

Introduction

In computer science, a heap is a specialized tree-based data structure which is essentially an almost complete tree 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, leftist 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 F_{k+2} , where F_k is the k th Fibonacci number.

Fibonacci heaps are similar to [binomial heaps](#) but Fibonacci heaps have a less rigid structure. Fibonacci heaps have a faster [amortized](#) 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

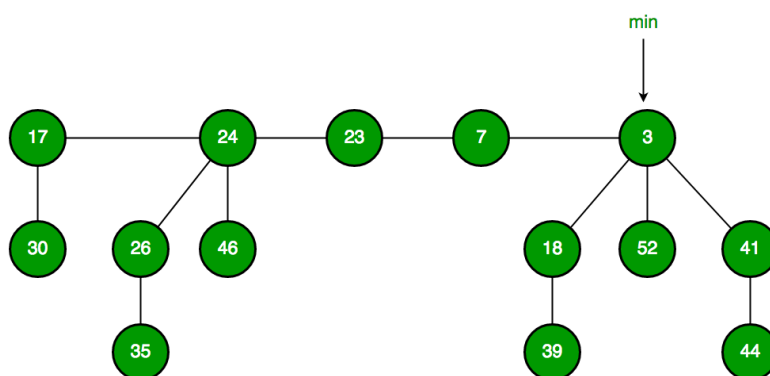
Fibonacci heaps are used to implement the [priority queue](#) element in [Dijkstra's algorithm](#) and Prim's algorithm. With Binary Heap, time complexity of these algorithms is $O(V \log V + E \log V)$. If Fibonacci Heap is used, then time complexity is improved to $O(V \log V + E)$.

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 :



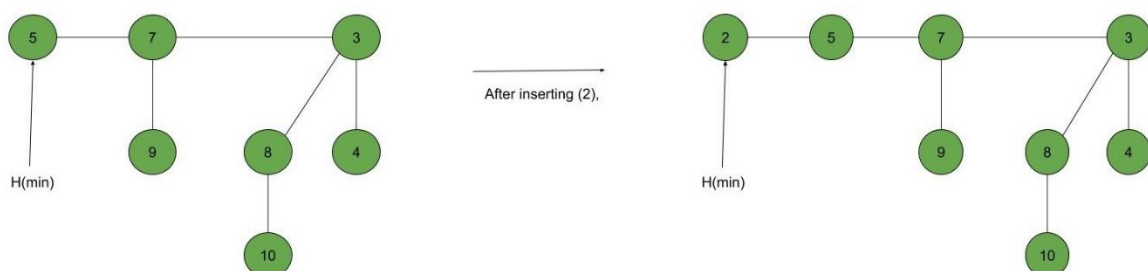
Operations

1. Insertion

Algorithm :

- Create a new node 'x'.
- Check whether heap H is empty or not.
- 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 :

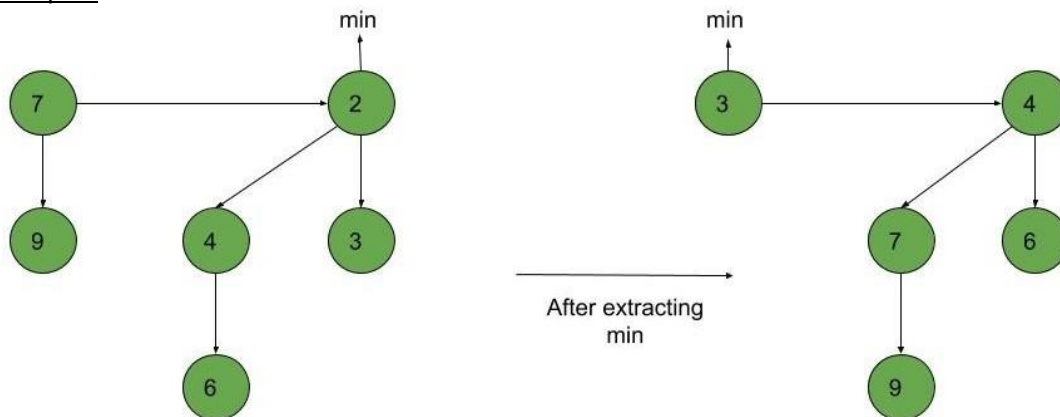


2.Extract min

Algorithm :

- Delete the min node.
- Set head to the next min node and add all the trees of the deleted node in the root list.
- Create an array of degree pointers of the size of the deleted node.
- Set degree pointer to the current node.
- 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.
- Repeat steps 4 and 5 until the heap is completed.

Example :

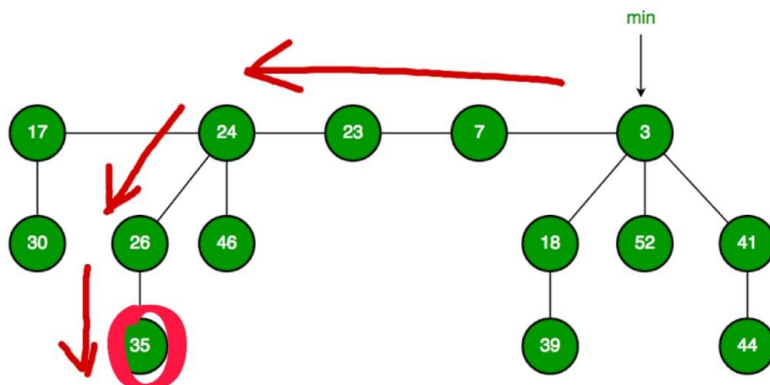


3.Search

Algorithm :

- Starting from the node pointer by min pointer, traverse along the root list and searches the element.
- 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.

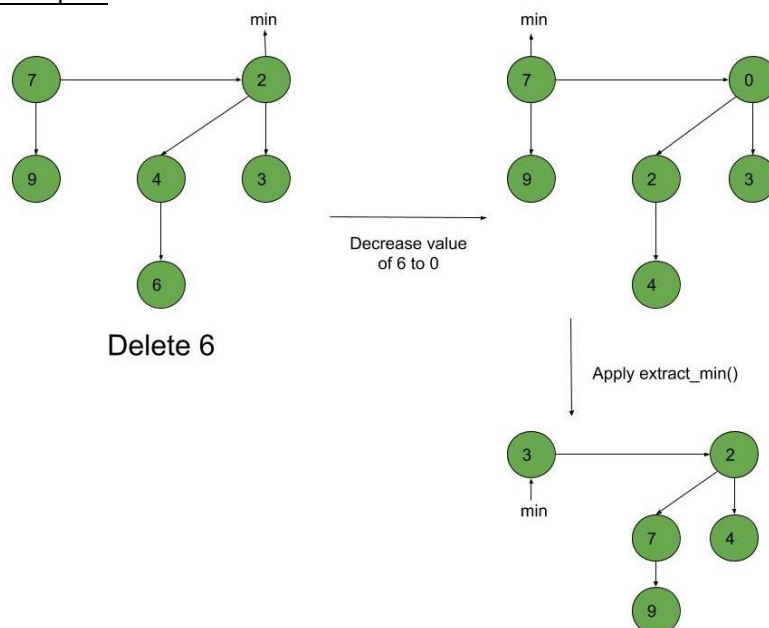
If $p[p[x]]$ is unmarked, mark it.

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 :

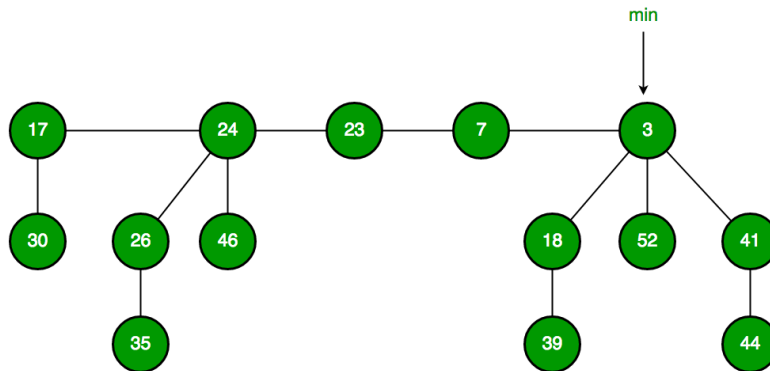


5. Find min

Algorithm :

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

Example :



Code:

```
1 #include <math>
2 #include <stdlib>
3 #include <iostream>
4 #include <malloc.h>
5 using namespace std;
6
7 struct node {
8     node* parent;
9     node* child;
10    node* left;
11    node* right;
12    int key;
13    int degree;
14    char mark;
15    char flag;
16 };
17
18 struct node* minimum = NULL;
19 int nodecount = 0;
20 int found = 0;
21
22 void insertion(int val)
23 {
24     struct node* new_node = (struct node*)malloc(sizeof(struct node));
25     new_node->key = val;
26     new_node->degree = 0;
27     new_node->mark = 'u';
28     new_node->flag = 'N';
29     new_node->parent = NULL;
30     new_node->child = NULL;
31     new_node->left = new_node;
32     new_node->right = new_node;
33
34     if (minimum != NULL)
35     {
36         (minimum->left)->right = new_node;
37         new_node->right = minimum;
38         new_node->left = minimum->left;
39         minimum->left = new_node;
40     }
41
42     if (new_node->key < minimum->key)
43     {
44         minimum = new_node;
45     }
46 }
```

```
C:\Users\Aravindh\Desktop\Fibonacci heap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug Fibonacci heap.cpp
44 }
45
46 else
47 {minimum = new_node;}
48 nodecount++;
49 }
50
51
52 void link(struct node* ptr2, struct node* ptr1)
53 {
54     (ptr2->left)->right = ptr2->right;
55     (ptr2->right)->left = ptr2->left;
56     if (ptr1->right == ptr1)
57         minimum = ptr1;
58     ptr2->left = ptr2;
59     ptr2->right = ptr2;
60     ptr2->parent = ptr1;
61     if (ptr1->child == NULL)
62         ptr1->child = ptr2;
63     ptr2->right = ptr1->child;
64     ptr2->left = (ptr1->child)->left;
65     ((ptr1->child)->left)->right = ptr2;
66     (ptr1->child)->left = ptr2;
67     if (ptr2->key < (ptr1->child)->key)
68         ptr1->child = ptr2;
69     ptr1->degree++;
70 }
71
72 void restructure()
73 {
74     int temp1;
75     float temp2 = (log(nodecount)) / (log(2));
76     int temp3 = temp2;
77     struct node* arr[temp3];
78     for (int i = 0; i <= temp3; i++)
79     {
80         arr[i] = NULL;
81         node* ptr1 = minimum;
82         node* ptr2;
83         node* ptr3;
84         node* ptr4 = ptr1;
85         do
86         {
87             ptr4 = ptr4->right;
88             temp1 = ptr1->degree;
89             while (arr[temp1] != NULL)
90             {
91                 ptr2 = arr[temp1];
92                 if (ptr1->key > ptr2->key)
93                 {
94                     ptr3 = ptr1;
95                     ptr1 = ptr2;
96                     ptr2 = ptr3;
97                 }
98                 if (ptr2 == minimum)
99                     minimum = ptr1;
100                 link(ptr2, ptr1);
101                 if (ptr1->right == ptr1)
102                     minimum = ptr1;
103                 arr[temp1] = NULL;
104                 temp1++;
105             }
106             arr[temp1] = ptr1;
107             ptr1 = ptr1->right;
108             while (ptr1 != minimum);
109             minimum = NULL;
110             for (int j = 0; j <= temp3; j++)
111             {
112                 if (arr[j] != NULL) {
113                     arr[j]->left = arr[j];
114                     arr[j]->right = arr[j];
115                 }
116                 if (minimum != NULL)
117                 {
118                     (minimum->left)->right = arr[j];
119                     arr[j]->right = minimum;
120                     arr[j]->left = minimum->left;
121                     minimum->left = arr[j];
122                     if (arr[j]->key < minimum->key)
123                         minimum = arr[j];
124                 }
125             }
126         }
127     }
128 }
```

```
C:\Users\Aravindh\Desktop\Fibonacci heap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug Fibonacci heap.cpp
87 node* ptr4 = ptr1;
88 do
89 {
90     ptr4 = ptr4->right;
91     temp1 = ptr1->degree;
92     while (arr[temp1] != NULL)
93     {
94         ptr2 = arr[temp1];
95         if (ptr1->key > ptr2->key)
96         {
97             ptr3 = ptr1;
98             ptr1 = ptr2;
99             ptr2 = ptr3;
100         }
101         if (ptr2 == minimum)
102             minimum = ptr1;
103         link(ptr2, ptr1);
104         if (ptr1->right == ptr1)
105             minimum = ptr1;
106         arr[temp1] = NULL;
107         temp1++;
108     }
109     arr[temp1] = ptr1;
110     ptr1 = ptr1->right;
111     while (ptr1 != minimum);
112     minimum = NULL;
113     for (int j = 0; j <= temp3; j++)
114     {
115         if (arr[j] != NULL) {
116             arr[j]->left = arr[j];
117             arr[j]->right = arr[j];
118         }
119         if (minimum != NULL)
120         {
121             (minimum->left)->right = arr[j];
122             arr[j]->right = minimum;
123             arr[j]->left = minimum->left;
124             minimum->left = arr[j];
125             if (arr[j]->key < minimum->key)
126                 minimum = arr[j];
127         }
128     }
129 }
```

```
C:\Users\Aravinth\Desktop\Fibonacci heap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug Fibonacci heap.cpp
131 }
132 else
133 {minimum = arr[j];}
134
135 if (minimum == NULL)
136 minimum = arr[j];
137
138 else if (arr[j]->key < minimum->key)
139 minimum = arr[j];
140
141 }
142 }
143
144
145 void extract_min()
146 {
147 if (minimum == NULL)
148 cout << "The heap is empty" << endl;
149
150 else
151 {
152 node* temp = minimum;
153 node* ptr;
154 ptr = temp;
155 node* x = NULL;
156 if (temp->child != NULL)
157 {
158 x = temp->child;
159 do
160 {
161 ptr = x->right;
162 (minimum->left)->right = x;
163 x->right = minimum;
164 x->left = minimum->left;
165 minimum->left = x;
166
167 if (x->key < minimum->key)
168 minimum = x;
169
170 x->parent = NULL;
171 x = ptr;
172 } while (ptr != temp->child);
173
174 (temp->left)->right = temp->right;
175
Line: 162 Col: 32 Sel: 0 Lines: 375 Length: 6517 Insert Done parsing in 0.406 seconds
```

```
C:\Users\Aravinth\Desktop\Fibonacci heap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug Fibonacci heap.cpp
175 (temp->left)->right = temp->right;
176 (temp->right)->left = temp->left;
177 minimum = temp->right;
178 if (temp == temp->right && temp->child == NULL)
179 minimum = NULL;
180
181 else
182 {
183 minimum = temp->right;
184 restructure();
185 }
186 nodecount--;
187 }
188
189
190 void cut(struct node* found, struct node* temp)
191 {
192 if (found == found->right)
193 temp->child = NULL;
194
195 (found->left)->right = found->right;
196 (found->right)->left = found->left;
197 if (found == temp->child)
198 temp->child = found->right;
199
200 temp->degree = temp->degree - 1;
201 found->right = found;
202 found->left = found;
203 (minimum->left)->right = found;
204 found->right = minimum;
205 found->left = minimum->left;
206 minimum->left = found;
207 found->parent = NULL;
208 found->mark = 'B';
209
210
211
212 void cut_rearrange(struct node* temp)
213 {
214 node* ptr = temp->parent;
215 if (ptr != NULL)
216 {
217 if (temp->mark == 'W')
218 (temp->mark = 'B');
219 else
Line: 217 Col: 27 Sel: 0 Lines: 375 Length: 6517 Insert Done parsing in 0.406 seconds
```

```
C:\Users\Aravindh\Desktop\Fibonacci heap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals) TDM-GCC 4.9.2 64-bit Release
Project Classes Debug Fibonacci heap.cpp
218 {temp->mark = 'B';}
219 else
220 {
221 cut(temp, ptr);
222 cut_rearrange(ptr);
223 }
224 }
225 }
226
227
228 void decrease_key(struct node* found, int val)
229 {
230 if (minimum == NULL)
231 cout << "The Heap is Empty" << endl;
232
233 if (found == NULL)
234 cout << "Node not found in the Heap" << endl;
235
236 found->key = val;
237
238 struct node* temp = found->parent;
239 if (temp != NULL && found->key < temp->key) {
240 cut(found, temp);
241 cut_rearrange(temp);
242 }
243 if (found->key < minimum->key)
244 minimum = found;
245 }
246
247
248 void revalue(struct node* minimum, int old_val, int val)
249 {
250 struct node* found = NULL;
251 node* temp = minimum;
252 temp->flag = 'Y';
253 node* ptr = NULL;
254 if (temp->key == old_val) {
255 ptr = temp;
256 temp->flag = 'N';
257 found = ptr;
258 decrease_key(found, val);
259 }
260 if (ptr == NULL) {
261 if (temp->child != NULL)
262 revalue(temp->child, old_val, val);
263 }
```

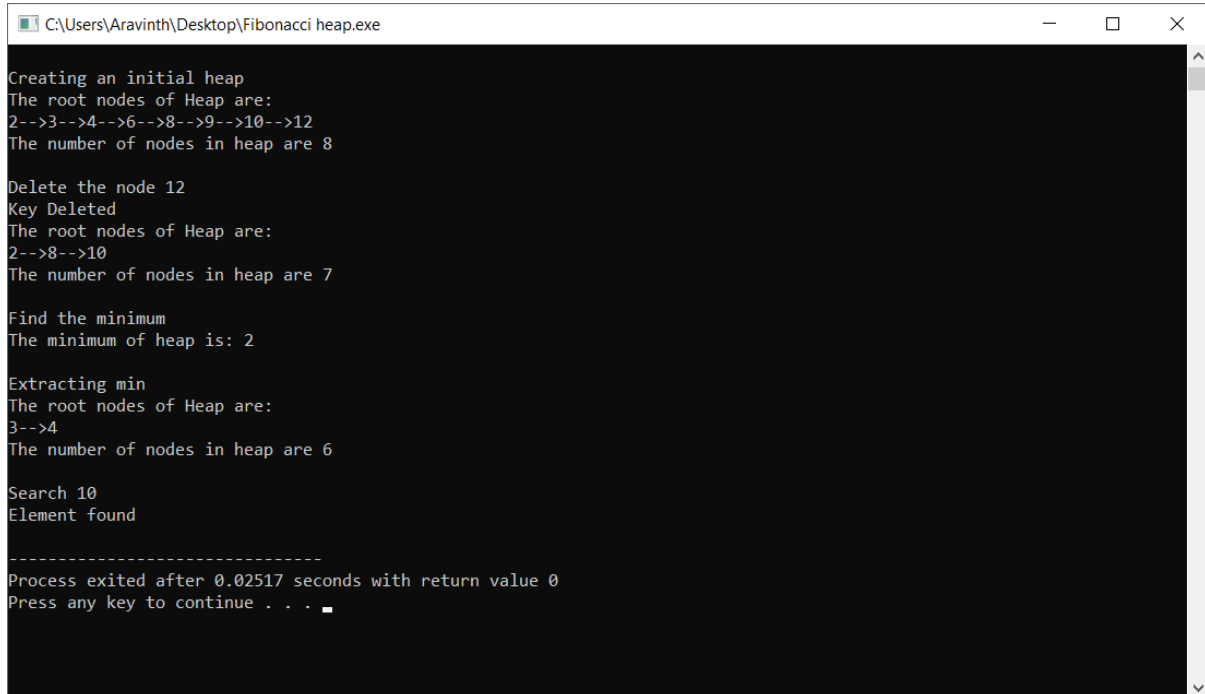
```
C:\Users\Aravindh\Desktop\Fibonacci heap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals) TDM-GCC 4.9.2 64-bit Release
Project Classes Debug Fibonacci heap.cpp
261 if (temp->child != NULL)
262 revalue(temp->child, old_val, val);
263 if ((temp->right)->flag != 'Y')
264 revalue(temp->right, old_val, val);
265 }
266 temp->flag = 'N';
267 found = ptr;
268 }
269
270
271 int search(struct node* minimum, int old_val)
272 {
273 node* temp = minimum;
274 temp->flag = 'Y';
275 node* ptr = NULL;
276
277 if (temp->key == old_val) {
278 ptr = temp;
279 temp->flag = 'N';
280 found = ptr;
281 }
282
283 if (ptr == NULL) {
284 if (temp->child != NULL)
285 search(temp->child, old_val);
286
287 if ((temp->right)->flag != 'Y')
288 search(temp->right, old_val);
289 }
290 temp->flag = 'N';
291
292 }
293
294
295
296
297 void deletion(int val)
298 {
299 if (minimum == NULL)
300 cout << "The heap is empty" << endl;
301
302 else {
303 revalue(minimum, val, 0);
304 extract_min();
305 cout << "Key Deleted" << endl;
306 }
```



```
C:\Users\Aravinth\Desktop\Fibonacci heap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug Fibonacci heap.cpp
301 |
302 | else {
303 |     revalue(minimum, val, 0);
304 |     extract_min();
305 |     cout << "Key Deleted" << endl;
306 | }
307 |
308 |
309 | void display()
310 | {
311 |     node* ptr = minimum;
312 |     if (ptr == NULL)
313 |         cout << "The Heap is Empty" << endl;
314 |     else {
315 |         cout << "The root nodes of Heap are: " << endl;
316 |         do
317 |         {
318 |             cout << ptr->key;
319 |             ptr = ptr->right;
320 |             if (ptr != minimum) {
321 |                 cout << "->";
322 |             } while (ptr != minimum && ptr->right != NULL);
323 |             cout << endl;
324 |             cout << "The number of nodes in heap are "<< nodecount << endl;
325 |         }
326 |     }
327 |
328 | void search_min()
329 | {
330 |     cout << "The minimum of heap is: " << minimum->key << endl;
331 | }
332 |
333 | int main()
334 | {
335 |     cout << "\nCreating an initial heap" << endl;
336 |     insertion(2);
337 |     insertion(3);
338 |     insertion(4);
339 |
Line: 341 Col: 5 Sel: 0 Lines: 375 Length: 6517 Insert Done parsing in 0.406 seconds
```

```
C:\Users\Aravinth\Desktop\Fibonacci heap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug Fibonacci heap.cpp
332 | void search_min()
333 | { cout << "The minimum of heap is: " << minimum->key << endl;
334 | }
335 |
336 |
337 | int main()
338 | {
339 |     cout << "\nCreating an initial heap" << endl;
340 |     insertion(2);
341 |     insertion(3);
342 |     insertion(4);
343 |     insertion(5);
344 |     insertion(6);
345 |     insertion(8);
346 |     insertion(9);
347 |     insertion(10);
348 |     insertion(12);
349 |
350 |     display();
351 |
352 |     cout << "\nDelete the node 12" << endl;
353 |     deletion(12);
354 |     display();
355 |
356 |     cout << "\nFind the minimum" << endl;
357 |     search_min();
358 |
359 |     cout << "\nExtracting min" << endl;
360 |     extract_min();
361 |     display();
362 |
363 |     cout << "\nSearch 10\n";
364 |     found = 0;
365 |     search(minimum, 10);
366 |     if(found == 1)
367 |         cout << "Element found\n";
368 |     else
369 |         cout << "Element not found\n";
370 | }
371 |
372 |
373 |
374 |
375 |
Line: 368 Col: 24 Sel: 0 Lines: 375 Length: 6517 Insert Done parsing in 0.406 seconds
```

Output:



```
C:\Users\Aravinth\Desktop\Fibonacci heap.exe
Creating an initial heap
The root nodes of Heap are:
2-->3-->4-->6-->8-->9-->10-->12
The number of nodes in heap are 8

Delete the node 12
Key Deleted
The root nodes of Heap are:
2-->8-->10
The number of nodes in heap are 7

Find the minimum
The minimum of heap is: 2

Extracting min
The root nodes of Heap are:
3-->4
The number of nodes in heap are 6

Search 10
Element found

-----
Process exited after 0.02517 seconds with return value 0
Press any key to continue . . .
```

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.

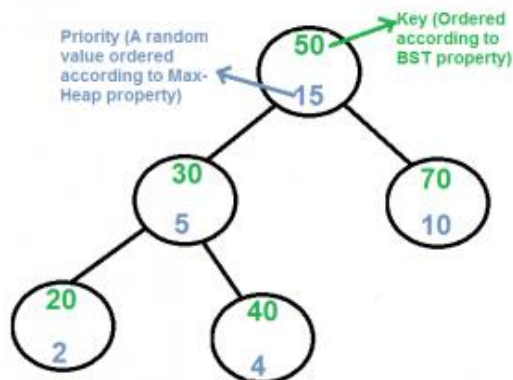
Limitations

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 :



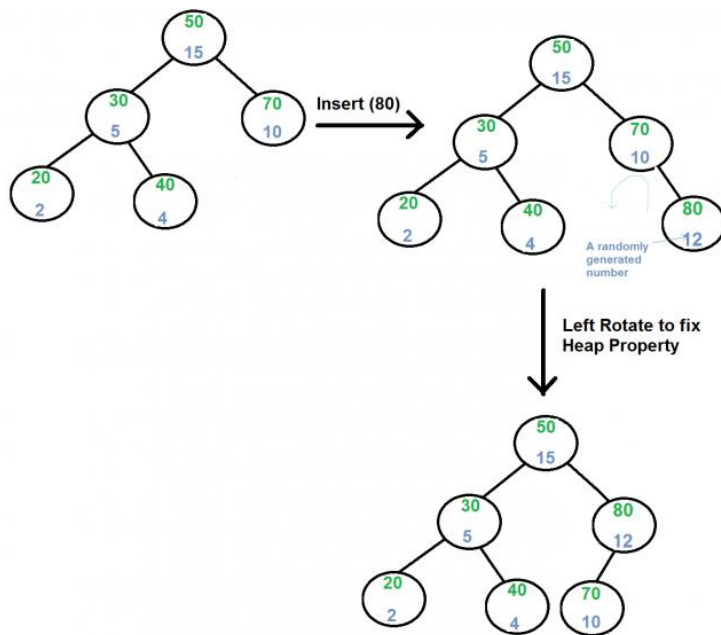
Operations

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 :



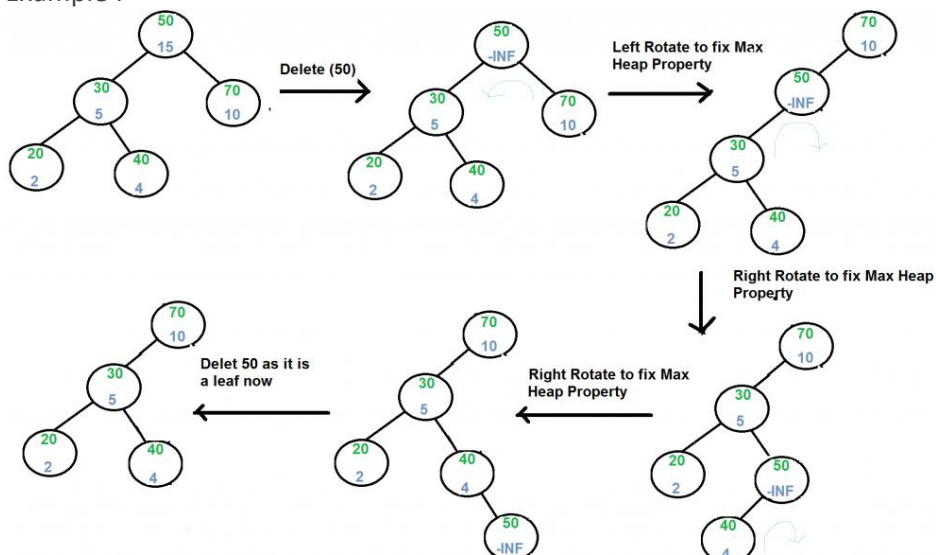
2.Deletion

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 :

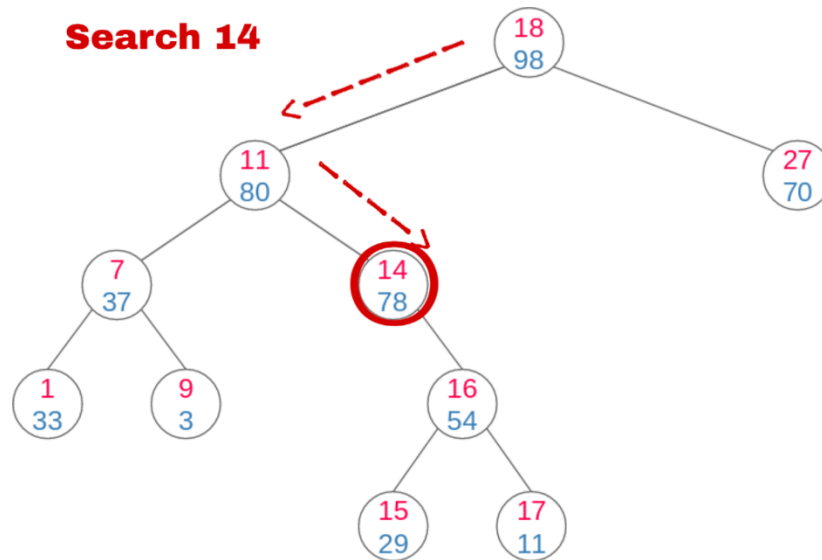


3.Search

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 :

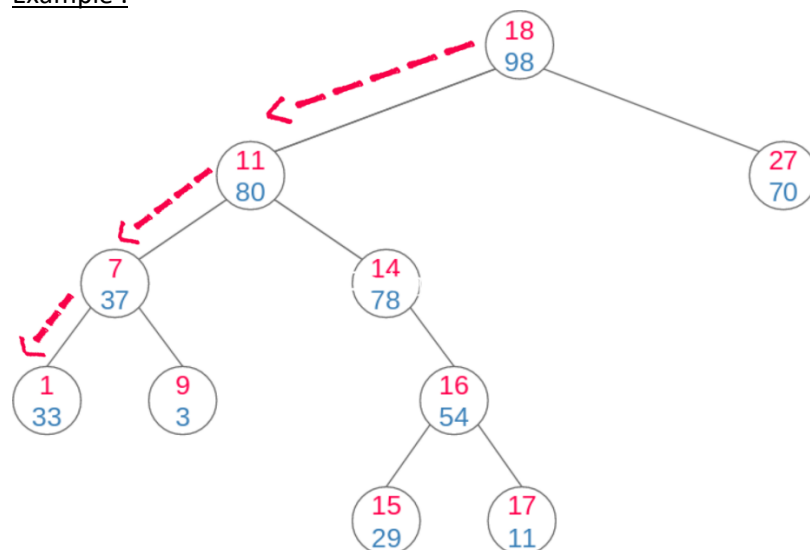


4.Find min

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 :

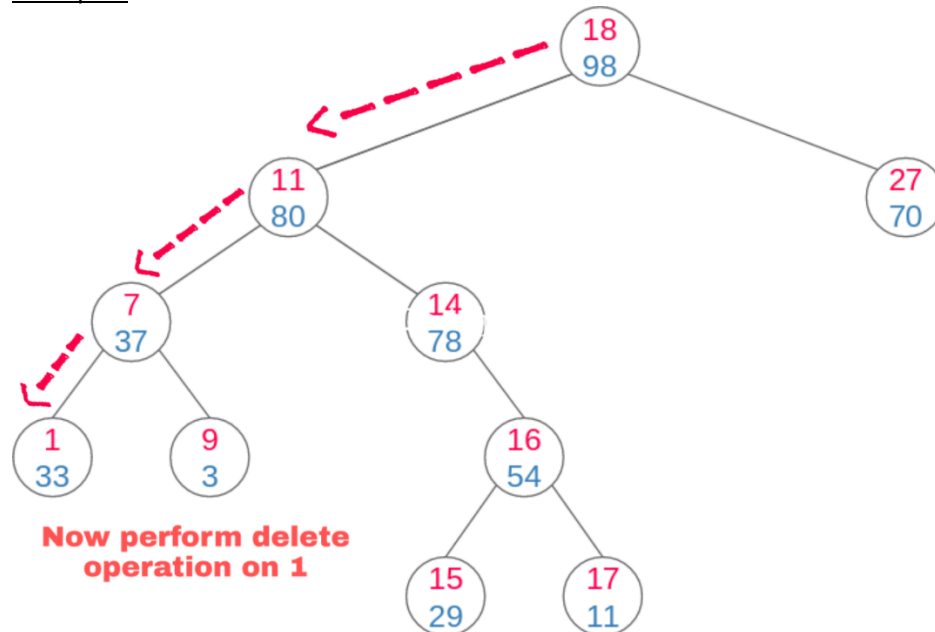


5.Extract min

Algorithm :

- 1) First perform Find min operation and get the minimum valued node.
- 2) Then perform Deletion operation.

Example :



Code:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct node
5 {
6     int key, priority;
7     node *left, *right;
8 };
9
10 node *rightrotate(node *y)
11 {
12     node *x = y->left;
13     node *temp = x->right;
14     x->right = y;
15     y->left = temp;
16     return x;
17 }
18
19 node *leftrotate(node *x)
20 {
21     node *y = x->right;
22     node *temp = y->left;
23     y->left = x;
24     x->right = temp;
25     return y;
26 }
27
28 node* newnode(int key)
29 {
30     node* temp = new node;
31     temp->key = key;
32     temp->priority = rand()%100;
33     temp->left = temp->right = NULL;
34     return temp;
35 }
36
37
38
39
40
41
42
43
```



```
C:\Users\Aravindh\Desktop\Treap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug Treap.cpp
126 node* search(node* root, int key)
127 {
128     if (root == NULL || root->key == key)
129         return root;
130     if (root->key < key)
131         return search(root->right, key);
132     return search(root->left, key);
133 }
134
135 void find_min(node* root)
136 {
137     while(root->left != NULL)
138         root = root->left;
139     cout << "The minimum value is " << root->key << endl;
140     cout << endl;
141 }
142
143 node* extractmin(node* root)
144 {
145     node *p, *parent;
146     p = root;
147     while(p->left != NULL)
148         (parent = p) = p->left;
149     cout << "Extracted min is " << p->key << endl;
150     if(p->right == NULL)
151         (delete(p))parent->left = NULL;
152     else
153     {
154         parent->left = p->right;
155         p->left = NULL;
156         delete(p);
157     }
158     return root;
159 }
160
161 int main()
162 {
163     srand(time(NULL));
164     struct node *root = NULL;
165     root = insert(root, 2);
166     root = insert(root, 3);
167     root = insert(root, 4);
168     root = insert(root, 6);
169     root = insert(root, 8);
170     root = insert(root, 9);
171     root = insert(root, 10);
172     root = insert(root, 12);
173     cout << "Inorder traversal of the given tree \n";
174     inorder(root);
175     cout << "\nDelete 12\n";
176     root = deletenode(root, 12);
177     cout << "Inorder traversal of the tree after deletion\n";
178     inorder(root);
179     cout << "\nfind minimum in the heap\n";
180     find_min(root);
181     cout << "Extract min\n";
182     root = extractmin(root);
183     cout << "Inorder traversal of the tree after extract min \n";
184     inorder(root);
185     cout << "\nSearch 10\n";
186     node* find_search(root, 10);
187     if(find == NULL)
188         cout << "10 is not present in the heap";
189     else
190         cout << "10 is found in the heap";
191     return 0;
192 }
```

```
C:\Users\Aravindh\Desktop\Treap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug Treap.cpp
165
166 int main()
167 {
168     srand(time(NULL));
169     struct node *root = NULL;
170     root = insert(root, 2);
171     root = insert(root, 3);
172     root = insert(root, 4);
173     root = insert(root, 6);
174     root = insert(root, 8);
175     root = insert(root, 9);
176     root = insert(root, 10);
177     root = insert(root, 12);
178     cout << "Inorder traversal of the given tree \n";
179     inorder(root);
180     cout << "\nDelete 12\n";
181     root = deletenode(root, 12);
182     cout << "Inorder traversal of the tree after deletion\n";
183     inorder(root);
184     cout << "\nfind minimum in the heap\n";
185     find_min(root);
186     cout << "Extract min\n";
187     root = extractmin(root);
188     cout << "Inorder traversal of the tree after extract min \n";
189     inorder(root);
190     cout << "\nSearch 10\n";
191     node* find_search(root, 10);
192     if(find == NULL)
193         cout << "10 is not present in the heap";
194     else
195         cout << "10 is found in the heap";
196     return 0;
197 }
```


Output:

```
C:\Users\Aravindh\Desktop\Heap.exe
Inorder traversal of the given tree
key: 2      priority: 82      right child: 3
key: 3      priority: 30
key: 4      priority: 88      left child: 2      right child: 6
key: 6      priority: 2
key: 8      priority: 89      left child: 4      right child: 9
key: 9      priority: 81      right child: 12
key: 10     priority: 35      left child: 10
key: 12     priority: 55

Delete 12
Inorder traversal of the tree after deletion
key: 2      priority: 82      right child: 3
key: 3      priority: 30
key: 4      priority: 88      left child: 2      right child: 6
key: 6      priority: 2
key: 8      priority: 89      left child: 4      right child: 9
key: 9      priority: 81      right child: 10
key: 10     priority: 35

Find minimum in the heap
The minimum value is 2

Extract min
Extracted min is 2
Inorder traversal of the tree after extract min
key: 3      priority: 30
key: 4      priority: 88      left child: 3      right child: 6
key: 6      priority: 2
key: 8      priority: 89      left child: 4      right child: 9
key: 9      priority: 81      right child: 10
key: 10     priority: 35

Search 10
10 is found in the heap
-----
Process exited after 0.04965 seconds with return value 0
Press any key to continue . . .
```

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.

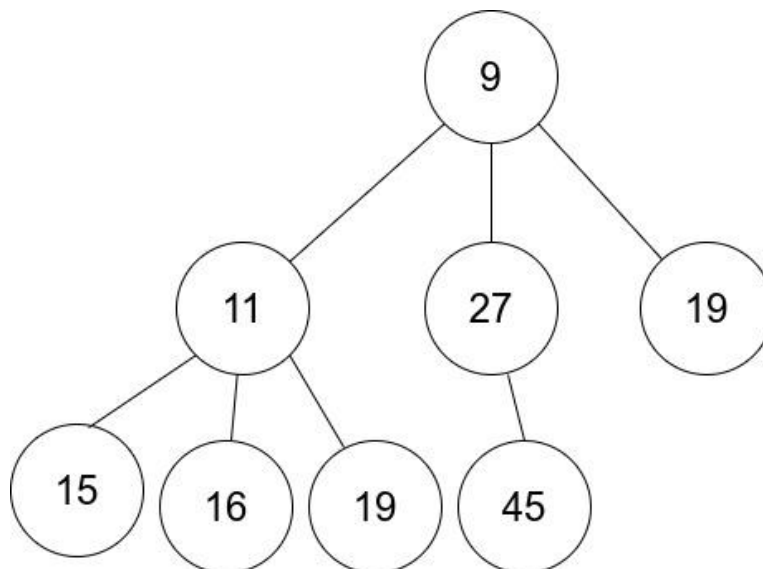
Applications

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

- Various Operating systems use this algorithm for jobs and process management
- Finding the order 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.



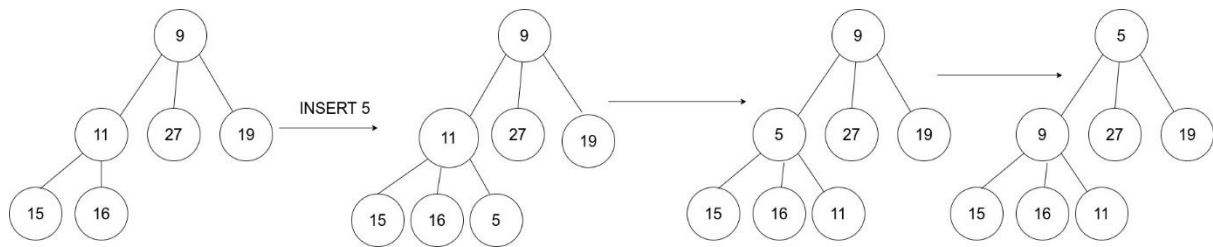
Operations

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:

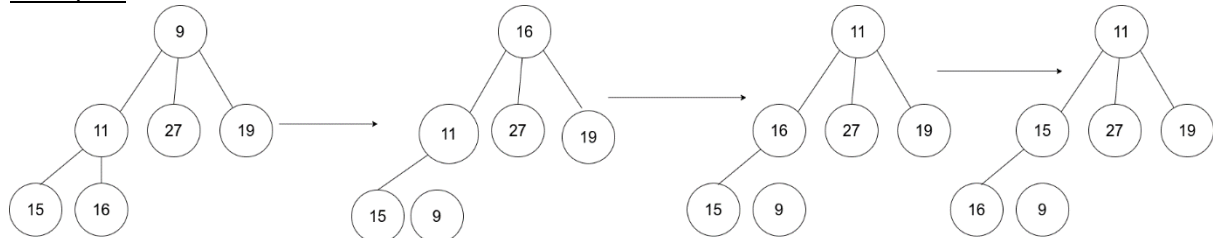


2.Extract min

Algorithm :

- i) Swap the root node with the last element in the heap.
- ii) $size \leq size-1$
- 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 :

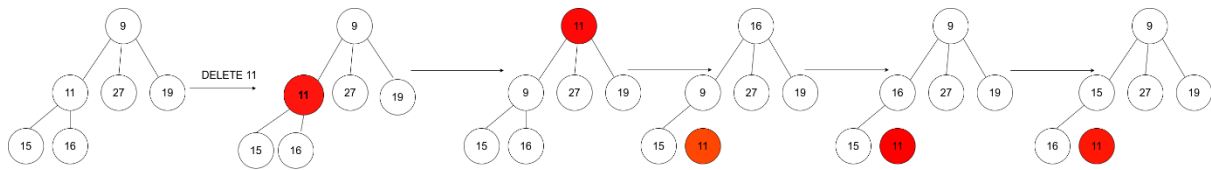


3.Deletion

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 :

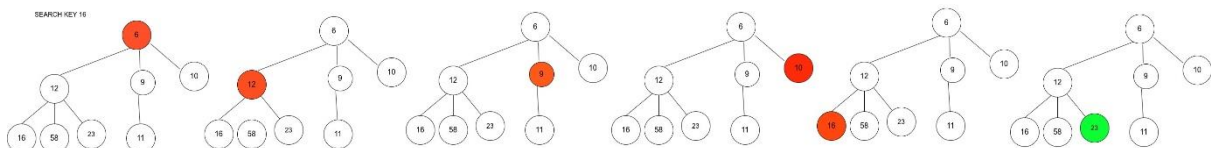


4. Search

Algorithm :

- Traverse through each level of the heap.
- If search key is found return index
- else return -1

Example:

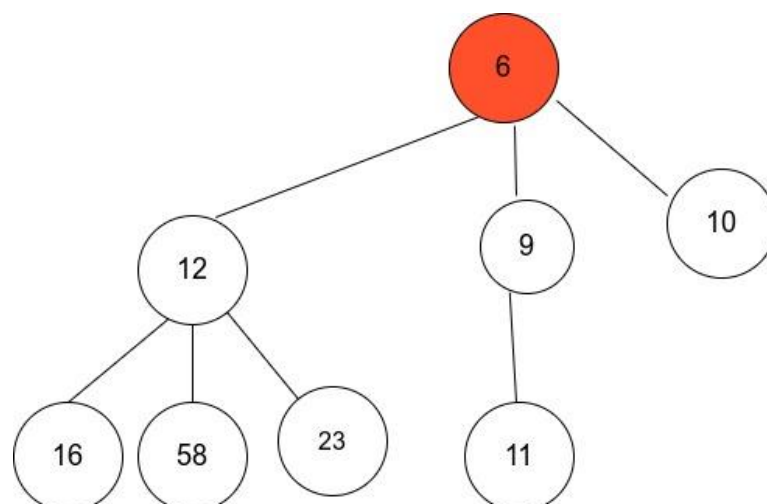


5. Find Minimum/Maximum

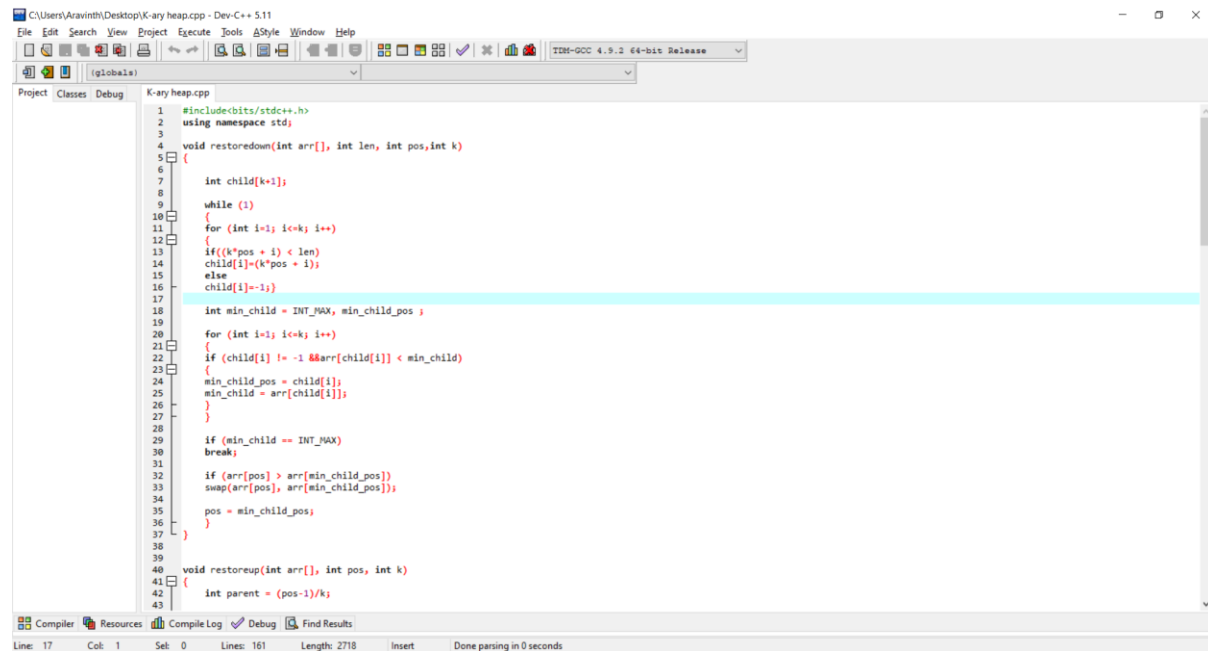
Algorithm :

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

Example:

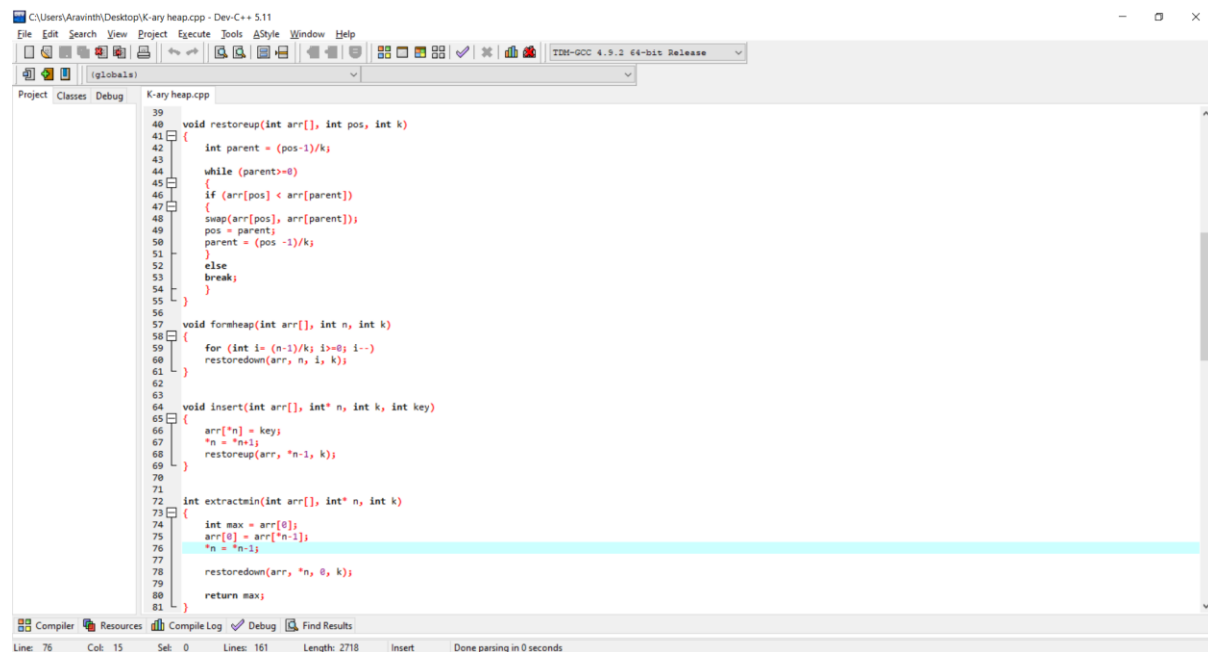


Code:



This screenshot shows a C++ IDE window titled 'K-ary heap.cpp - Dev-C++ 5.11'. The code defines a 'restoreup' function that maintains the heap property for a k-ary heap. It starts by including `<bits/stdc++.h>` and using the `std` namespace. The function takes an array `arr`, its length `len`, a position `pos`, and a branching factor `k`. It calculates the child index `child[k+1]` and enters a loop to find the minimum child. Once the minimum child is found, it swaps the element at `pos` with the minimum child and updates `pos` to the child's position. The status bar at the bottom indicates 'Line: 17', 'Col: 1', 'Sel: 0', 'Lines: 161', 'Length: 2718', and 'Done parsing in 0 seconds'.

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 void restoreup(int arr[], int len, int pos, int k)
5 {
6     int child[k+1];
7     while (1)
8     {
9         for (int i=1; i<=k; i++)
10         {
11             if ((k*pos + i) < len)
12                 child[i] = (k*pos + i);
13             else
14                 child[i] = -1;
15         }
16         int min_child = INT_MAX, min_child_pos;
17         for (int i=1; i<=k; i++)
18         {
19             if (child[i] != -1 && arr[child[i]] < min_child)
20             {
21                 min_child_pos = child[i];
22                 min_child = arr[child[i]];
23             }
24         }
25         if (min_child == INT_MAX)
26             break;
27         if (arr[pos] > arr[min_child_pos])
28             swap(arr[pos], arr[min_child_pos]);
29         pos = min_child_pos;
30     }
31 }
32
33 void restoreup(int arr[], int pos, int k)
34 {
35     int parent = (pos-1)/k;
36     while (parent > 0)
37     {
38         if (arr[pos] < arr[parent])
39         {
40             swap(arr[pos], arr[parent]);
41             pos = parent;
42             parent = (pos-1)/k;
43         }
44         else
45             break;
46     }
47 }
```



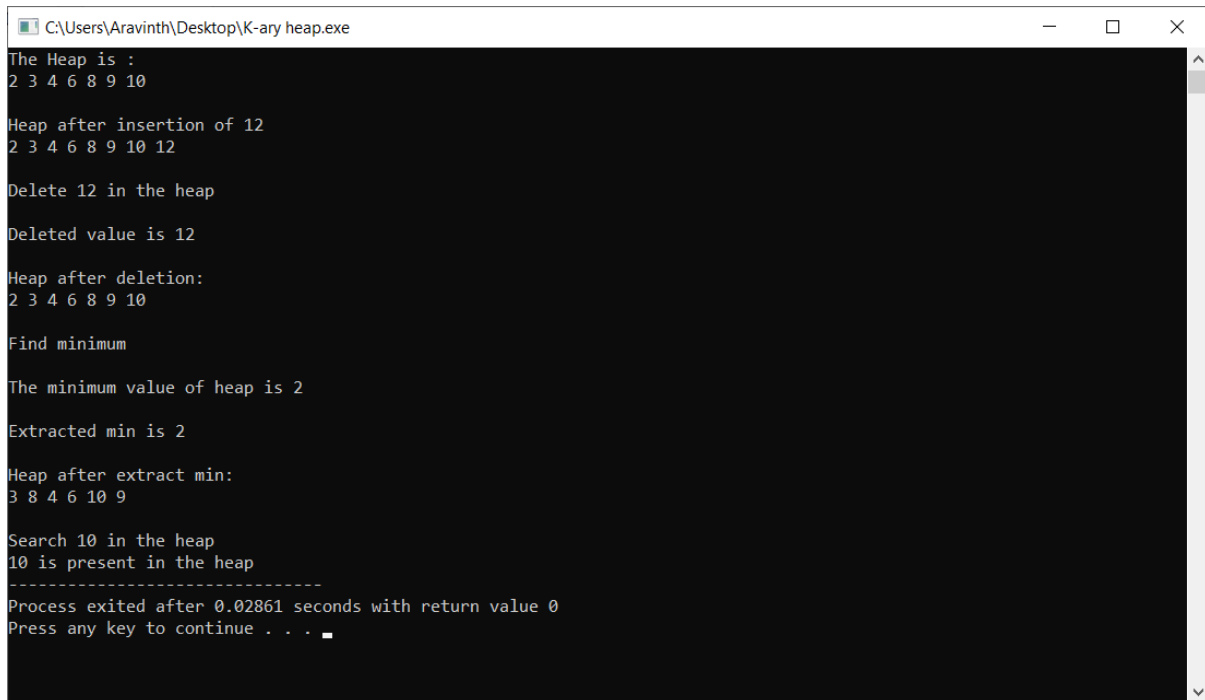
This screenshot shows the continuation of the C++ code in the same IDE. It defines three more functions: 'formheap', 'insert', and 'extractmin'. 'formheap' builds a k-ary heap from an array by calling 'restoreup' on each non-leaf node. 'insert' adds a new element to the heap and restores the heap property. 'extractmin' removes the root element and restores the heap property. The status bar at the bottom indicates 'Line: 76', 'Col: 15', 'Sel: 0', 'Lines: 161', 'Length: 2718', and 'Done parsing in 0 seconds'.

```
39 void formheap(int arr[], int n, int k)
40 {
41     for (int i = (n-1)/k; i >= 0; i--)
42         restoreup(arr, n, i, k);
43 }
44
45 void insert(int arr[], int* n, int k, int key)
46 {
47     arr[*n] = key;
48     *n = *n + 1;
49     restoreup(arr, *n, k);
50 }
51
52 int extractmin(int arr[], int* n, int k)
53 {
54     int max = arr[0];
55     arr[0] = arr[*n-1];
56     *n = *n - 1;
57     restoreup(arr, *n, k);
58     return max;
59 }
```

```
C:\Users\Aravindh\Desktop\K-ary heap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug K-ary heap.cpp
80     return max;
81 }
82
83
84 int findmin(int arr[], int n)
85 {return arr[0];}
86
87
88 int search(int arr[], int n, int k)
89 {
90     for (int i=0; i<n; i++)
91         if(arr[i] == k)
92             return i;
93     return -1;
94 }
95
96
97
98 int deletion(int arr[], int* n, int k)
99 {
100     int pos = search(arr, *n, k);
101     if(pos == -1)
102         return -1;
103     swap(arr[0], arr[pos]);
104     int ans = extractmin(arr, n, k);
105     return ans;
106 }
107
108
109 int main()
110 {
111     const int size = 100;
112     int arr[size] = {2, 3, 4, 6, 8, 9, 10};
113     int n = 7;
114     int k = 3;
115
116     forheap(arr, n, k);
117
118     cout<<"The Heap is : \n";
119     for (int i=0; i<n; i++)
120         cout<< arr[i] <<" ";
121
122     int key = 12;
123
124     Compiler Resources Compile Log Debug Find Results
Line: 115 Col: 1 Sel: 0 Lines: 161 Length: 2718 Insert Done parsing in 0 seconds
```

```
C:\Users\Aravindh\Desktop\K-ary heap.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug K-ary heap.cpp
115     forheap(arr, n, k);
116
117     cout<<"The Heap is : \n";
118     for (int i=0; i<n; i++)
119         cout<< arr[i] <<" ";
120
121     int key = 12;
122     insert(arr, &n, k, key);
123
124     cout<<"\nHeap after insertion of "<<key<<"\n";
125
126     for (int i=0; i<n; i++)
127         cout<< arr[i] <<" ";
128
129     key = 12;
130     cout<<"\nDelete "<<key<<" in the heap\n";
131     int ans = deletion(arr, &n, key);
132     cout<<"\nDeleted value is "<<ans;
133
134     cout<<"\nHeap after deletion: \n";
135     for (int i=0; i<n; i++)
136         cout<< arr[i] <<" ";
137
138     cout<<"\nFind minimum ";
139     cout<<"\nThe minimum value of heap is "<<findmin(arr, n)<<"\n";
140
141     ans = extractmin(arr, &n, k);
142     cout<<"\nExtracted min is "<<ans;
143
144     cout<<"\nHeap after extract min: \n";
145     for (int i=0; i<n; i++)
146         cout<< arr[i] <<" ";
147
148     key = 10;
149     cout<<"\nSearch "<<key<<" in the heap\n";
150     int find = search(arr, n, key);
151     if(find == -1)
152         cout<<key<<" is not in the heap";
153     else
154         cout<<key<<" is present in the heap";
155
156     return 0;
157 }
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
255
```

Output:



```
C:\Users\Aravinth\Desktop\K-ary heap.exe
The Heap is :
2 3 4 6 8 9 10

Heap after insertion of 12
2 3 4 6 8 9 10 12

Delete 12 in the heap

Deleted value is 12

Heap after deletion:
2 3 4 6 8 9 10

Find minimum

The minimum value of heap is 2

Extracted min is 2

Heap after extract min:
3 8 4 6 10 9

Search 10 in the heap
10 is present in the heap
-----
Process exited after 0.02861 seconds with return value 0
Press any key to continue . . .
```

Performance Comparison

Note:

The outputs are taken using this method:

```
auto begin1 = high_resolution_clock::now();
insertion(n);
auto end1 = high_resolution_clock::now();

auto period1 = duration_cast<nanoseconds>(end1 - begin1);

cout << "\nTime taken for insertion: "<< period1.count() << " nanoseconds\n\n";
```

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

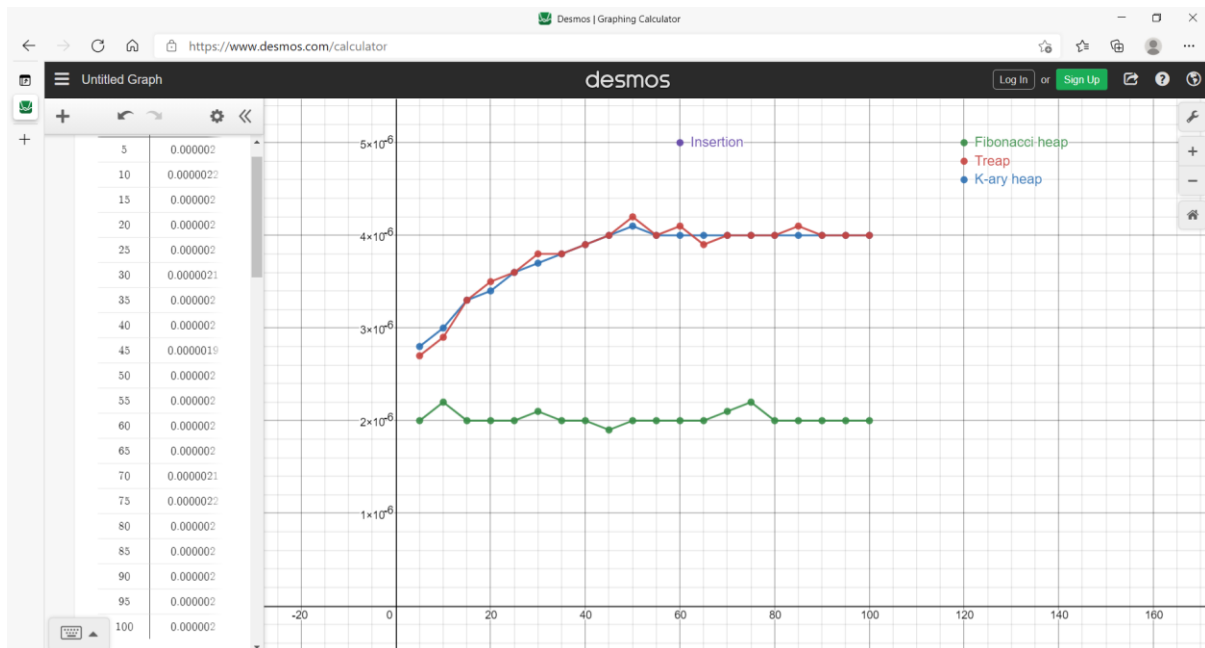
70	0.000004	0.0000037	0.000001	0.0000027	0.0000043
75	0.000004	0.0000038	0.000001	0.0000029	0.0000044
80	0.000004	0.000004	0.000001	0.000003	0.0000044
85	0.000004	0.000004	0.000001	0.000003	0.0000044
90	0.000004	0.000004	0.000001	0.000003	0.0000045
95	0.000004	0.0000039	0.0000009	0.000003	0.0000045
100	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:

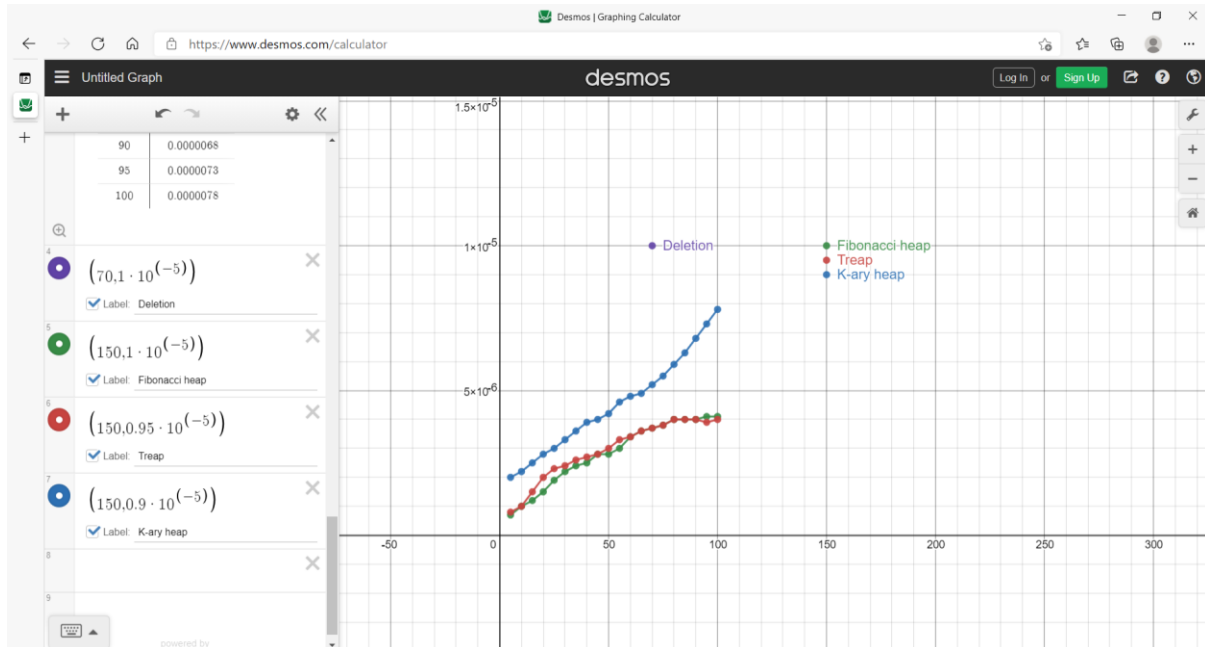


Inference:

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:

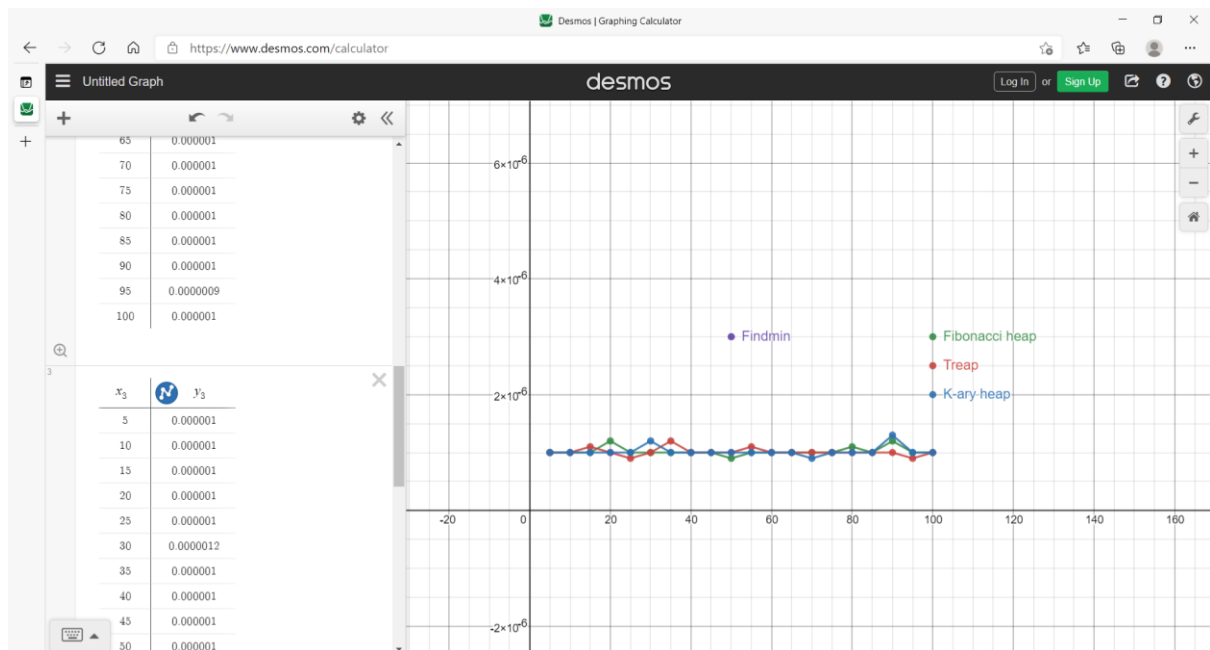


Inference:

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 $y=n\log(n)$ graph. Therefore the time complexity of k-ary heap is $O(n*\log(n))$.

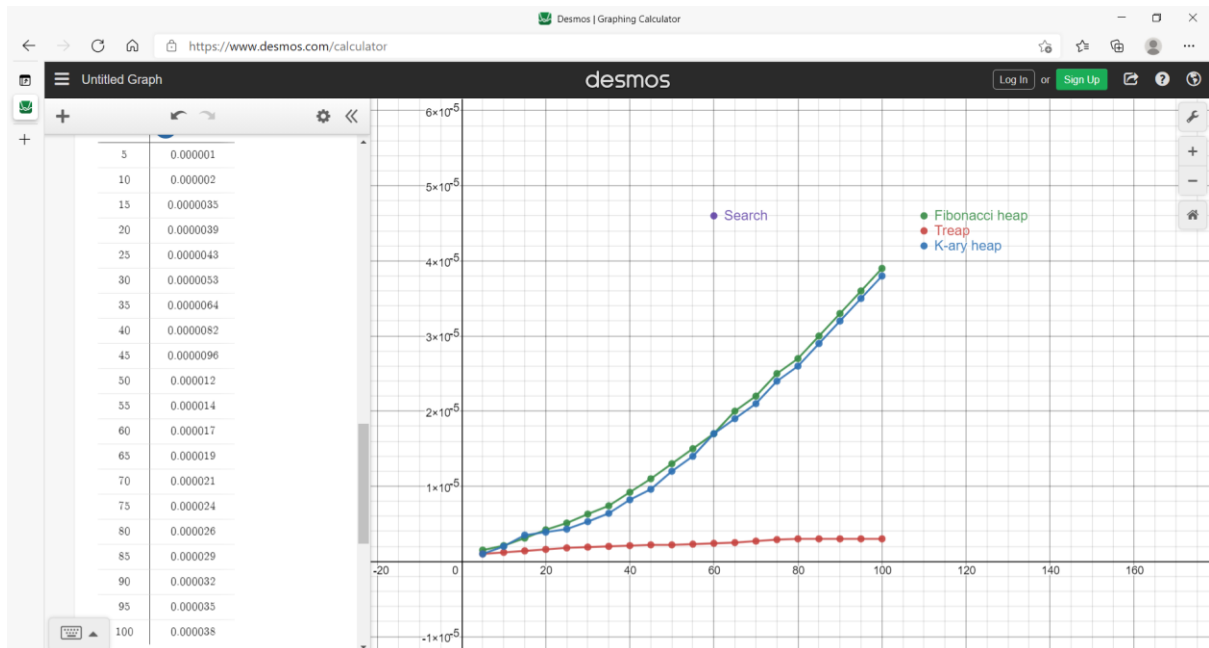
Find min:



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:

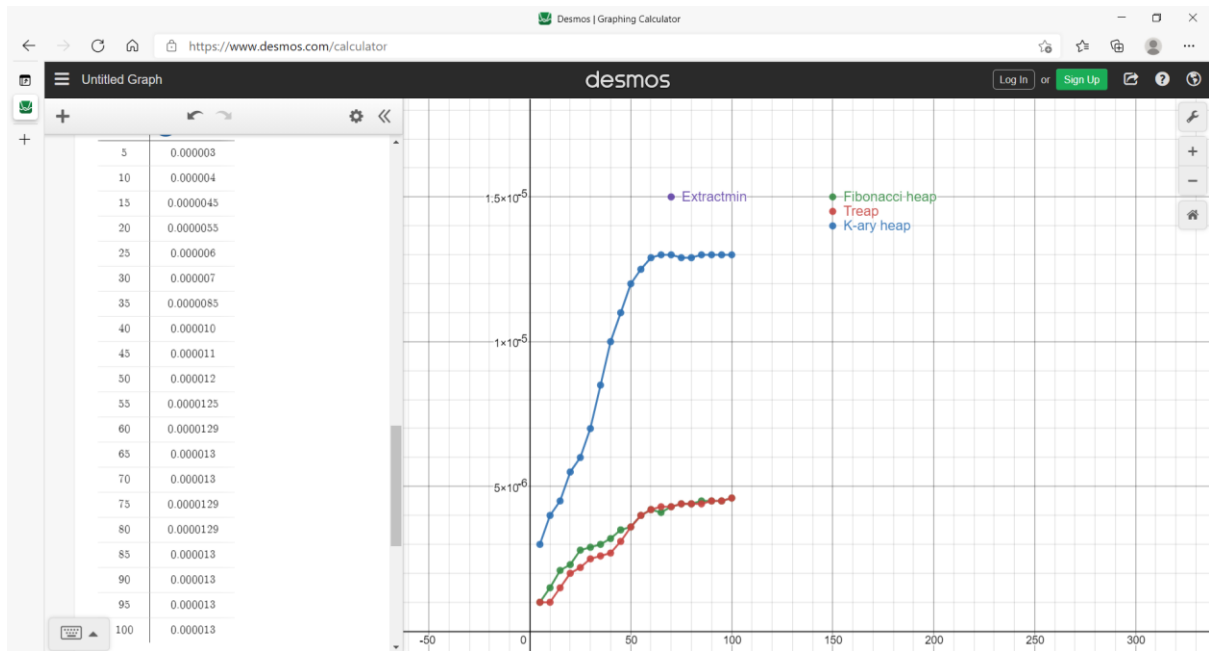


Inference:

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:



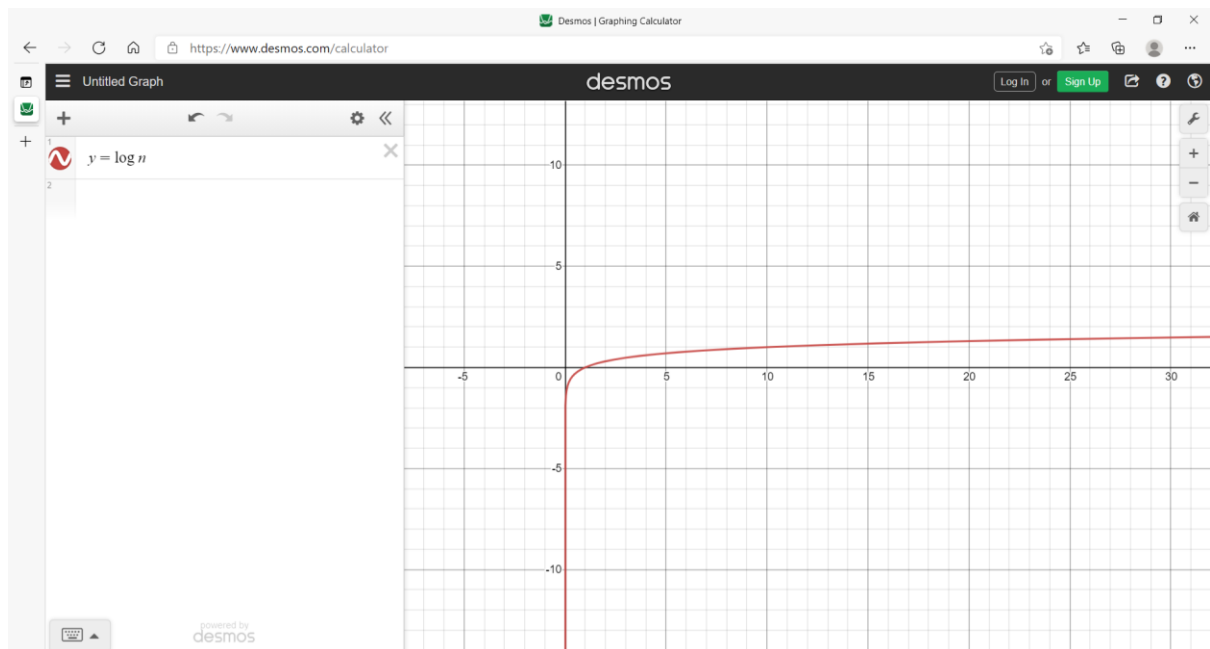
Inference:

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 $O(\log(n))$.

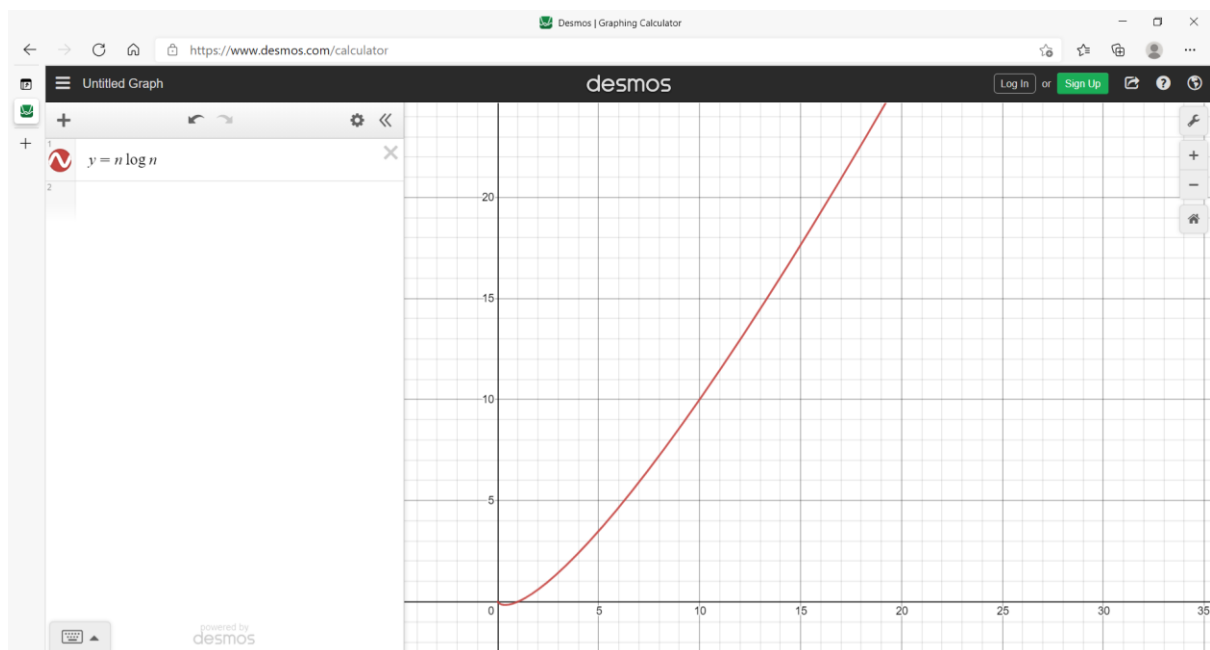
K-ary heap is slower than the other two. But there is a similarity between them, it increases then becomes constant. The graph's magnitude is thrice the magnitude of the other graph. Since, here we used K value as 3, it is thrice. It varies with the value of k. Therefore, the time complexity of K-ary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

References:

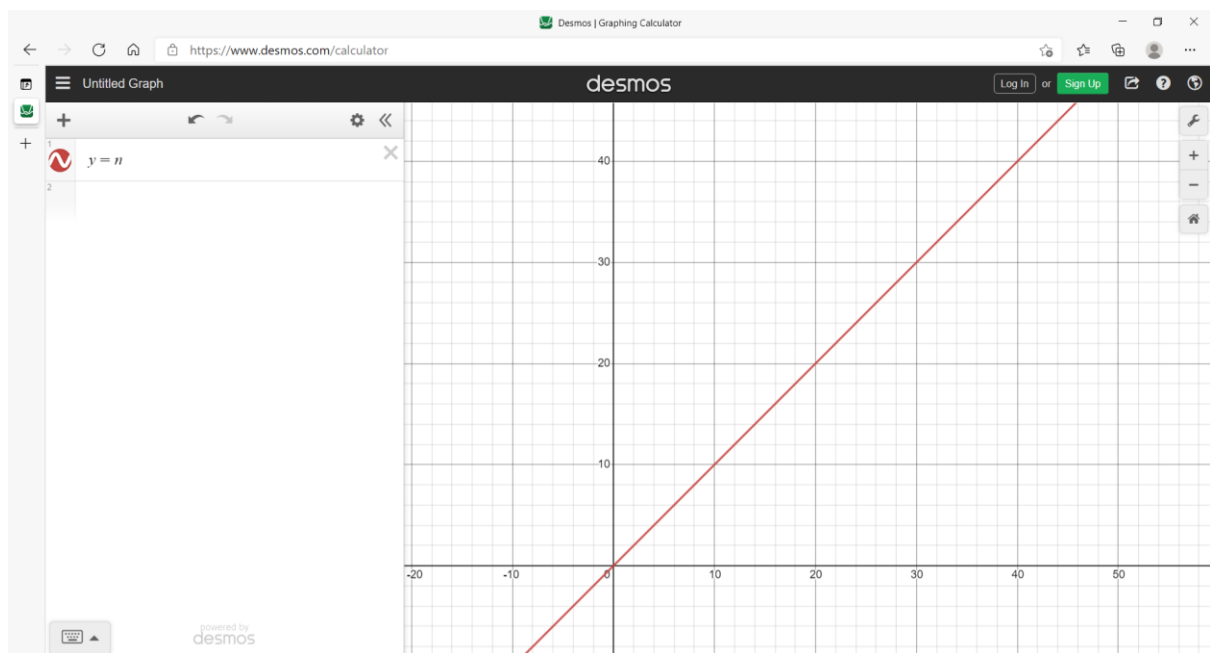
($y = \log(n)$) graph:



($y = n \cdot \log(n)$) 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.