CS5800: Algorithms — Spring '21 — Virgil Pavlu

Homework 2A
Due : Gradescope

Name: Shri Datta Madhira (NUID:001557772)

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

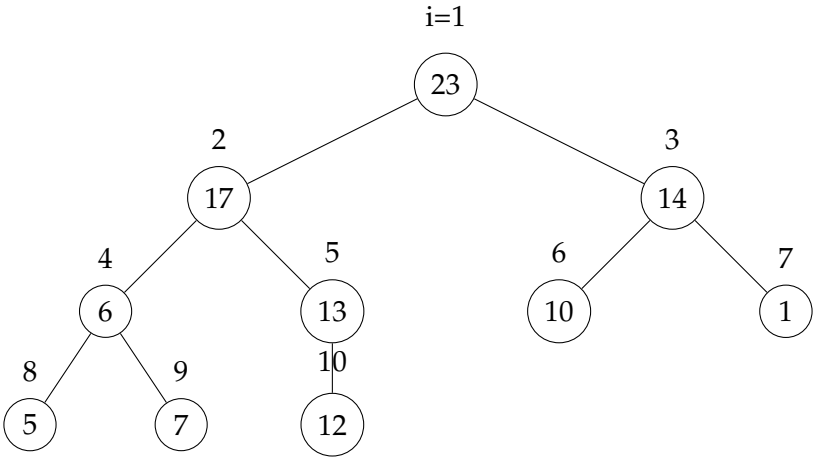
1. ***(5 points)*** *Exercise 6.1-4, explain why.*

Solution:

The smallest element in a max-heap will reside **at the leaf**. As the max-heap property selects the largest value among the parent, left child, and right child; the smallest value among these is usually floated down. And if we apply this recursively to the entire sub-tree, we will end up floating the smallest element down to the leaf. Also, because all the elements are distinct, there cannot be a similar value as the parent of the smallest value. Therefore, leaf is the only spot.

2. ***(5 points)*** *Exercise 6.1-6, explain why.*

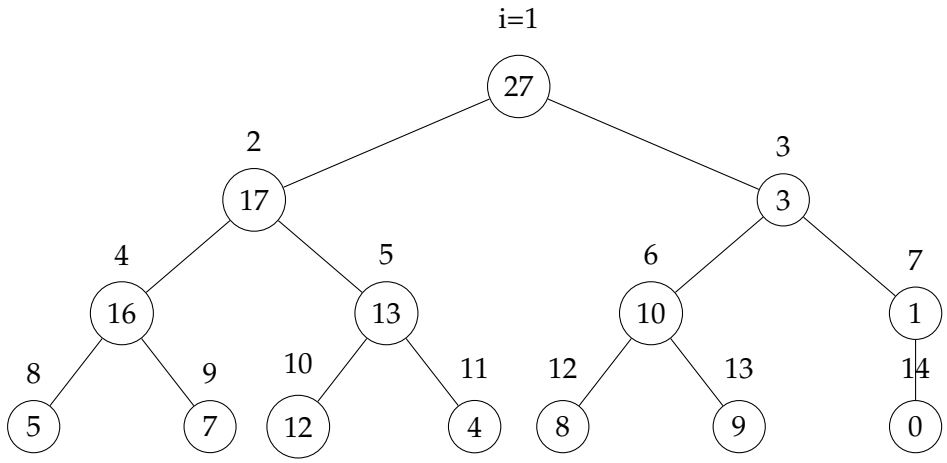
Solution: The array given in the question can be represented as a tree that looks as follows,



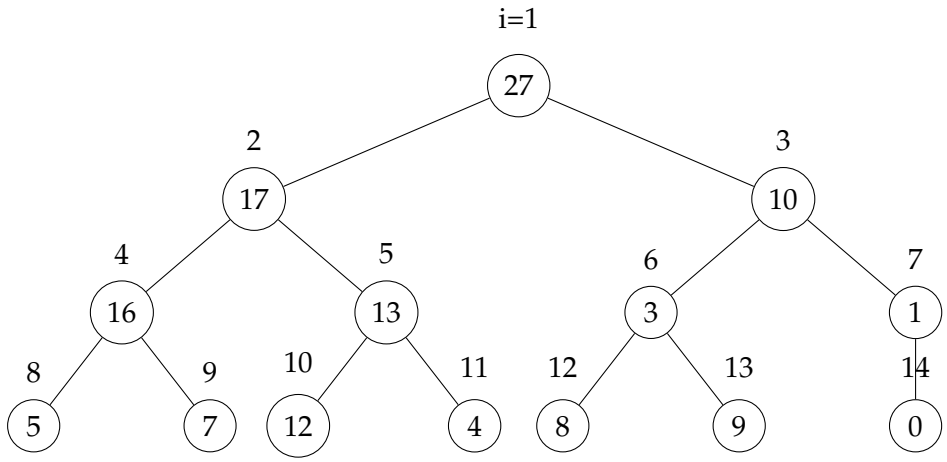
As we can see, the max-heap property is violated by elements **6 (position 4)** and **7 (position 9)** as **6 < 7**. Therefore, the values in the given array are not a max-heap.

3. ***(10 points)*** *Exercise 6.2-1.*

Solution: Initial Tree,

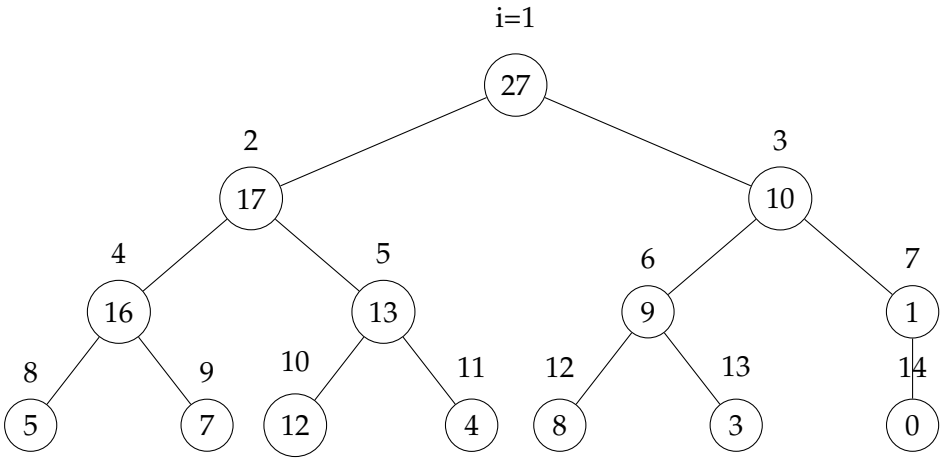



```
MAX_HEAPIFY (A, 3):
    left = 6, right = 7
    A[3] < A[6], A[3] > A[7] --> largest = 6
    swap (A[3], A[6])
```



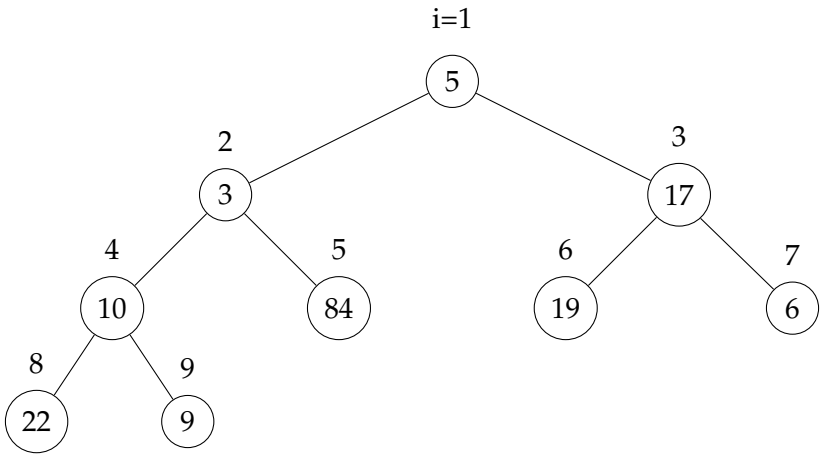

```
MAX_HEAPIFY (A,6):
    left = 12, right = 13
    A[6] < A[13], A[6] < A[12] and A[12] < A[13] --> largest = 13
    swap (A[6], A[13])
    MAX_HEAPIFY (A, 13):
        Leaf Node. So Return.
```

Therefore, the final tree after all the MAX-HEAPIFY operations will look as follows,

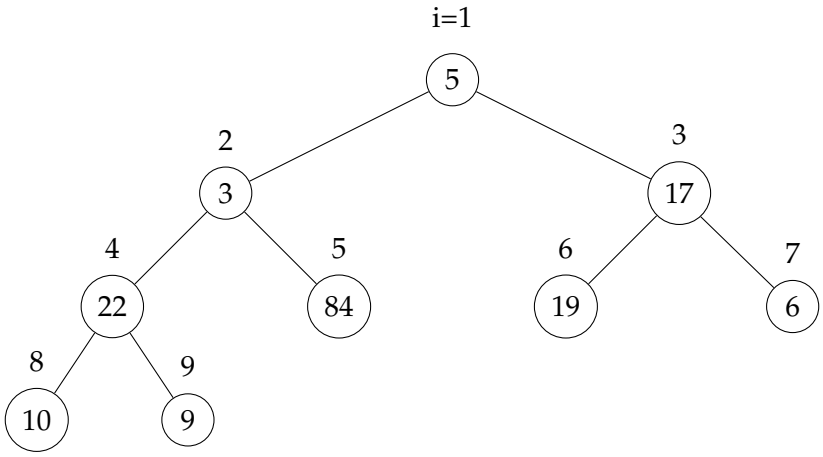

i=1
27
2
17
3
10
4
16
5
13
6
9
7
1
8
5
9
7
10
12
11
4
12
8
13
3
14
0

4. ***(10 points)*** *Exercise 6.3-1.*

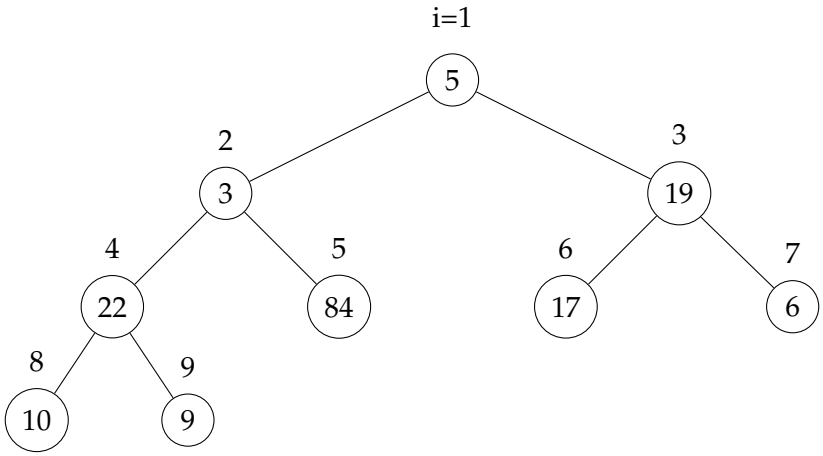
Solution: Initial Tree,



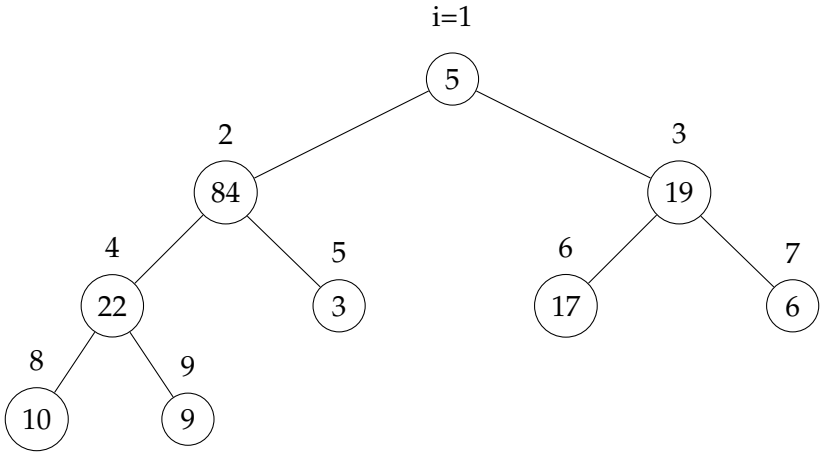

```
BUILD-MAX-HEAP (A):
    for i = 4 down to 1:
        i = 4 ----> MAX-HEAPIFY(A, 4):
                        left = 8, right = 9
                        A[4] < A[8], A[4] > A[9] ---> largest = 8
                        swap (A[4], A[8]) ----> MAX-HEAPIFY(A, 8):
                                                    Leaf Node. So, Return.
```




```
    i = 3 ----> MAX-HEAPIFY(A, 3):
                    left = 6, right = 7
                    A[3] < A[6], A[3] > A[7] ---> largest = 6
                    swap (A[3], A[6]) ----> MAX-HEAPIFY(A, 6):
                                                Leaf Node. So, Return.
```

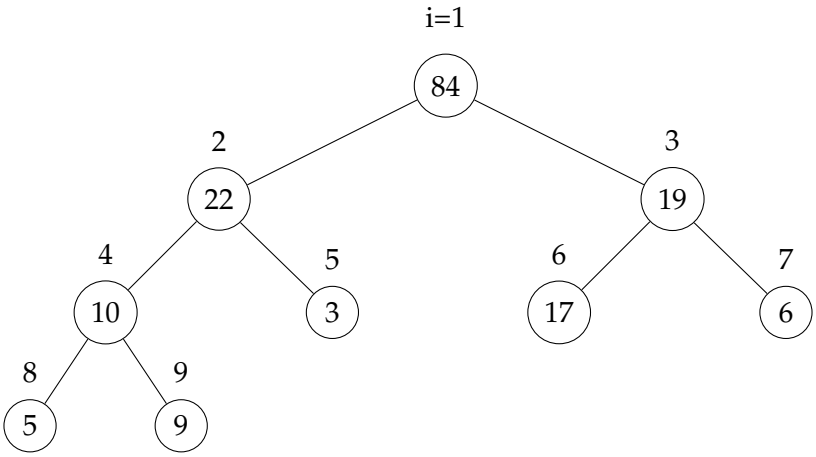



```
i = 2 ----> MAX-HEAPIFY(A, 2):
                left = 4, right = 5
                A[2] < A[5], A[2] < A[4], A[4] < A[5] ---> largest = 5
                swap (A[2], A[5]) ----> MAX-HEAPIFY(A, 5):
                                            Leaf Node. So, Return.
```




```
i = 1 ----> MAX-HEAPIFY(A, 1):
                left = 2, right = 3
                A[1] < A[2], A[3] < A[2] --> largest = 2
                swap (A[1], A[2]) ----> MAX-HEAPIFY(A, 2):
                                            left = 4, right = 5
                                            A[2] < A[4] --> largest = 4
                                            swap (A[2], A[4]) ---->
                                            MAX-HEAPIFY(A, 4):
                                                left = 8, right = 9
                                                A[4] < A[8] ---> largest = 8
                                                swap (A[4], A[8]) ---->
                                                MAX-HEAPIFY(A,8):
                                                    Leaf Node. So, Return.
```

After *"BUILD – MAX – HEAP"* is performed on the given array, the binary tree representation of the heap would look something as follows,



5. ***(15 points)*** *Problem 6-2.*

Solution:

(a) It can be represented as the normal array. Array[1] will be the root, Array[$2 \cdots$ d] are level-1 of the d-ary heap, and so on.
The children of the parent can be represented by **CHILDREN(i) = id - j + 2** where $1 \leq j \leq d$ and **PARENT(i) = floor($\frac{i-2}{d}$ + 1)**

(b) Let 'h' be the height of the tree and 'n' be the total number of nodes. The first layer of a d-ary heap contains **'d'** nodes. The second layer contains $\boldsymbol{d^2}$ nodes. The second layer contains $\boldsymbol{d^3}$ nodes, and the 'h' layer contains $\boldsymbol{d^h}$ nodes. This can be represented as,

$$1 + d + d^2 + \cdots + d^{h-1} \leq n \leq 1 + d + d^2 + \cdots + d^h$$

$$\frac{d^h - 1}{d - 1} \leq n \leq \frac{d^{h+1} - 1}{d - 1}$$

$$d^h - 1 \leq n(d-1) \leq d^{h+1} - 1$$

$$d^h \leq n(d-1) + 1 \leq d^{h+1}$$

$$h \leq \lg_d [n(d-1) + 1] \leq h + 1$$

Here, n(d - 1) + 1 will always be less than nd. So, substituting that we get,

$$h \leq \lg_d nd \leq h + 1$$

$$h \leq \lg_d n + \lg_d d \leq h + 1$$

$$h \leq \lg_d n \leq h + 1$$

Therefore, the height 'h' of the b-ary heap in terms of 'n' and 'd' is bound by $\lg_d n$, in other words $Height(h) = \boldsymbol{\Theta(\lg_d n)}$.

(c)
```
EXTRACT-MAX (A):
    max = A[1]
    A[1] = A[n]
    n -= 1
    MAX-HEAPIFY(A,1)
    return max
```

In the above function every step takes a constant time except for the MAX-HEAPIFY step that takes $\Theta(\lg_d n)$ because it has to traverse the tree to find the correct position for the new root. Therefore, the running time for EXTRACT-MAX is $\boldsymbol{\Theta(\lg_d n)}$.

(d)
```
INSERT (A, key):
    n += 1
    A[n] = key
    i = n
    while A[i] > A[PARENT(i)]:
        swap(A[i], A[PARENT(i)]
        i = PARENT(i)
```

In the above function every step takes a constant time except for the while loop which takes $\Theta(\lg_d n)$ because it has to traverse the tree from the end of the tree towards the root. Therefore, the running time for INSERT is $\boldsymbol{\Theta(\lg_d n)}$.

(e)
```
INCREASE-KEY (A, i, key):
    if A[i] > key:
        "ERROR: Current value is greater."
    A[i] = key
    while A[i] > A[PARENT(i)]:
        swap(A[i], A[PARENT(i)])
        i = PARENT(i)
```

In the above function every step takes a constant time except for the while loop which takes $\Theta(\lg_d i)$ because it has to traverse the tree from the end of the tree towards the root. Therefore, the running time for INSERT is $\boldsymbol{\Theta(\lg_d i)}$.

6. ***(5 points)*** *Exercise 7.2-1.*

Solution:

$$T(n) = T(n-1) + n$$

$$T(n) = T(n-2) + (n-1) + n$$

$$T(n) = T(n-3) + (n-2) + (n-1) + n$$

We can guess the pattern of the above sequence as,

$$\Rightarrow T(n) = T(n-k) + (n-k+1) + (n-k+2) + \cdots + (n-1) + n$$

Taking n = k we get,

$$T(n) = T(0) + (1 + 2 + \cdots + (n-1) + n)$$

$$T(n) = T(0) + \frac{n(n+1)}{2}$$

$$T(n) = T(0) + \frac{n^2 + n}{2}$$

$$\boldsymbol{T(n) = \Theta(n^2)}$$

7. ***(5 points)*** *Exercise 7.2-2, explain why.*

Solution: When all the elements are the same, we will always end up with the entire array on one side (unbalanced). So, an array of size 'n' will be divided into (n-1) and the pivot. So, we will get $T(n) = T(n-1) + \Theta(n)$. Therefore, the running time of QUICKSORT when all the elements have the same value is $\boldsymbol{\Theta(n^2)}$.

8. ***(5 points)*** *Exercise 7.2-3.*

Solution: Same as the question above, when the pivot ends up at the extremes, the array get skewed in one direction making the recurrence relation $T(n) = T(n-1) + \Theta(n)$. Therefore, the running time of QUICKSORT when all the elements are in the decreasing order is $\boldsymbol{\Theta(n^2)}$.

9. ***(15 points)*** *Exercise 7.4-1.*

Solution:
Given,

$$T(n) = \max_{0 \le q \le \{n-1\}} (T(q) + T(n-q-1)) + \Theta(n)$$

Let us assume, $T(n) \ge cn^2$

$$T(n) \ge \max_{0 \le q \le \{n-1\}} (cq^2 + c(n-q-1)^2) + \Theta(n)$$

$$\ge c. \max_{0 \le q \le \{n-1\}} (q^2 + (n-q-1)^2) + \Theta(n)$$

As mentioned in the textbook, the above equation is at its maximum at q = n-1. So, we get,

$$T(n) \ge c(n-1)^2 + \Theta(n)$$

$$T(n) \ge cn^2 - 2nc + c + \Theta(n)$$

The biggest term of the above equation is cn^2. So, we can say that

$$T(n) \ge cn^2$$

$$\boldsymbol{T(n) = \Omega(n^2)}$$

Hence Proved.

10. ***(15 points)*** *Exercise 7.4-2.*

Solution:
The best case for QUICKSORT happens when the array is partitioned exactly in half. This gives us the recurrence relation similar to that of MERGE SORT which is,

$$T(n) = 2T(n/2) + \Theta(n)$$

Assuming $T(n) \geq cn\lg n$,

$$T(n) \geq 2(c(\frac{n}{2})\lg(\frac{n}{2})) + \Theta(n)$$

$$T(n) \geq cn\lg(\frac{n}{2}) + \Theta(n)$$

Since we can choose a value for 'c' large enough that makes $cn\lg(\frac{n}{2})$ greater than and comprehensively dominate $\Theta(n)$, we can ignore it from the equation. Therefore, we get,

$$T(n) \geq cn\lg n$$

$$\boldsymbol{T(n) = \Omega(n\lg n)}$$

Hence Proved.

11. ***(10 points)*** *Exercise 7.4-3.*

Solution: To show that the expression $q^2 + (n-q-1)^2$ achieves maximum at q = 0 and q = (n - 1) over q = 0, 1, 2, $\cdots$ n - 1, we have to do the following steps,

1. **Evaluate the expression at the endpoints**, q = 0 and q = (n - 1).
   When $\boldsymbol{q = 0} \Rightarrow q^2 + (n-q-1)^2 = (n-1)^2$, and
   when $\boldsymbol{q = (n-1)} \Rightarrow q^2 + (n-q-1)^2 = (n-1)^2$.

2. **Finding the critical points.** For this we need to calculate the $f'(x) = 0$.
   $f'(q) = 0 \Rightarrow 2q + 2q - 2(n-1) = 0 \Rightarrow q = \frac{n-1}{2}$

3. **Evaluate the expression at critical point.** So,
   When $\boldsymbol{q = \frac{n-1}{2}} \Rightarrow q^2 + (n-q-1)^2 = \frac{(n-1)^2}{2}$

4. **Compare the values obtained in steps 1 and 3. The largest is maxima and smallest is minima.**
   As we can see, the values for the expression at q = 0 and q = (n - 1) are **greater than** the value at the critical point, q = $\frac{n-1}{2}$.
   So, q = 0 and q = (n - 1) are maxima for our expression.
   Hence, we have proved that the expression $q^2 + (n-q-1)^2$ achieves maximum over q = 0 $\cdots$ n - 1 when q = 0 and q = n - 1.

The maxima-minima method is referenced from the website "https://openlab.citytech.cuny.edu/mat1475coursehub/lessons/lesson-15-maxima-and-minima/".

12. ***(Extra credit 10 points)*** *Problem 6-3.*

Solution:

(a)

$$\begin{pmatrix} 2 & 3 & 4 & 5 \\ 8 & 9 & 12 & 14 \\ 16 & - & - & - \\ - & - & - & - \end{pmatrix}$$

(b) If **Y[1,1] = ∞** the matrix will be empty because there will be no elements less than ∞.
If **Y[m,n] < ∞** then that means ∞ cannot be a part of the matrix. That means, the matrix is completely filled.

(c)