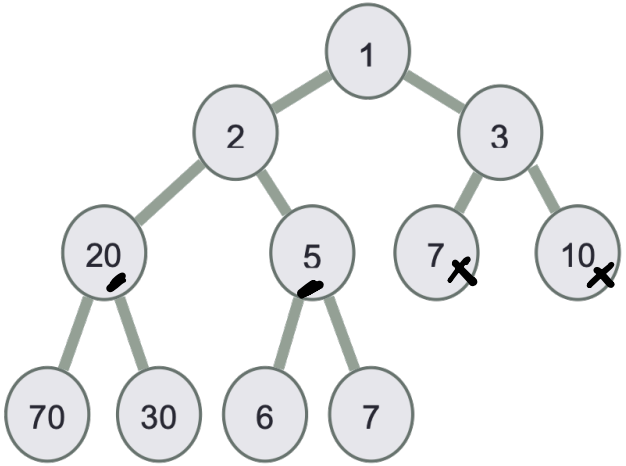
For a key at index i

- index of its parent is $(i-1)/2$
- index of left child is $2i+1$
- index of right child is $2i+2$

Handout activity 1: Complete the table

min heap vector	1	2	3	20	5	7	10	70	30	6	7
Index of key	0	1	2	3	4	5	6	7	8	9	10
Index of parent	-1	0	0	1	1	2	2	3	3	4	4
Index of left child	1	3	5	7	9	11	13				
Index of right child	2	4	6	8	10	12	14				

Internally the “heap binary tree” is really just a vector!



1	2	3	20	5	7	10	70	30	6	7
0	1	2	3	4						

not a leaf node

2

What is the largest index of an internal node in a min-heap with n elements?

A. log n

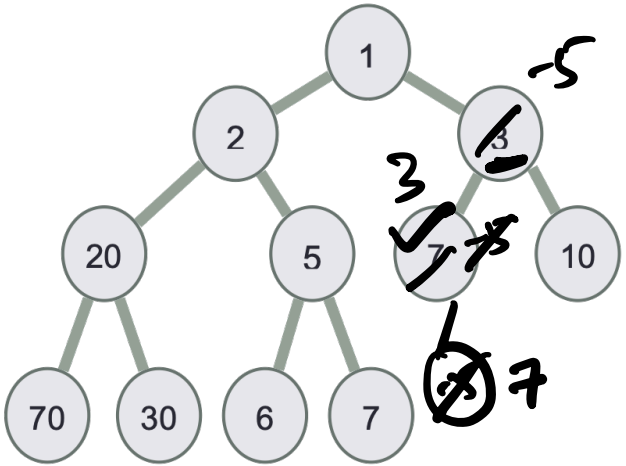
B. (n - 1) /2

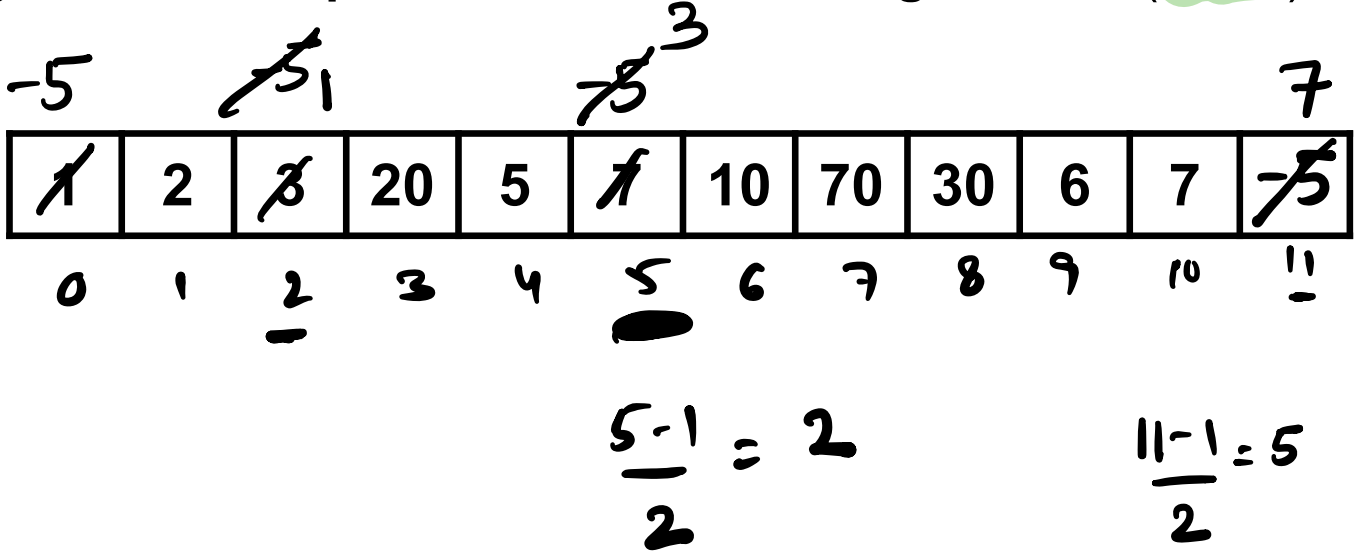
C. n - 1

D. None of the above

The answer is $\left(\frac{n}{2} - 1\right)$

Activity 2: push(-5) into the given min heap and draw the resulting vector (2 min)

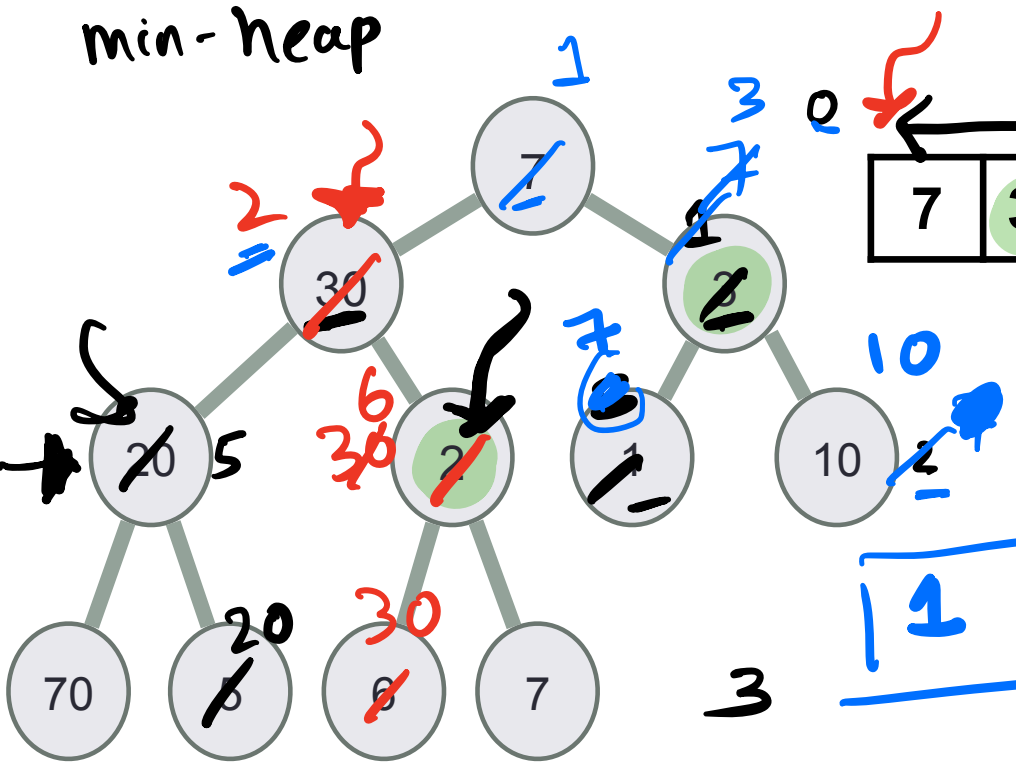




What is the Big-O running time of n push operations?

```
procedure push(x: key value)
  insert x in the first open spot in the tree (Claim O(1), why?)
  while(x has a parent && parent(x) > x):
    swap(x, parent(x))
  return
```

Heapify: A fast way to turn an arbitrary vector to a heap:



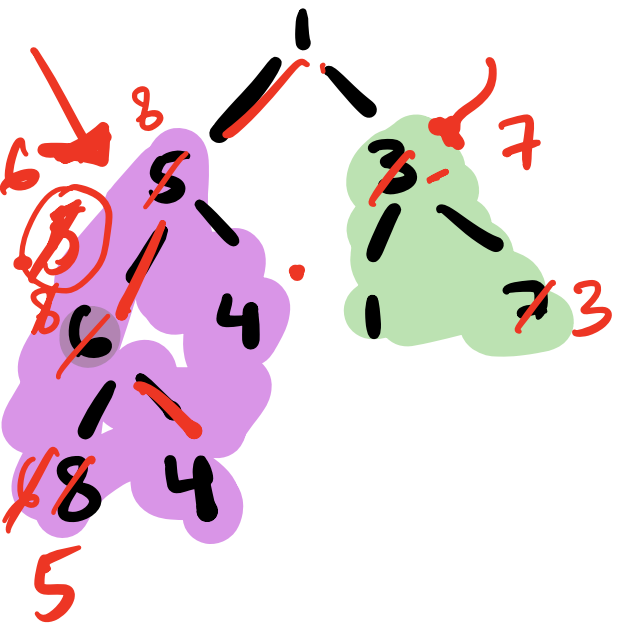
$T(n) = O(n)$

0	1	2	3	4	5	6	7	8	9	10
7	30	3	20	2	1	10	70	5	6	7

1 2 3 5 6 7 10 70 20 30 7

High-level approach: Starting from the level containing the last internal node and moving upwards through all the internal nodes, sift the root of each subtree downward as in the **bubble-down process** until the **heap property** is restored.

Activity 3: Heapify the vector to turn it into a max-heap (3 min)



Level 0

Level 1

1	5	3	6	4	1	7	8	4

Level 2

Level 3

What is the resulting vector?

A. 8 6 7 5 4 1 3 1 4

B. 8 1 7 5 4 1 3 6 4

C. 1 4 1 5 4 3 7 8 6

D. Something else

The number of swaps to get key 6 in the right position

The number of swaps to get the root node (key 1) in the right position 3

Activity 4: Running time of heapify (2 min)

1	5	3	6	4	1	7	8	4

Let $T(n)$ be the running time of heapify for an input vector of size n

Let h be the height of the binary tree represented by the vector.

The number of nodes is: 9
The height of the tree is: 3
The maximum number of swaps to restore nodes at each level
Level 3 : 0
Level 2 : 1
Level 1 : 2
Level 0 (root): 3
Generalize your answers to a tree of height h

Note: The max. number of swaps to restore the heap property for a node at level l is $(h-l)$

$$T(n) \leq \sum_{l=0}^{h-1} 2^l \cdot (h-l)$$

Let $j = h-l$

$l = 0$, $j = h$

$l = h-1$, $j = 1$

$$= \sum_{j=1}^{h} 2^{h-j} \cdot j$$

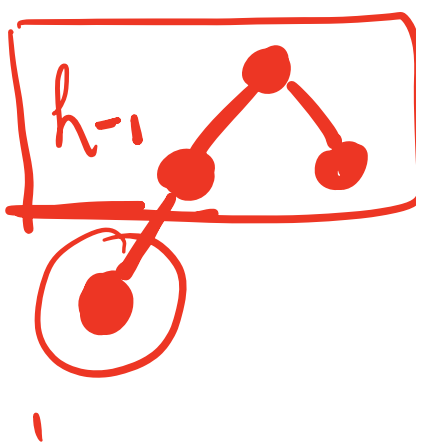
$$= \sum_{j=0}^{h} 2^{h-j} \cdot j$$

$$= 2^h \sum_{j=0}^{h} \frac{j}{2^j}$$

converges to 2

$$= 2^h \cdot 2$$

$$= 2^{h+1}$$



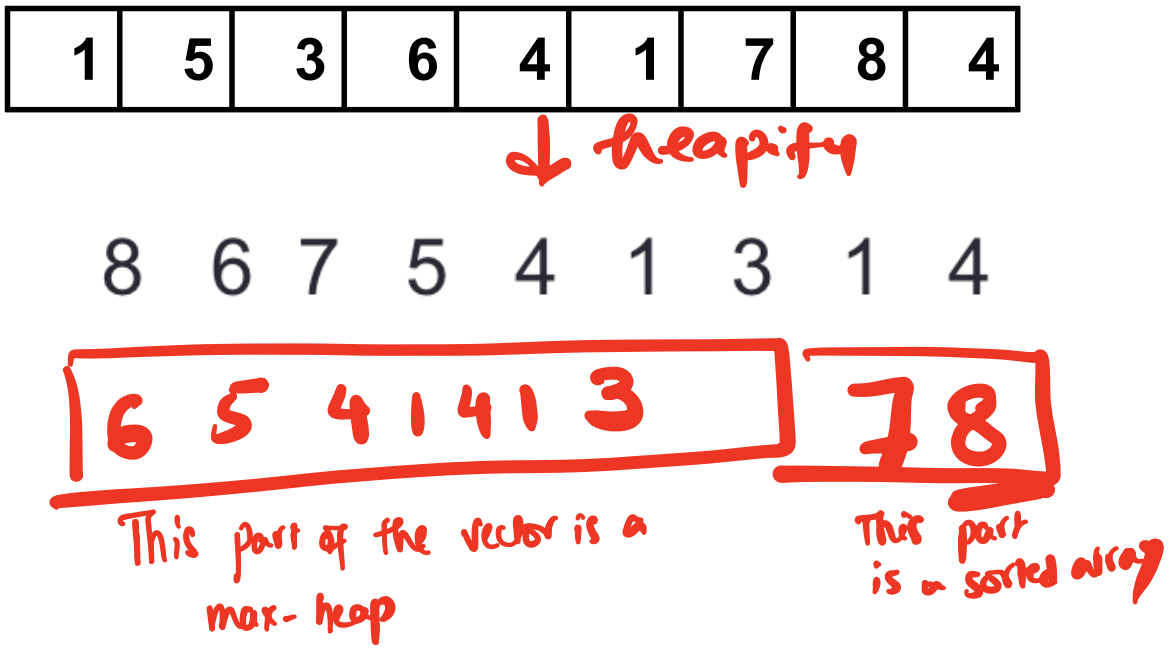
We observe that $2^h \leq n \leq 2^{h+1} - 1$

Therefore, $2^{h+1} \leq 2n$

Therefore, $T(n) \leq 2n$

$T(n) = O(n)$

Heap Sort Algorithm



- Heapify the input vector
- At this point, the maximum element is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of the heap by 1. Finally, heapify the root of the tree.
- Repeat step 2 while the size of the heap is greater than 1.

std::priority_queue template arguments

```
template <
      class T,
      class Container= vector<T>,
      class Compare = less <T>
      > class priority_queue;
```

The template for priority_queue takes 3 arguments:

1. Type elements contained in the queue.
2. Container class used as the internal store for the priority_queue, the default is **vector<T>**
3. Class that provides priority comparisons, the default is **less**

Comparison class: A class for comparing objects

```
class myCompare{
        bool operator()(int& a, int & b) const {
                return a > b;
        }
};

int main(){
    myCompare cmp;
    cout<<cmp(20, 10)<<endl;
}
```

If cmp(x, y) returns true, priority queue will interpret this as:

x has lower priority than y

Which element will be at the top of such a priority queue?

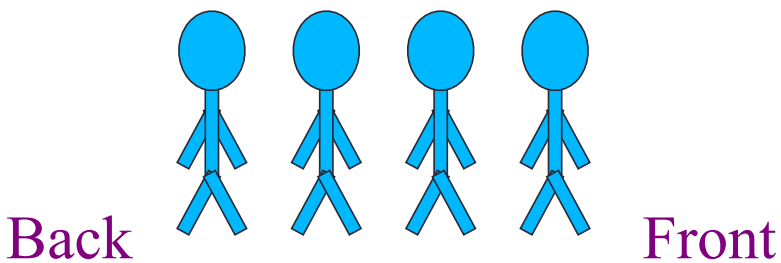
std::priority_queue template arguments

```
//Template parameters for a max-heap
priority_queue<int, vector<int>, std::less<int>> pq;

//Template parameters for a min-heap
priority_queue<int, vector<int>, std::greater<int>> pq;
```

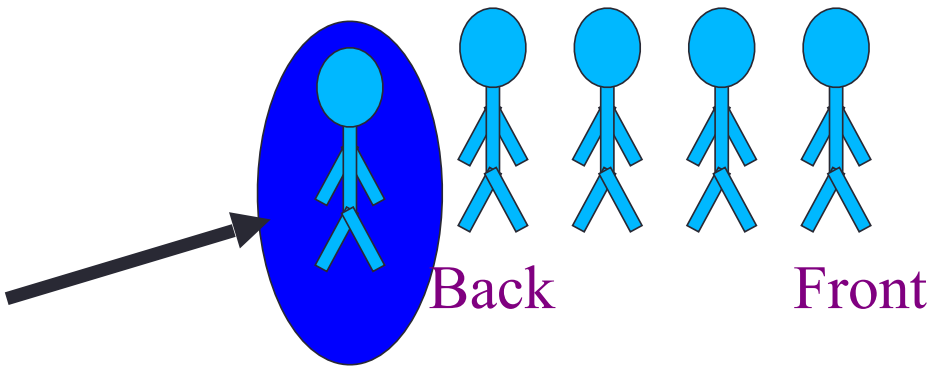
Queue Operations

- A queue is like a queue of people waiting to be serviced
- The queue has a **front** and a **back**.



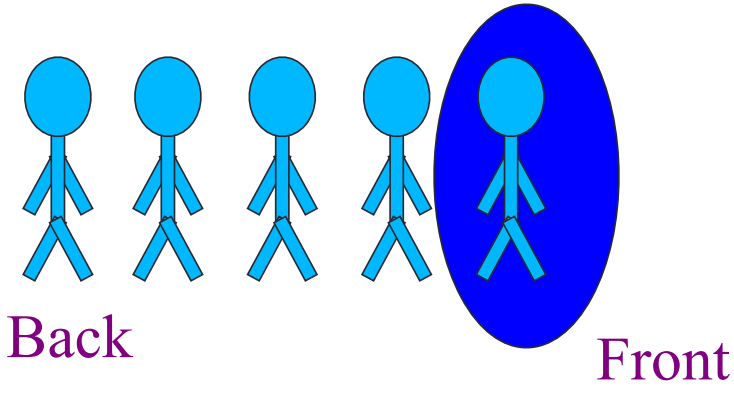
Queue Operations

- New people must enter the queue at the back. The C++ queue class calls this a **push**, although it is usually called an **enqueue** operation.



Queue Operations

- When an item is taken from the queue, it always comes from the front. The C++ queue calls this a **pop**, although it is usually called a **dequeue** operation.

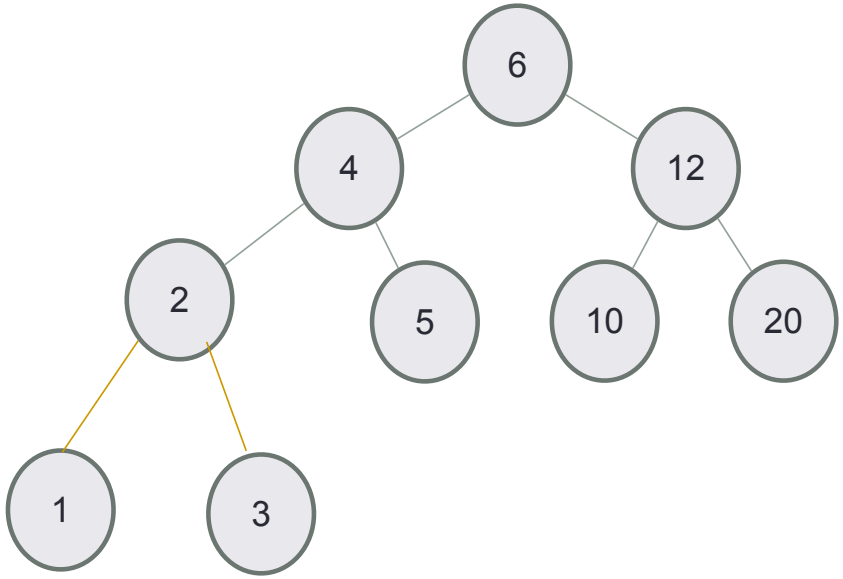


Queue class

- The C++ standard template library has a queue template class.
- The template parameter is the type of the items that can be put in the queue.

```
template <class Item>
class queue<Item>
{
public:
        queue( );
        void push(const Item& entry);
        void pop( );
        bool empty( ) const;
        Item front( ) const;
        …
```

Breadth first traversal



- Take an empty Queue.
- Start from the root, insert the root into the Queue.
- Now while Queue is not empty,
  - Extract the node from the Queue and insert all its children into the Queue.
  - Print the extracted node.