HEAPS &HEAP SORT

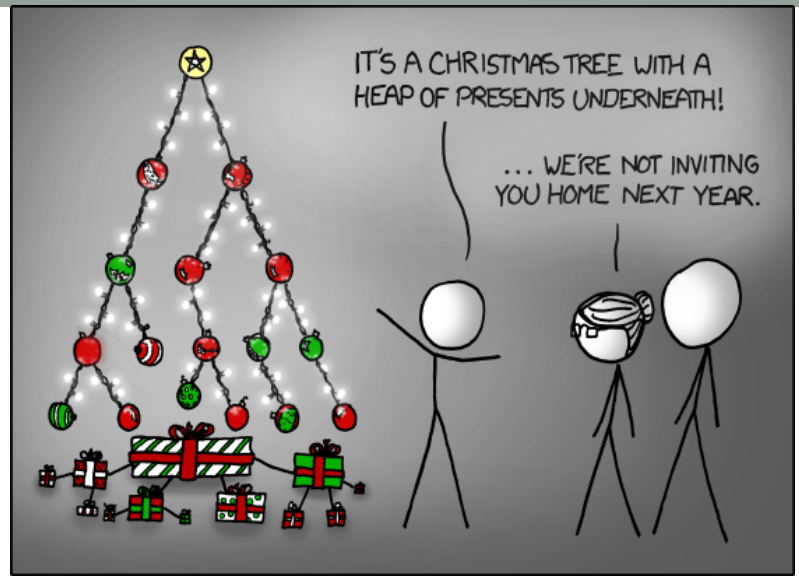
Problem Solving with Computers-II

Make a copy of the handout for today's lecture:
https://bit.ly/CS24-Heaps-lect15

Review: Heap or priority_queue

- What are the operations supported?
- What are the running times?

push O(log n)
pop O(log n)
top O(1)




```
//Declare a (max) heap
priority_queue<int> pq;
```

Note: This lecture we used Visual so.net to understand the heap operations & the heapify algo

Application — sorting

```
void selection_sort(vector<int>& v){
    int n = v.size();
    for (int i = 0; i < n; i++){
        int index = i;
        for (int j = i + 1; j < n; j++){
            if(v[j] > v[index]){
                index = j;
            }
        }
        if(index != i){
            int temp = v[index];
            v[index] =  v[i];
            v[i] = temp;
        }
    }
}
```

Running time: O(n^2)

Space complexity: O(1)

Can we do better?

Application — simple heap sort

```
void simple_heap_sort(vector<int>& v){
    priority_queue<int> pq;
    for(auto& elem : v){
        pq.push(elem);
    }
    int i = 0;
    while(!pq.empty()){
        v[i] = pq.top();
        pq.pop();
        i++;
    }
}
```

O(1)

n x O(logn) = O(nlogn)

+

O(1)

= n x (O(1) + O(logn) + O(1))

= n * O(logn) = O(nlogn)

O(logn)

O(1)

Running time: O(n logn)

Space complexity: O(n)

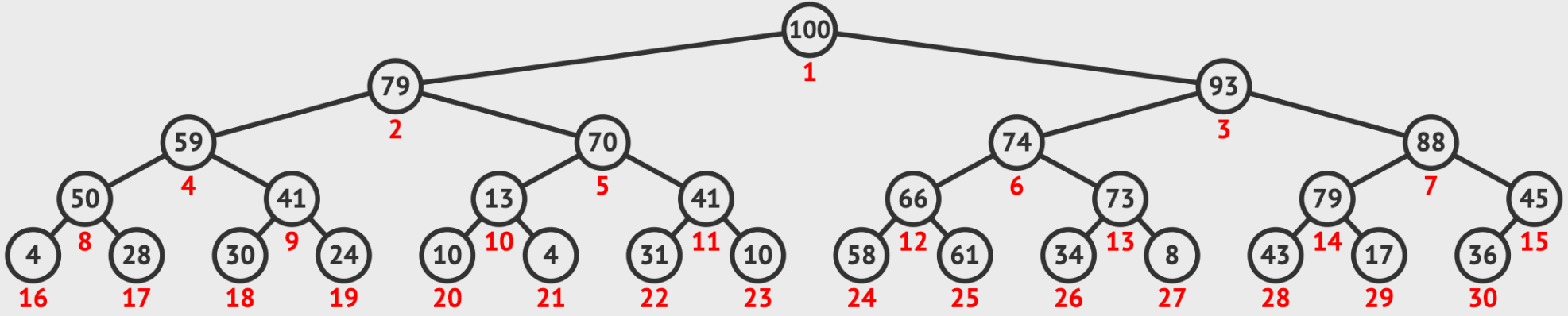
because we used a priority-que

Can we do better? Yes we can improve the space complexity.

Review: Two important properties of a heap

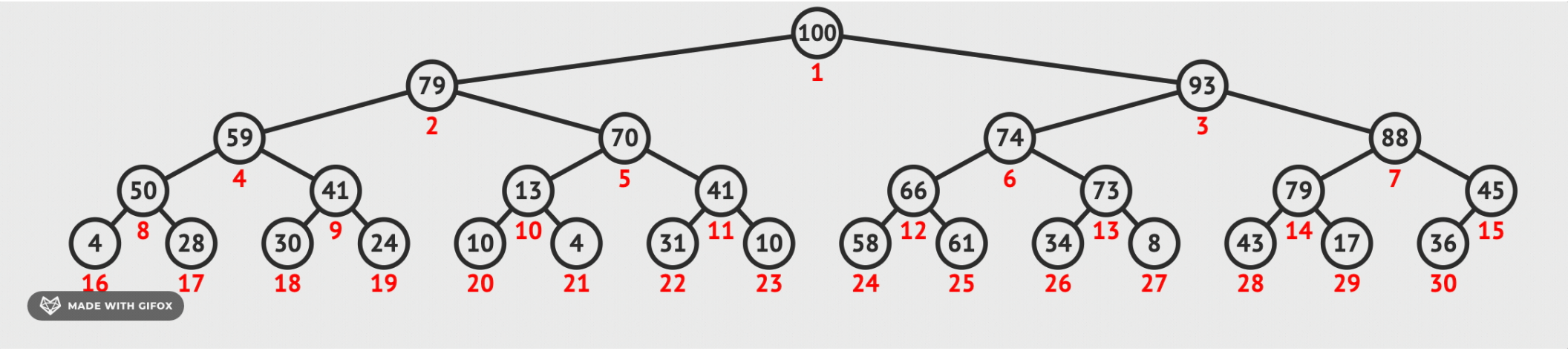
Shape property: Complete binary tree

Heap property : For max heap.
For each node x, key(x) ≥ key(children(x))



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

Activity 1 (5 min): Observe the animation, then answer the following questions



Complete the entries in the vector representation of the binary heap

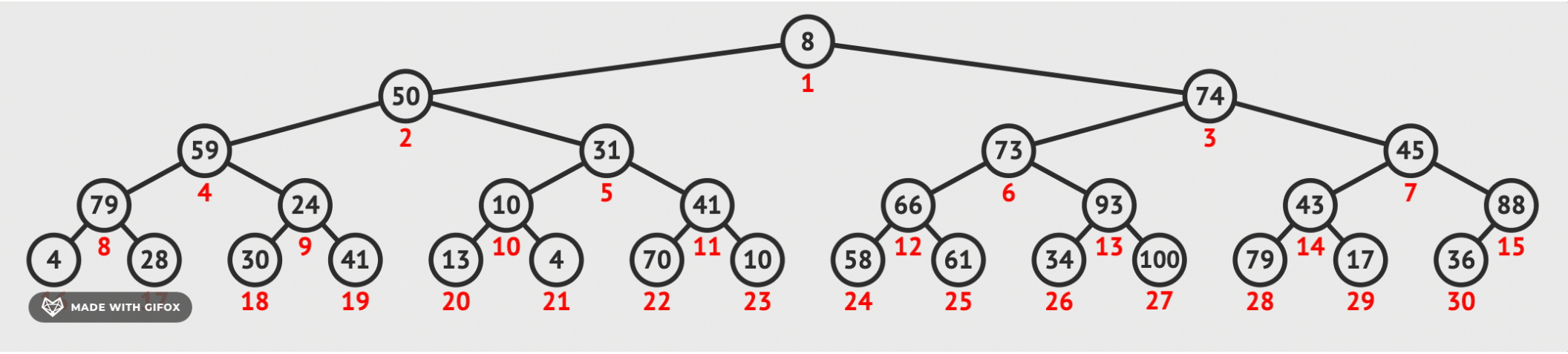
100	79	93	59	70	74	88	50	41	13	41	66	73	79	45	4	28	30	24	10	4	31	10	58	61	34	8	43	17	36	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

For a key at index i in the vector (assume indices start at 0), what is:

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

How do we know if a key at index i has a left child? $2i+1 <$ size of vector

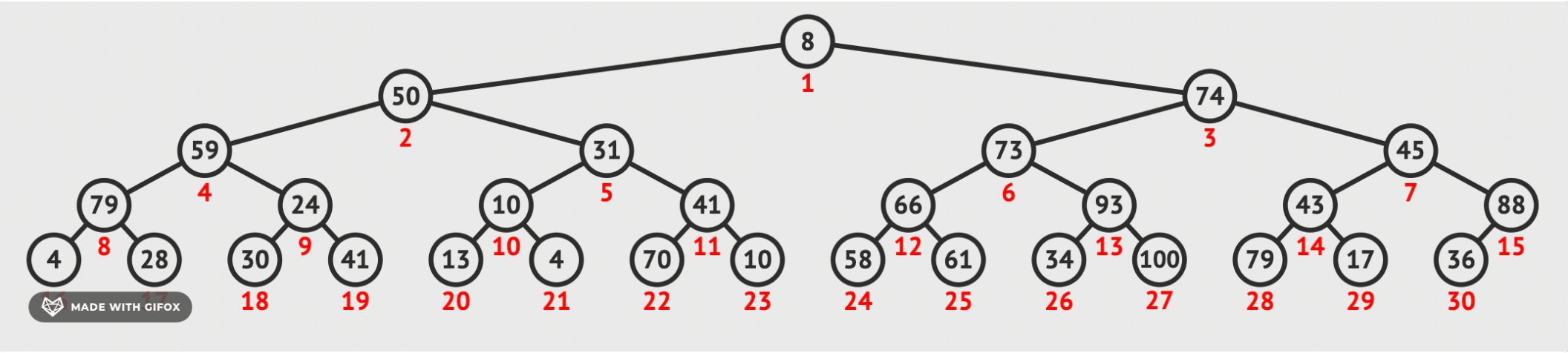
Heapify: A fast way to turn an arbitrary vector into a heap



Activity 2 (5 mins): Observe the visualization of heapify, then describe the algorithm in your own words

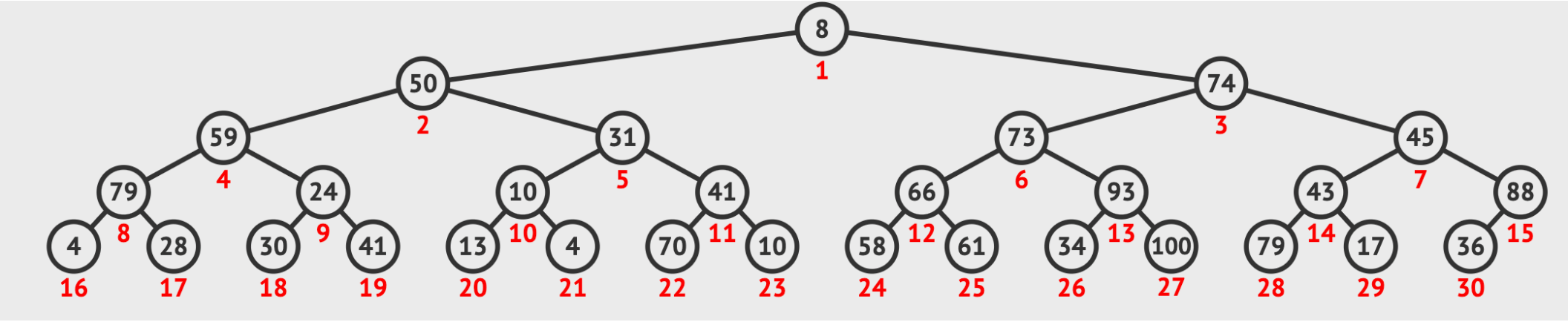
To recreate the visualization, go to: https://visualgo.net/en/heap

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



High-level approach: Given an arbitrary vector of keys. Starting from the internal node with the largest index in the vector, and moving upwards in the tree through all the internal nodes (level by level), sift the root of each subtree downward as in the **bubble-down process** until the **heap property** is restored.

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



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

A. log n
B. (n - 1) /2
C. n - 1
D. n/2 - 1
E. None of the above

Heapify the vector below to convert it into a max-heap (3 min)

1	5	3	6	4	1	7	8	4

What is the resulting vector?

A. 8 7 6 5 4 4 3 1 1

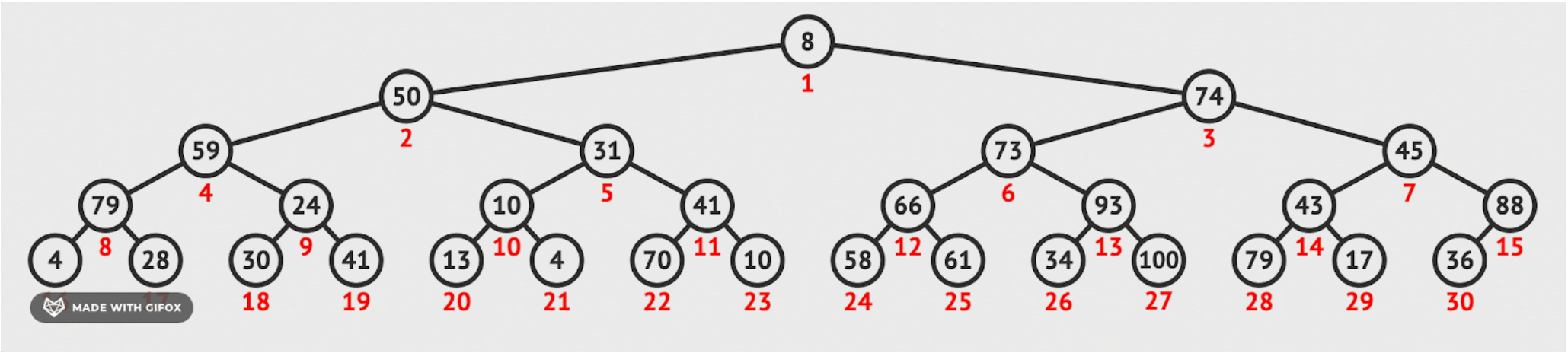
B. 8 1 7 5 4 1 3 6 4

C. 8 6 7 5 4 1 3 1 4

D. Something else

Refer to the in class handout on how you can use the visualization tool we have been working with to arrive at the answer.

Activity 3: Running time of heapify (10 min)



$T(n)$: Running time of heapify

$T(n) \propto$ Total no. of swaps. Let n be no. of keys
h : height of tree

In the heapify algo, all nodes at level ℓ, need $(h-\ell)$ swaps

$$T(n) = c \sum_{\ell=0}^{h-1} 2^{\ell} (h-\ell)$$

Let $j = (h-\ell)$

when $\ell = 0$, $j = h$

$\ell = h-1$, $j = 1$

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

$$= 2^h \sum_{j=1}^{h} j/2^j$$

$$\leq 2^h \sum_{j=1}^{\infty} j/2^j$$

$$\leq 2^h \cdot c$$

$$= c \cdot 2^h$$

$$T(n) = c \cdot 2^h \qquad 2^h < n$$

$$\leq c \cdot n$$

$$= O(n)$$

Heap Sort Algorithm

1	5	3	6	4	1	7	8	4

(See code written in next lecture)

- Step 1: Heapify the input vector with n keys
- Step 2: Let S be the number of keys in the heap. Extract the max element (root key) by swapping it with the last key in the vector. Reduce the size of the heap by 1. At this point, the first (S - 1) keys in the vector represent the heap and the remaining are the sorted portion of the vector Finally, restore the heap property of the root using the bubble down process
- 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 _____ 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;
```