Heap Sort

Print this page

- **Heap sort is an efficient** sorting algorithm with average and worst case time complexities are in O(n log n).
- Heap sort does not use any extra array, like merge sort.
- This method is based on a data structure called **Heap.**
- Heap data structure can also be used as a priority queue.

Heap:

- A binary heap is a complete binary tree in which each, node other than root is smaller than its parent.
- Heap example:

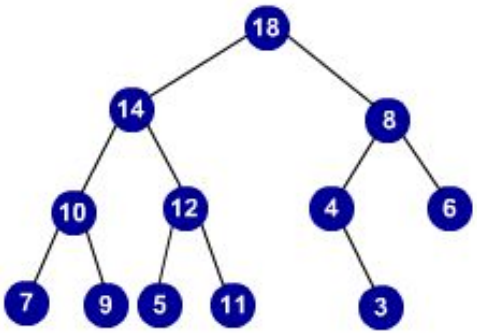


Fig 1

Start Next

Heap Sort

Print this page

Heap Representation:

- An efficient representation of the heap is using array.
- The root is stored at the first place, that is a[1].
- The children of the node i is at the locations 2*i and 2*i +1.
- In other words, the parent of a node stored in i th location is at floor(i/2).
- The array representation of a heap given in the figure below.

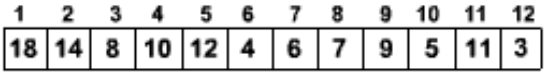


Fig 2

Prev Next

Heap Sort Print this page

Heapification

- Before discussing the method for building heap of an arbitrary complete binary tree, we discuss simpler problem.
- Let consider a binary tree in which left and right subtrees of the root are satisfying the heap property, but not the root. See the following figure.

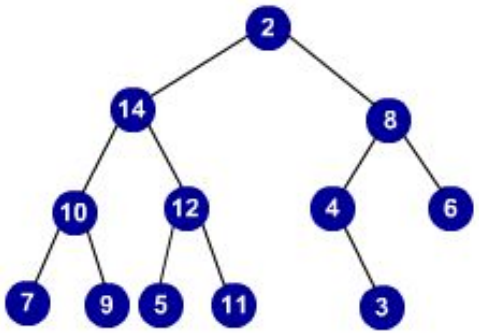


Fig 3

Heap Sort

Print this page

- Now question is how to make the above tree into a heap?
- Swap the root and left child of root, to make the root satisfies the heap property.
- Then check the subtree rooted at left child of the root is heap or not.If it is, we are done. If not, repeat the above action of swapping the root with the maximum of its children.
- That is, push down the element at root till it satisfies the heap property.
- The following sequence of figures depicts the heapification process.

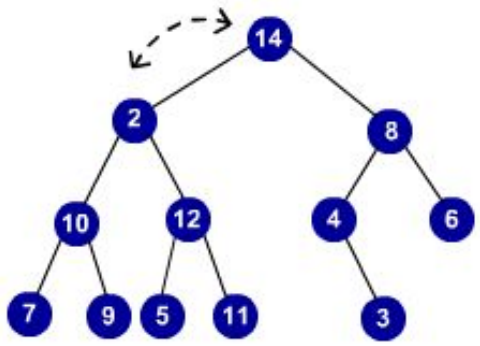


fig 4

Prev **Next**

Heap Sort

Print this page

algorithm Heapification (a,i,n)

```
left = 2^i
right = 2^i +1
if (left <n) and (a[left] > a[i]) then

        maximum = left

else

        maximum = i

if (right <n) and (a[right] > a[maximum]) then

        maximum = right

if (maximum != i) then

        swap (a[i],a[maximum])
        hepfication(a, maximum)
```

• The time complexity of the heapification is in O(log n).

Prev Next

Heap Sort

Print this page

Build Heap

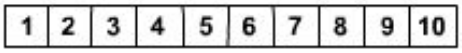
- Heap building can be done efficiently from bottom up fashion.
- Given an arbitrary complete binary tree, we can assume each leaf is a heap.
- Start building the heap from the parents of these levies. That is, heapify subtrees rooted at the parents of leaves.
- Then heapify subtrees rooted at their parents. Continue this process till we reach the root of the tree.

Prev **Next**

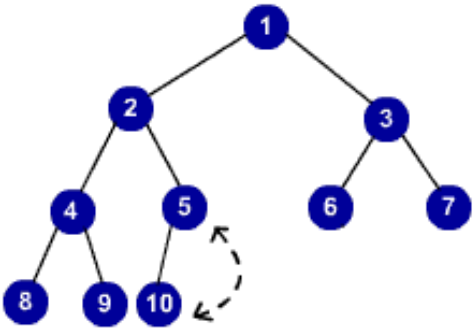
Heap Sort

Print this page

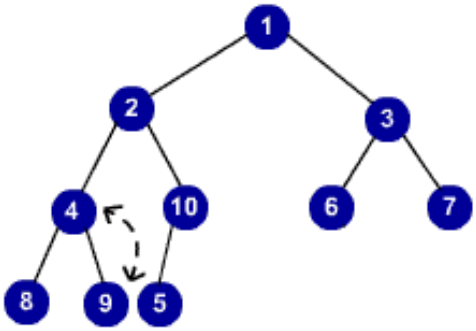
The following sequence of the figures illustrates the build heap procedure.



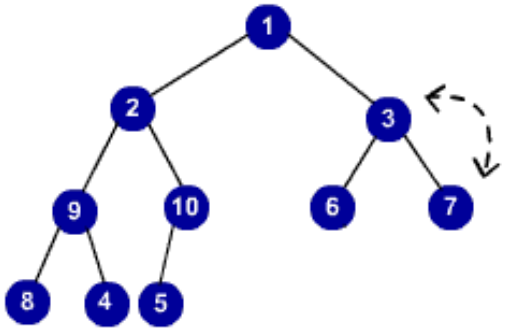
(a)



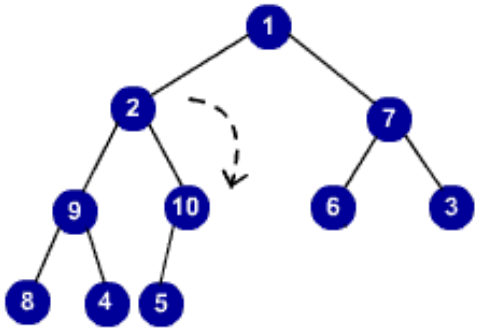
(b)



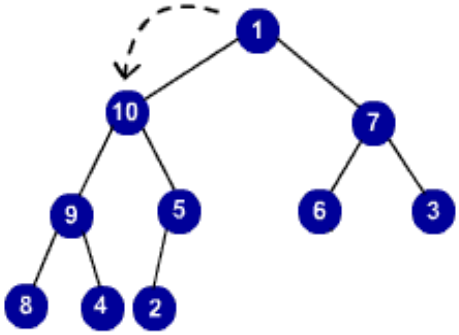
(c)



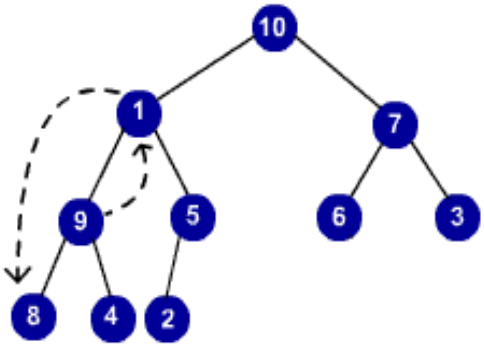
(d)



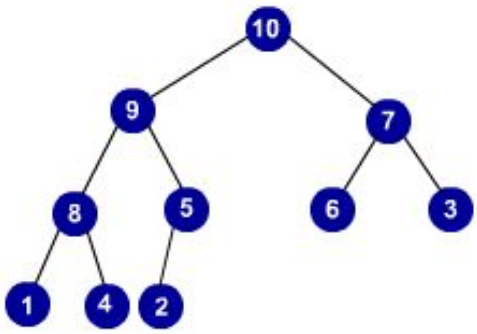
(e)



(f)



(g)



(h)

Prev Next

Heap Sort

Print this page

algorithm build_heap(a,i,n)

```
for j = floor(i/2) down to 1 do
        heapification(a,j,n)
```

• The time complexity of the build heap is in O(n).

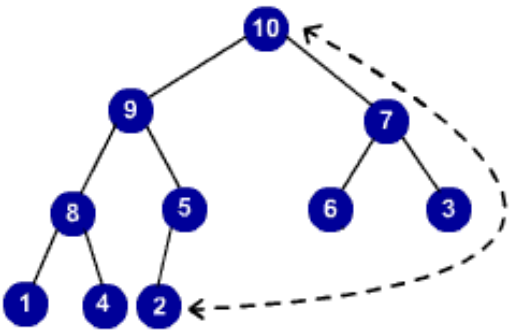
Prev **Next**

Heap Sort

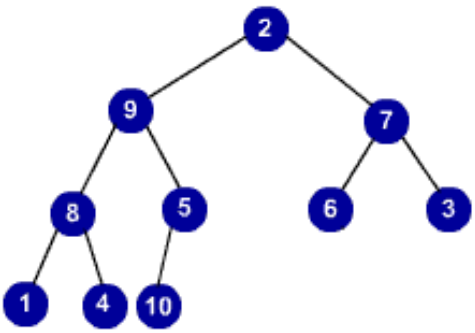
Print this page

Heap sort

• Given an array of n element, first we the build heap.
• The largest element is at the root, but its position in sorted array should be at last. So, swap the root with the last element.
• We have placed the highest element in its correct position. We left with an array of n-1 elements. repeat the same of these remaining n-1 elements to place the next largest element in its correct position.
• Repeat the above step till all elements are placed in their correct positions.



(a)

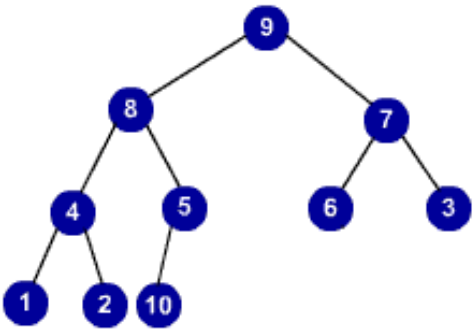


(b)

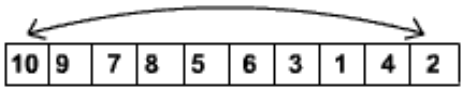
Prev Next

Heap Sort

Print this page



(c)



(d)

(e)

• Pseudocode of the algorithm is given below.

Algorithm Heap_Sort(a,i)

```
build_heap(a,i)
for j = i down to 1 do

        swap (a[1],a[j])
        heapification(a,1,j-1)
```

• The time complexity of the heap sort algorithm is in O(n log n)

Prev Next

Heap Sort

Print this page

Priority Queue

• Let consider a set S of elements {s1,s2,...}, such that each element si has priority pi.
• We want to design an data structure for these elements such that the highest priority element should be extracted/ deleted efficiently.
• We can use heap for this purpose since highest element, is always at the root, which can be extracted quickly. • Pseudocode for the extracting the maximum from the priority queue P is given below. Let the global variable size maintains the number of elements in P.

Algorithm Max_Extract(P)

```
max = p[1]
P[1] = P[size]
size = size -1
heapification(P,1,size)
return (max)
```

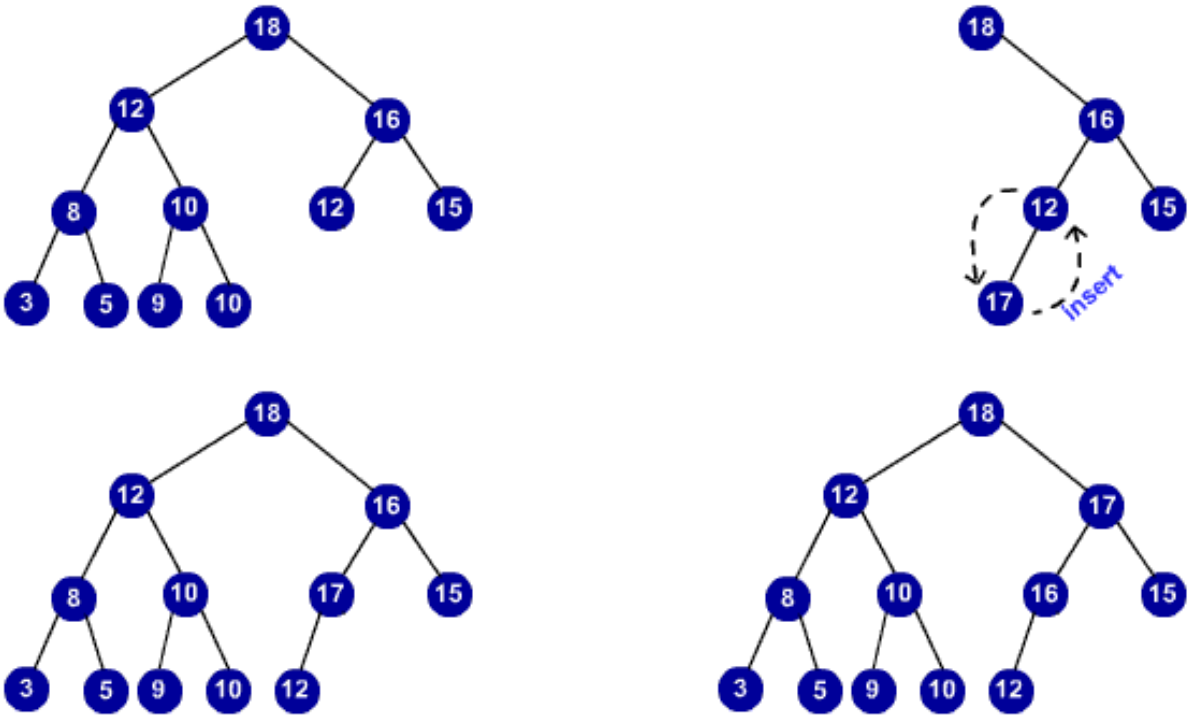
- After extracting the maximum, we have to maintain remaining elements in the priority queue. So we hepify before returning the maximum.
- Other operation to be supported is to insert an element into the priority queue.
- Inserting an element into the priority queue can be done easily.
- Insert the new element as a new leaf, and push this up till it satisfies the heap property.

Prev Next

Heap Sort

Print this page

- The following figure illustrates the inserting procedure.



Final Heap

Prev Next

Heap Sort

Print this page

• The pseudo code is given below.

Algorithm Insert(P, x)

```
size =size +1
i = size
while ( i > 1 ) and (x > P[floor(i/2)]) do

        P[i] = P[floor(i/2)]
        i = floor(i/2)

P[i] = x
```

Prev Next

Heap Sort

Print this page

Problems

• Write a program for heap sort.
• Given an array of n elements sorted in non-ascending order. Is it a heap?
• Heapify the array {9, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10}. Show that tree structure after each operation.
• Design an heap sort algorithm to sort in non-ascending order.
• Modify the priority queue algorithms as that the smaller the value of the priority higher the priority.

Prev **End**