**Introduction**

In computer science, a heap is a specialized tree-based data structure which is essentially an almost completetree that satisfies the heap properties. Heaps are used in many famous algorithms such as Dijkstra’s algorithm for finding the shortest path, the heap sort sorting algorithm, and more. Essentially, heaps are the data structure you want to use when you want to be able to access the maximum or minimum element very quickly.

The heap is one maximally efficient implementation of an abstract data type called a priority queue, and in fact, priority queues are often referred to as "heaps", regardless of how they may be implemented. In a heap, the highest (or lowest) priority element is always stored at the root. However, a heap is not a sorted structure; it can be regarded as being partially ordered. A heap is a useful data structure when it is necessary to repeatedly remove the object with the highest (or lowest) priority. There are many different types of heaps each having specialized operations or optimized way for performing certain operations. Examples of heap: binary heap, pairing heap, Fibonacci heap, leftish heap, treap, k-ary heap, weak heap, radix heap, and many more.

In this project, we analysed and compared the working of Fibonacci heap, treap and k-ary heap. Starting with the description on above mentioned heaps, we have attached detailed algorithm for operations like insertion, deletion, finding minimum, search and extracting min performed on those heaps. Also, we have done the performance analysis for all the above-mentioned operations.

**Fibonacci heap**

**Description**

Binomial heap, Fibonacci Heap is a collection of trees with min-heap or max-heap property. In Fibonacci Heap, trees can have any shape even all trees can be single nodes (This is unlike Binomial Heap where every tree must be Binomial Tree).

Fibonacci Heap maintains a pointer to minimum value (which is root of a tree). All tree roots are connected using circular doubly linked list, so all of them can be accessed using single ‘min’ pointer. The main idea is to execute operations in easier way. For example, merge operation simply links two heaps, insert operation simply adds a new tree with single node. But the extract minimum operation is the most complicated operation. It does delayed work of consolidating trees. This makes delete also complicated as delete first decreases key to minus infinite, then calls extract minimum.

Fibonacci heap are mainly called so because Fibonacci numbers are used in the running time analysis. Also, every node in Fibonacci Heap has degree at most O(log n) and the size of a subtree rooted in a node of degree k is at least Fk+2, where Fk is the kth Fibonacci number.

Fibonacci heaps are similar to [binomial heaps](https://brilliant.org/wiki/binomial-heap/) but Fibonacci heaps have a less rigid structure. Fibonacci heaps have a faster [amortized](https://brilliant.org/wiki/amortized-analysis/) running time than other heap types. Although Fibonacci Heap looks promising time complexity wise, it has been found slow in practice as hidden constants are high.

**Applications**

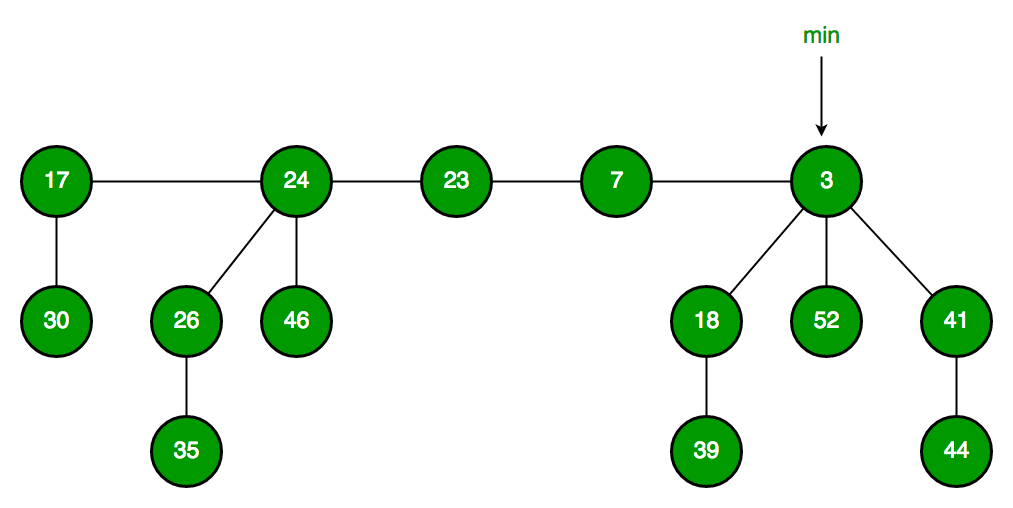
Fibonacci heaps are used to implement the [priority queue](https://brilliant.org/wiki/priority-queues/) element in [Dijkstra’s algorithm](https://brilliant.org/wiki/dijkstras-short-path-finder/) and Prim’s algorithm. With Binary Heap, time complexity of these algorithms is O(VLogV + ELogV). If Fibonacci Heap is used, then time complexity is improved to O(VLogV + E).

**Limitations**

Fibonacci heaps have a reputation for being slow in practice due to large memory consumption per node and high constant factors on all operations. Recent experimental results suggest that Fibonacci heaps are more efficient in practice than most of its later derivatives, including quake heaps, violation heaps, strict Fibonacci heaps, rank pairing heaps, but less efficient than either pairing heaps or array-based heaps.

**Example**

Fibonacci heap :



**Operations**

**1.Insertion**

Algorithm :

i) Create a new node ‘x’.

ii) Check whether heap H is empty or not.

iii) If H is empty, then:

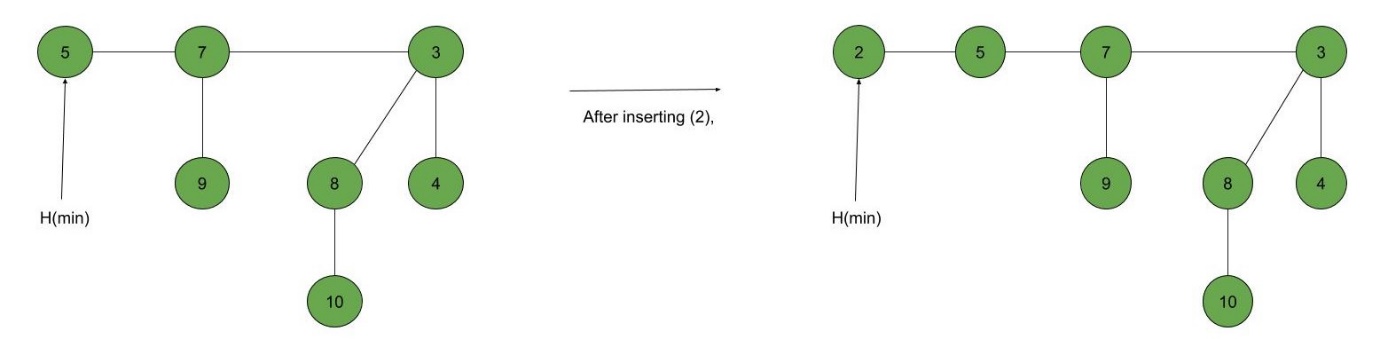
Make x as the only node in the root list.

Set H(min) pointer to x.

Else:

Insert x into root list and update H(min)*.*

Example :



**2.Extract min**

Algorithm :

i) Delete the min node.

ii) Set head to the next min node and add all the trees of the deleted node in the root list.

iii) Create an array of degree pointers of the size of the deleted node.

iv) Set degree pointer to the current node.

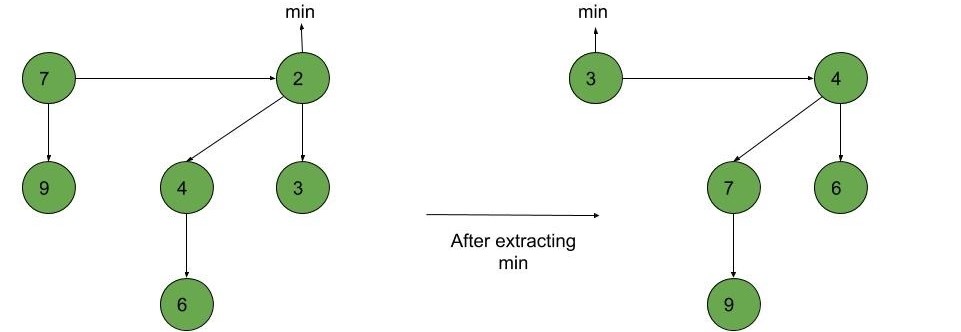
v) Move to the next node.

If degrees are different, then set degree pointer to next node.

If degrees are the same, then join the Fibonacci trees by union operation.

vi) Repeat steps 4 and 5 until the heap is completed.

Example :



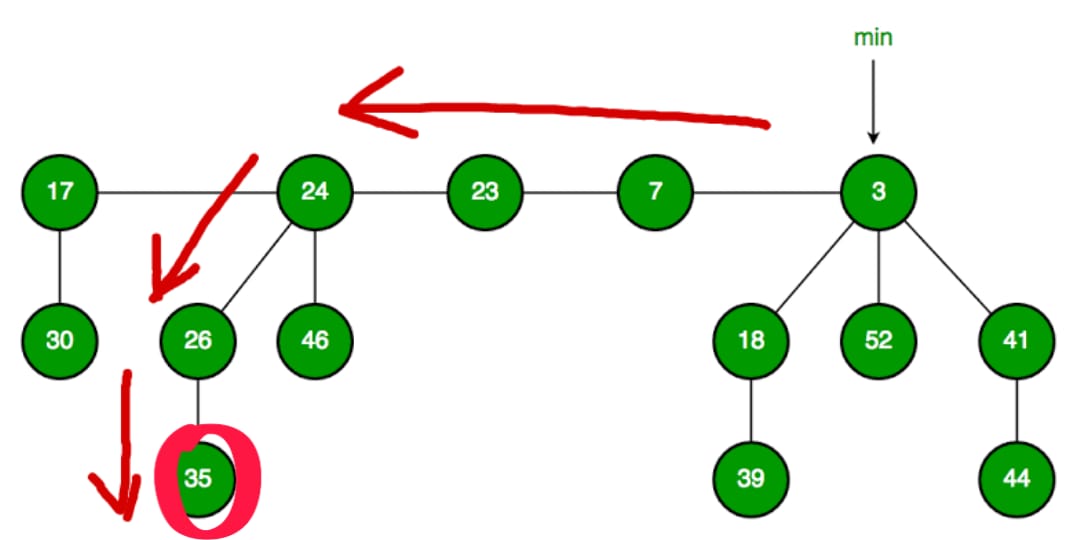
**3.Search**

Algorithm :

i) Staring from the node pointer by min pointer, traverse along the root list and searches the element.

ii) If a node on the root list has children, then traverse on the children list and search.

Example : Searching 35



**4.Deletion**

Algorithm :

i) Decrease the value of the node to be deleted ‘x’ to a minimum by using decrement operation.

Decrement operation :

a) Decrease the value of the node ‘x’ to the new chosen value.

b) CASE 1)

If min-heap property is not violated, Update min pointer if necessary.

CASE 2)

If min-heap property is violated and parent of ‘x’ is unmarked, Cut off the link between ‘x’ and its parent.

Mark the parent of ‘x’.

Add tree rooted at ‘x’ to the root list and update min pointer if necessary.

CASE 3)

If min-heap property is violated and parent of ‘x’ is marked,

Cut off the link between ‘x’ and its parent p[x].

Add ‘x’ to the root list, updating min pointer if necessary.

Cut off link between p[x] and p[p[x]].

Add p[x] to the root list, updating min pointer if necessary.

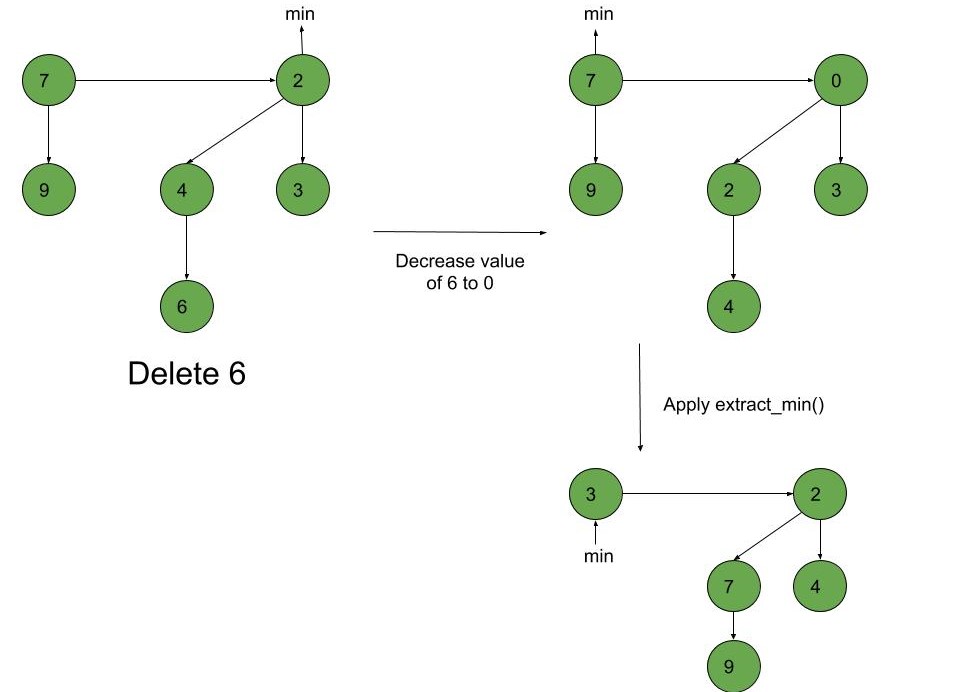
If p[p[x]] is unmarked, mark it.

Else, cut off p[p[x]] and repeat steps 4.2 to 4.5, taking p[p[x]] as ‘x’.

ii) By using min-heap property, heapify the heap containing ‘x’, bringing ‘x’ to the root list.

Iii) Now, apply Extract min algorithm to the Fibonacci heap.

Example :

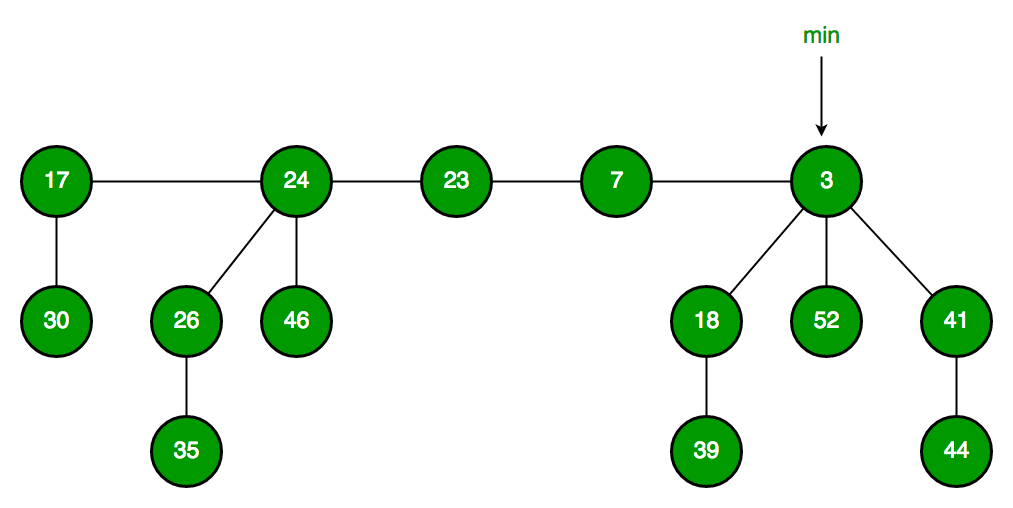


**5.Find min**

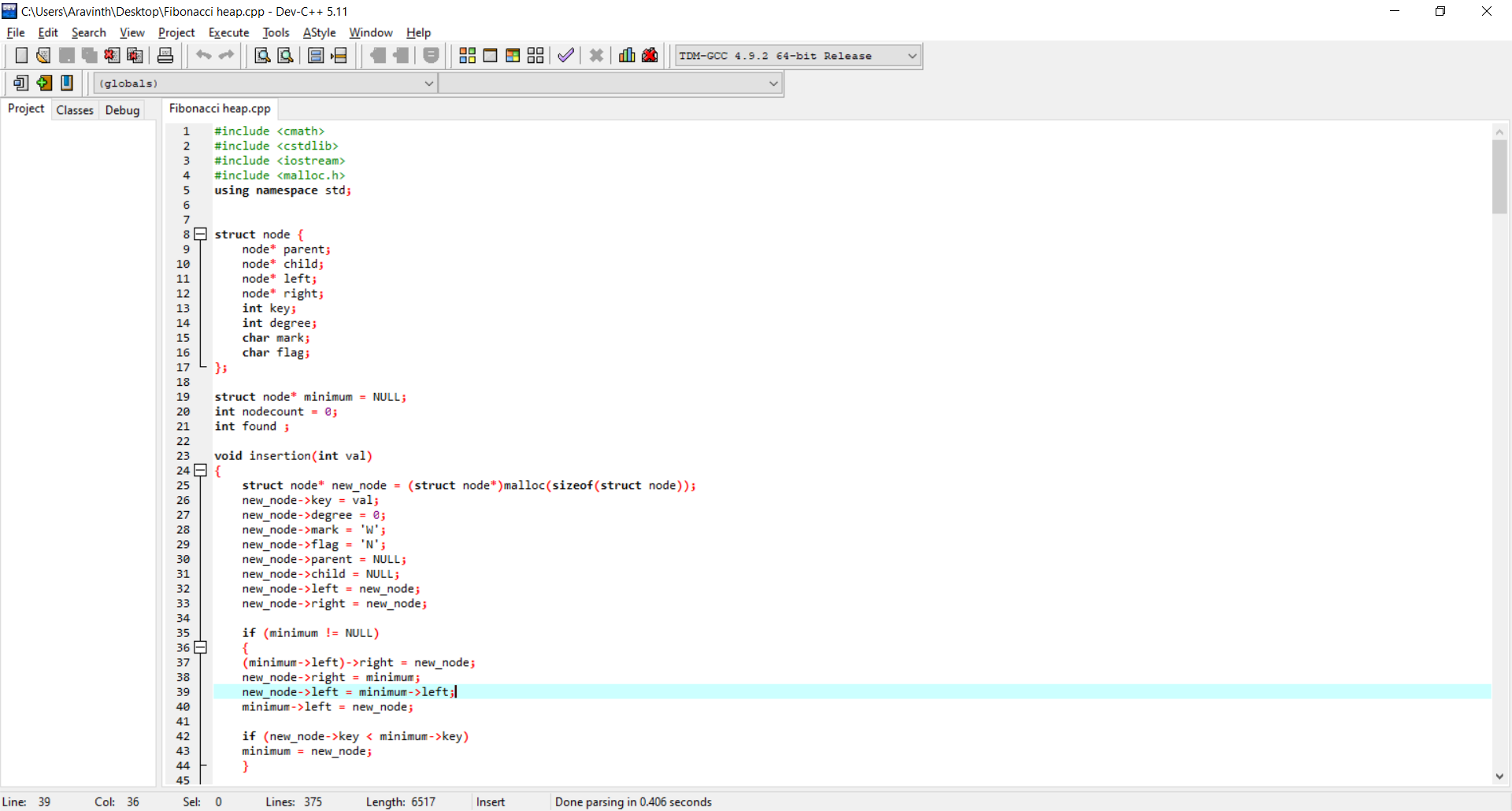
Algorithm :

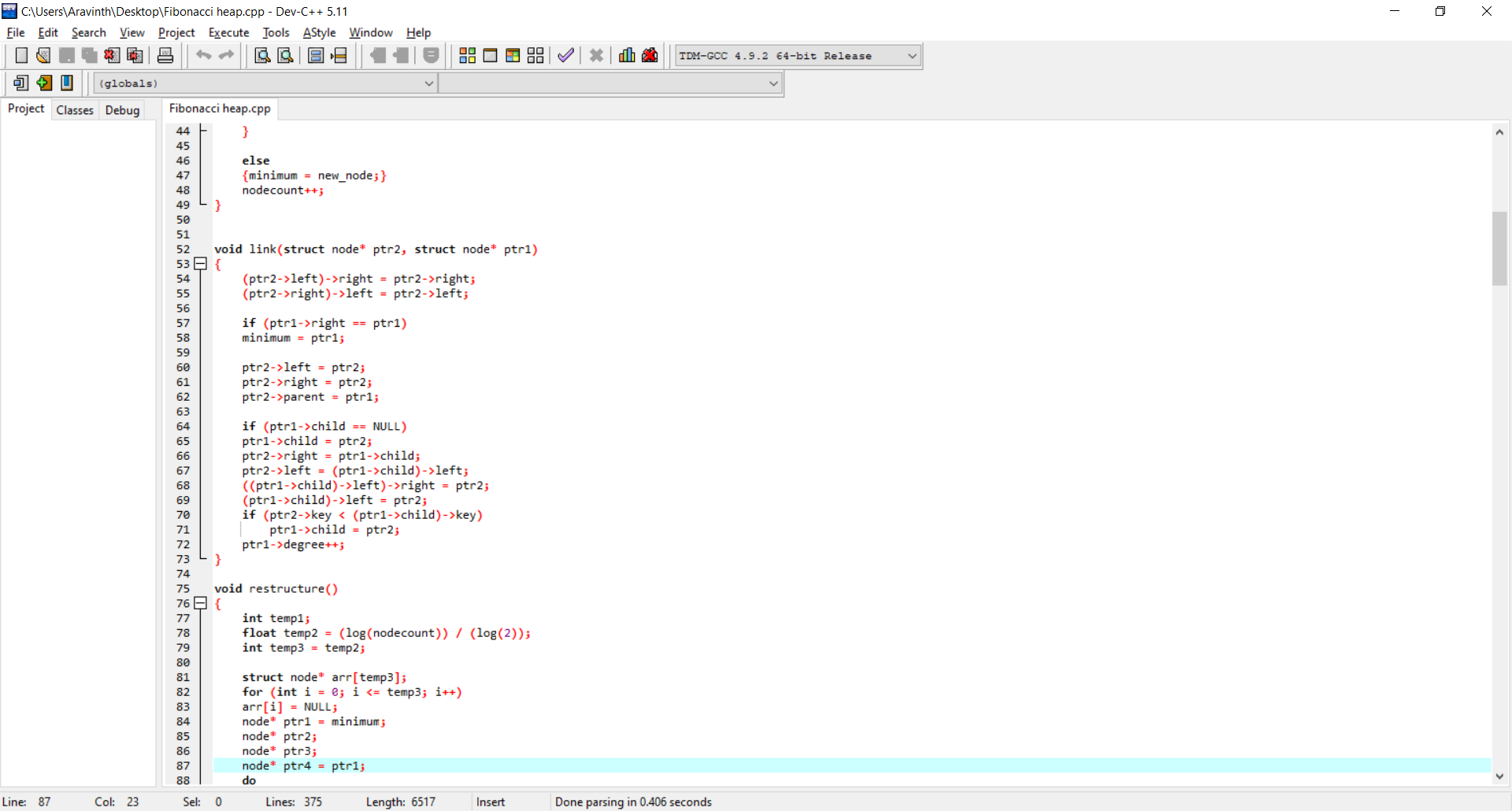
i) As the min pointer points to the minimum valued node, just return the value present in the node.

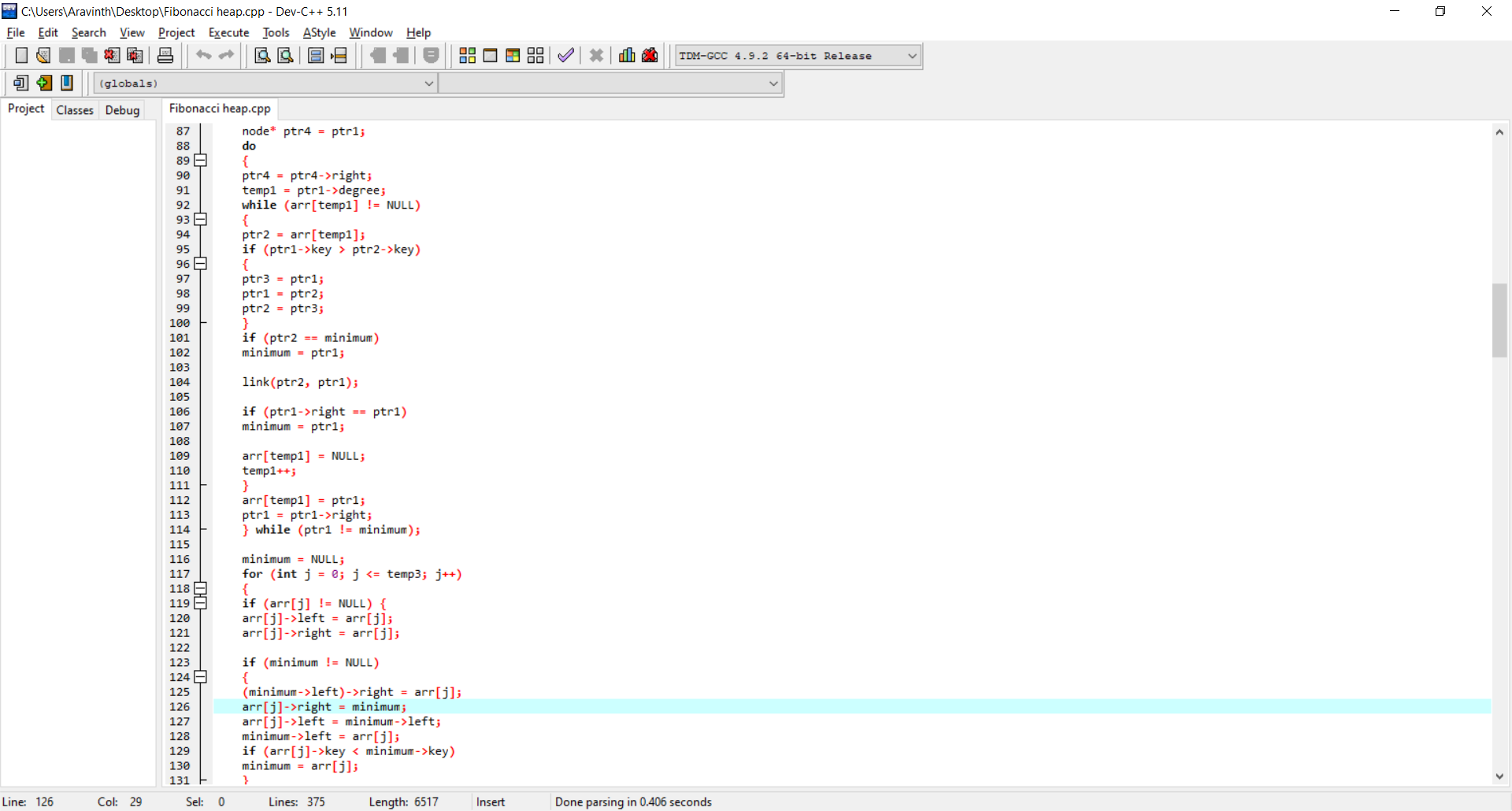
Example :

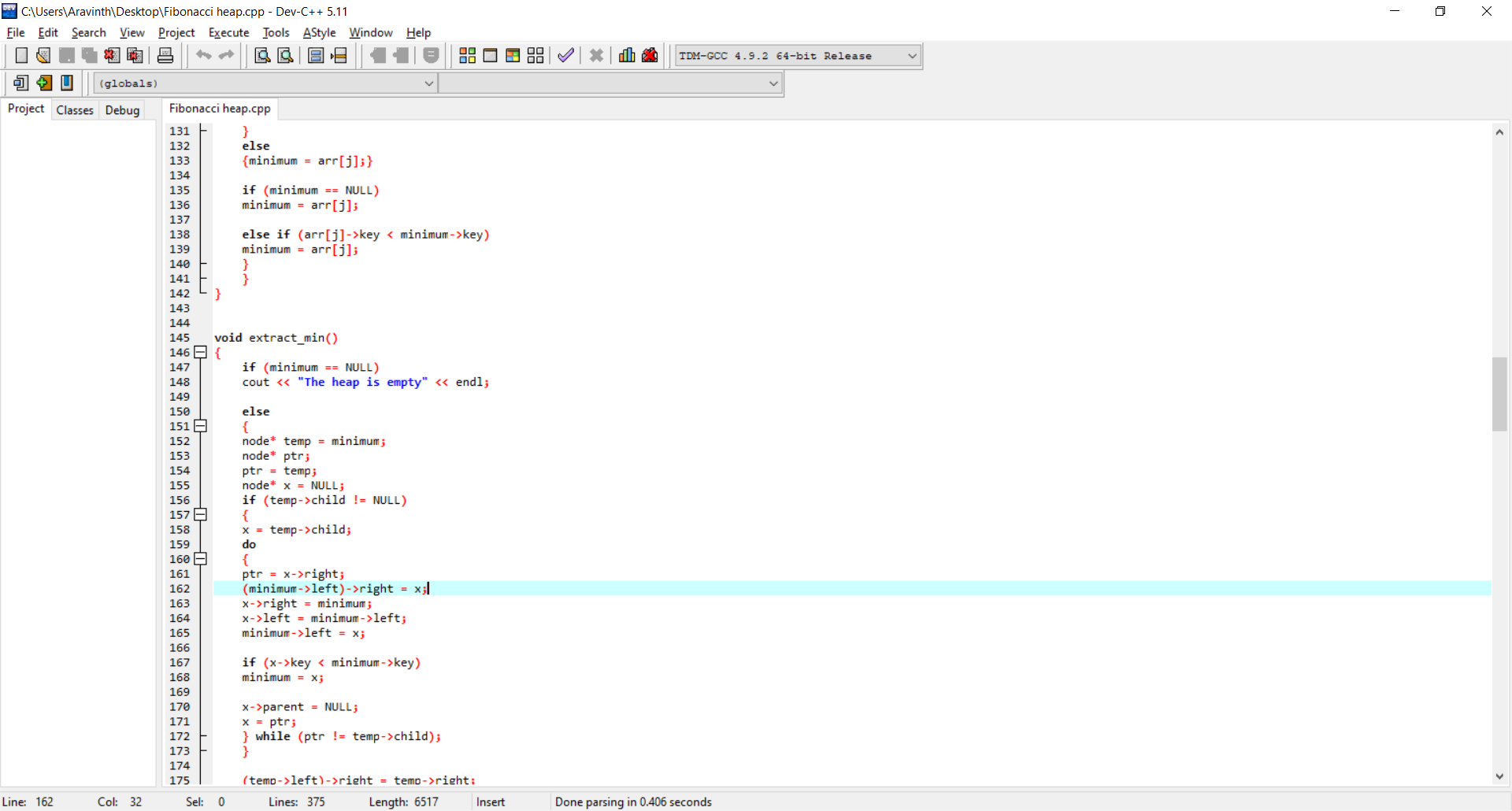


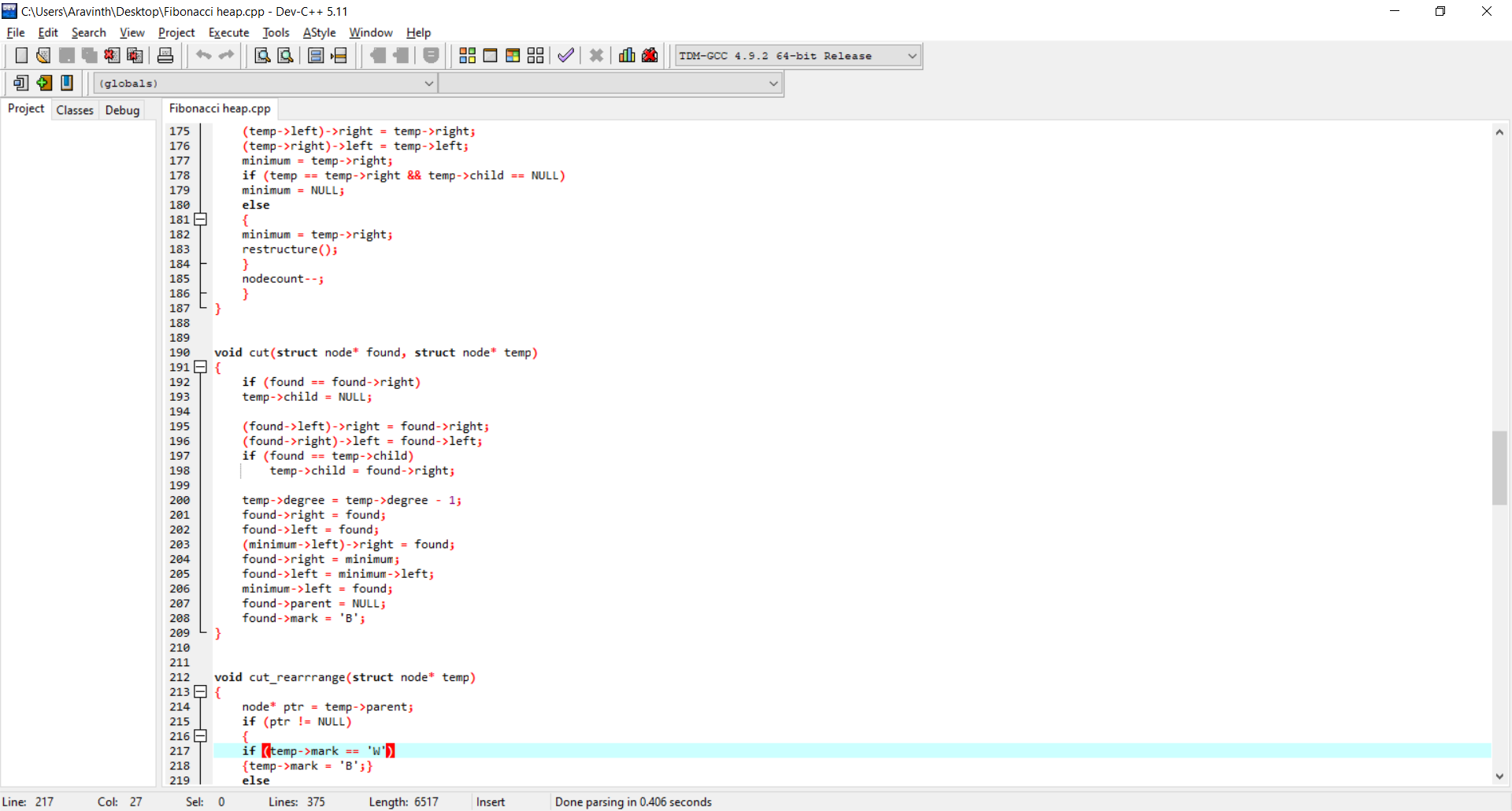
Code:

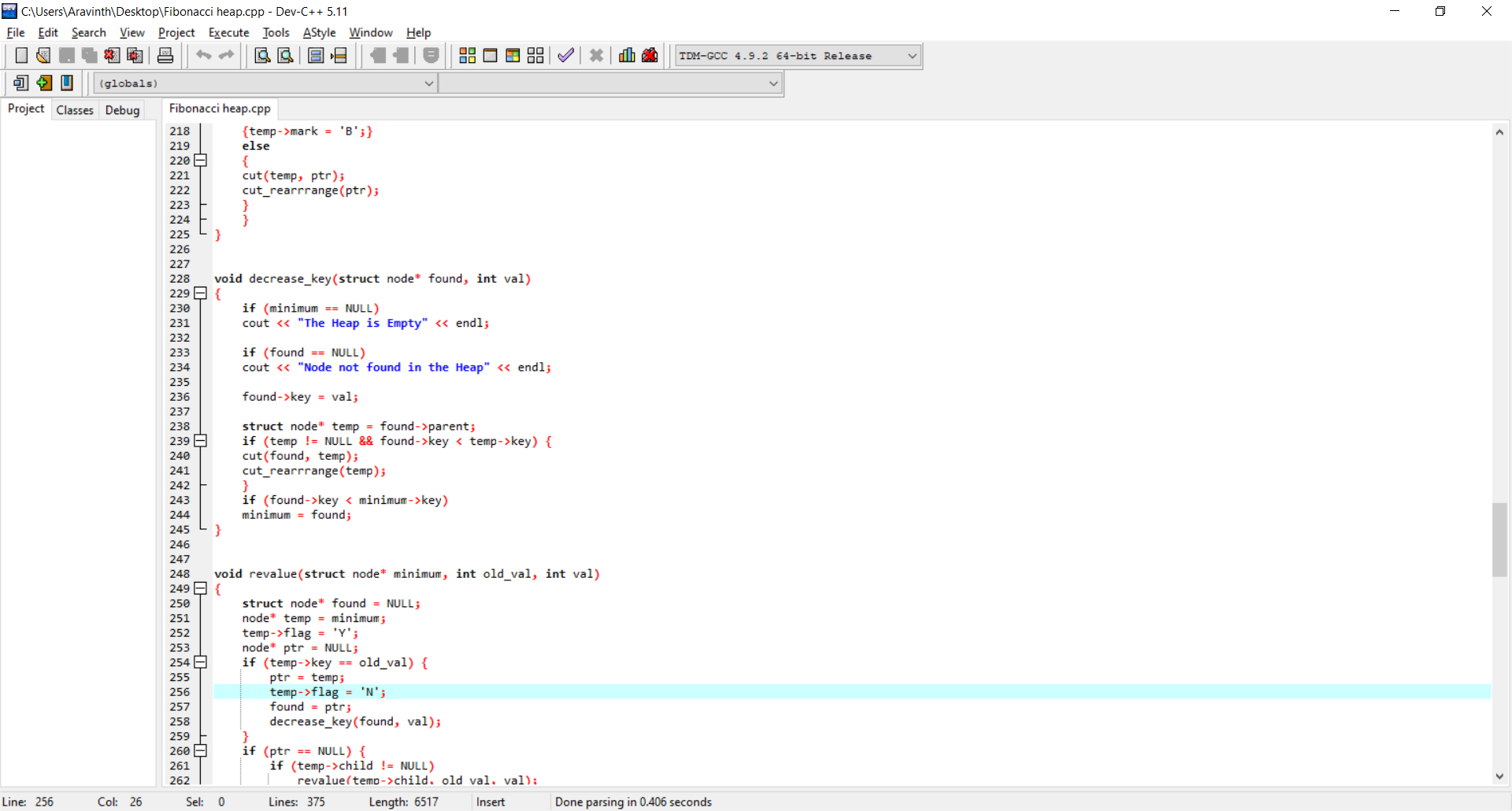


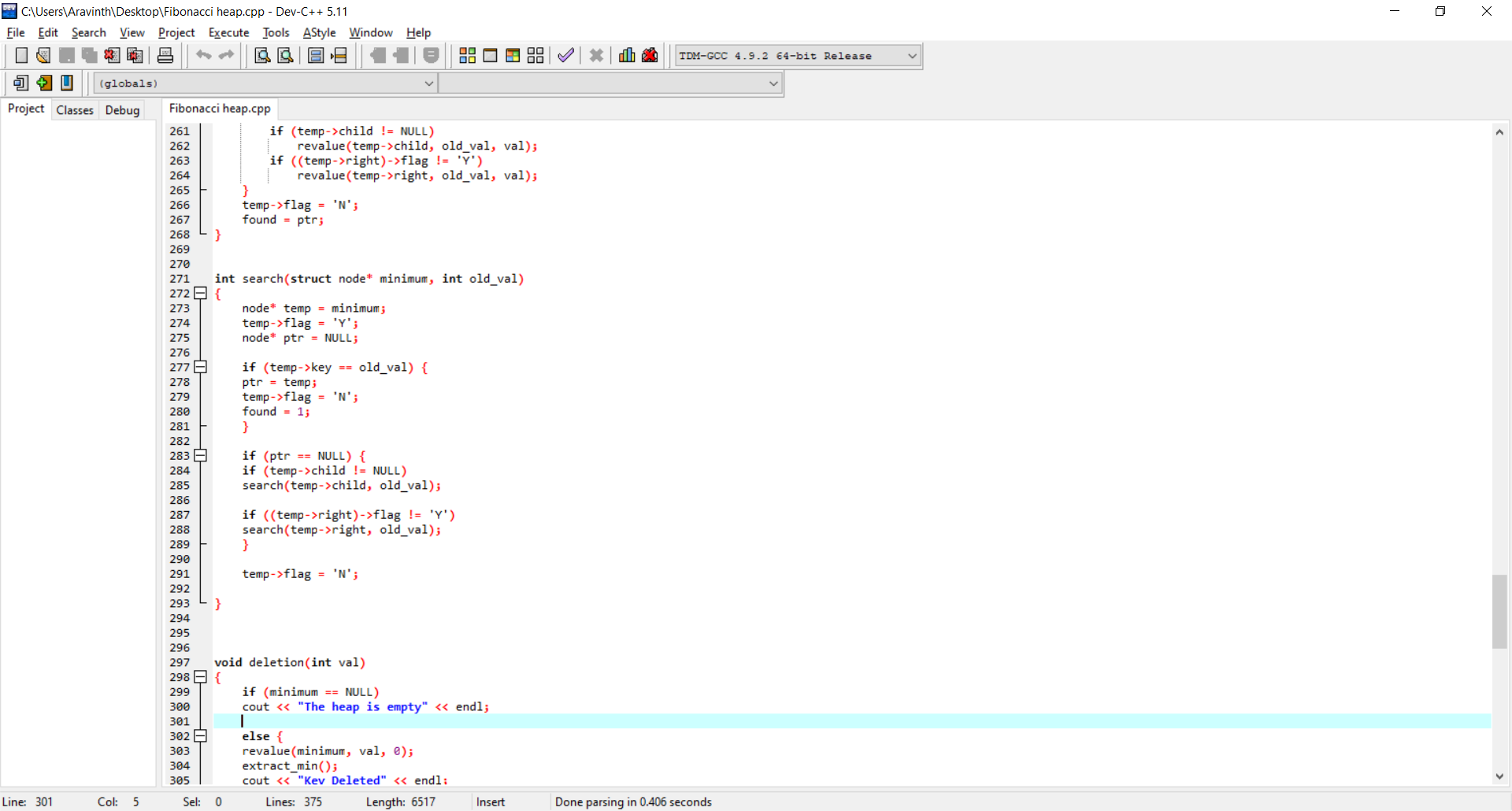


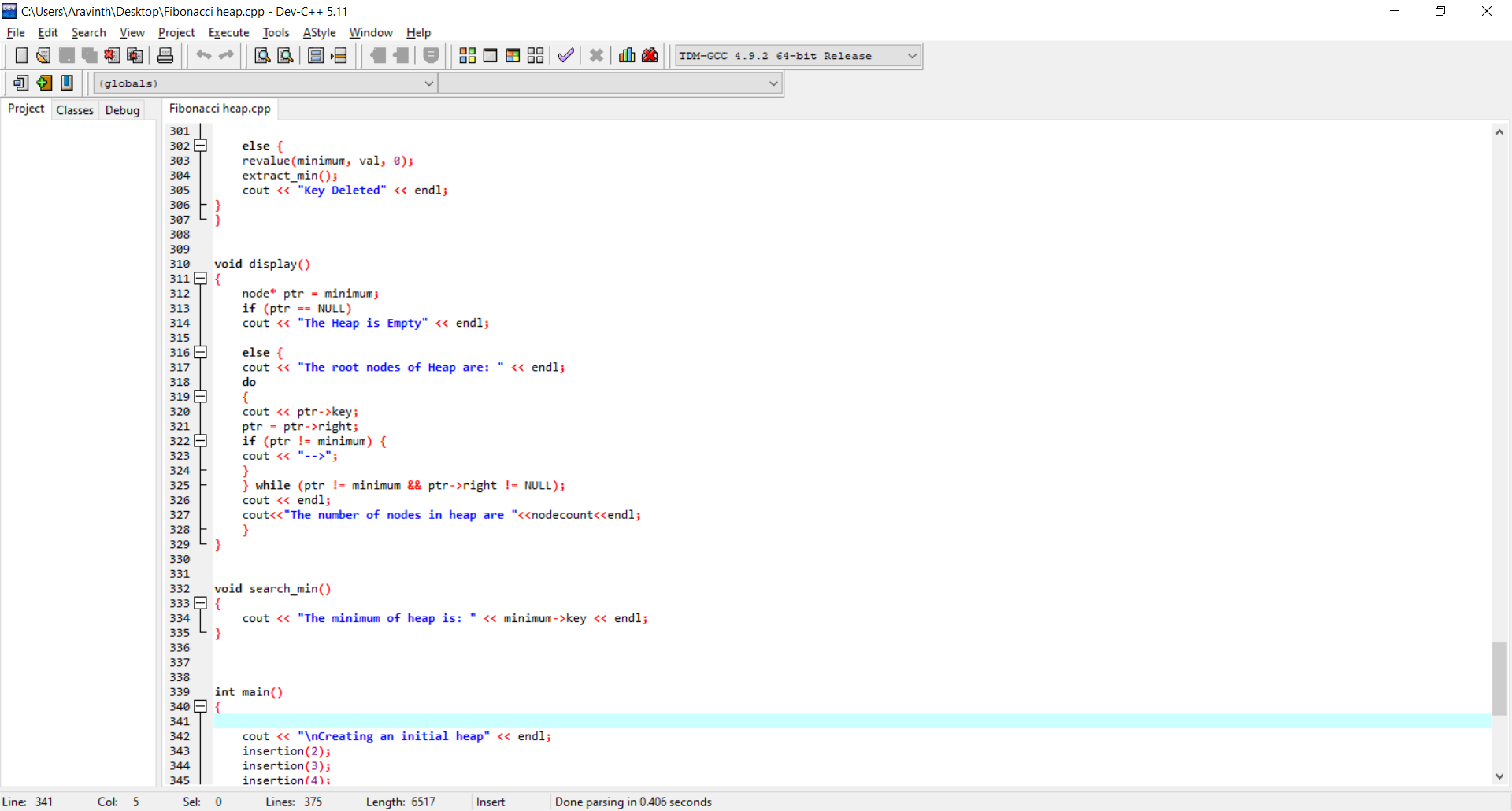


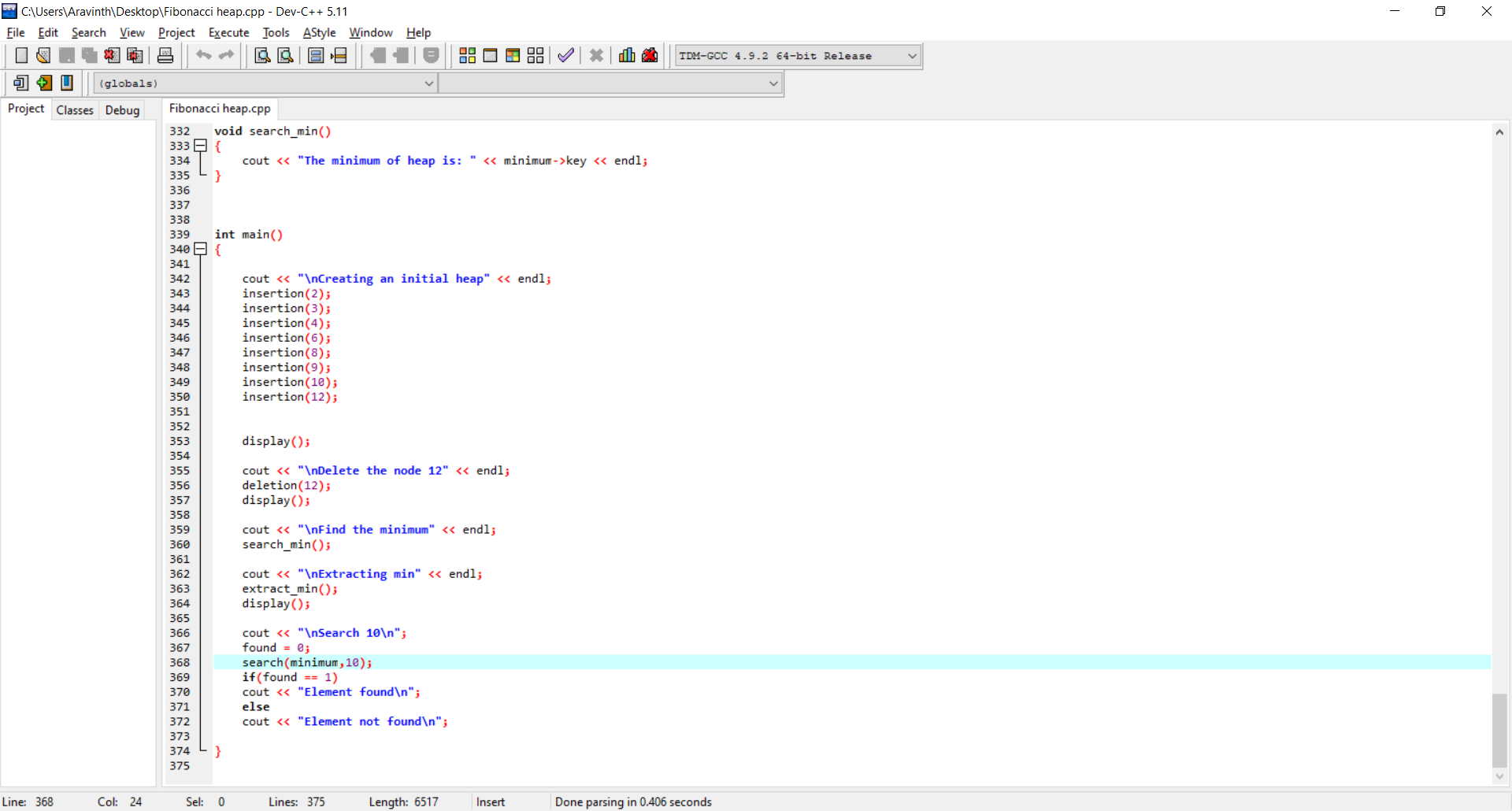




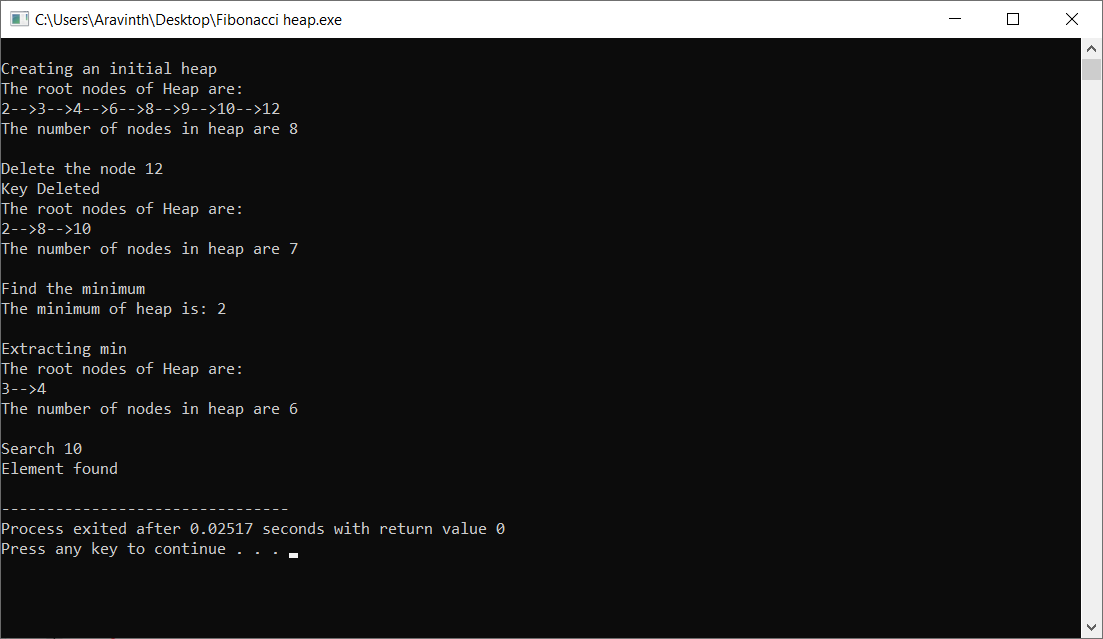








Output:



**Treap**

**Description**

Like Red-Black and AVL Trees, Treap is a Balanced Binary Search Tree, but not guaranteed to have height as

O(Log n). The idea is to use Randomization and Binary Heap property to maintain balance with high probability. The expected time complexity of search, insert and delete is O(Log n).

Every node of Treap maintains two values.  
1) **Key**: Follows standard BST ordering (left is smaller and right is greater)  
2) **Priority:** Randomly assigned value that follows Max-Heap property.

Like other self-balancing Binary Search Trees, Treap uses rotations to maintain Max-Heap property during insertion and deletion.

**Applications**

If hash values are used as priorities, treaps provide unique representation of the content.The win of treaps is that the code is considerably simpler than red-black trees. Red-black trees are known for being fast, but this implementation of treaps is competitive in speed and a lot shorter and simpler.

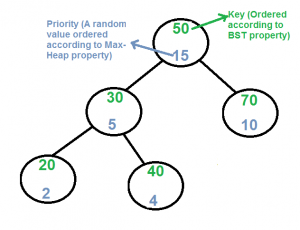
**Limitations**

Treaps permit easy implementations of find, insert, delete, split, and join operations. All these operations take worst case time O(H), where H is the height of the treap. However, treaps (like BST’s) can become very unbalanced, so that H = O(N), and that is bad. Maintaining a strict balance condition (like the AVL or red-black property) in a treap would be impossible in general, if the user supplies both key values and priorities: remember that treaps with unique keys and priorities are unique.

However, if priorities are generated randomly by the insert algorithm, balance can be maintained with high probability and the operations stay very simple that is the idea behind randomized search trees.

**Example**

Treap :



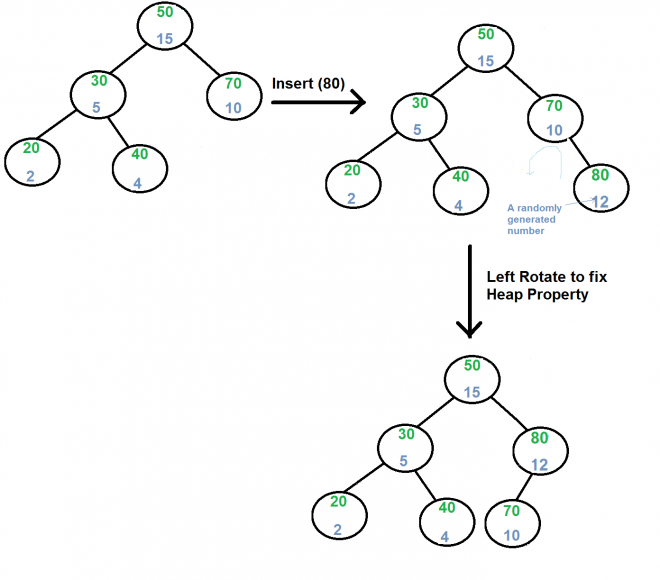
**Operations**

**1.Insertion**

Algorithm :

1) Create new node with key equals to x and value equals to a random value.  
2) Perform standard binary search tree insertion i.e., a new key is always inserted at the leaf. Start searching a key from the root until hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.  
3) Use rotations to make sure that inserted node's priority follows max heap property.

Example :



**2.Deletion**

Algorithm :

1) If node is a leaf, delete it.  
2) If node has one child NULL and other as non-NULL, replace node with the non-empty child.  
3) If node has both children as non-NULL, find max of left and right children.  
 a) If priority of right child is greater, perform left rotation at node  
 b) If priority of left child is greater, perform right rotation at node.

The idea of step 3 is to move the node to down so that we end up with either case 1 or case 2.

Example :



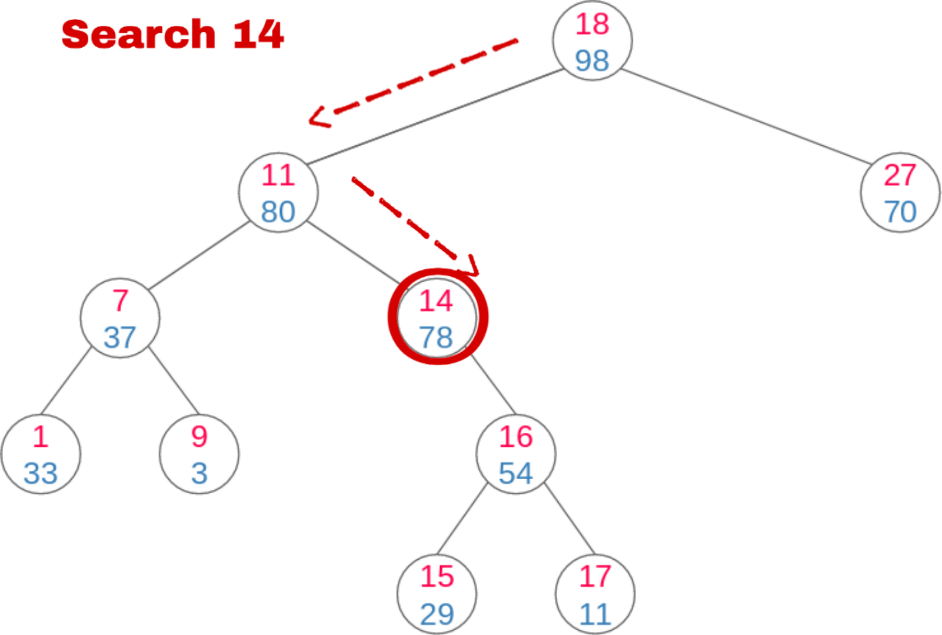
**3.Search**

Algorithm :

1) If the key is the same as root, then we return root.

2) If the key is not root, then we compare it with the root to determine if we need to search the left or right subtree. 3) Once we find the subtree, we need to search the key in, and we recursively search for it in either of the subtrees.

Example :



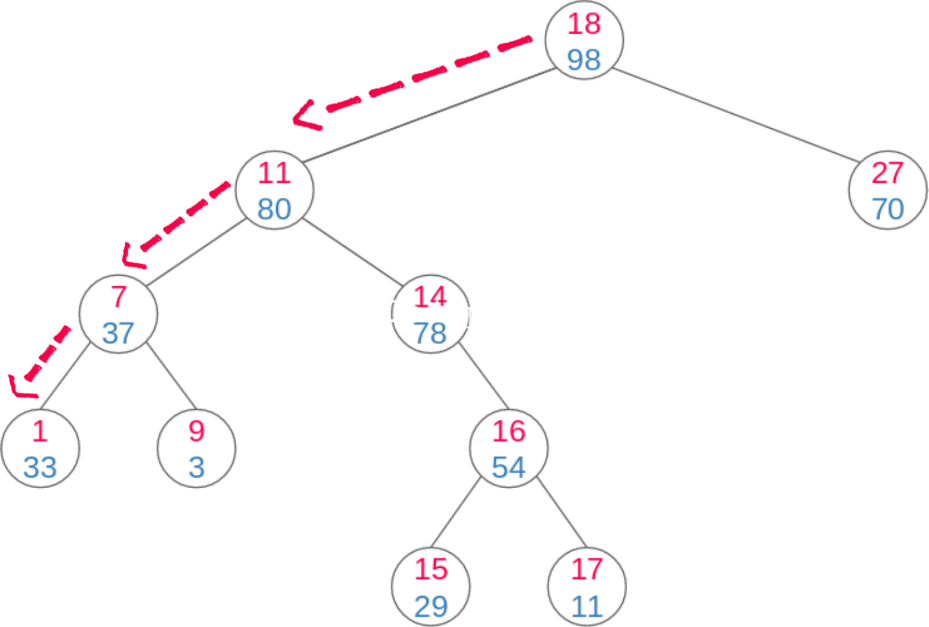
**4.Find min**

Algorithm :

1) As treap has a structure of BST. To get min, recursively traverse on the left side of the tree.

2) Once you reach NULL, its parent node is the minimum. Return the minimum.

Example :



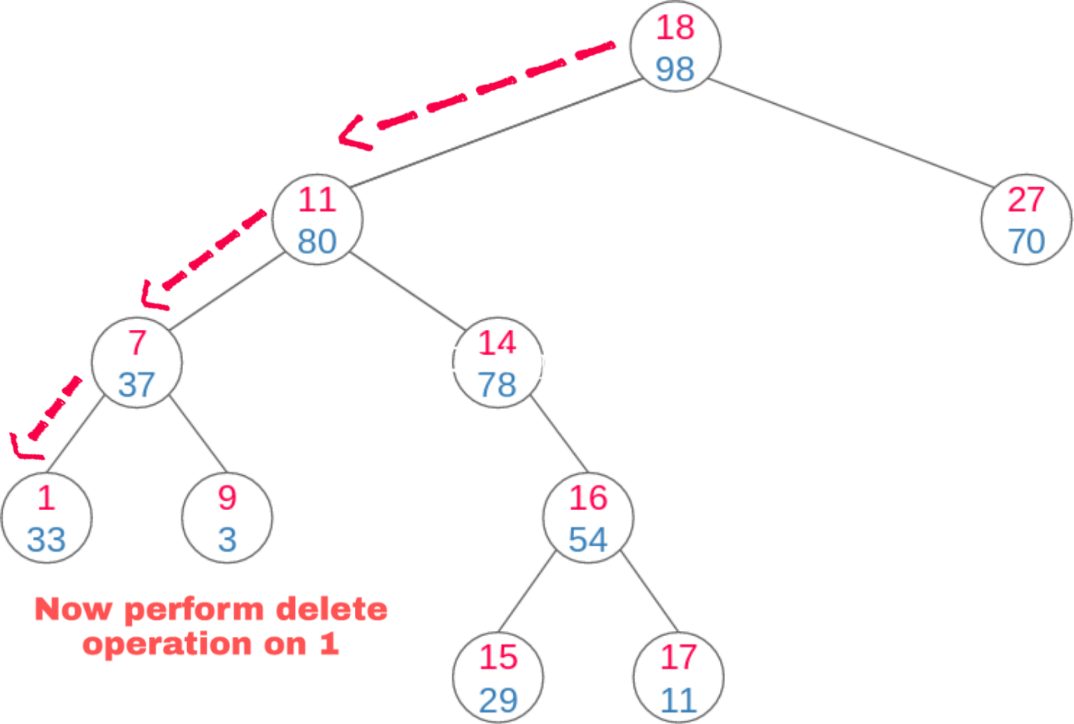
**5.Extract min**

Algorithm :

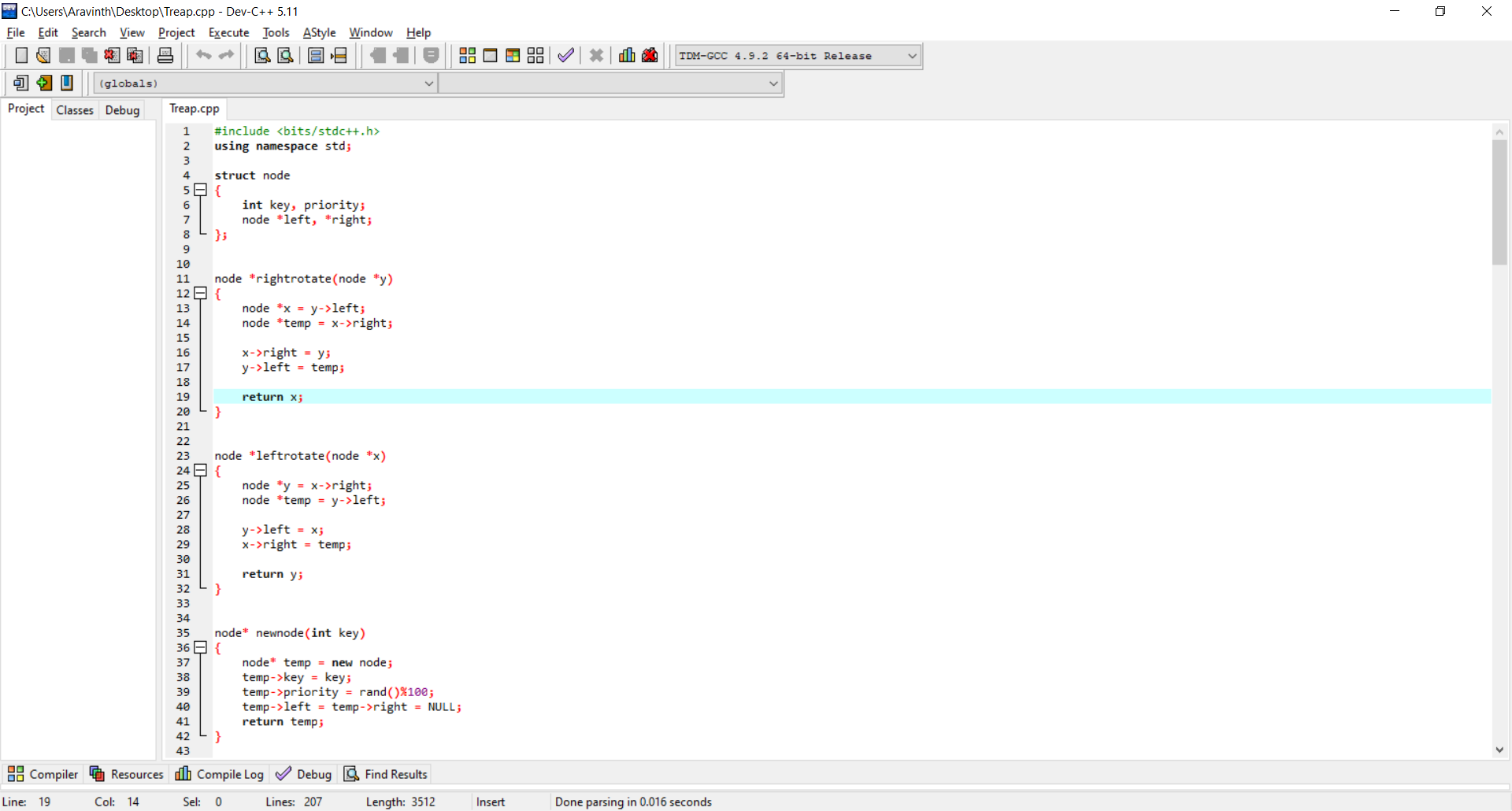
1) First perform Find min operation and get the minimum valued node.

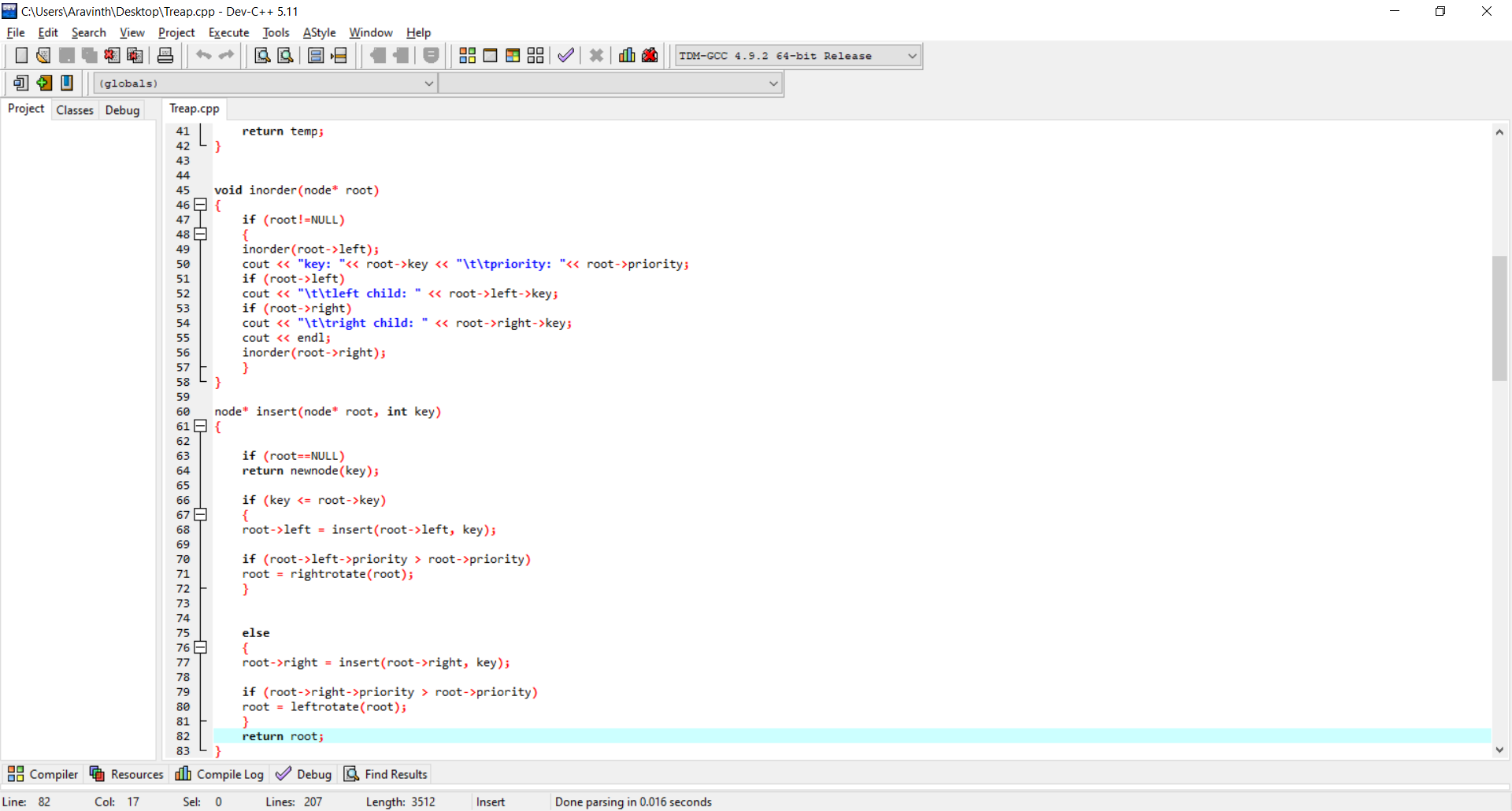
2) Then perform Deletion operation.

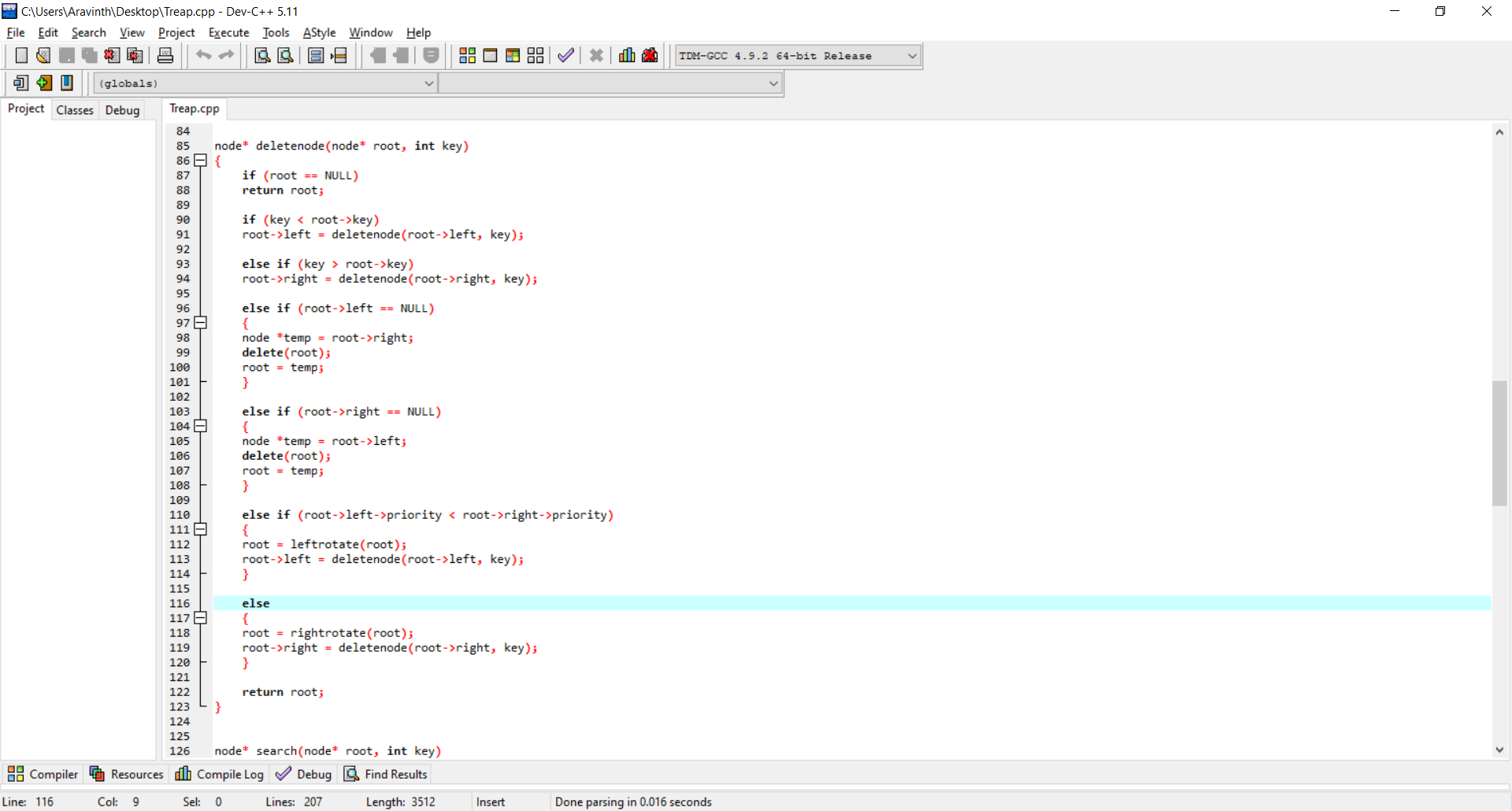
Example :

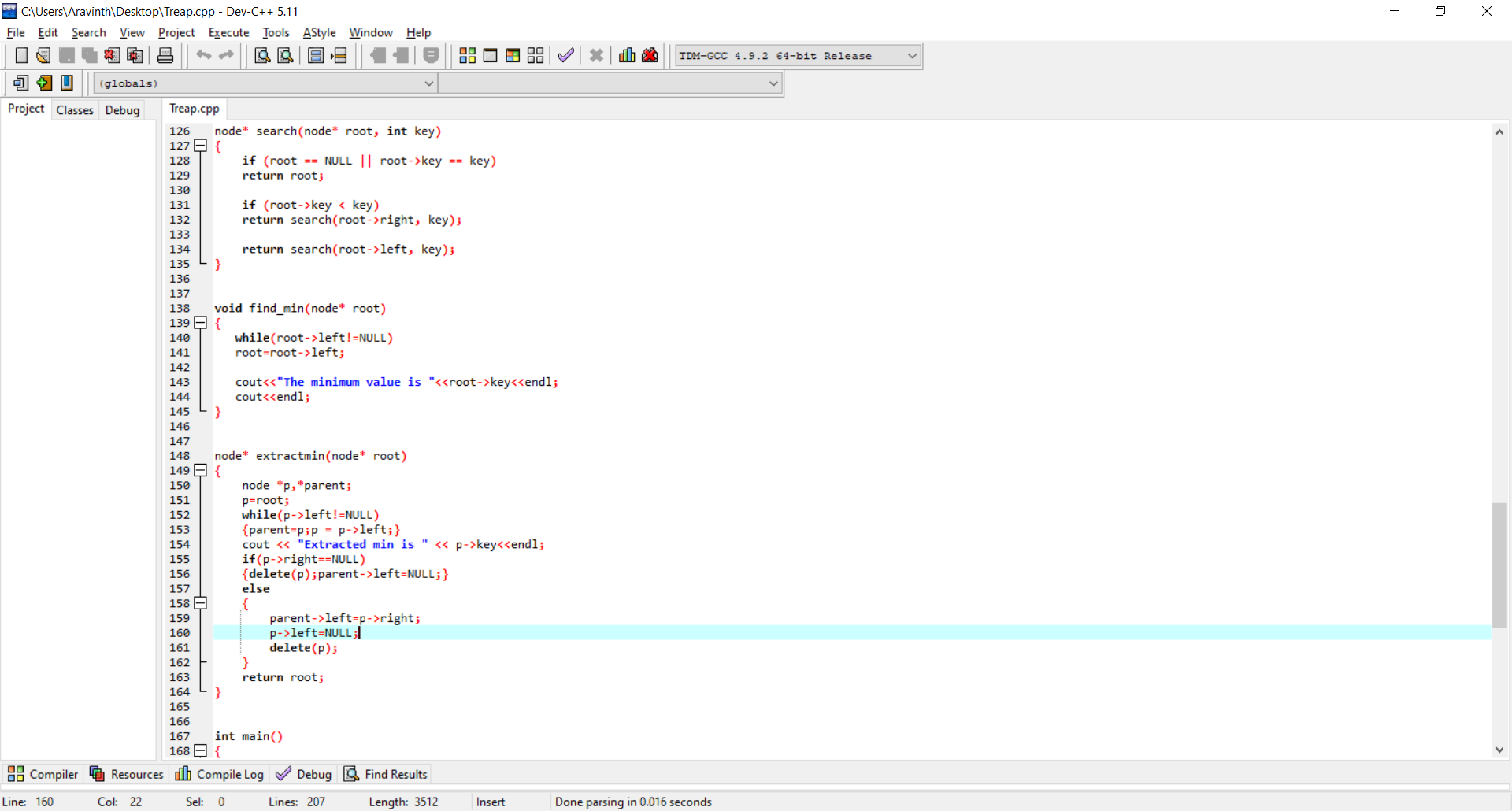


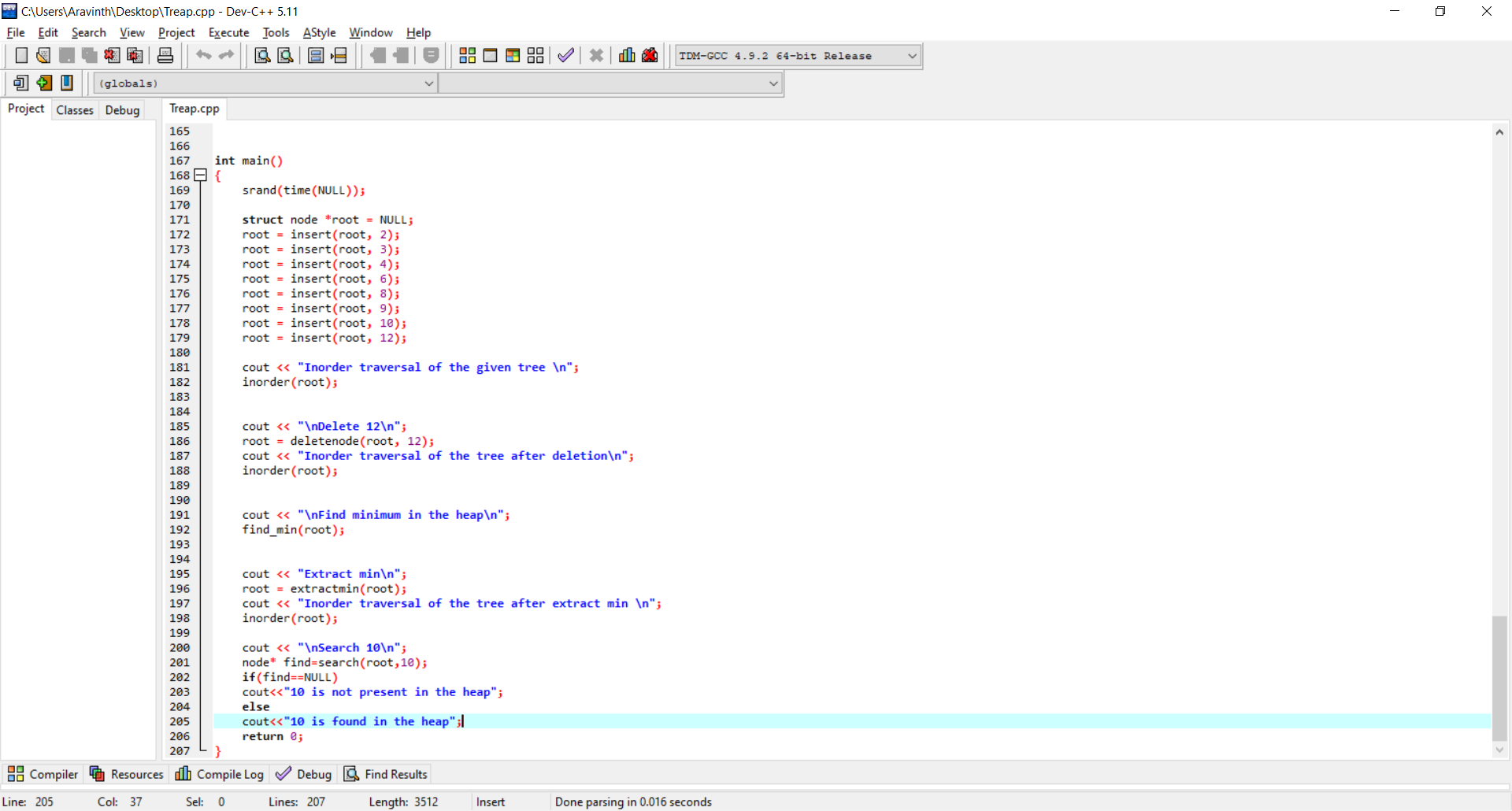
Code:



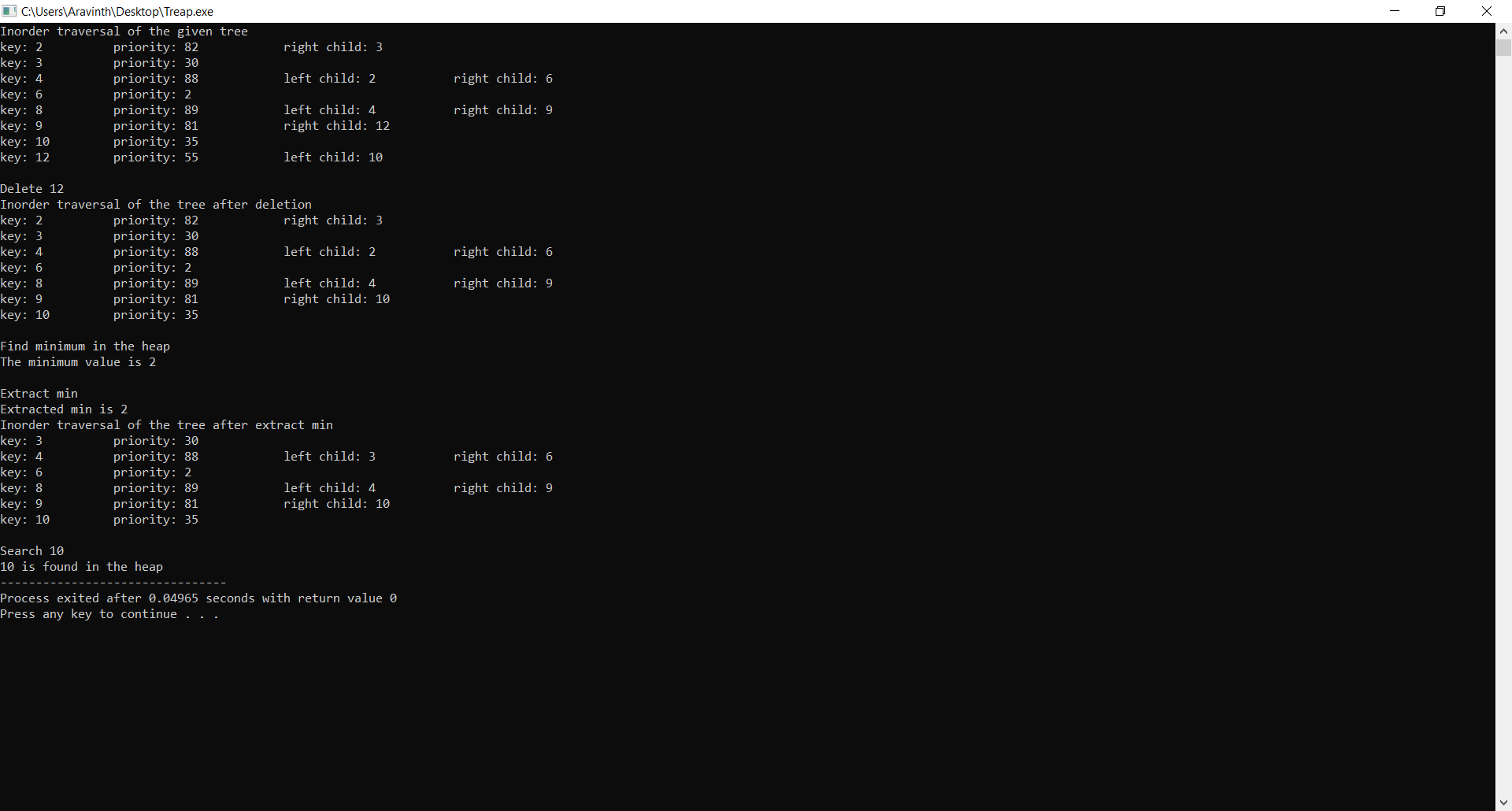








Output:



**K-ary heap**

**Description**

K-ary heap is a generalization of binary heap where k=2, in which every node can have maximum of k nodes. It is a complete tree where each level has maximum number of nodes except the last level.

Basically K-ary heap follows two properties:-

* + - Each level is filled in left to right manner where all levels having maximum number of nodes except the last one.
    - It can be whether a max k-ary heap or min k-ary heap.
* Max k-ary heap:- Key value at root node is always greater than the key values in its descendants.
* Min k-ary heap:- key value at root node is always less than the key value in its descendants.

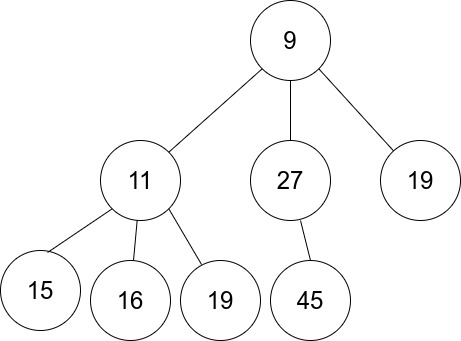
**Applications**

Applications of the k-ary heap prominently focuson priority queue nature of the k-ary heaps.

* Various Operating systems use this algorithm for jobs and process management
* Finding the orde in statistics
* In real life scenario, it can be applied to a Sim card store where customers who have made payment already will be dealt first since they will take less time. It will save time for many customers in line and avoid unnecessary waiting.
* Dealing with priority queues in Prim’s algorithm.
* Priority Queues implementation in Data Compression or Huffman Coding

**Limitations**

Tree must be complete means that every level of tree must be filled. Nodes in lowest level must be as far to the left of the tree as possible. When do any operation we need to make sure that heap property is maintained.



**Operations**

**1.Insertion**

Algorithm :

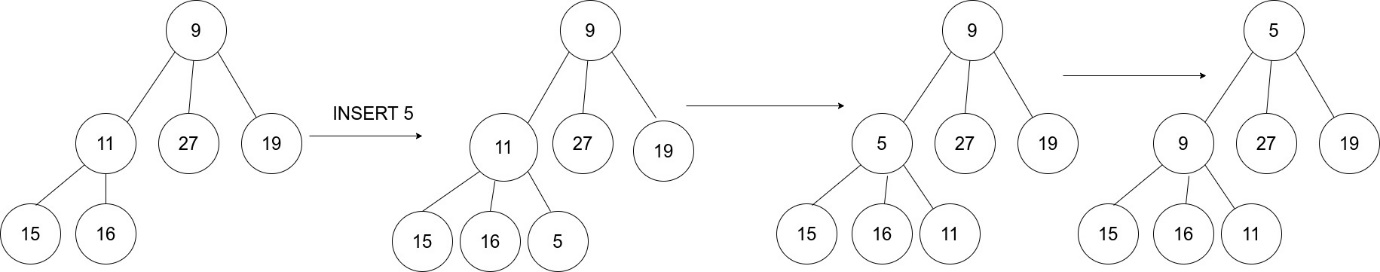
i) Insert the element at n+1 index.(last index+1)

ii) check with it’s parent whether satisfy heap property.

iii) *If not satisfied then swap with it’s parents.*

iv) go to step 2 and do it till parent becomes root of the heap.

Example:



**2.Extract min**

Algorithm :

i) Swap the root node with the last element in the heap.

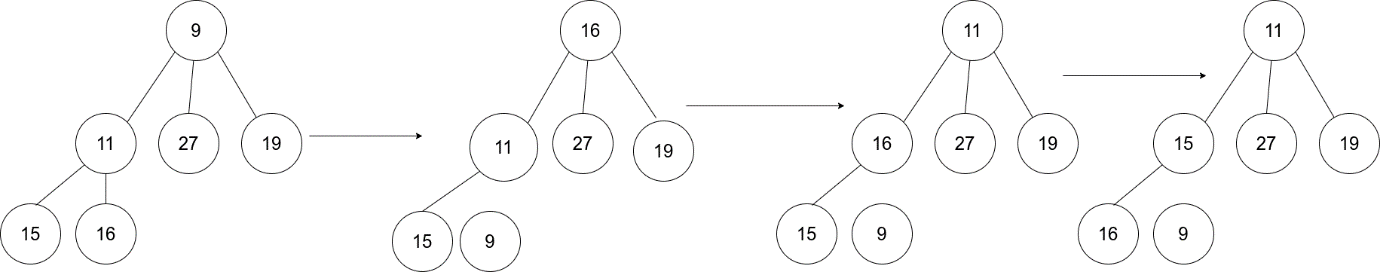
ii) size<=size-1

iii) Get the minimum key element among the children.

iv) If minimum key of children less than the parent key swap.

v) Repeat the step 3 till the heap property is achieved.

Example :



**3.Deletion**

Algorithm :

i) Search for the given key value in the heap.

ii) If the key is found swap the key value and root value.

iii) Swap the root key and last element of the heap.

iii) Get the minimum key element among the children.

v) If minimum key of children less than the parent key swap.

v) Repeat the step 3 till the heap property is achieved from root to leave node.

Example :



**4. Search**

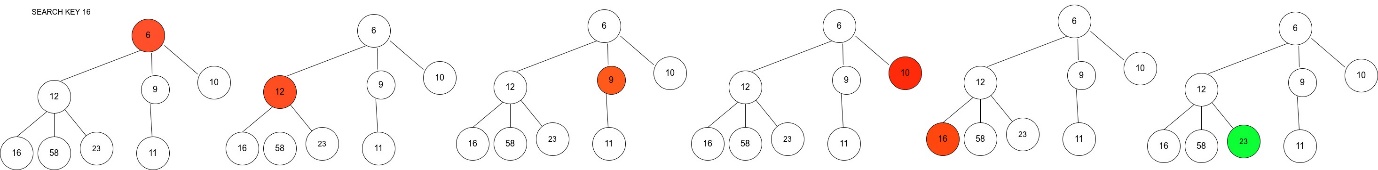
Algorithm :

i) Traverse through each level of the heap.

ii) If search key is found return index

iii) *else return -1*

Example:



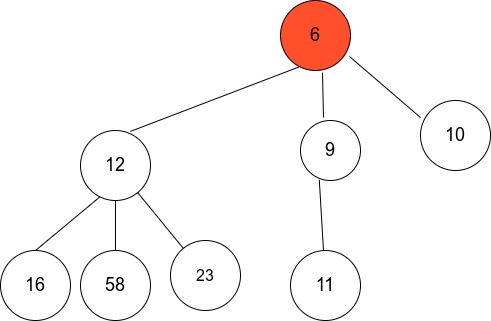
**5. Find Minimum/Maximum**

Algorithm :

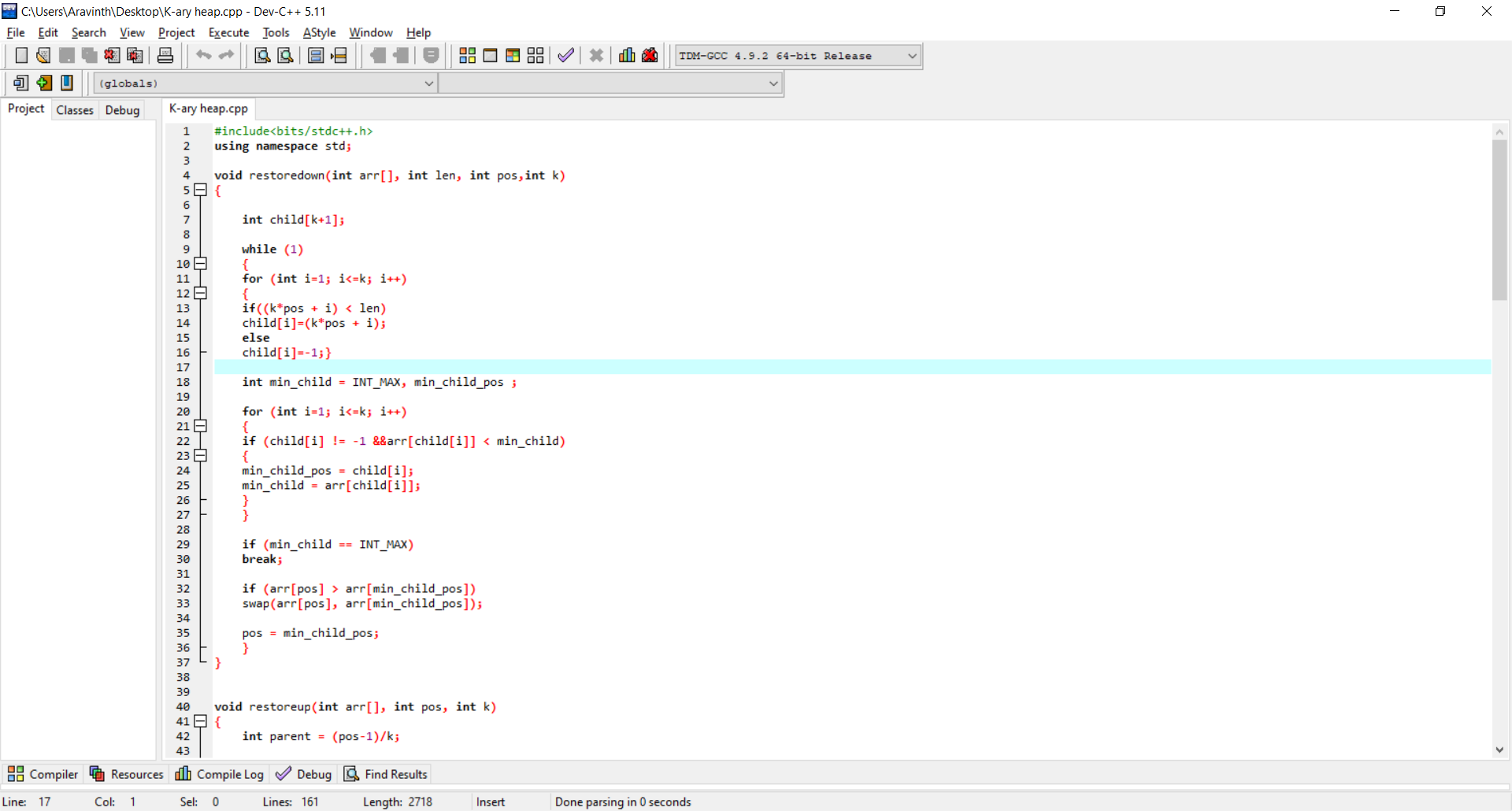
i) If the k-ary heap is min heap then min value will be root key value.

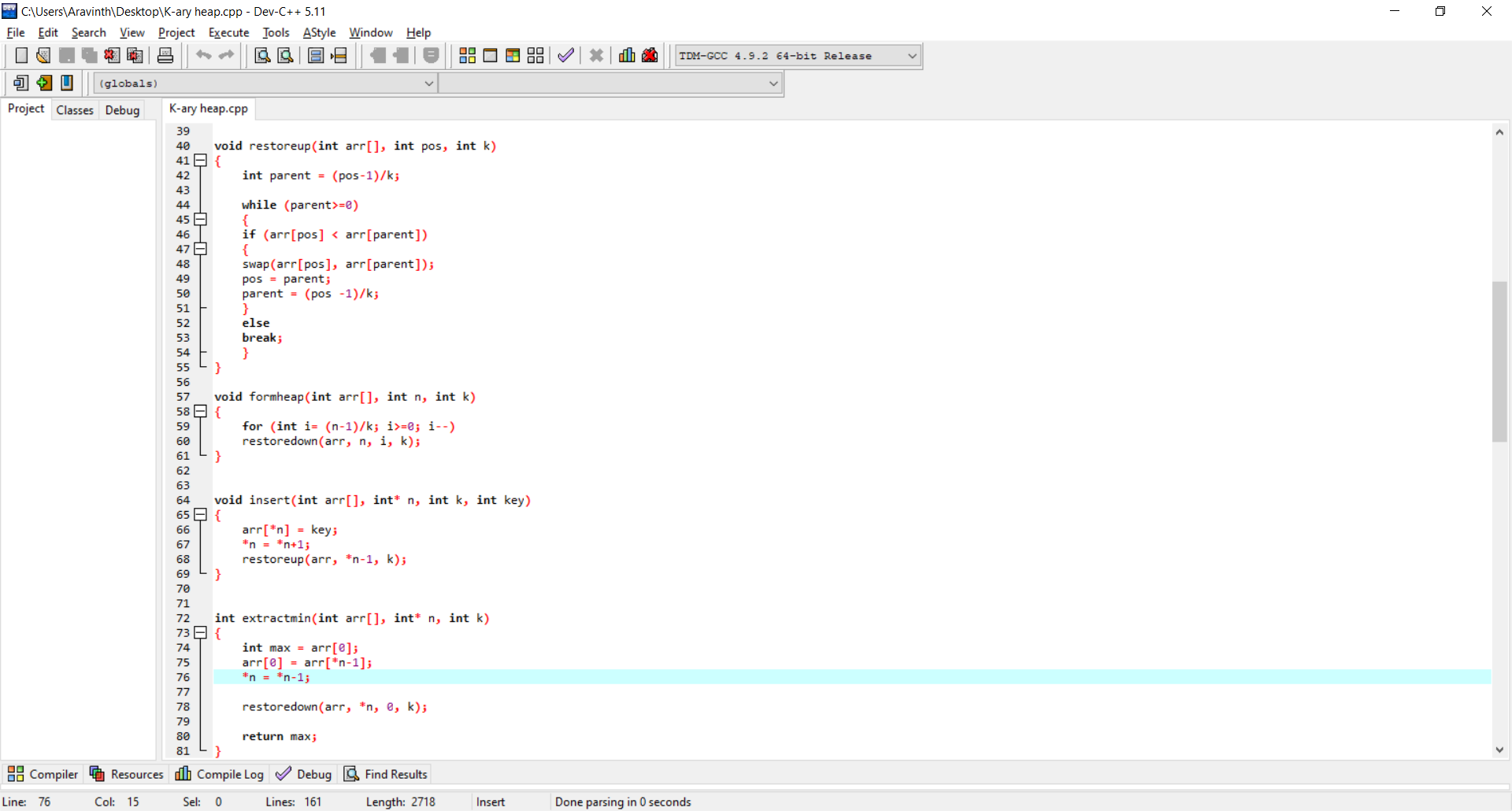
ii) If the k-ary heap is max heap then max value will be root key value.

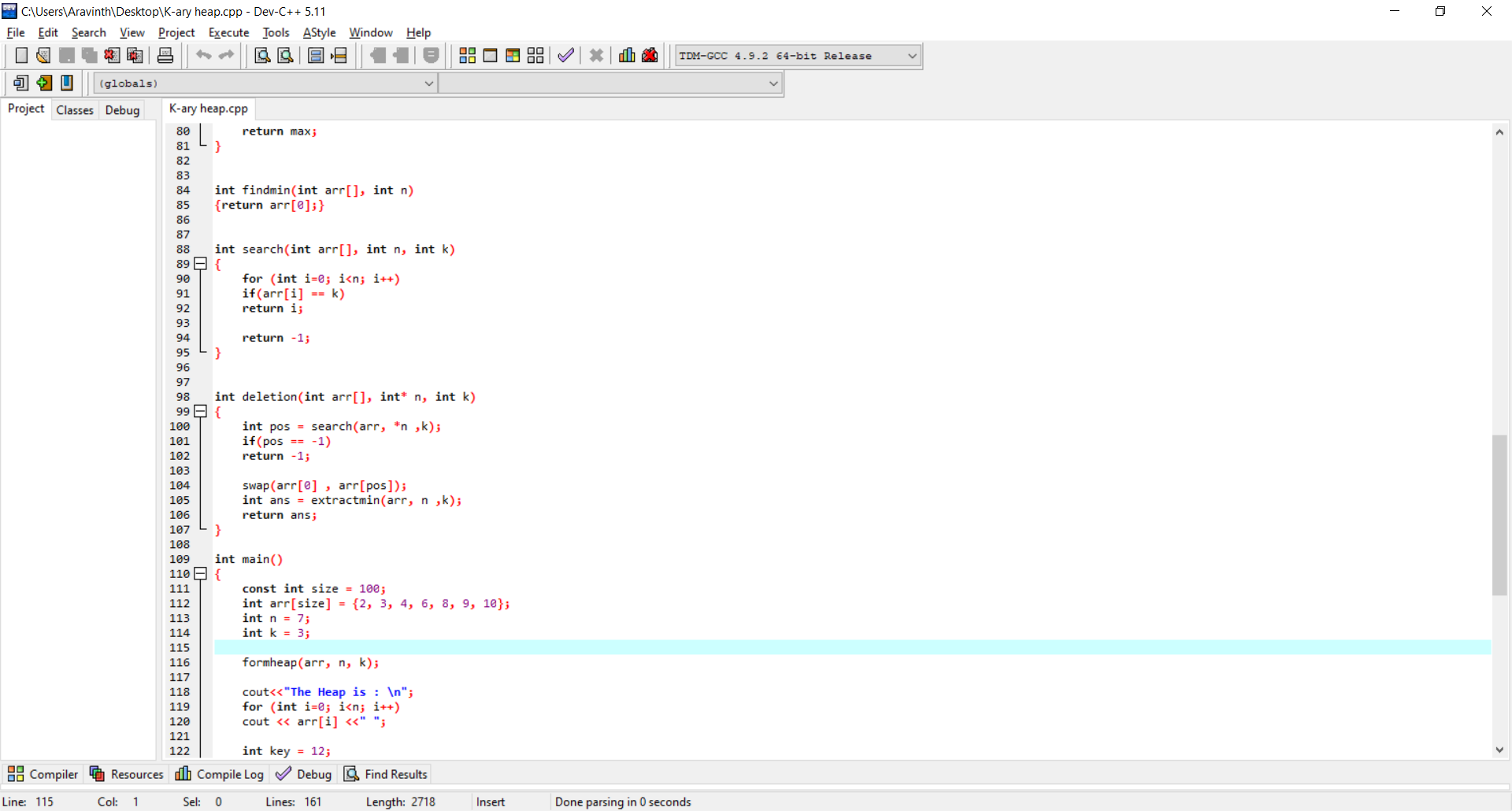
Example:

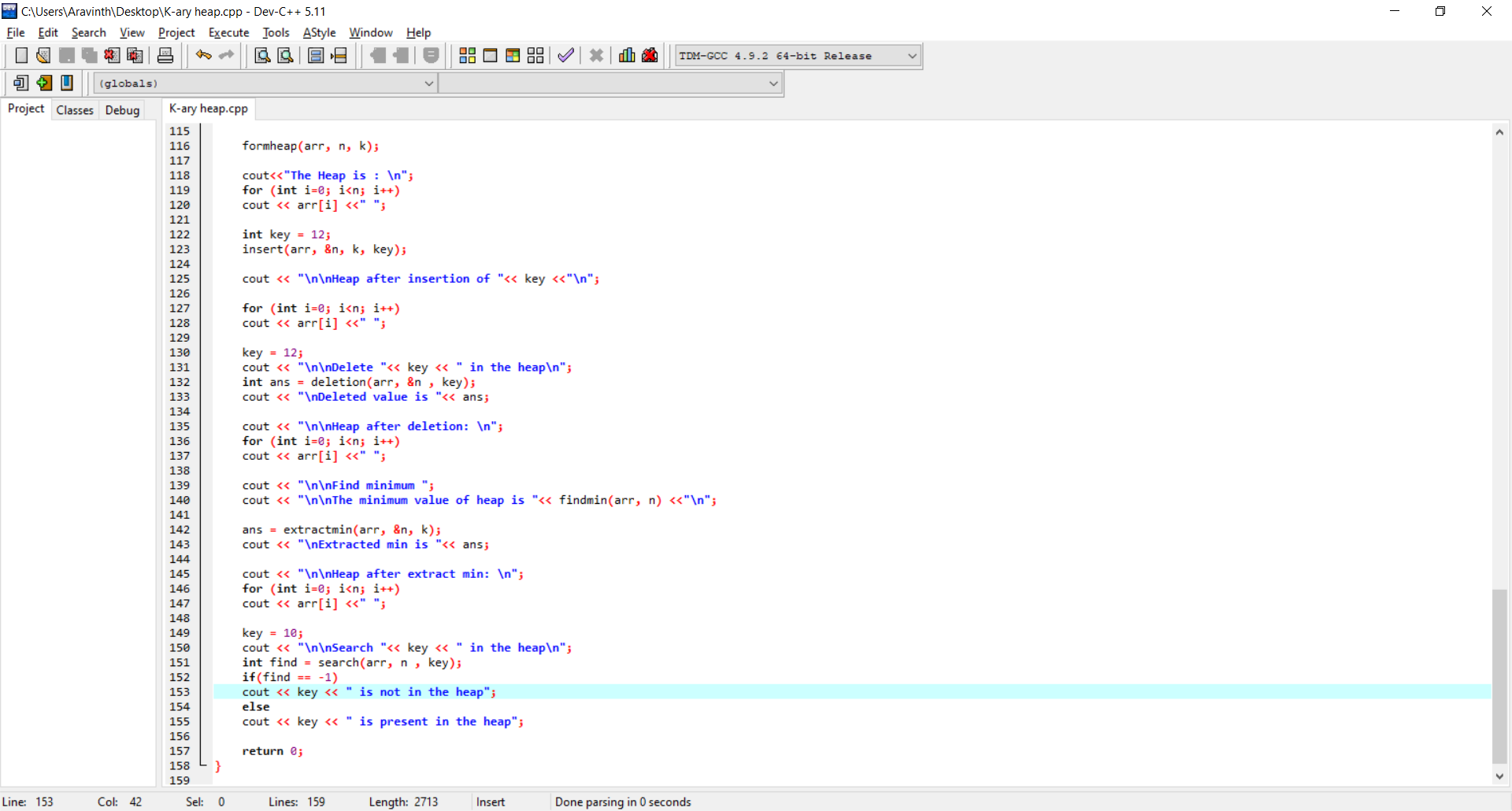


Code:

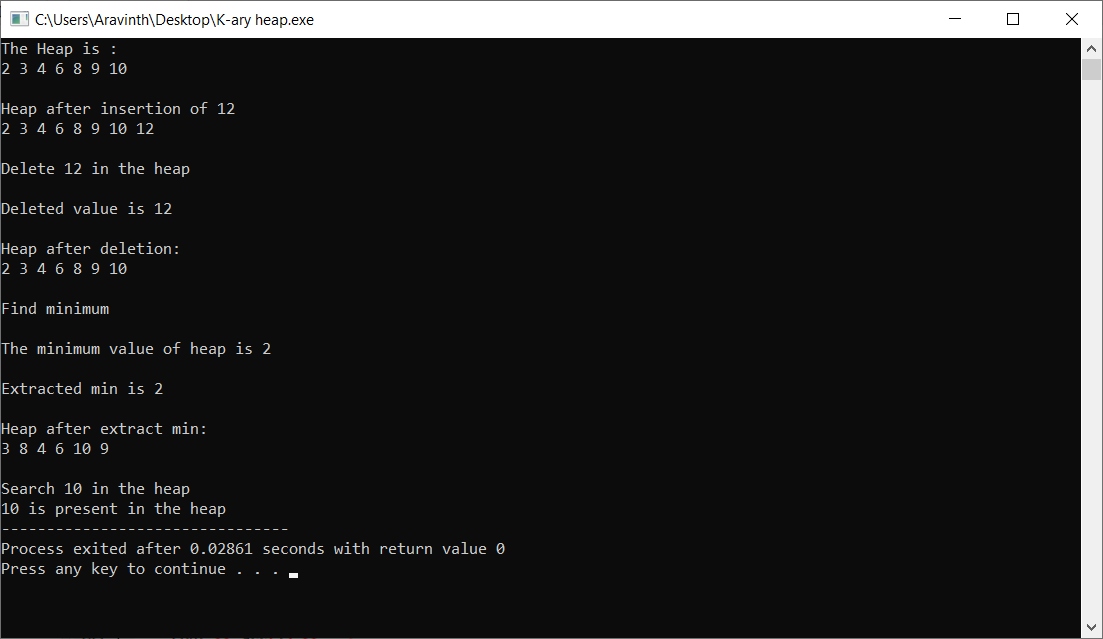








Output:



**Performance Comparison**

Note:

The outputs are taken using this method:

![Graphical user interface

Description automatically generated with medium confidence](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAeAB4AAD/4RDmRXhpZgAATU0AKgAAAAgABAE7AAIAAAAJAAAISodpAAQAAAABAAAIVJydAAEAAAASAAAQzOocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEFyYXZpbnRoAAAABZADAAIAAAAUAAAQopAEAAIAAAAUAAAQtpKRAAIAAAADMjEAAJKSAAIAAAADMjEAAOocAAcAAAgMAAAIlgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADIwMjE6MDU6MjAgMjM6MjE6NDkAMjAyMTowNToyMCAyMzoyMTo0OQAAAEEAcgBhAHYAaQBuAHQAaAAAAP/hCxtodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvADw/eHBhY2tldCBiZWdpbj0n77u/JyBpZD0nVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkJz8+DQo8eDp4bXBtZXRhIHhtbG5zOng9ImFkb2JlOm5zOm1ldGEvIj48cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6ZGM9Imh0dHA6Ly9wdXJsLm9yZy9kYy9lbGVtZW50cy8xLjEvIi8+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRlRGF0ZT4yMDIxLTA1LTIwVDIzOjIxOjQ5LjIwNzwveG1wOkNyZWF0ZURhdGU+PC9yZGY6RGVzY3JpcHRpb24+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iPjxkYzpjcmVhdG9yPjxyZGY6U2VxIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpsaT5BcmF2aW50aDwvcmRmOmxpPjwvcmRmOlNlcT4NCgkJCTwvZGM6Y3JlYXRvcj48L3JkZjpEZXNjcmlwdGlvbj48L3JkZjpSREY+PC94OnhtcG1ldGE+DQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgIDw/eHBhY2tldCBlbmQ9J3cnPz7/2wBDAAcFBQYFBAcGBQYIBwcIChELCgkJChUPEAwRGBUaGRgVGBcbHichGx0lHRcYIi4iJSgpKywrGiAvMy8qMicqKyr/2wBDAQcICAoJChQLCxQqHBgcKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKir/wAARCACWAuEDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwD6RooooAKK5/8A4Tvw2NYGltqiJeNJ5QjeN1y/puK4/WppfF+gQ29/PLqcKxadII7psHETHseKOlw62NqiuC1f4taHpXjGx0Z5FaGeNmmuNsn7ogEgBQvzZ9q39U8b+HdGW3OpanHB9pTzIgUdiy+uACQPrR0uHWxvUVBZXttqNnFd2MyTwSqGSRDkMKwfE/iYeHtY0ZbiaOGyunmFw7qSQFjLDGOetGzswWqujpaKxovF2hTaG2sRajG1ijbWl2twfTbjOfbFFr4u0K90mfU7bUY2tLf/AFshVl2fUEA/pRsG5s0Vzum+PfDWsSzxabqizyW8fmyqIpAVX1wVHpUHhHxK2sWOq391qFnc2lvcv5Ulujr5cQzw4YA7gBzR/wAOB1NFYWmeNfDus6gtjpmpx3Fyyb1RVYbhjOQSMHj0qno3iu+1HxheaPe6T9hiigWaF3lzIykkfMoGF6diaOtg6N9jqaKKrvqNlFIUkvLdHHVWlUEfhmldIpRctkWKKAQRkHIPcVyeseM7bQPGJs9Xuo7bT/sCzhjGzMXMhXsCcYA7UxWZ1lFcp4q8X21h4RGo6VqlpDJdAfZLi4R2iY577QSK1bvxBYaNokF/rd5FDG6rmRVJDMR/CAMn8qO/kLt5mtRXE6l8RdLn023u/D+owzxLqMNtdO0bAIrHkfMBziui0TxHpPiKKWTRrxbpYWCyYVlKn3DAGnZgalFR3E8VrbST3DhIolLux/hA5JrF0/xt4d1TUvsFhqcc1ztLhAjAEDqQSMHr60utg8zeorEsfGWgalqjadZalHLdKSNmxhnHXBIwfwNLF4w0KfXG0iG/V75W2mJY3OD6bsY/WgNjaorCufGnh6z1gaXc6nGl4WC7NrEAnsWA2g/jWfr+pazpni/Rlgv4G02/mMT2ptvmGFzkSbv6ULVrz0B6X8jraKptq1il3c2zXKCa1iWWZOcojZwT/wB8n8q5bxF8UND0J9IKzrPFqTAiQK+FiK5DjCnPbj3o7f1uOzO1ork/EvjO0tfBUmq6TqVvA8yMLS4uoZPL3j+8AuQPwrWbXrTTfD0Go61ewxo0alpVBKsT/dGMn8qO/kLt5mtRWHH408PS6OdUTVIvsayCJpWDLtcnABBGRkn0qfRfE2j+IhN/Y18l0YCBIArKVz04YA9qANWikJCqSxAAGST2rFsPGGhapqrabp+oLNdrnKLE+OOvzYwfzo62DzNuiuT0LUtZHjfVdI1a/hvIIIUmhaO28opu7H5jn61f/wCE18PDWv7K/tSL7bv2eXtbG703Y25/GhapNdf+GB6N+Ru0Vian4x0HR79LLUdQWK5fGI1jdzz0ztBx+NUB4sW08Uazb6rcwwafYxROjlecsoJ9z16Urr+vIDqqKo3WtadZQW013dLFHdECEsD85PQfrUmoanZaVai51G4WCEsF3tnGT0FLmS6mipVJWtF67ab+haorM1XxJpGhwwS6repbpcZ8osCd+Bk4AHpUEfi/QZdCbWI9SiNgrbTNhuG9NuM59sVRFnextUVh2XjPw/qEMUlpqcbrNL5KZVlJfGcYIBHTvWst3BIWEcyPtXcdpzgZI/oaTaW4lrsTUVg+HtYu9ee8vFRYtPD+XaHGWkx1c+3TApPD+t3d3qF/perRJHe2chIMZ+WSM8qw/AgH3FQqidvM654SrDnT3ja67X/y2fY36Kpw6vY3GpzafDco93AA0kQzlQen8quVaaaujmlCUHaSsFFFFMkKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKRwxRghw2OD6GlooA8Jvvh74z1XWIrnUrW7uLiDWEuftMmqDyGgEgICQb8AhR3XNXNa8EeLmt/FWmafpEdxDrF0lxFdm6jVQo6rtLbs/hivVzr1oPEo0MrN9rMH2gHZ8m3JH3vXjpWnSS91Lp/wAN/wDIjb96/wDXX/M8913Q9etfEvhnWtK0v+0hp1u0FxbpcRxsMoRkFyAear6jo/iWx8cXniHT9BXVE1LT0tzA1zErWrqWOMswBB3dvSvSqKcve38/xvf8yYrlVl5fhb/I5vwB4duPCvgqx0q8dXnhXMmz7qk9h7VW8Z2GozaxoOo6bo39rjT55HkgEscZGUwrAuQMg11tV72/tdNt/Pv50gi3BN7nAyTgD86bk3LmGtFb+tTzLUvAus6rodzd3Fo0N1Pqgv2023u/KbaFxtEqMAH56g4qnrWiQaH4B1KR9G1DS7u9u7ZR9t1L7bLK4lQJhy7Ac4+8cevFeuXFzHbWcly+THGhc7RkkAZ4rJgfSPHXhaKWa1afT7xFcRXCFGx1GRnINT0aXl+n+RSeqb/rr+pxvg25v9P8SajFrdjfza9LZpIn2ia2zLEpbao8o7V53ctjOas6DaeJSviO31Dw1LZJqjTSxSteQuAWBwpCsTnnr0rrdF8K6L4daRtGsEtmlwHbczE+2WJNa9N2f3WIWn33OGs/Ct7a3Xgxo7OKNNKgdboIyjYzRbeMdcse1QWf/CT/APCyJtWm8KTRWU1ult5hvoCV2ljuwGzjnp1rso9XtJdYk0yJzJcxIHkCjIQHpk+tXqObmal2v+N7/mXOEoq0la6X3boK5bUPBS3+pS3ZvIU8xt21tMtnI/4EyEn8TW3baxZ3Wq3OnRuRdWwBeNhgkHoR6ir1ZuMaiuzop1a+EleOja6pbb9RsSeXEqZztGM4A/QVzlzoc8/j59Va3jeAaX9nSVipIfexIHccHrXS0VbVzn5n/X3nm8ui+IdP+EsHh600Nr29kiaORY7qJBF8+QSWYA/hWd4tudYl0/wra/2bc6brSXRW2hjmhkkJWJskMSY+gP3v516zWZrfh3SfEdvHDrVkl0kTb0DMVKnGMggg0225OXdp/cL/AIP4nnOjabLrOnS6VJYTNqVrrcF1qa3kkL+YAFJb5Ds+7j5R6dOa7XRdGu7Lxtr+ozQqlterAIWDD5toYNwOR1HWtTR9C03QLQ22kWiW0RbcQpJJPqSSSaNY0PTtfs1tdXtvtEKtvC72Xn1ypB71XN2/rb/IXSz/AK1uO1y1lvdAv7W3XdLNbuiAnGSVIHNZ9loCr4Sg0+SCGG5itmiRgqnyiRzgjp+FN0zwL4c0a/S903TfJuI87X86RsZ9ixFdBUOKafn+l/8AMd3deX/A/wAjynwX4Ck0nUtNTVvD+pmXTsiO/fWzLb7gpXcsBkO3I7beM1o2ekeIbLxeH0LSrzSrCe533z3F9FPDMncqm4tGT/sge9ei0VV3dMlq9zzVvD/iGDT9Q8ORaJHc217eGcao08exVLhvmQneWGOw6mrniqDxG2taJ/ZXhybULfS5A7Ti8hj835QMAMwP5131FStLeVvw2G9b/P8AHc8+1qw8TJr2pX2laILoatp8VuQ1zGv2Z03/AHst8w+f+HPSmy+GdatvAvhaKCyFzqOiiFpbRZkXeQmxgGJ28ZJ69q9Dop9Puf3aobd9/wCtLfkch4ql1/VvAtxbWfhuZr2+ieJrY3cIMGehLbsH8DWVNpfiG70rQb5tBKXmjTBjp0t1E3njYV3KwbaDz3PrXolFC0ba62/D/hxbpJ+f4nmt34U1nW47y9vNKhtXvtStJnsWkRyscbpuLkEqxwpPHbiui0nQ7uz+IGs6o8KpZ3VrbRROrD5mQybhgcjG4V1FUdX0Ww12x+x6rb/aINwbZvZeR7qQe9NOysv62/yG9dGO1ezfUdFvbKKTynuLd4lf+6WUjP61z/hGTXdPtLXRtR8ONbQ2sRT7bHdRGNyOhCBtwz7jrVrTvAXhrSb+O90/TRDcRHKP58jY4x0LEV0VLYW55/Zp4mbx9f38/heeCyvYFtxP9ugJjwD82A2fy5qlD4c8QjSE8MPokIto7/7R/apnjKFfN8zOzO/fzjOOtem0ULRJf1vcHq7/ANbWOFsbPX/DHiLVHtdCOsW2ozrMLmG5ijaMYxtYOwJA6jHqaz9f8EaveeOrvxLYQxySQCJ7a3nkVorjaoDKVJwGHOGOMEDmvSqKXVPqg7+Zi6uNQntdPa30i1u5fMVp47gqfI6ZKk9SPb0qXxAl3JpYFhpttqMvmL+5ucbQM8nnuK1aKlwvfXc6IV3Hk0Xu+uvrr+Vjm9Z0WfUvEHhu7NrE8NhJJJMDtxGTHhcA+h9K5h/CGtWpnvrexjuJYNce/iszKgE8ZUDgk4Dcd69LorS9tv61v+hhJ82/9b/5nmOq+GfEWspqGtR6THY6jLNA1vZtNGzDYVBZnB25wCetafhfwxq2kXerWdwu3T5rZPssvmBiHJYuuM54Jz+Nd3RWbhFx5eg4zalzdTlvh/cIPDEWmvhLrTv3FxEeCrDvj3qPSWGp/EPUdQtTutrWBbUyL0dwcnnvjOPwrpE060jvJrqOBVnnULI44LgetOtLK2sIPJs4Vhj3FtqjuTkn8zUqEvd5un/DHoVMXTc6tSKd5/hfV/8AAOV0m+ubn4k6jHbraz6eLdSbiCJch8n5S4GSfYniuxpkcMUOfKjRNxydqgZNPqqcXGNmc2JrRrSTjG1kl9y3CiiirOYKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPPY9VsNN+LQtk8SPJBcWpaS2uNR3xpLuIwqE4U9OBWJrZ1T+z/E2ux6/qccmnamY7SCO5ZYkX5MgqDhhz0PT8a9Mbw/oz3P2h9IsGn3bvNNshbPrnGc1aNlatFJG1tCY5G3OhjGHPqR3NEfdS8v87/8AAKurvs/8jzrx1ezWGuR6jd6tK1jDEpOnWGqm2uSeOfL3KHHfBOcdAaoeLtT1jU/E9vZWmqR6TYf2fHPC15qU1j5jsWz86A7iAFyD0/GvTrjRdLvLkXF3ptnPMuMSywKzDHTkjNPvNL0/UVRdQsba6WP7gnhV9v0yOKVtLErT7jgprfWbyfwzpeo6/KWnt3N3caZcMiz4xgq4wfxqDx94Sa28BLHPrWr3KWl0knmteOGEZkyS5z820dCemK9Ijs7aIRCK3iQQjbHtQDYPQelPlhjuIWinjSWNxhkdQQw9CDRLW/rcFpucettosPgWdYPE929tjd9tOrszo+37ol3ZH0zXO2viq5k+EunzWesGecvFFe3azebLDGzAMxOSQcHr2616ONE0pbM2i6ZZi2ZtxhFumwn124xmlt9G0u0hkitdNs4I5RiRI4FUOPcAc03q2/T8P6sC0SPL9S1GXS7DxDB4b8SX2o2keleety921wYJvm+7KScHAHy54/GtvRIr/R/H0Fi2r3+oR3ekS3MqXc7SL5qvGAVBOFHzHgV28Gk6da20ltbWFrDBJ9+KOFVVvqAMGp/s8ImEwij81V2B9o3BfTPpwOKfT7/1/r5FJrqv60/y/E5XwAqvo97ePzdz3kxnY9chyoB+gArkdE/tSC18K6zNr2p3M+oaiLeaKW5YxGM7+NmcZ+Xr1r0Kw0A6Z4iu76zuNtpdrultSOBJ/eHpkYrUFnbKkaLbQhYm3RqIxhD6j0NZUU4wSa2t+H+Z242rCrXdSOqlr6abfL9Dl/EQ+z+PPD09sP38rSRSAd49jHn8QKk1vUEtPHGiwSXt7F9oDKsMY/cyH/a56/hWnBoZHiWfWLyfzn2CO2jxgQr3/E+tLL4fhudei1S7uJrhrfJt4X27ISepGBnJwOpNZqEtPW/y/r8zpjiKKcFJ35YNerd7Lbpf8Hboa1FFFdB44UUUUAFFFFABRRRQAUUUZoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACjpRSModCrdGGDQB5bf/ABgl0zxdFpdxaabPayXq2nmWl3I8qlmABKmIJ35G+r938UZbbSfE16ulI39h3KwKpnI84Hufl4/WmL8HbRWWIa3e/YY9Q/tCK0EaBUk37z82NzAn1JxUmq/CO11ObVFXXL62stUlWae0jRMGQdG3Ebvwzilryrv/AMN/wR6c3l/w/wDwDJ1rxP44/wCFkaFb6XZWH2a7t3kjtn1B1SYbScviM7SO2M10GqeNtcTXTomgaLaX2o2tqtzfCa8aKOIHICqwQlj8p6gdqu634GGp3uk31hq9zpl9paGOK4ijR9ylSCCrgjv6VBqvw9e+1g6tY+IL3TdQmthbXU8EcZ+0KM4JVhgHk8gCnLa0fP8AW36Ex7vy/S/6mz4S8Rw+LPDNpq9vE0InXLRt1Ru4rO8deNV8IW1osUUMt3eOViW4d0jGBklmRWYcdMKa2fD+hWfhvQrbStNUrb26BV3HJPuT61W8ReGxrptJ7e+m06/s3L293AqsyZGCNrAggjjkU525vd2HHbU5e3+Jl3deF7e/ttGWe9l1BbEwpKwjZipO5GZQSOO6ip4viFe2unasNc0mKDVNPnit1t7W4Mscry42Ycqpx8wzxxz1rdPhme6srGLV9Ynv5rK8W7WdoY4yxUEBSFAGOT71V1LwHZak2rPJdTpJqUsM29cZgeLG0r68qDzQ7f18v+CNWt/Xn/wDnPD03iPUPiRri6tFaaTfNpUHlfZpmuUTLyYY7lTn29utbvgOW88zW7W+v7i++y37xRyXD7mCgnj6U7SvA02l6hqGp/2/e3Gp31slu11LGh2bSxDKmNo+90xjineGvB9/4e1O6upfEt3qEd3I0s0MttCgZz/FlVBH06Udfl+opa6+a/Kx1THapPXAzXPW3jGC5v0tV02+RnfbvYxbR78SE/pXREAjBGQexrPj8P6NDOJodJsY5VO4SLbIGB9c4rOSldWeh00JUIxl7WN30/q5oV5S/irV/DWqeLr2z0mO9sLXUfMupZbgxlV8mPiMBTuPscdRzXq1cze+CbW+0vXrJ7qZU1uUyysAMxkoqYX8F71pHSV/63RgrWszG8cXFzJrPhySbTYptNN6hSdb1o5UkKNjKBCGGM/xCr3i7xlf6Jq8Wm6RZ2E9w1ubl21C8NtGEyRhSFbLZHTAqTxJ4KvPEEtn5PiO70+GzdZIoYraFxvAI3ZZSehPHSuM8dWEy+LbVdba9mtksAq30OhrqBlk3HIMflusZxzkAVGyt5v8v+B+Ql3fZfn/AMEtSeJde1TWPCuq2+jwf2hdWV4Us/tTCJeYuWfbkDj+6etdz4P8Qy+JdDN5c2gtLiOeS3mhV96q6MVbDYGRkdcCsnwppF9qNno2s680kV9ZwTQpGYlj3o5XBZQAFPyDgAU5b7Svh3btZTJqt39suJroNbadLcbS7liCY1IHJ71q7LT+t3+gtWl3/wCH/wCAbHirXZfDuiG+gtDeSeYsawhtpbPvWJN4x1rQ9F1LUPFui29klqiNA1pdNMkpY4wSUUggkdAeM1OmraX49t2sLePVbXyJEmLXWnTW4ODwAZFAP4Vu63ottr2izabebhHIBh1OGRgQQw9wQKzaaQ01c5Dwn8SJ9emvoL3T4Vktrc3CtYvLKjKP4SZI0+bjoAfrVnw144vtWs7u/wBVtNNtrKFGdRb3jyT8f3o2jXH5mtnSdE1jT7eeK98TXWoF49kTS2sKGE/3htUbj9c1m2XgFV1ubVdb1WXVrmW3a33PbRQ4RuoPlqN3QYznHbqab307fjr/AMAS217lfTfG2sy3mnTarocFrpOqtizniui8oypZfMQqAuQOzNzUNjpVjpfxkuDp9slv9o05ZZtn8bb35PvxV7S/AP2G6svtmt3l/Zacc2NnKqKsHBUfMoDPgHHzE+vWk/4QnU/+Eu/t7/hK7zzNoj8n7JBt8sEnZnbnueetGnOmvP7mra/MNeVr0++9/wAh1543ktfD+qakLFWNjqJshH5uN4EgTdnHHXOKxdV1zxzH8SorLS9O02W1az8yOGbUXRXGT8zYiOG7Y56da0NU+GiancXKnXb6GwubsXj2UaoF83duJ3Y3EH0zitfXPC82p6la6jpurz6Te20Zi86GKOTehz8pDgjvQuj/AK2/zKlbVL+tf8jD8eS6gJtCN7plvLZfbYzJJHfMjxy4OMLswy9epFauveJNVh1pdH8MaZb6hfrD5832u4MEaJ0+8FYlskcY/GmeJvB1/wCI1tU/4SW7s47dlkCRW0LbpFz8xLKfXp0pb3wbeXNxbX1t4iu7PVI4fImvY4IiZ09GQrtHIByBSW1vN/l/n+AuvyX5/wBfMzpfH+pz2+kR6RoaT6hqDTRPDNcbEhePGcsFOV564zjtW74Q8Qz+ItJmnvrNbO7trh7aeKOTzFDocHaxAyPwFRaf4MstOm0uWCeYtp/nHLnJmaXG5mP4dqqNe6X8P1miuF1S7/tC6lu82uny3GwsckHy1OPbNVdLf+mK19v63/4Bs+JNdi8OaLJfyxNM25Y4ol6vIx2qvtkkVU0k65rFpcweLdI062t5VARLa7afzAeoYFFx26ZrPfUtF+Iun3Gji31a3AAkEtxp89rsYHKsrOoBIIBxWlp2h6xZWNzBc+J7q8eRAsMslrCpgx3AVQG/4FnpSto7/wBf19w77WOa8Ez2Phbw74ilSIxWVjfz7YoxnCh2woB/KtHS/F2tnVLOLxDottZWmooXtJre6MrcDOJFKLtOCOhNJo/w/l0+HUbbUPEF1qdpqO9p4JbeKMF3JJYMigg8+tWNK8EPZ31tPqmt3eqx2SFLOGZERYQeP4AC3AAy2elJXsr9l+CsxO2tu7/4H/BGaH4m1/xFdxXmn6RZLoErHbdTXjLOy9mEQQjnj+Kuc8EeKJv7SvND0q2+2Xf9pzPcmWQokEPmHJzgknHQAYz3FdRo3g660C7RNM8QXceko5ZdNaGN0AP8Icjfj8aig+HtlaXkF5a3txDdw3jXPnoAC6s25om9VOSOemcjmhaST6Wt57ob2aXf9GbVxqGox+ILezh0wy2MiFpLzzMeWewx3o1LUNRtdSsoLLTDdW8zETziTb5Ix1x3ouNGafxBb6oL+5jWFChtlfEb57ketGpaM2oalZXYv7m3FoxYxRPhZeOjDv1rP37fPy2/r5nbF0OaN7fC76PfXz32208tzB8QeLtbsvEs2j6BolvfyQ2aXby3F0YlALMMcKxz8vH9KrReP7/WbLS18L6RFdahfWn2x4ru4MMcUfA5cKxJywwMevNdI/h6F/EF3qpmkElzZpaMmBgKrMcj3+c/lWEnw6WystMTRdau9OvNPtvsq3ccaO0sfGQysCvUA5xnitVtr/W//AOOVun9bf8ABK9v8S1XadT017cRwStOI33sskZYFFHG7O3g8degrW1DxZF/wh95q9gNwQ+XFnuTjH86rD4c6eF0v/SrgvY3DXEjsQTcuxy27PQEk8Djt0qWPwPBaeEtQ0W3uJJEupnnjMgA8tjjCjHYY/Wsqqm4e7v/AFt+n4m+GdJVour8N191/wDLf1L2k6CIfDBsb2aSea6QtcTE/MztySPoTxVXwXf3E1heafeyGafTbhrfe3VlwCpP4HH4VPpGrXv/AAjDz6jYTJe2itHJCFJ81l4yvqDjj61H4Y0y60zR7q7uYTJf30jXMkIYAgkABcnjoBUq3OnDa3/Dfqd9Tm9nWVZq7krbb9beVvlsXtB1S81SK5a+sRZtFMY0AlD7wO/HT6Vq1zPgzQLnRU1Ga6UxG+uTMtuX3mIc8EjIJ5rpq0p83IubexxYyNKNeSpW5fLbb1f5hRRRVnIFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFI7bEZjkgDPFAC0V8+X/jjXG8TW2o6Pe3UNi+tLZtDc6m0jMPMCt/o+zCr1wd1aOseNtVstE8bCTWpobu3v0jsl8zDop7KOuKV/d5v66f5jt71v66/5HtEupWUGoQ2E11El3OC0UDOA7gdSB3q1Xi/iDSY3+JfhW51DX9UtVvLR2ab7b5YjbYThSR8ufStLWdUlufHVzo194lvdH02x01J7aaG52NdMSwLM/wDFjaOKcvdWvn+F/wDImPva+n42/wAz1aiuW+G2qalrPgHTr3WizXUkfzOy4LjsxHvVTx1cXA1nw/ZR61LpFtdTSrcSxSBCyiPOMnofQ9qck1LlHHVX/rQ7SivHb7xDr1r4Zu7XStRkvIBq4tItRubkxnyiuTmYKSOcfNg1fspfFOn+CdXkvtXgYiSIWktvqBvZIssobdIVXPU8Y6VLfuuXb/gP9R21S/rsep1m6Pr9jrjXi6e7sbKdrebchXDr1x61k6T4dn0+KW5ufEmp3EFzbKJFuZ8+U3Uujfw9f0rnvDetyWWi+KIrPVm1S4sriX7L59wJZCozzx1FDdm0+iv+Ilqlbq/0PSaoanrNppMlml4zKbycW8O1c5cgkZ9Ohrz7TdRns7zQ7yw8R3er3GooWu7KWbzEQ7Nx2r/yzweOfTFYQul1O58J6nfa/eXWpXWrK0lgZ/3MOAwwI/4ccdfWqSvK3nYNLX8r/gevjWtMbWW0kX9udQVPMNr5g8wL67euKvV57cPounfFrT1tZrOAvbTGULIoJkZ8nPPUk16EDkZHIpLWKf8AW4PSTQUVyeu33iCDVXTTRemAAY8nTo5Vz/vNKp/SuksHmewha53eaUBfegQ59wCQPzNRGfM3odNXDunTjU5k79FuvXQsE4BJqhpGtWeuW0s+nszRxTyQMWXb86MVb9Qa5Xxpfyv4o0vR59Vn0fTp4Jppbi3l8p3Zdu1A/b7x471yng7Vf7L0nRDBqUwsJtfvkmnmk/1qfvSpcnHUgHnvWkfe/rzsc7Vl/XZv9D2SivKf+Ek1DU9H8RS6drEoCa2lvbTqc+WnlpkL7ZJrf1Szl8H+GdVv5tc1a/hkjQIk9xmSJyQMrJ/CMnnjgVLdlf8ArVXC3vcq/rU7eoLO9ttQtxPYzx3ERJG+NtwyPevKvAt3reoeJNT0DVtVlMLWCTrJbaq108bFscSlF28dgDWh8LdJtbPTTv1/UHuo5JUeymvQVXkc7OufeqtrZ9r/AI2J6fP9Lnc2ev2N/rl9pNu7m6sApnBQgDcMjB79a068+8O6jHY+NfENjDrcmoosSSW8dzdrIdxUEqv49qx7XVbtrGw1tfEt3Lq1xfCKXS/NzEF3ldvldRwBz75qY6pf11sOWjf9dLnrNFeR6lPrSafrOvL4g1BHstZMEFqkmIhH5qqVZf4uDx6V62h3IpPUjNV0v/Wyf6g9HYWiuC169k1Dx8+i6hrVzothDaLPE9tP5DTuSQw3nj5cDj/arO8d6hcwSaZpGianceetqZzcy6ubVZI14yXVG3seDjHNTf3b/wBf1oO2tv6/rU9OqG6u7extnuLyZIIUxukkbAGTjr9TXmFvqfiDXfhZo16uqRwXDSP9pf7ebd5VVyMLIVOTgegz7U/W1t/FHwjuJrG/1yNbWQK3n3A81yrqDvIyGA68HtRO8VLyCOrS7nqKOsiK6EMrDII7ilri9Oju9N8Es/hbUZtbuGRW3Xd2Jmj+UcLx+O04+tZUl3HbfCzVtQ0TUrv+0nGZ7ib5Zo5dwBBHbHpmlKVqjgjqpYZzw/tr9Urevfsu3c9Jorzi3ub3w14pvba91jUNRtDosl7KJpNzLIpAJj/u98Cuf8JeINdXxxYwT35lstStZpljbVWvGUKMrkGNRGeemTVf8H8L/wCRydL/ANdP8z2eivHtIvNds7HwfrEuvX97Nqs3lz280mYmUxOwAX1yo5qaPWLyTTotcPiS7GrtqPktpQl/dBfO2bPK6/dH3vxqnGzsOSs7f11/yPW6K8w1/UbzSvF7ahrGp3h0zKLFb6ZqADxsf+ekBAyPcEn2qj4q1HxVqPjbUbPSb63sYrNY/swn1ZrMMSoO4qI2Egz7j0qVrYO567RXneoXt9eeJbDRNd1uTSoBpyTvc2U/k/aJiWDASHsAAcd80urTl/EOn6FN4jvLHShZNMl8l0FkuXBUDMvQ8EnHej+vuv8A5C/r77f5nodFeU2t9rOsLoOnLrt5FDNfXUDXcR2yXESBsc+vH3vxqNda1W30WXTJ9ZukhGvNYPqMkmZoodgP3+xz3p2v8/8ANL9Qem/9b/5HpdnrNpf6nfWFuzGexZVnBXABZdwwe/FX68n8ORPHN44i0DWpJ5RJCsd9f3JJ3eWAcyY49AcH8a2PBN/bXVrqelapd6jcXEEStdvd3guYcNkZikHbg9QDUva67J/gVb3redj0Cqa6taSau+mRyb7qOMSSIozsB6Z9K5bwZdXu67h095LzQoci1ubnKyZH8K/3l9zj6VN4DAbR7/UWXzL2e6mMv94lXKqv5AViqt2rdm/uPSqYFUVUc3flsl6vv2sk7rudfRXjXhS+8ZaxrVpqLanZxpJcH7Rby6u5OwE5UWxjG1un8RqW5utbi0y618eIdQMkOvm1ithJiERG68sqy/xcHj0roSu0u/8Aml+p5lt7f1v/AJHqes6va6FpFxqeoMy21sheQqu4gD2qO61/SbG4tIL3ULe2mvTi3ilkCtKeuFB615D46vzq+ieLJdW127tJLKT7Pa6bDNsSRdisCyfxZLHn2rp/G39kWknh+4d7WK9e+h812cByojfGcnpzSWyfdr8bf5ie2nZ/gekUU2ORJo1kidXRhkMpyD+NU9Ul1OK3U6Pa21zLn5luJzEAPqFak3ZXKhFzkor8dC9VPUtVtdJS3a8ZlFxOlvHtXOXY4FS2bXL2kbX0UcU5HzpE5dQfYkDP5VynxMBOiaYFuhZn+1bb9+2MR/N15Ip9V8g5bNpnZUV5Le65q2iWPiWDQdVuNYgtY4DDdXM3mGJ3bDDzAOmM9AcU7Q7jxZpuj63e3GrWU8EenvLAP7Va+dZQpIbJjXavTilfRsSWqX9f1qesVmWev2N/rl9pNu7m6sApnBQgDcMjB79a858DS6jqvii3nOq64lpFasbmHUL+NxJIcYKKrH5Rz1x1Fbfha+htfHuu6cNekvodsbwrc3ayEMVBIX9eKb0av2f4E30flY76qJ1vS11kaS1/bjUSnmC18weYV9dvXFeY2NxrUGn6HrkviDUJ5bvWPsr27yfufKMzLgr3OB1rY1N9HsPixo32eW0hkkiuGnxIoJkLL156mnazSfe34FPS67f52/Q7f+17MayNKaXbdmPzVjYY3r6j16VdrkfHYEDaPewj/S4r+JI8dSrMAw/ImpvF+orp7aVJNqN1YrJchSLeMMJMj7rcjA/OsPaWvfo7ffb/ADPRjg1VVJ0/tX89V6dGdRRRRWx5oUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBmDw1oQunuRounCd2DtL9kTczA5BJxknPOaWfw5ol1dvdXOjafNcONrTSWqM7D0JIya0qKAKN9omlapDHFqemWd5HF/q0uLdZAn0BBxTLzw9ouoJEl/o9hdLCNsSzWqOIx6LkcfhWjRQA1I0ijVIlVEUYVVGAB9KwfEXhaDxFq2kT30Nrc2tjJI8lvcxCQSbk2jggjg810FFHW4bFQaRpo03+zhp9qLHG37L5K+Vj02YxUdpoWkWFq9tY6VZW0EhDPFDboisR0JAGDV+igCOW3hnt2gnijkhZdrRuoKkehB4xVCy8M6Dplx5+naJp1pNjb5kFpGjY9MgZrTooAoWWhaTpt1Jc6dpdlaTy/6yWC3RGfvyQMmki0DR4bprmHSbGO4dxI0qWyB2YdGJxnPJ5960KKAMmTwn4dmuzdTaBpclwz7zM1nGXLZzu3YznPetYAKoCjAHAA7UUUAFFFFAFW/0rT9VjWPVLC2vURtyrcQrIFPqAwPNRy6HpNxYmzn0uzltWYuYHt0KFic52kYznnNXqKAKcOkabb25gt9PtYoSwcxpAqqWHAOAOvA5qxPbw3Vu8F1DHNC4w8cihlYehB4NSUUb7gUrDRdL0s50zTbOz+Xb/o8Cx8ZzjgDjNNj0HSItQkv4tKskvJAQ9wtugkbPXLYyav0UAZNv4V8PWl2t1a6DpkNwpys0dnGrg+u4DNTx6FpEOpNqMWlWUd833rpbdBKfq2M1fooArtp1k8MkT2du0cj+Y6GJSrtnO4jHJzzmsC4Hjv7TJ9jPh0Qbj5fmrPu29s4OM109FAzJ/saLV9OgXxZp2mX1zHyR5AkjU/7O8Eiprjw/o13FBFdaRYzx2wAgSS2RhEB02gj5fwrQooEZ1x4f0a7s47S60iwntoiTHDJbIyIT1wpGBVmHT7O2sRZW9pBFahdogjiCx49NoGMVYooAqafpOnaRC8Wk2FrYxu25ktoVjDH1IUDJp72FnIsyvaQMs5zKDGCJP971/GrFFA02tiFrO1eYzPbQtKYzGXMYLFD1XPp7VVs/D+jafL5lhpFjavkndDbIh568gd60KKBFZdOsljgRbO3CWxzAoiXERxjK8ccEjj1qFdC0hdTOpLpdkL49boW6eaeMffxnp71foo8wM2Tw5ok2ofb5tH0+S8zu+0taoZM+u7GafqGg6Pq0qS6rpVjeyRjCPc2ySFfoWBxV+igCle6Lpep28cGpabZ3cMX+rjngWRU+gIwKZceH9Gu7GGyutJsZrWDHlQSWyMkf+6pGB+FaFFAECWNpGIBHawqLcYhCxgeUMY+X0444qObSNNuLaa3n0+1lgnbfLE8KlZG9WBGCfc1booAoLoWkJZyWiaXZLbSgLJCLdAjgdAVxg4pbLQ9J063kt9P0uztIZf8AWRwW6Ir/AFAGDV6igBscSQxLHCixoowqqMAD2FY2naBJpXiC7u7O4Asrwb5LYj7sndl9M1t0VLim0+xrCtOCkk9JbmcPD2irqX9oLo9gL3du+0i1TzM+u7Gc1YOm2LQmFrK3MTSeaUMS7S+d27GOuec+tWaKoyM+40DR7u7N1d6TYz3DLsM0tsjOV9MkZx7VHe+GdB1O48/UtE067mwF8ye0jkbA6DJFalFAEcFvDa26QWsMcMMYwkcahVUegA6VJRRQAVXvtPstTtTbalZwXkBOTFcRLIpP0IxViigCnaaRpthZNZ2On2ttbPndBDAqI2euVAxWPq3gvTZ/Dep6boVnaaPNfW7Qme0t1iIJGBkqATXSUUPUadndHnvh/wCH15Za7Y32oW+h2aWUTRgaTbGNrjdj/WEjkcdPeust/Cvh60u1urXQdMhuFOVmjs41cH13AZrWopk2K39nWQhjiFnb+XE/mRp5S4Rs53AY4Oec1Tm8K+Hri9N5PoWmS3TPvM72cZct1zuIzn3rVopDMe70R7/xHbX93MGtrNcw24H/AC0P8R/p9KXVvD8etXls99cO9rbuJRa7V2lx0YnGe/TOK16Kjkj+NzoWJqxcWnblVl5f8Hz3CiiirOcKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKx7zxd4e0/WI9JvtasrfUJWVUtZJlEjE9AF680Xni3w/p+sR6VfazZQahLjZayTKJGz0wvXtRuGxsUVUfVbCL7V5l3Cv2NQ1zlx+5GM5b045rm9d+J3hjw/q9hY32q2aG8BYu1yqiJMDDHPY54+lFx2Z19FV7DULTVLGK8065juraZd0c0TBlceoIqS4uIbS2kuLmRYoYlLO7nAUDqSaHpuJJydkSUVR03WLTVdLXULRz9lYErI67QQO/wBPem6Prdjrts9zpkhmgVyglA+VyDglT3HvS5k3Y1lRqRUm4v3dH5PzNCiobySaKyme1QSTKhKIf4j2Fct4e1vxXqMhTVtIisydPSZWMbgCcjlDk9Ae3WumnQlUg5pqy8zFu39f13Ovorn/AAh4ik1/SJHvkjgv7WZ4LqJAQEZSemecEYP41Lb+MvDl5fpY2et2M93J/q4EnUs/0FEsLWjOUOVtx3tr/SFzK1zborip/HGrw+Lk8P8A/CLlriRDKj/b0AMYON3Tj6Vb1DxY0XjDTNGsbrSy8pP223luB58fTGxc89+1a/Ua6aTS1V909F10bBySv5HVUVjXvi/w7pt01tf61Y20653RyzqrDHXinXnizw/p1hb319rFlb2tz/qZpJgFk+h71isPWdrQeu2j19B3Rr0VFbXMF5bR3FrKk0Mg3JIhyGHqDWJ4v8ZaZ4N0lrzUpohIceVA0oRpeccZ9Ov4VNOjUq1FSgryelgurXOgorBs/Gvh290M6tHrFkLNMCWXz1KxMRnaT2PNUNd8dWUPgy91vwzeWWpm1ZFykm9ASwGDtPua2jg8RKahyNa21XVgpJ7HW0Vl2HiDT7p4LV723GoSW6zvbBxuUFc5x1xTLDxZ4f1WeaHTdZsrqSAFpUimDFAOpOOlZOhVV/denkCkmrmvRWNY+L/Dup36WOna1Y3N067lhinVmYYzkAe1bNTOnOm7TTT8wTT2CiuP1rxNqp8bW/hrQVs45mtmupZ7xGdQoKjACsOfmrXvfFOi6NcQWWt6xY2t7IgIiklCFyeMgE5xmtnhKqUbK7krpLV27i5km12Nmis2+8RaPpm/+0NTtbbZGJW82ULhD0b6VdtLu3vrSK6s5kngmUPHJG2VdTyCDWDpzjHmadh3WxLRWD4x8RL4c0CW5S6sLe7ZSLYX8wjjd/QnIqZPENnZ6HZ32u31nafaEHz+aBGzYzhSTzwD+VarD1XTVRLRuyC+tjYorKtfE+h32lTanZ6taT2MGfNuI5gUTAycntS6X4l0TWreafSNVtLyKDmV4JQwT6kdOlS6FVJtxem+j0DmRqUVk6b4q0HWVnbStXs7xbdS8xhmDeWB3OOlIvizw+90LZdZsjO0RmEYmXcUHVsenvR9XrJtODuvJhdGvRXH6F8TvDeu61d6bb6jaJLFN5UH+kKftPXlB+FWLbxX9u8eHR7C80u4tIoSZlScG4SQZ4256dO1bSwOIg3GcWrK+vYXPG1zqKKyLzxZoGn6oum32sWVvevjbbyTBXOenFN1Hxf4d0m6a21PWrG0nUhTHNOqsCRkDB9RWSw9Z2tB66rR7DujZoqleazpunW8M99fQW8U7BYnkkChyegB75q6DnpWTjJK7QXQUVj+Ir7VrCGzbRbJbtpLuKOcMpOyIsAzcHsCTSeG9S1O90RrnxDaLYXCyOCm0qAo6Hk1q6MlS9tdW9dfuC+tjZorP0jW7HXbV7nTJTLAjlPNx8rEHBIPce9Jpeu2Gsy3KabL54tZPLkkUfJu7gHviuZSTt5m8qFWPNzRfu7+XqaNFNllSCF5ZnCRopZmY4CgdTWTp3i3w/q+omw0zWbK7u1QOYIZlZwpGc4HbFV5GPmbFFY8Hi3w/c64+jW+s2UmpISGtFmUyLjr8vWo7vxp4ZsdQSxvNdsILuSTy0hknUMzZxtA9c8ULW1uoG5RXMX/AIrSPxtp2hWF7pjvIGN3BJOBcIMDaUXPPftWjq/irQdAuIbfW9Xs7Ca4/wBUlxMEL9uAevUULVXDrY1qKyNc8U6J4ds47jWNVs7FJh+5a4lCCQ4zxnrXN2/xDuP+Ff2PiFtKe9e4jMri1ysSKGxnecgdRgd+fSlfRvsOzdvM7uiobS4F3Zw3AXaJUDgE9MjNOuLiG0t5Li6lWKGNSzu5wFHqTTl7u4lrsSUVj6Z4t8P61a3NzpOs2V5BajM8kEwZYv8AeI6dDSad4w8O6tZXN3pmtWN3b2gLXEsM6ssQAySxHTigDZoqo+rafHPFC95Css0bSxoXGXRcZYeoGRz71QsvGXhvUb42dhrljcXIj80xRzqzbMZ3Y9Mc0AbVFcbo/wAUvC+seI7nRoNVsluI5xBAPtKk3LED7o+vH4VrXuuzDxJbaNp0CyykebcyN92GP/Ekj9alySS8zWnRnVk4x6av5G5RWJf67NpniS0s7uBfsV6NkU69Vl/un68fnW3RGSlsFSlOmoyltJXQUVV1LUrTSNPlvdRnWC3iGWdjj8PrTRqtp/ZA1OWXyLTy/NMk3y7V9TnpRzJX12EqVRxUknZu3z7Fyiqml6nb6xp0V9ZFjBKMozLtyPWq/iO81Gw8PXl1olqt3fRRFoYGUkO2OBgc1pTi6klFdSZRlCTjJWaNOisHw7qetX93qEetaetpHC6i3ZUYeYCOTyecGtqeeK1t5J7iRYoo1LO7nAUDkkmrqUpU58j1flqRF8yuiSisjT/Fnh/Vr5rPTNYsrq5VdxhhmVmA65wKZZeMPDmo6ibCw1uxuLsZ/cRzqz8deKp4esr3g9NXo9g5kbVFYieMvDcmoR2Ka5YNdyOY0hE67mYHBAHrkYqS88WaBp+qLpt9rFlb3r4228kwVznpxR9Xr3tyO++z2DmSNeisnVfFWg6HdR22savZ2U0oykc8wRmHsDTtU8S6JokMEur6raWUdwMxNPKFEn0z160lQqu1ovXbR6+ncLrY1KKz73X9J07TE1G/1G2t7KTGy4kkCo2emDVdvFvh9dFGrtrNkNOLbRdGYeWTnGN3TrQqFWSuove2z37eoXRsUVkjxToJ0uPUhq9mbKR9iXHnDYzegPrTrHxPoepadPf6fqtpc2lvnzZ4pQyR455Pah0KyV3F9tnv2C6NSiuI0f4h2uuePLrStMurG70uCzWc3UD7iHJbIJBxjAH51u2PivR9dW4h8OavY393HGWCRSh8HtkA9M4rWrgsRSdpxeyfpfa/YSlFu1zaorn/AAh4ik1/SJHvkjgv7WZ4LqJAQEZSemecEYP41NbeMvDl5qEdhaa5YzXcv3IUnUs30FRLDVozlDlb5d7ahzK1zaorndD8QXd14k1XRdWjhiuLRhJB5QIEkJ6NyTzyAferF54y8N6feG0vdcsYLhX8sxSTqGDYBxj1wR+dOWFrKfIo3dr6a6PqPmWt+htUVyniPWvEthqUkeh6THeWwszKkhRmzLvQbeD/AHWY/hW62qwWen2s+rTRWbzKgIkbb85A+UZ75OKipRdOnGpJqz8/zHFOUuWKuy9RQCCARyDRWIBRRRQAUUUUAFFFFAHn3hpdQ8N6neafqXhy9vXub95U1K2WN49jtkFizBhtBA4B6Vm3WlapBa61oLeG7i+n1K7M0OpDZ5IVsY3sTuBG09FPWvU6KOln2t+X+Q76387/AD1/zPLtdsfEGnHxHZWOh3Wpvq1nHHDcRMojVlhCNvJOQeuMA/hWlq+k3tpp/hXUE0ia/k0xVW5tbdVaU5VQcbiAcYPeu/oobb/D8BeXlb8Ev0KunXBu9PhnNnNZb1z9nnVVdPYhSR+RrJ8X+HZvEGmBLW4CTwnfHFKN0MjDkB19PftXQUVM4qaszWhWnQqKpDdHMXOlXnibwelrextpV0BhoVO6MkdiO6n8KveF21AaLHb6tYR2Vxb/ALorCcxuBwGX2PXFbNFJQtLmNp4qU6bpWVr3Xl6f8G4UUUVZyHDSwS+H/iYzwQyNY69ARIUX5Y50HU/VQBXKWHgrV7TwposcekvHew62bmfG3csZz8xOenSvZKK9anmlSmkopX0v52TS/B2+REoKV13/AFVjjbrSL9/i9ZaqlsxsY9PeJp8jAYsOPWsvxj/aU3jzQ7ux8N6ldQaXMzTTwpHiQMFxty4J6HrivRqKyp4+UJxm4p8seXrs7/5jcU7+f/AX6HkWseEdU1LxDc3o0eQxz6mk2XC58sQyjJ57Fl/OqK+DNbtbrSLy/stWktY7eaCSDTJlWaImV2BwSAVIYd69rorrjnVeMVHlVkrde1u5Lppu/wDW9zD8HaXbaP4Zt7Sxtr21hGWEN8waVc+pBI/WoPH2k3GteCtQs7GFZrpkBjQ9WIYHA9ziujorzFiZrEfWOt7/AI3LiklY881ebxBe/D+GHRNG1DTrqJ4oriJkjWZ0AAYx4YjPoSRXK/8ACHa9LpPixY9P1LGorbm1F3KrTNhsnJBwCBXttFehRzadCLjTgt79e6ffyI5NtdjyPRfB2u6XpOu+H/sjySalZk2+rsQCGaPHlyHOeDwCM9qPDPhLYtub3SPEcep2tpJEktzPG1sCeoXDZwcDqK9copyzivJSul729r9rd+q6bAqaVv67f5Hjmi+C9X07SfBIj0p4J7K7eS927QY1ZG5PPOSa9joorjxmNqYySlNbX/Ft/qOMFF6Hmk3w8spvi+NRk0KJtMayZmlIG0z70IPXOetUtX0PWbS+8V2i+H5dU/ttAtneRlSkIMYTa5JBUAjPANesUV0xzaumnLWyS1v0d77+Q+VXb+Z5xpng68i8c6dJqVs1xZW2kiB5WIKNJnoR34zXfWGn2ml2Udnp1ulvbxjCRRjCqParNFceIxdXEW5+n/B/zFGCjt/WiX6HJfEqO7u/Bd7p+naVdajcXkLxxi3VCIzjgtuYYH0zXMaro2qa/wCEfC9m2hXcT2F9EbmG4CDCqjAtwxyM4r1Sit8PmEsPTjCMVo2769VbuOUeb7mvvPLW8JyNeeL49V0e/m0y9uYngisSoeTCIMrkgDBB6+lVrHQfGWpeC9f0a4MyW7xqlg96ixXDYJyGKFhjGMHNet0Vt/a9W1uVP4XrrZxtqvW2vcXIr387nlnhvwwohlmk0nxFBqaaY9uGv5kaEkpjYuGJ69Miuk8FeE7ax8L6V/a2nINStt7b5QC6MWPf6AV19FY4jMq1ZNbXa6vzVvTUUaaj/Xp/kcHoFre6H461xLjQrqW21G5WWC8gRDGgw2d2WBHUdAapXD6n/wALct9Xi8Map9iitmtWlWOLDMWb5vv/AHeevX2r0mimswfO5uCbceV79rX3G4aNd3f8bnkV9oWtpbeI9HPh+a8n1a782DUgV8tEOPvEncCuD0B61NP4K1H+0/FMkmnm4afSUtrWcgfvXEajA59RXq9Fbf2xWS92KX3/AN3z/uryDkV7/P8AG5wc9jfQeBdAtbnw0utXMJjWWCZwDBj+P6iu7X7o4xx09KWivPrYh1lqrat9evzt93zCMeVJeVgrI8TaG2v6PJZpcvAx5XHKsfRh3Fa9FckoqSszelVnSmqkN0cvZabf6x4PbStWgOk3Ea+STaN+7cDgMo4+U46cdaseEItRstNfTdUsIrc2jbI5oOI516hgOx9RXQUUlC0ubqdNTGSnCVNxVm7+j8v+DczvEP8AyLOp/wDXpL/6Aa838LWFzrUHhVdP0C60e30238yS8uERRKWjx8m1iTk884616rc28d3ay2867opUKOM4yCMGmWVnDp9jBZ2q7IIIxHGuScKBgDJq7LW/9b/5nG3pY8o8MeEZbXUrGy1zT/EzzWdyZFnS5RrHcCcMBkNjnpiuy8P+GohNrR1XToyLu/aVTIoO9QxKn+VdZRR6/wBbf5Cev9ev+Z594rub5PGWjS2HhbVbuHTXcyT20cW1wQuNpLgnGO+Kh1C31Gx13Wpn8NXmqprMKCCSNUPkHBG2Tcw2j5h93PSvR6KSWlvX8R31ueXR6LrXh7U7KaTRJNXzpS2aCBlZbaXcT8xYjCc4yAT7UPbazpnwbtdDbw9qF1qE8DI0VqEYQnfn5izD9M16jRRL3k0+v/B/zCLs0+3+VjnbHU9RTwRHdW+iXS30MQRbC6KpIxHHVSwx3HNUfFtlq3iDwPA0VgBeJLHcTWDSf6xVOTHnvmuwood3NyuVGUVS9ny/PW9rWt2/C5wetC98U+EbqCx8PX2mywSRSG3vEjj+0hWyVGxm9O+O1Z95pWpeKDqt1aeH59GV9IkskjvAiPLIwbHCFhtGRznPtXplFP8Ar8LEptNP+t7nl1tb6/rHirTribw9d2FpZaTcWrPcsgZpGCAAAE8Hbwc/lUuleEbuwtvAYh0vyWsCTfbQAYyYHB3evzGvTKKpSt/Xr/mLpbyt+hw2nWt1oXxB1LfoV1c22qTRyRXlvGhjg4CnflgRjGeAetW9OddP+JGqRXZCvfxRvbs38QXIIH5iuuqnfaTZajNbS3cIeS1k8yFwSGQ4x1H16Vi4v3Wun5HXh60Yc8Z7SVtPk1+KOe8cf6XNpGm2/N1LexSqB1VEYMzfTANdYowoB7Cqw020GqHUfKBujH5fmE5IX0HpVqnCLTbfViq1oypQpR+zf73v8jlvF/h681FrfUtMZJ7qyYSJZ3PMUuOuP7rY6HmjxFod54k0G3mhH2W9hAkWzuPmhZh/C4H866mipdKLv5/ma08fVpqFrXhs/J9PT8fMo6NcXNzpMEl/ZfYbgrh4MghT7H0q9RRWpxTkpSckrXCsvxLbTXvhbU7a1jMk01rIkaDqzFSAK1KKuE3Cakugk7O55TpvhLVtOl8GPZaaYHsbeZbphgCMt0Dc81j2fhzxZfa9oV7qllqjXNres9680qeQoKMAY1Bzj6469K9uor2Y51WV24pt3V9b6uT7/wB78EY+yVrf1tY8WHgfWI/C92sWkOt/J4hF2CNu4xiYHfnPTaKtXWga5FY+INFfw/NeXGq3Ikh1IFTGinH3iTuBXB6A9a9foo/tqs3eUU9b9d9PPy2H7NXv/XX/ADPKPFXh7VbPVILzwxa6udaigihe5SOOS0uAFAw4LgjHUnB6UeI/DOtXHi6y1fU7bUru3k05beaPRplV45ASTwxA2nd616vRUQzerFR91aJq/Wz8/LoHs108vwPKrrwtcaTN4V1DSNG1C807Td6yadOytcR7l4bBO0kYx171p+IodR1C/wDD+uW3h68ktrGZzPpkioJRkEBwu7acdetehUVm8zqSlGUoptXXXZ3v183ruHs1a3yPHL/wbrGp6ZPcNpDxpfa3HdfYGK7oYgCCW5x+AzUniHwdrlxqXiZtOsXFrPPazRRqQBcKhQuoHrwetev0VtHOq8ZJqK09e8X3/uoHTT3/AK3/AMzym10S/wBS8W6pqGo+HL210ufSFthAmwSuQWyoAbAbn19Ku/D/AE/xDpevzWi297D4cSI+SmqRoJ42yNqqys25QM9cdq9JorKpmk6lOVNxVmkurtbqr9fMPZr+vl/kcNLBL4f+JjPBDI1jr0BEhRfljnQdT9VAFcpYeCtXtPCmixx6S8d7DrZuZ8bdyxnPzE56dK9koop5pUppKKV9L+dk0vwdvkOUFK67/qrHEeNYZ9I8Q6N4nsYXl8mUWt4ka5LQvwD+DFT+Fctqvg3U7xfHlxFpLSTakE+xPxmUBB0545zXsFFKhmdShGKildaX8lJSS+9fcNxTd/6/r/IpaLFNDolnHdArMkKhw3UHFc58Rbl7bR7V7e1tbyZbqLbDMu5jlgMqPX37V2FQyWdtNcRzy28TzR/ckZAWX6HqK8iterfpd3/G51YKrHDVYTkrqPTuPhJMEZYbSVGR6cU+iiqMHqwooooEFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAf//Z)

**Fibonacci heap:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | **Insertion** | **Deletion** | **Find min** | **Search** | **Extract min** |
| 5 | 0.000002 | 0.0000007 | 0.000001 | 0.0000015 | 0.000001 |
| 10 | 0.0000022 | 0.000001 | 0.000001 | 0.0000021 | 0.0000015 |
| 15 | 0.000002 | 0.0000012 | 0.000001 | 0.0000031 | 0.0000021 |
| 20 | 0.000002 | 0.0000015 | 0.0000012 | 0.0000042 | 0.0000023 |
| 25 | 0.000002 | 0.0000019 | 0.000001 | 0.0000051 | 0.0000028 |
| 30 | 0.0000021 | 0.0000022 | 0.000001 | 0.0000063 | 0.0000029 |
| 35 | 0.000002 | 0.0000024 | 0.000001 | 0.0000074 | 0.000003 |
| 40 | 0.000002 | 0.0000025 | 0.000001 | 0.0000092 | 0.0000032 |
| 45 | 0.0000019 | 0.0000028 | 0.000001 | 0.000011 | 0.0000035 |
| 50 | 0.000002 | 0.0000028 | 0.0000009 | 0.000013 | 0.0000036 |
| 55 | 0.000002 | 0.000003 | 0.000001 | 0.000015 | 0.000004 |
| 60 | 0.000002 | 0.0000034 | 0.000001 | 0.000017 | 0.0000042 |
| 65 | 0.000002 | 0.0000036 | 0.000001 | 0.000020 | 0.0000041 |
| 70 | 0.0000021 | 0.0000037 | 0.000001 | 0.000022 | 0.0000043 |
| 75 | 0.0000022 | 0.0000038 | 0.000001 | 0.000025 | 0.0000044 |
| 80 | 0.000002 | 0.000004 | 0.0000011 | 0.000027 | 0.0000044 |
| 85 | 0.000002 | 0.000004 | 0.000001 | 0.000030 | 0.0000045 |
| 90 | 0.000002 | 0.000004 | 0.0000012 | 0.000033 | 0.0000045 |
| 95 | 0.000002 | 0.0000041 | 0.000001 | 0.000036 | 0.0000045 |
| 100 | 0.000002 | 0.0000041 | 0.000001 | 0.000039 | 0.0000046 |

**Treap:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | Insertion | Deletion | Find min | Search | Extract min |
| 5 | 0.000003 | 0.0000008 | 0.000001 | 0.000001 | 0.000001 |
| 10 | 0.000003 | 0.000001 | 0.000001 | 0.0000012 | 0.000001 |
| 15 | 0.0000033 | 0.0000015 | 0.0000011 | 0.0000014 | 0.0000015 |
| 20 | 0.0000034 | 0.000002 | 0.000001 | 0.0000016 | 0.000002 |
| 25 | 0.0000036 | 0.0000023 | 0.0000009 | 0.0000018 | 0.0000022 |
| 30 | 0.0000037 | 0.0000024 | 0.000001 | 0.0000019 | 0.0000025 |
| 35 | 0.0000038 | 0.0000026 | 0.0000012 | 0.000002 | 0.0000026 |
| 40 | 0.0000039 | 0.0000027 | 0.000001 | 0.0000021 | 0.0000027 |
| 45 | 0.000004 | 0.0000028 | 0.000001 | 0.0000022 | 0.0000031 |
| 50 | 0.0000041 | 0.000003 | 0.000001 | 0.0000022 | 0.0000036 |
| 55 | 0.000004 | 0.0000033 | 0.0000011 | 0.0000023 | 0.000004 |
| 60 | 0.000004 | 0.0000034 | 0.000001 | 0.0000024 | 0.0000042 |
| 65 | 0.000004 | 0.0000036 | 0.000001 | 0.0000025 | 0.0000043 |
| 70 | 0.000004 | 0.0000037 | 0.000001 | 0.0000027 | 0.0000043 |
| 75 | 0.000004 | 0.0000038 | 0.000001 | 0.0000029 | 0.0000044 |
| 80 | 0.000004 | 0.000004 | 0.000001 | 0.000003 | 0.0000044 |
| 85 | 0.000004 | 0.000004 | 0.000001 | 0.000003 | 0.0000044 |
| 90 | 0.000004 | 0.000004 | 0.000001 | 0.000003 | 0.0000045 |
| 95 | 0.000004 | 0.0000039 | 0.0000009 | 0.000003 | 0.0000045 |
| 100 | 0.000004 | 0.000004 | 0.000001 | 0.000003 | 0.0000046 |

**K-ary heap:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | **Insertion** | **Deletion** | **Find min** | **Search** | **Extract min** |
| 5 | 0.0000027 | 0.000002 | 0.000001 | 0.000001 | 0.000003 |
| 10 | 0.0000029 | 0.0000022 | 0.000001 | 0.000002 | 0.000004 |
| 15 | 0.0000033 | 0.0000025 | 0.000001 | 0.0000035 | 0.0000045 |
| 20 | 0.0000035 | 0.0000028 | 0.000001 | 0.0000039 | 0.0000055 |
| 25 | 0.0000036 | 0.000003 | 0.000001 | 0.0000043 | 0.000006 |
| 30 | 0.0000038 | 0.0000033 | 0.0000012 | 0.0000053 | 0.000007 |
| 35 | 0.0000038 | 0.0000036 | 0.000001 | 0.0000064 | 0.0000085 |
| 40 | 0.0000039 | 0.0000039 | 0.000001 | 0.0000082 | 0.00001 |
| 45 | 0.000004 | 0.000004 | 0.000001 | 0.0000096 | 0.000011 |
| 50 | 0.0000042 | 0.0000042 | 0.000001 | 0.000012 | 0.000012 |
| 55 | 0.000004 | 0.0000046 | 0.000001 | 0.000014 | 0.0000125 |
| 60 | 0.0000041 | 0.0000048 | 0.000001 | 0.000017 | 0.0000129 |
| 65 | 0.0000039 | 0.0000049 | 0.000001 | 0.000019 | 0.000013 |
| 70 | 0.000004 | 0.0000052 | 0.0000009 | 0.000021 | 0.000013 |
| 75 | 0.000004 | 0.0000055 | 0.000001 | 0.000024 | 0.0000129 |
| 80 | 0.000004 | 0.0000059 | 0.000001 | 0.000026 | 0.0000129 |
| 85 | 0.0000041 | 0.0000063 | 0.000001 | 0.000029 | 0.000013 |
| 90 | 0.000004 | 0.0000068 | 0.0000013 | 0.000032 | 0.000013 |
| 95 | 0.000004 | 0.0000073 | 0.000001 | 0.000035 | 0.000013 |
| 100 | 0.000004 | 0.0000078 | 0.000001 | 0.000038 | 0.000013 |

**Graph comparisons for each operation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Insertion: |  |  |  |  |  |

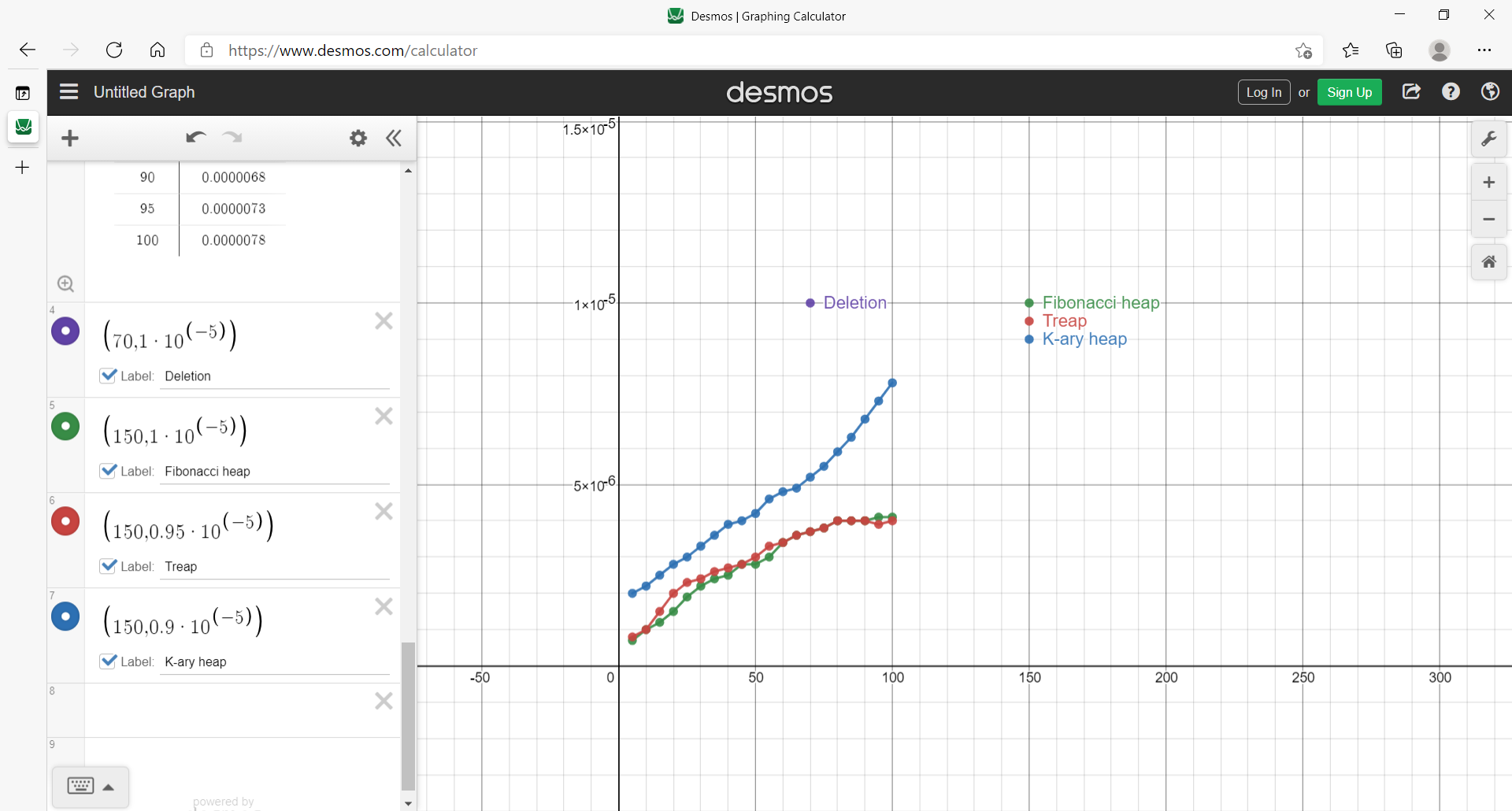


Inference:

Insertion is faster in the fibonacci heap as it inserts element directly to the root list. From the graph we can see that the graph is constant. Therefore the time complexity is O(1).

In treap and k-ary heap the value keeps on increasing and becomes constant after some N. This is similar to y=log(n) graph. So the time complexity of treap and k-ary heap are O(log(n)).

Deletion:

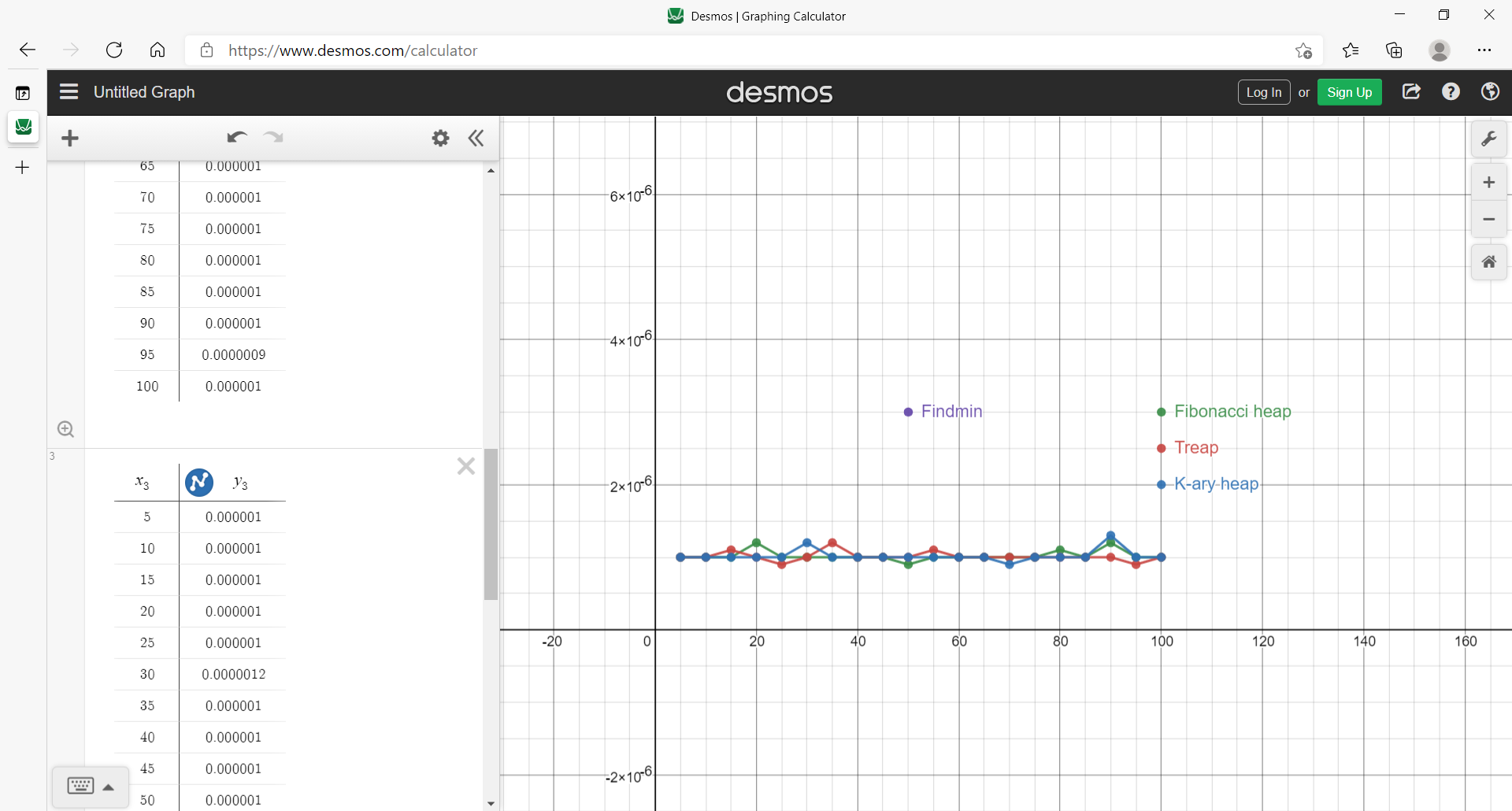


Inference:

In treap and fibonacci heap the value keeps on increasing and becomes constant after some N. This is similar to y=log(n) graph. So the time complexity of treap and k-ary heap are O(log(n)).

K-ary heap deletion is slower than treap and fibonacci heap. The graph is similar to y=nlog(n) graph. Therefore the time complexity of k-ary heap is O(n\*log(n)).

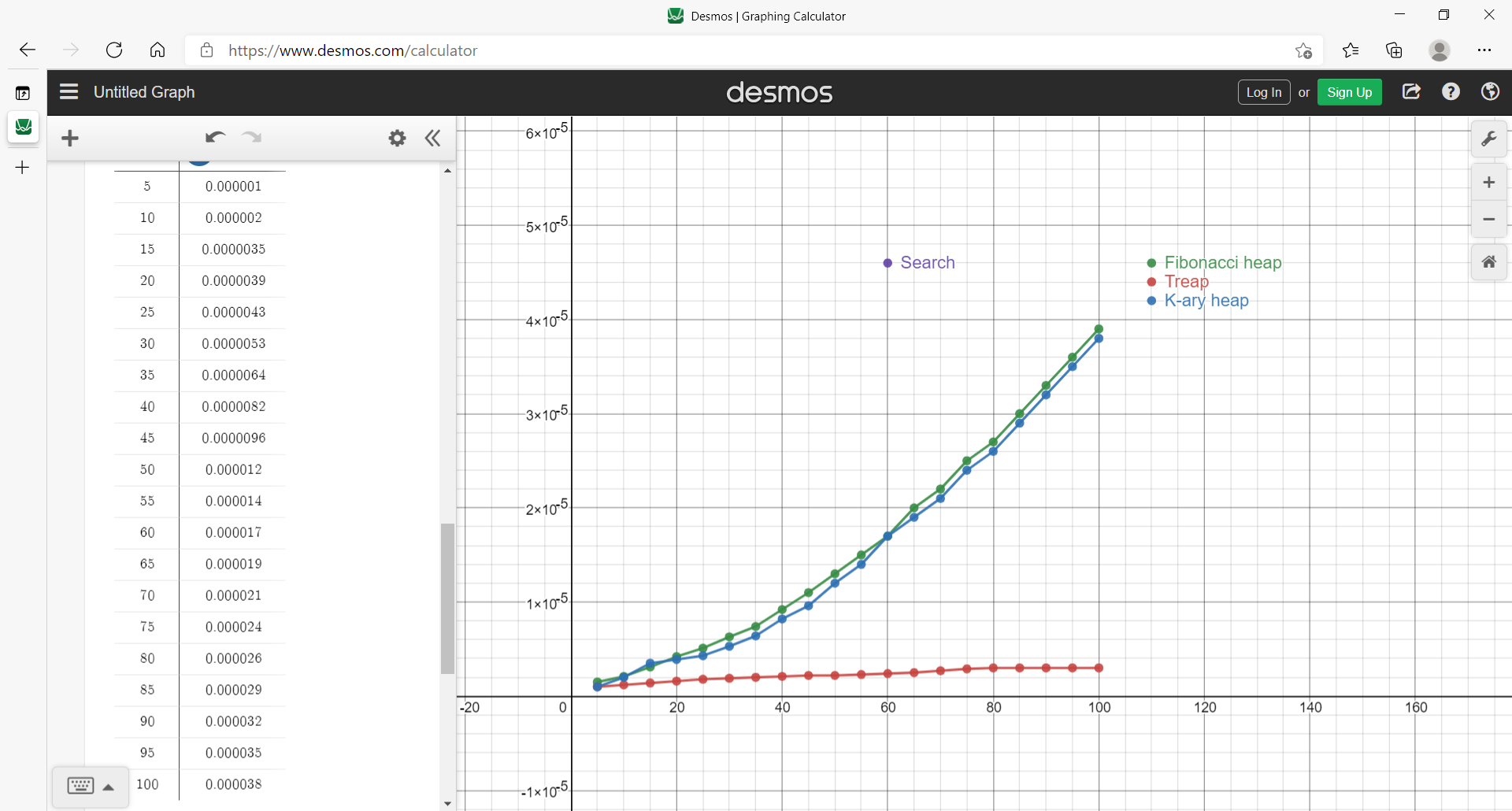
Find min:



Inference:

Here, all the graphs are similar. They are constant for varying n. So the time complexity of Fibonacci heap, treap and k-ary heap are O(1).

Search:

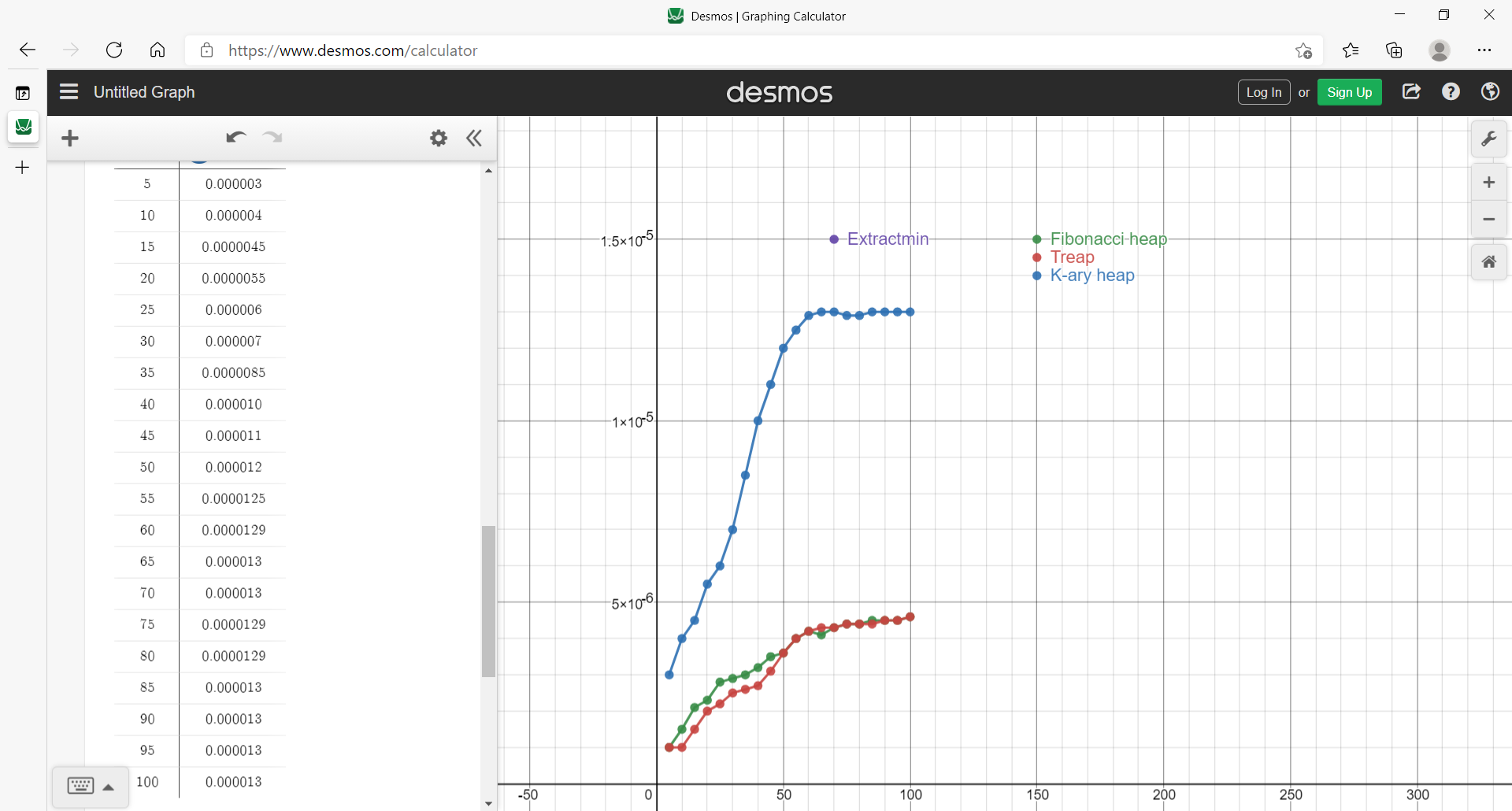


Inference:

Treap is faster than k-ary heap and fibonacci heap by a large margin. It’s value increase and becomes constant after sometime. Therefore, the time complexity is O(log(n)).

Fibonacci heap and k-ary heap graph increases linearly. Fibonacci and k-ary heap are worst in searching a element. The graph is similar to y=n graph. Therefore the time complexity of Fibonacci and k-ary heap are O(n).

Extractmin:



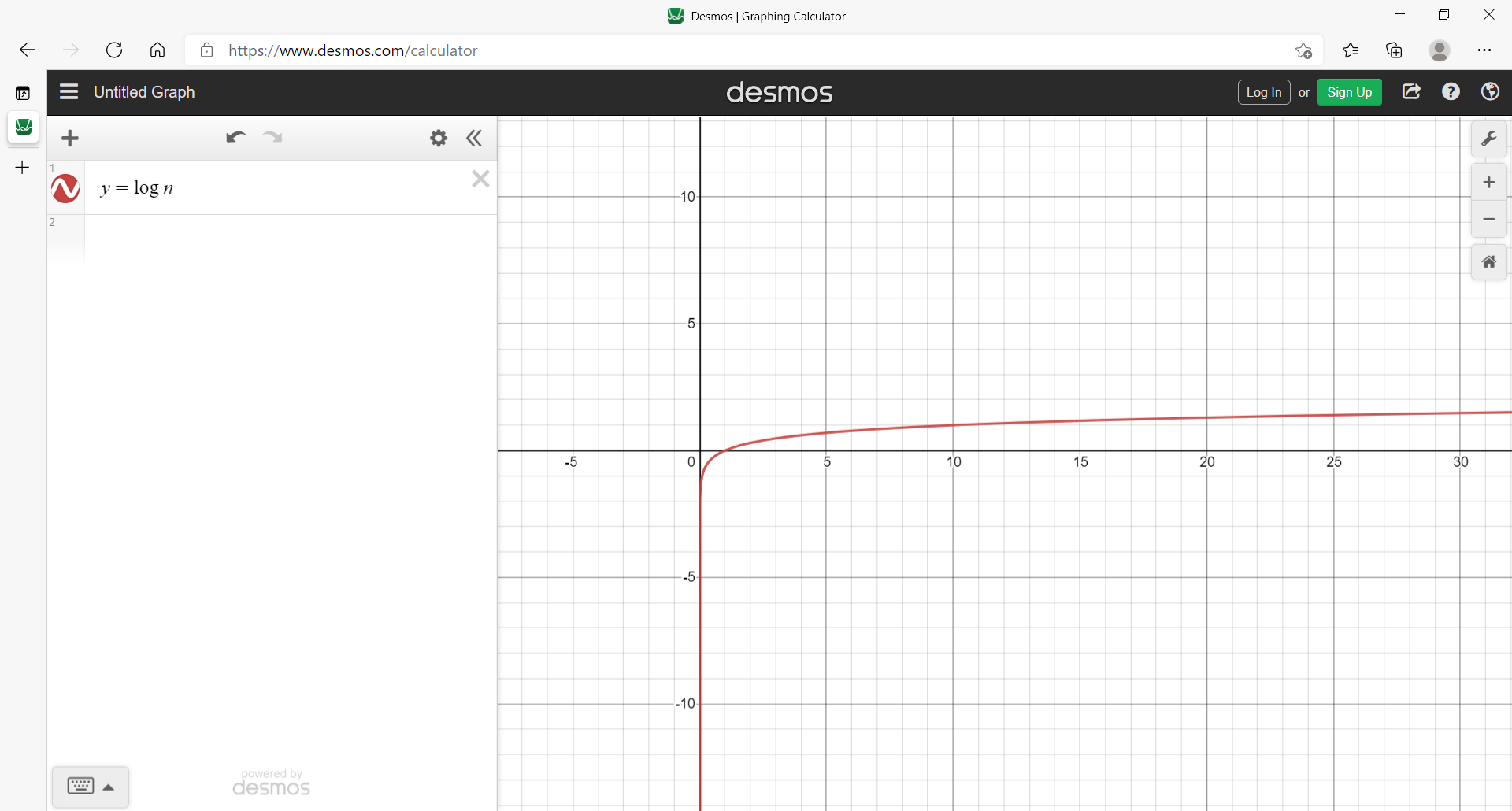
Inference:

In treap and k-ary heap the value keeps on increasing and becomes constant after some N. This is similar to y=log(n) graph. So the time complexity of treap and k-ary heap are O(log(n)).

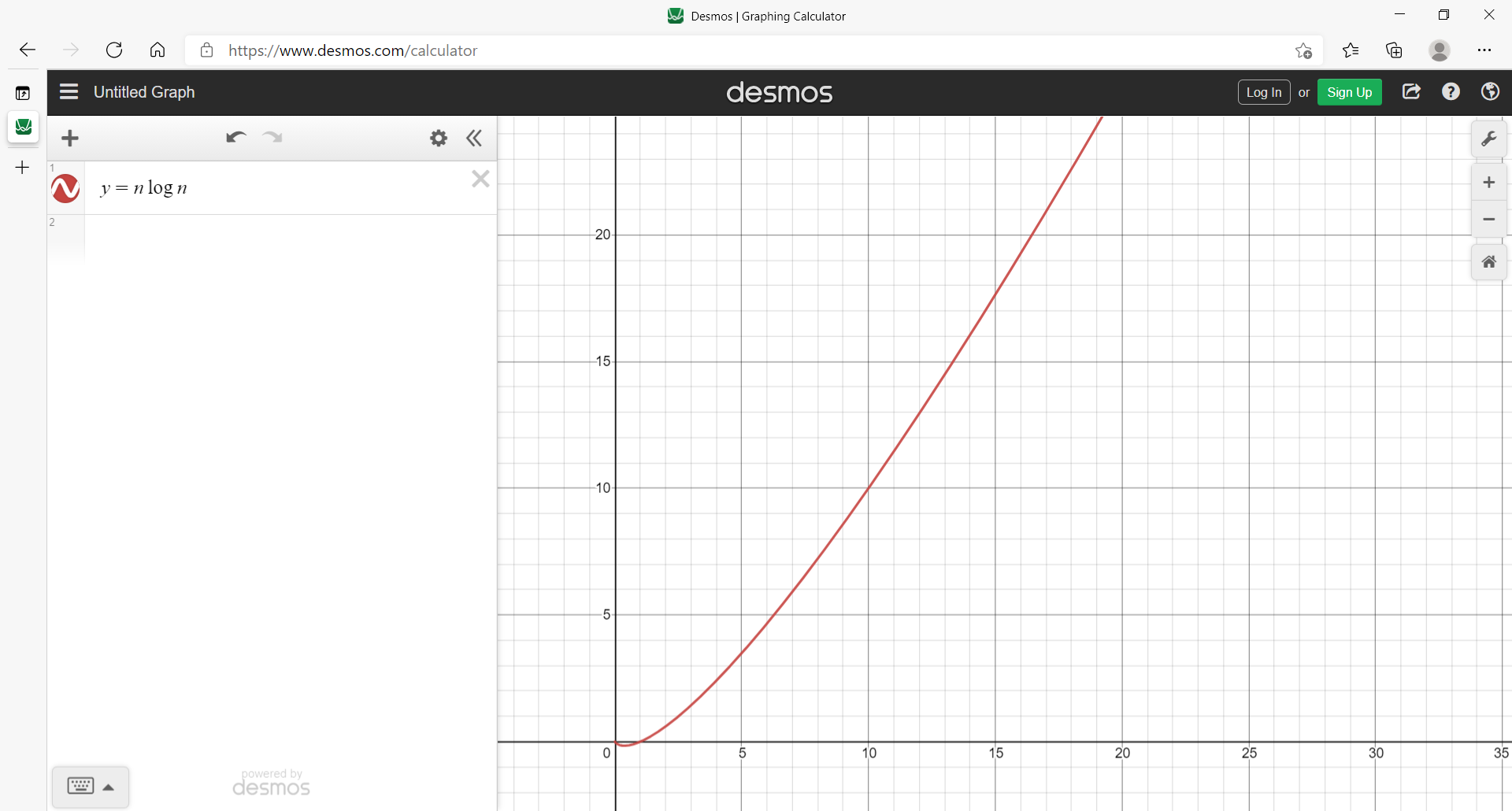
K-ary heap is slower than the other two. But there is a similarity between them, it increases then becomes constant. The graph’s magnitude is thrice the magnitude of the other graph. Since, here we used K value as 3, it is thrice. It varies with the value of k. Therefore, the time complexity of K-ary heap is O(k\*log(n)).

References:

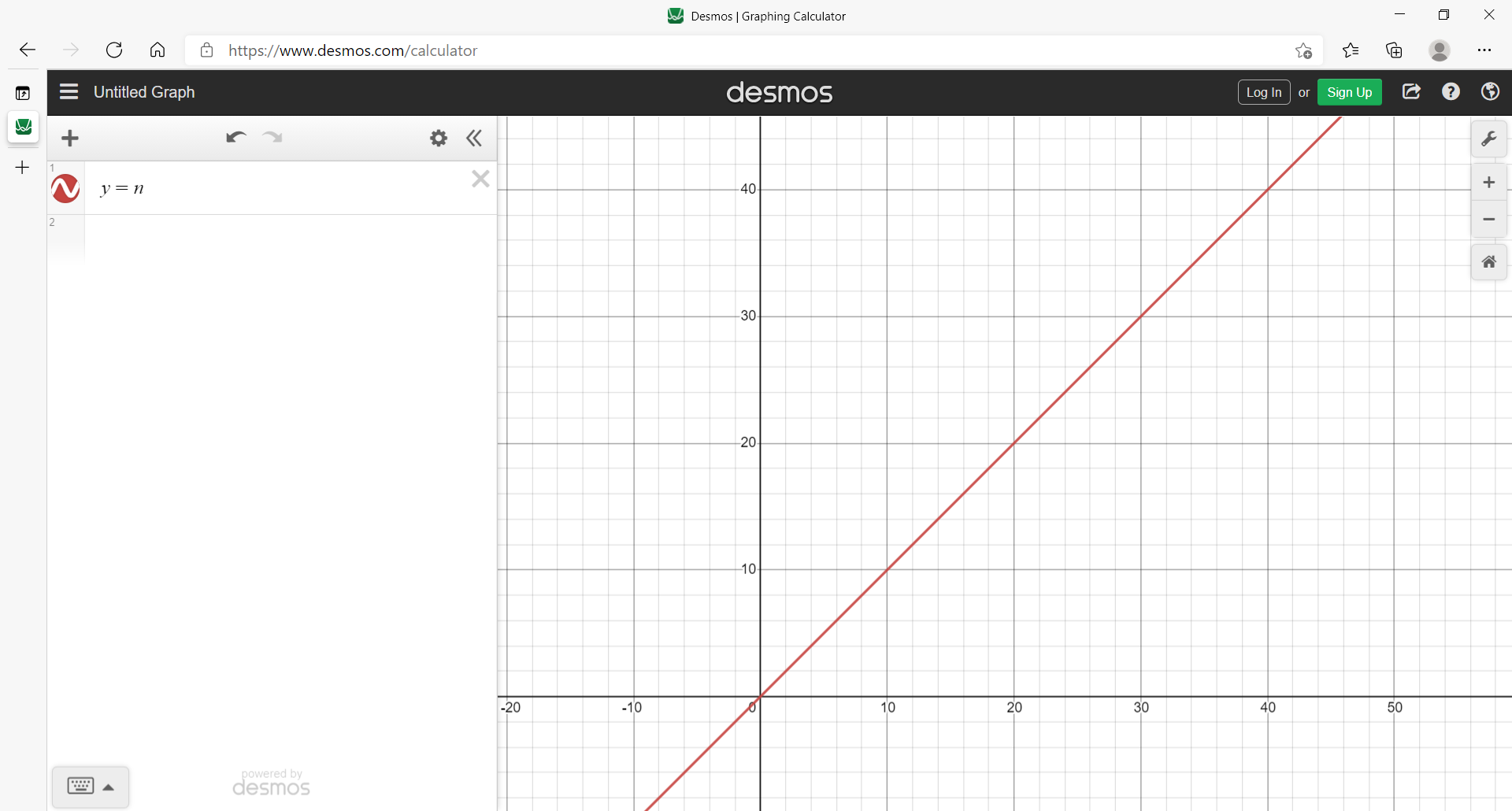
(y=log(n)) graph:



(y=n\*log(n)) graph:



(y=n) graph:



**Summary table**

Time complexities of each operation:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S.No.** | **Data Structure** | **Insertion** | **Deletion** | **Find min** | **Search** | **Extract min** |
| 1 | Fibonacci heap | O(1) | O(log(n)) | O(1) | O(n) | O(log(n)) |
| 2 | Treap | O(log(n)) | O(log(n)) | O(1) | O(log(n)) | O(log(n)) |
| 3 | K-ary heap | O(log(n)) | O(n\*log(n)) | O(1) | O(n) | O(k\*log(n)) |

**Conclusion**

So, with the help of performance analysis, we get know that treap is performs better compared to Fibonacci heap and k-ary heap in all the operations. Fibonacci heap’s insertion is simple but that alone is not enough. Relatively,

k-ary heap’s performance is the worst. Thus, it is always better to prefer treap than these two heaps.