Heap sort and Priority Queue

Dr. Sreeja S R

Heap-Sort

Goal:

– Sort an array using heap representation

Idea:

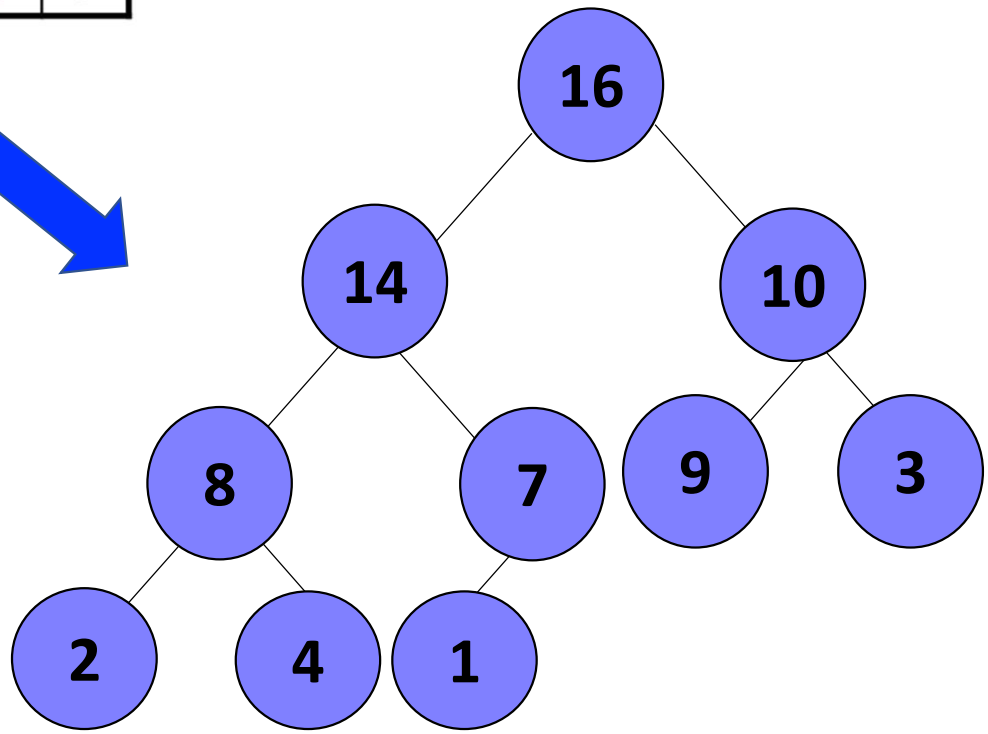
1. **BUILD-MAX-HEAP** from unordered array
2. Find maximum element $A[1]$
3. Swap elements $A[n]$ and $A[1]$:
   now max element is at the end of the array!
4. Discard node n from heap (by decrementing heap-size variable)
5. New root may violate max heap property, but its children are max heaps. Run **MAX-HEAPIFY** to fix this.
6. Go to Step 2 unless heap is empty.

Heap-Sort - Example

A

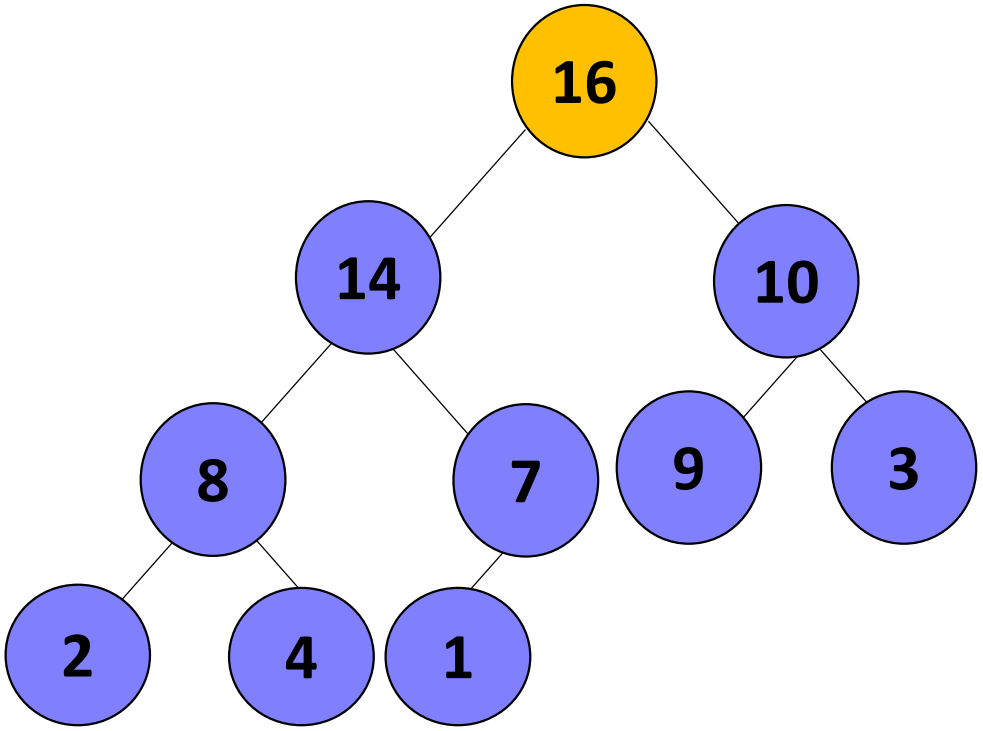
4	1	3	2	16	9	10	14	8	7

1. BUILD-MAX-HEAP

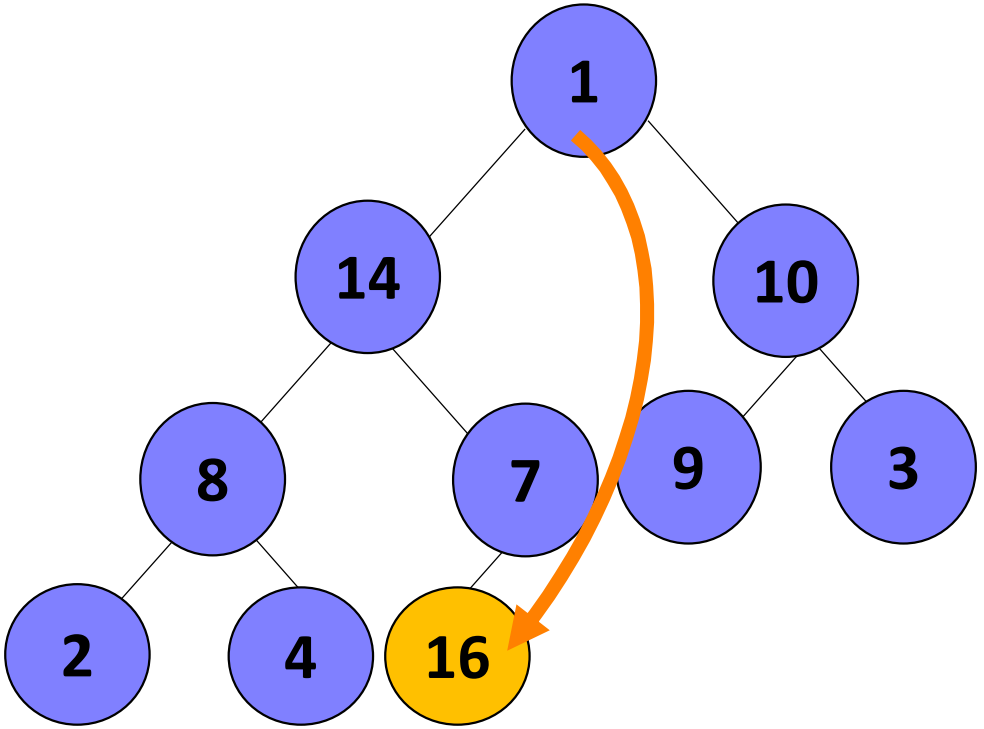


Heap-Sort - Example

2. Find maximum element $A[1]$

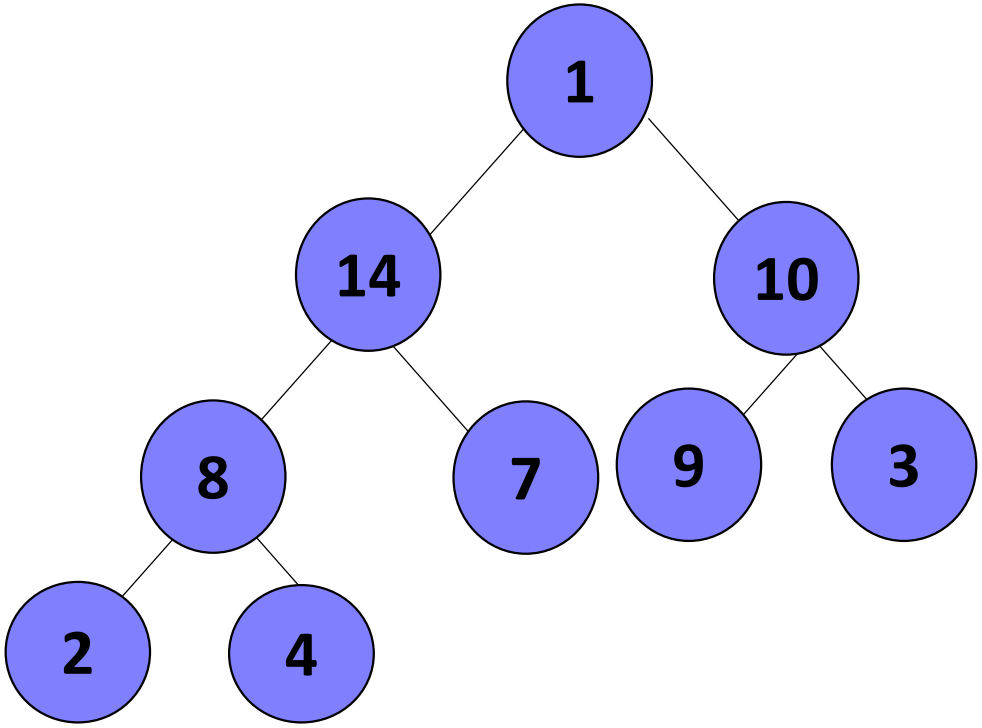


3. Swap elements $A[n]$ and $A[1]$

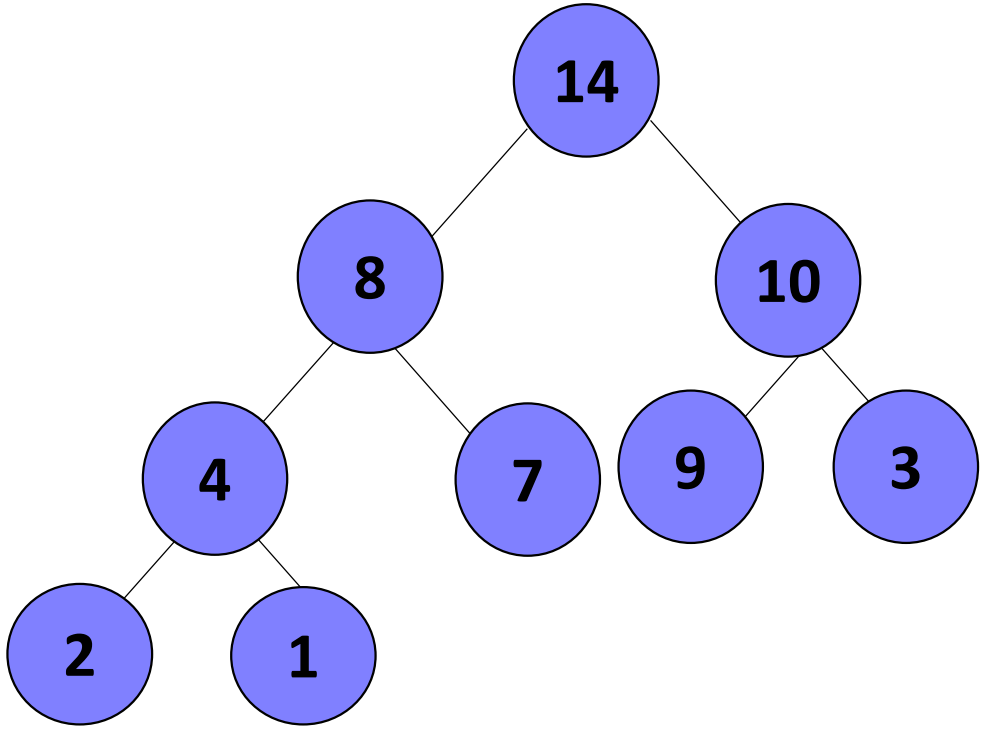


Heap-Sort - Example

4. Discard node n from heap

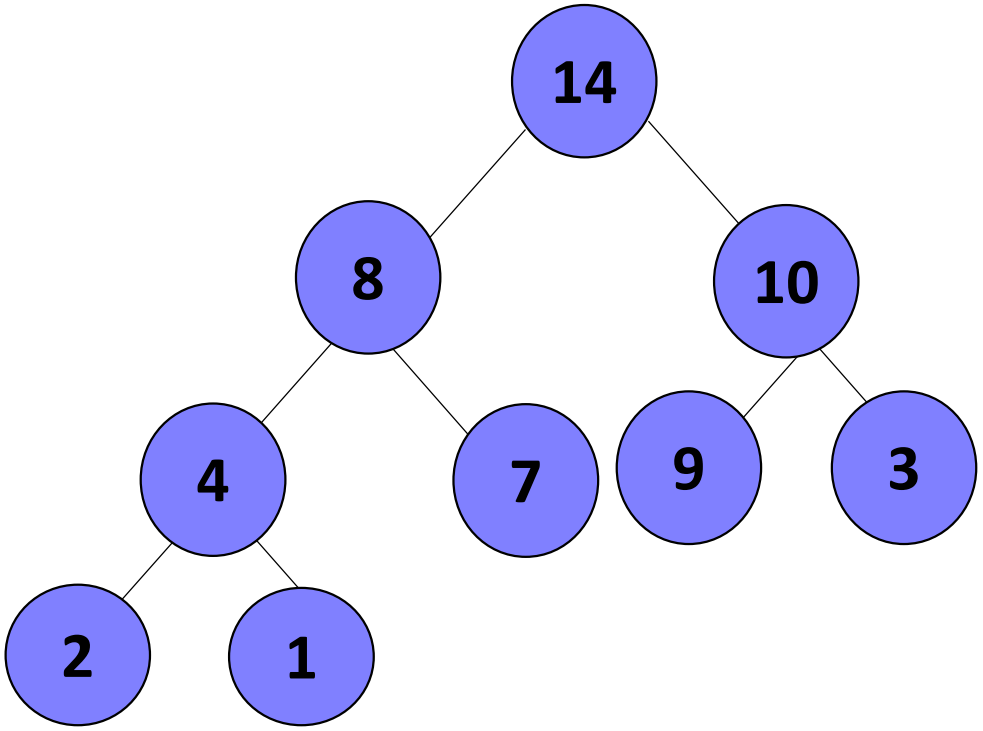


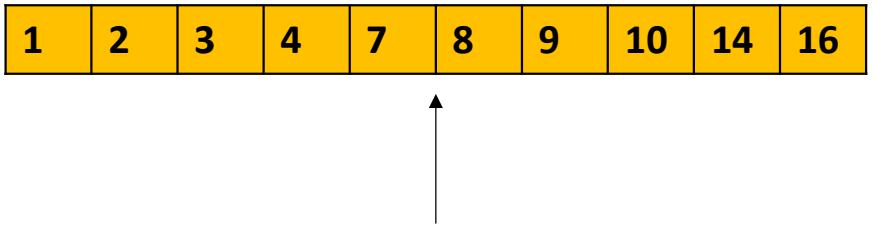
5. Run **MAX-HEAPIFY**



Heap-Sort - Example

6. Repeat the process until heap is empty





The elements which is removed from the heap

Running Time of HEAPSORT

1. BUILD-MAX-HEAP(A) — O(n)
2. **for** i ← length[A] **downto** 2
3. **do** exchange $A[1]$ $A[i]$
4. MAX-HEAPIFY(A, 1, i - 1) → O(log n)

O(n-1)

Running time of HEAPSORT: O(n logn)

HEAPSORT

```
 void heapify(int arr[], int n, int i)
{
  // Find largest among root, left child and right child
  int tmp;
  int largest = i;
  int l = 2*i + 1;
  int r = 2*i + 2;

  if (l < n && arr[l] > arr[largest])
    largest = l;

  if (r < n && arr[r] > arr[largest])
    largest = r;

  // Swap and continue heapifying if root is not largest
  if (largest != i)
  {
    //swap(arr[i], arr[largest]);
    tmp=arr[i];
    arr[i]=arr[largest];
    arr[largest]=tmp;
    printf("\n\tMaxHeap: ");
    printArray(arr,n);
    heapify(arr, n, largest);

  }
}
```

HEAPSORT

```
void heapSort(int arr[], int n)
{
    int i,tmp;
   // Build max heap
   for (i = n / 2 - 1; i >= 0; i--)
     heapify(arr, n, i);

   // Heap sort
   for (i=n-1; i>=0; i--)
   {
     //swap(arr[0], arr[i]);
     tmp=arr[0];
     arr[0]=arr[i];
     arr[i]=tmp;
     printf("\n\tHeap Sort::");
     printArray(arr,n);
     // Heapify root element to get highest element at root again
     heapify(arr, i, 0);
   }
}
```

HEAPSORT

```
void printArray(int arr[], int n)
{
  int i;
  for (i=0; i<n; ++i)
    printf(" %d ", arr[i]);
    printf("\n");
}

int main()
{
  int arr[] = {15,19,10,7,17,16};
  int n = sizeof(arr)/sizeof(arr[0]);
  printf( "UnSorted array is \n");
  printArray(arr, n);

  heapSort(arr, n);

  printf( "Sorted array is \n");
  printArray(arr, n);
  return 0;
}
```

Priority Queues

Properties:

- Each element is associated with a value (priority)
- The key with the highest (or lowest) priority is extracted first.

Major operations:

- insert(S, x) : insert element x into set S
- max(S) : return element of S with largest key
- extract_max(S) : return element of S with largest key and remove it from S
- increase_key(S, x, k) : increase the value of element x' s key to new value k

Priority Queue problem

- Where might we want to use heaps? Consider the Priority Queue problem
  - Given a sequence of objects with varying degrees of priority, and we want to deal with the highest-priority item first.
- Managing air traffic control
  - Want to do most important tasks first.
  - Jobs placed in queue with priority, controllers take off queue from top
- Scheduling jobs on a processor
  - Critical applications need high priority
- Event-driven simulator with time of occurrence as key.
  - Use min-heap, which keeps smallest element on top, get next occurring event.

Running time of priority queue with different representations

- Priority queue using **linked list**
  - Remove a key - O(1)
  - Insert a key - O(n)
  - Increase key - O(n)
  - Extract max key - O(1)

- Priority queue using **heaps**
  - Remove a key - O(log n)
  - Insert a key - O(log n)
  - Increase key - O(log n)
  - Extract max key - O(logn)

Thank you!