Homework : Heaps

1. In an array representation of a binary heap, for an item in position *i*, where are the parent, left child, and right child located?
2. For **max** heap, show the result of inserting 7, 20, 2, 15, 5, 21, 6, 12, 1, 8, 3, 9, 10 and 24, one at a time, into an initially empty heap. **Show the heap after each insertion!**
3. Show the heap from the previous question after deleting seven most maximum elements.
4. Show heap after inserting 99, 1, 55, 16, 28, 33, 599, 19 and 0, one at a time, into the heap in the previous question.
5. What is the greatest number of nodes that can appear at level *n* of a heap?
6. What is the maximum number of nodes that can appear in a heap of height *n*?
7. Given a heap of size *n* what is the minimum number of comparisons required to find the second largest element in the max heap?
8. Given a heap of size *n* what is the minimum number of comparisons required to find the smallest element in the max heap?
9. What is the running time of the `insert` operation on a heap containing `n` elements ? Give result in big-oh and explain how you come up with it.
10. Write pseudocode and find worst case running time (big O) of the following operations.

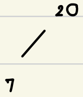
    - `build_heap : build a priority queue from a given set of items`
    - `find_max   : find the largest item in the priority queue`
    - `delete     : remove the largest item from the priority queue`

1. item in position i :

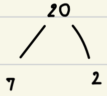
parent : (i - 1) / 2
left child : 2i + 1
right child : 2i + 2

2 insert 7

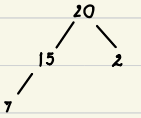
insert 20



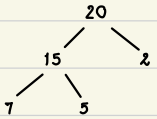
insert 2



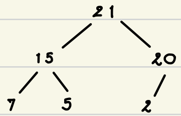
insert 15



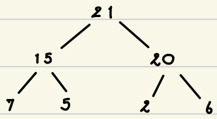
insert 5



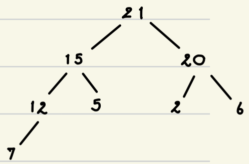
insert 21



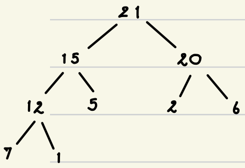
insert 6



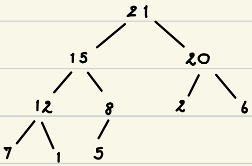
insert 12



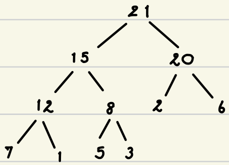
insert 1



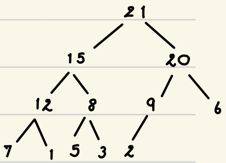
insert 8



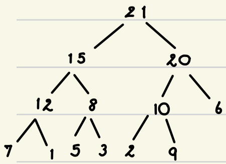
insert 3



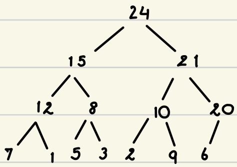
insert 9



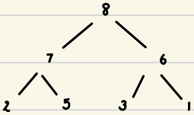
insert 10



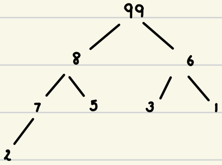
insert 24



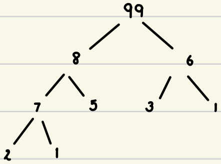
3



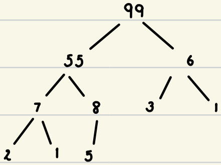
4 insert 99



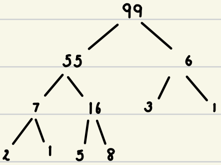
insert 1



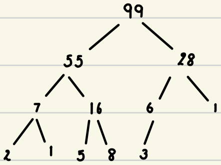
insert 55



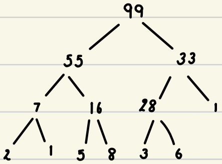
insert 16



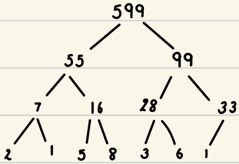
insert 28



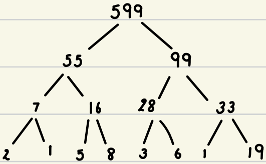
insert 33



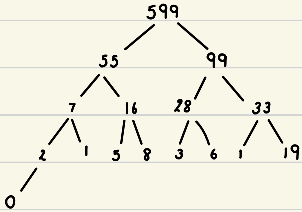
insert 599



insert 19



insert 0



5. 2^{n-1}

6. $2^{n} - 1$

7. 1

8. log n

9. O (log n) because the worst case is that we have to swap number for every level, for the heap of size n is log n

10.
```
build_heap (items):
    heap ← list
    for item in items
        heap.append (item)
          n ← length of heap - 1
          while n > 0 then
              temp ← n/2
              if heap [n] > heap [temp] then
                  st ← heap [n]
                  heap [n] ← heap [temp]
                  heap [temp] ← heap [n]
                      n ← temp
                  end if
              else
                  break
          return heap
```

O(n) : $n \log_2 n$

```
find_max (heap)
    return heap [0]
```

O(n) : 1

```
delete(heap)
    st ← heap.pop()
     temp ← 0
     while temp * 2 + 1 < length of heap
         if temp* 2 + 2 < length of heap
             max ← max ( heap [temp], heap [temp* 2 + 1], heap [temp* 2 + 2 ])
               if max ← heap [temp* 2 +1] then
                   heap [temp] ← heap [temp *2 +1]
                   heap [temp* 2 + 1] ← st
                   temp ← temp* 2 +1
               if max ← heap [temp * 2 + 2] then
                 heap [temp] ← heap [temp *2 +2]
                 heap [temp * 2 + 2] ← st
                 temp ← temp * 2 + 2
               else
                 break
          else
             if heap [temp*2 +1] > heap [temp]
               heap [temp] ← heap [temp *2 +1]
               heap [temp* 2 +1] ← st
               temp ← temp *2 +1
             return heap

     O(n) : log n
```