Welcome To My Presentation

My Presentation Topics is

Heap Sort

Presented By

Tushar Sarkar

Student ID : 18CSE35

Second Year Second Semester

Department of CSE, BSMRSTU.

Content

- Heap Sort?
- What is Heap?
- Max/Min Heap.
- Procedure on Heap.
- Example of Heap Sort.
- Pseudo Code
- Complexity

Heap Sort

- Heap Sort is a popular and efficient sorting algorithm in computer programming.
- Learning how to write the heap sort algorithm requires knowledge of two types of data structures - arrays and trees.
- Heap sort works by visualizing the elements of the array as a special kind of complete binary tree called a heap.

What is Heap?

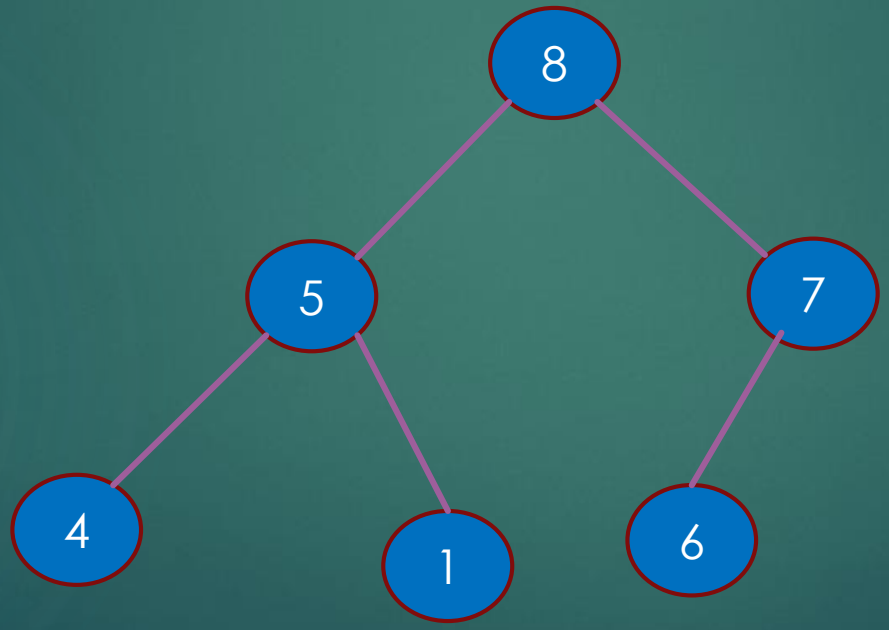
- A Heap is a special Tree-based data structure in which the tree is a complete binary tree.
- Generally, Heaps can be of two types:

1.Max Heap

2.Min Heap

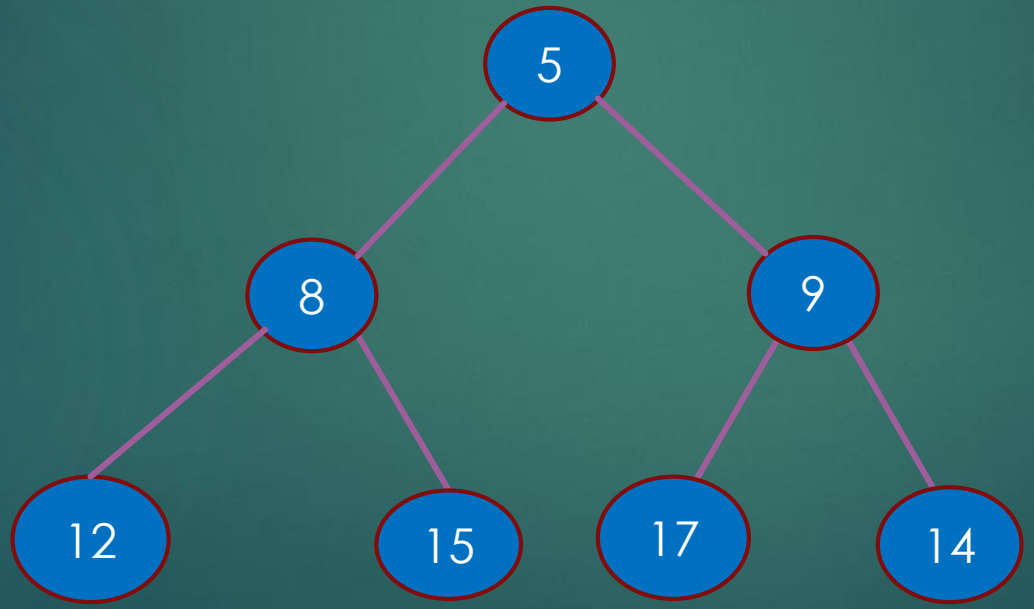
Max Heap

- Root node must be greater than those child node.
- Store data in ascending order



Min Heap

- Root node must be lower than child node.
- Store data in descending order



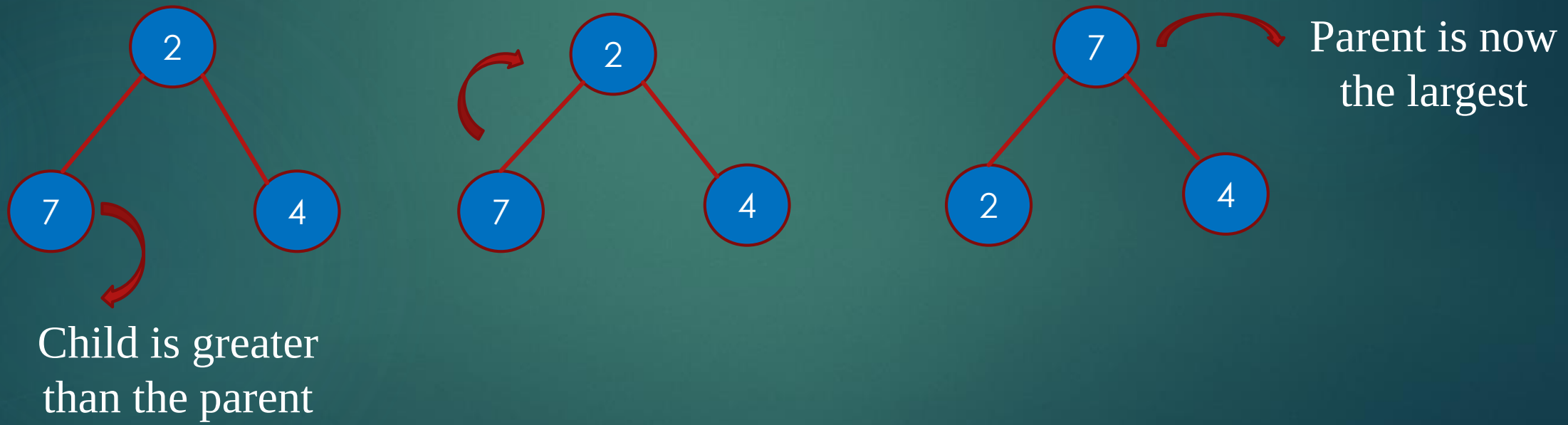
Procedures on Heap

- Following procedures are apply on heap-

1. Heapify
2. Build Heap
3. Heap Sort

Heapify

- Starting from a complete binary tree, we can modify it to become a Max-Heap by running a function called heapify on all the non-leaf elements of the heap.



Build Heap

- To build a max-heap from any tree, we can thus start heapifying each sub-tree from the bottom up and end up with a max-heap after the function is applied to all the elements including the root element.

```
Buildheap(A){
        heapsize[A] ← length[A]
        for i ← length[A]/2          //down to 1
        do Heapify(A, i)
}
```

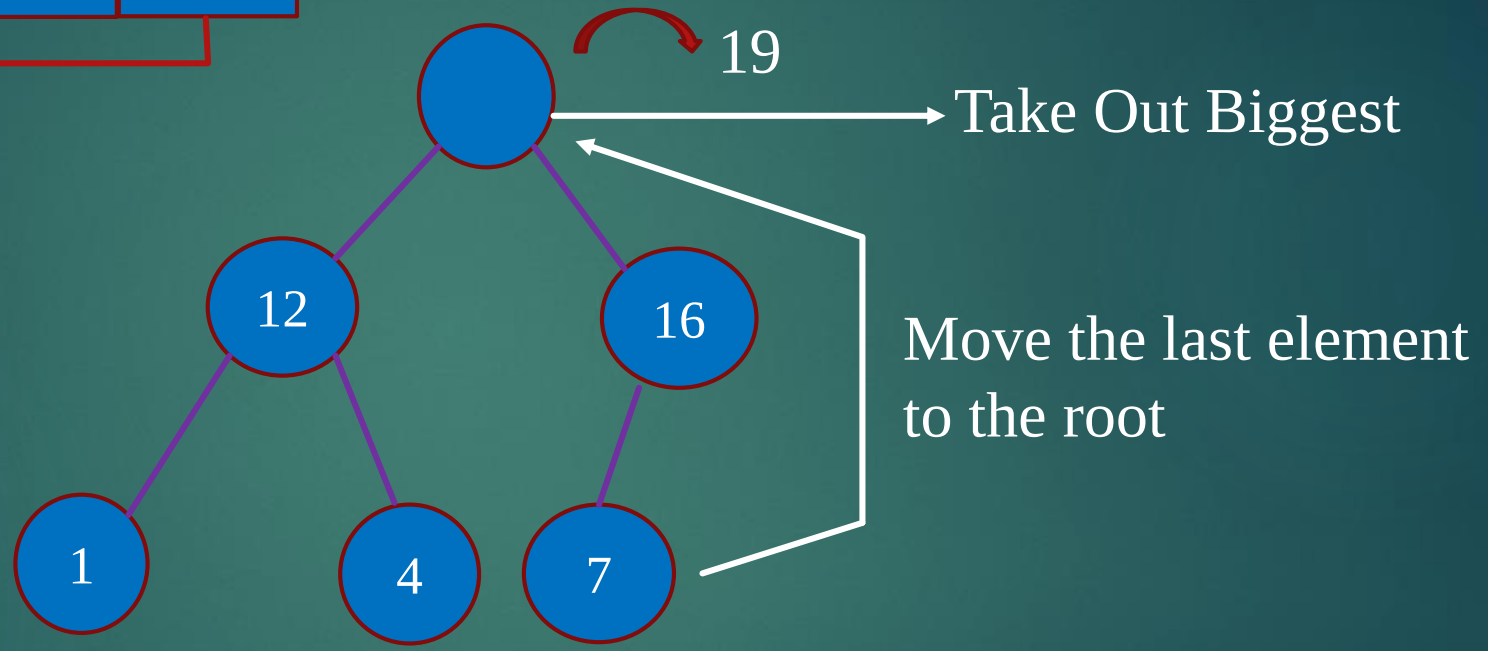
Heap Sort Algorithm

- The heap sort algorithm starts by using procedure BUILD-HEAP to build a heap on the input array A[1 . . *n*]. Since the maximum element of the array stored at the root *A*[1], it can be put into its correct final position by exchanging it with *A*[*n*] (the last element in *A*).

```
Heapsort(A){
        Buildheap(A)
            for i ← length[A]      //down to 2
        do swap A[1] ←→ A[i]
                heapsize[A] ← heapsize[A] - 1
                Heapify(A, 1)
}
```

Example of Heap Sort

Array of A

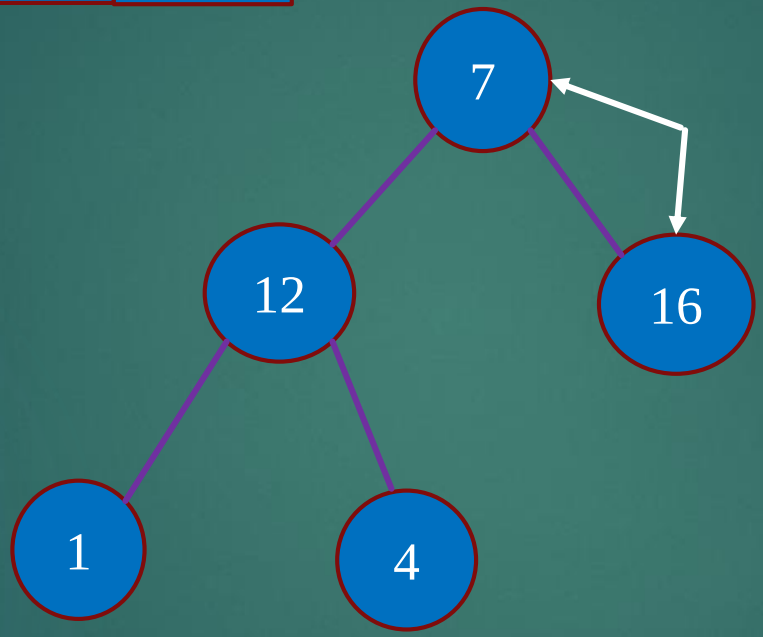


Sorted: 19

Example of Heap Sort

7	12	16	1	4

Array of A

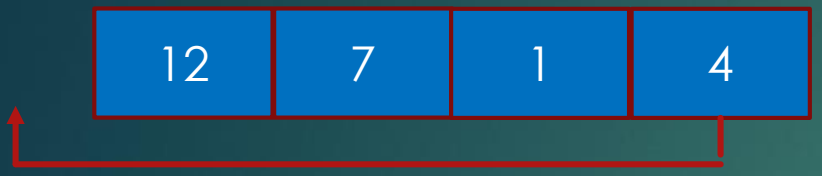


Swap

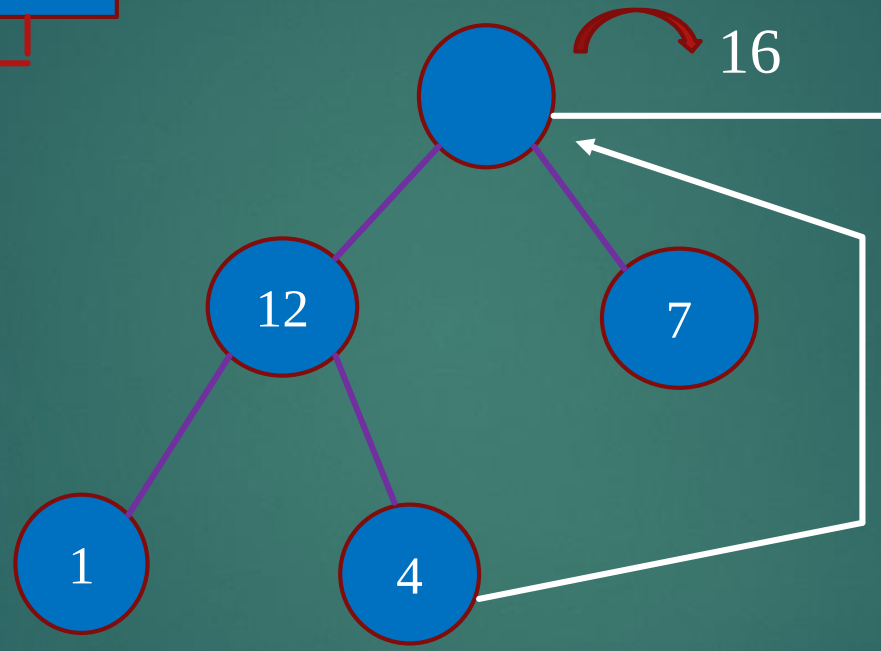
Heapify

Sorted: 19

Example of Heap Sort



Array of A



Take Out Biggest

Move the last element to the root

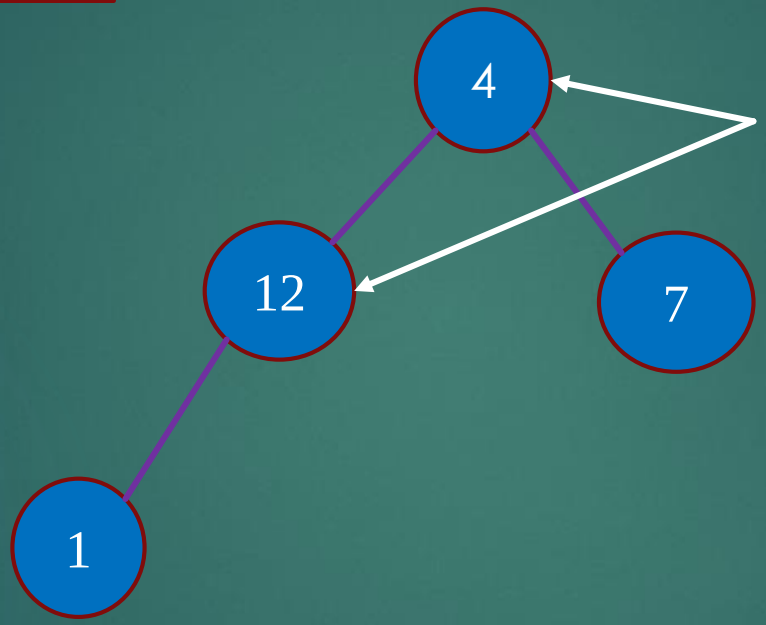
Sorted:

16	19

Example of Heap Sort

4	12	7	1

Array of A



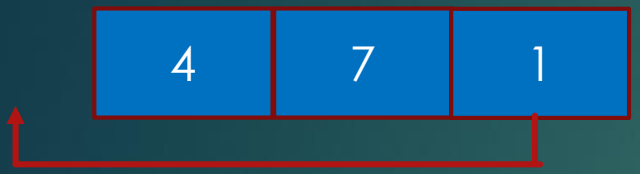
Swap

Heapify

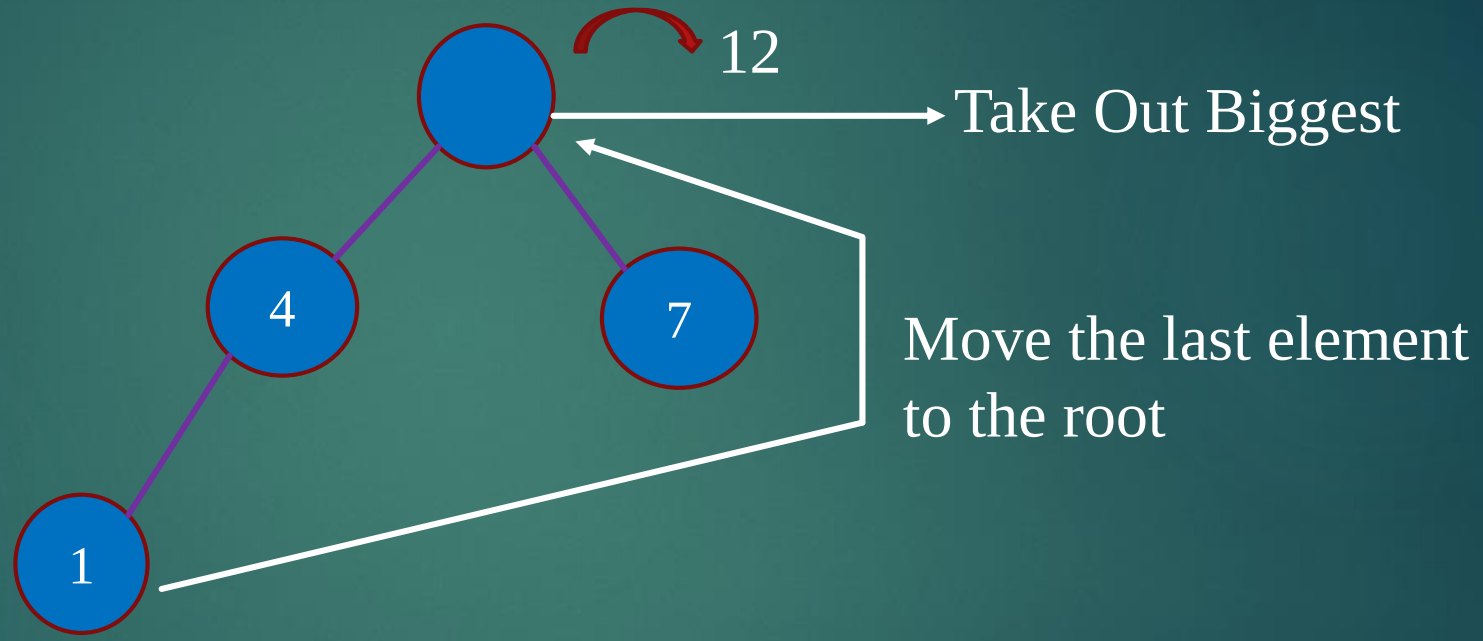
Sorted:

16	19

Example of Heap Sort



Array of A



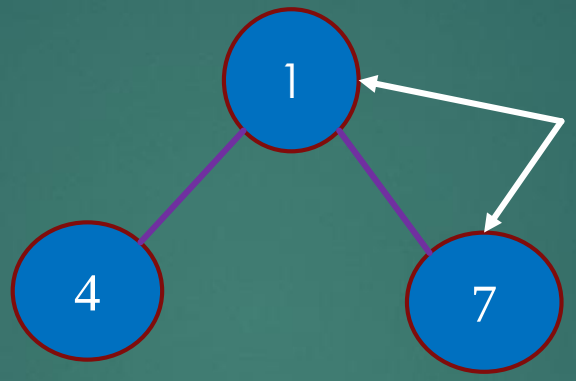
Sorted:

12	16	19

Example of Heap Sort

1	4	7

Array of A



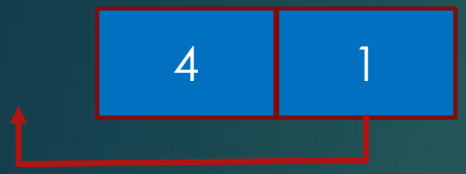
Swap

Heapify

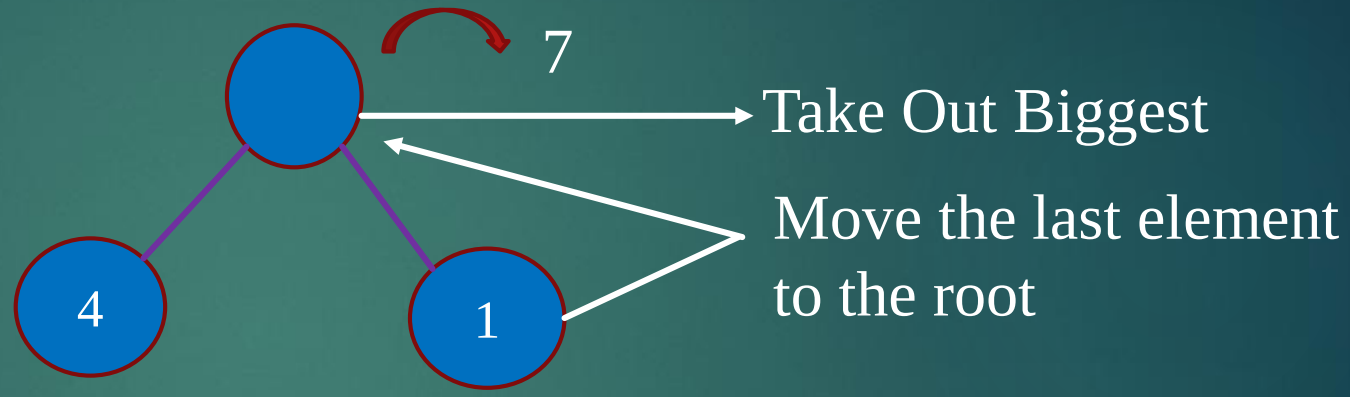
Sorted:

12	16	19

Example of Heap Sort



Array of A



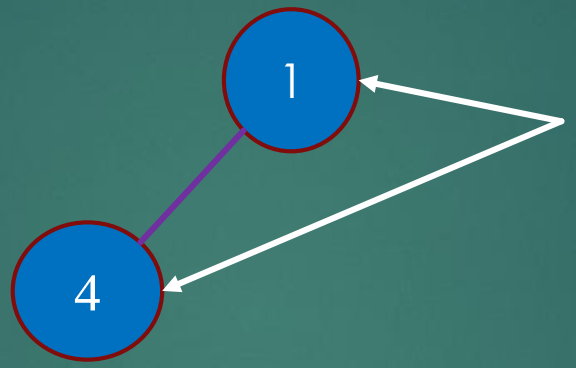
Sorted:

7	12	16	19

Example of Heap Sort

1	4

Array of A



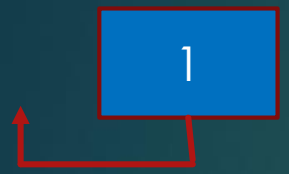
Swap

Heapify

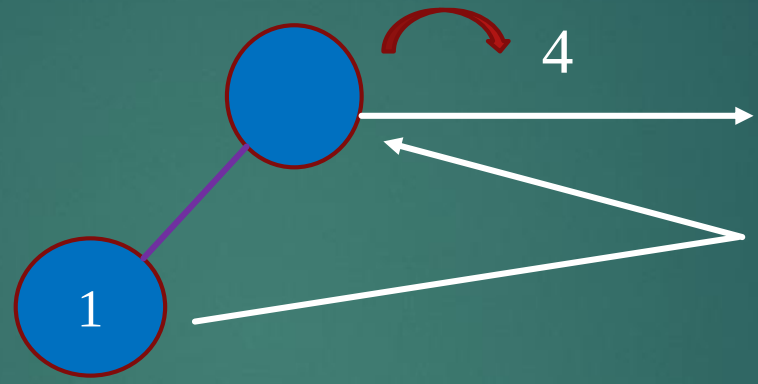
Sorted:

7	12	16	19

Example of Heap Sort



Array of A



Take Out Biggest

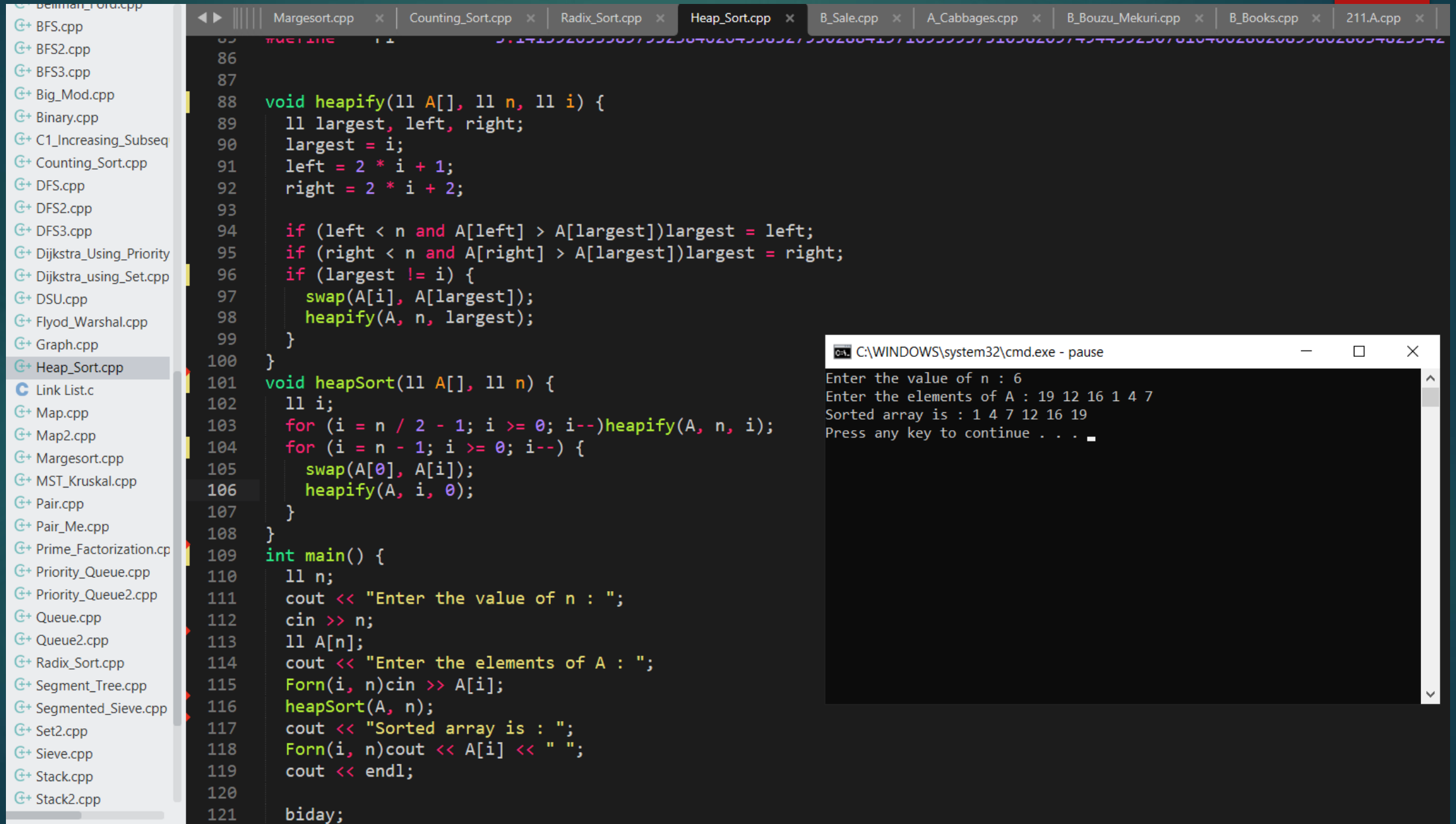
Move the last element to the root

Sorted:

4	7	12	16	19

Example of Heap Sort

Pseudo Code


BFS.cpp
BFS2.cpp
BFS3.cpp
Big_Mod.cpp
Binary.cpp
C1_Increasing_Subseq
Counting_Sort.cpp
DFS.cpp
DFS2.cpp
DFS3.cpp
Dijkstra_Using_Priority
Dijkstra_using_Set.cpp
DSU.cpp
Flyod_Warshal.cpp
Graph.cpp
Heap_Sort.cpp
Link List.c
Map.cpp
Map2.cpp
Margesort.cpp
MST_Kruskal.cpp
Pair.cpp
Pair_Me.cpp
Prime_Factorization.cp
Priority_Queue.cpp
Priority_Queue2.cpp
Queue.cpp
Queue2.cpp
Radix_Sort.cpp
Segment_Tree.cpp
Segmented_Sieve.cpp
Set2.cpp
Sieve.cpp
Stack.cpp
Stack2.cpp
Margesort.cpp
Counting_Sort.cpp
Radix_Sort.cpp
Heap_Sort.cpp
B_Sale.cpp
A_Cabbages.cpp
B_Bouzu_Mekuri.cpp
B_Books.cpp
211.A.cpp
86
87
88 void heapify(ll A[], ll n, ll i) {
89 ll largest, left, right;
90 largest = i;
91 left = 2 * i + 1;
92 right = 2 * i + 2;
93
94 if (left < n and A[left] > A[largest])largest = left;
95 if (right < n and A[right] > A[largest])largest = right;
96 if (largest != i) {
97 swap(A[i], A[largest]);
98 heapify(A, n, largest);
99 }
100 }
101 void heapSort(ll A[], ll n) {
102 ll i;
103 for (i = n / 2 - 1; i >= 0; i--)heapify(A, n, i);
104 for (i = n - 1; i >= 0; i--) {
105 swap(A[0], A[i]);
106 heapify(A, i, 0);
107 }
108 }
109 int main() {
110 ll n;
111 cout << "Enter the value of n : ";
112 cin >> n;
113 ll A[n];
114 cout << "Enter the elements of A : ";
115 Forn(i, n)cin >> A[i];
116 heapSort(A, n);
117 cout << "Sorted array is : ";
118 Forn(i, n)cout << A[i] << " ";
119 cout << endl;
120
121 biday;
C:\WINDOWS\system32\cmd.exe - pause
Enter the value of n : 6
Enter the elements of A : 19 12 16 1 4 7
Sorted array is : 1 4 7 12 16 19
Press any key to continue . . .

Complexity

Best Case	Worst Case	Average Case
O(n*logn)	O(n*logn)	O(n*logn)

Thank You