# Question 1

## Solution to (a)

Since every binary heap is a complete binary tree, every level, if fully filled, has $2^h$ nodes, where $h$ is the height of the tree. Therefore, the maximum number of nodes in a binary heap of height $h$ is

$$\begin{aligned} 1 + 2 + ... + 2^h &= \sum_{i=0}^{h} 2^i \\ &= \frac{2^{h+1} - 1}{2 - 1} \\ &= 2^{h+1} - 1 \end{aligned}$$

The minumum number of nodes in a binary heap of height $h$ is the maximum number of nodes in a binary heap of height $h - 1$ plus one, which is $2^h - 1 + 1 = 2^h$.
For $h = 6$

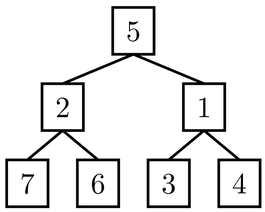
- Maximum number of nodes: $2^{6+1} - 1 = 127$
- Minimum number of nodes: $2^6 = 64$

## Solution to (b)

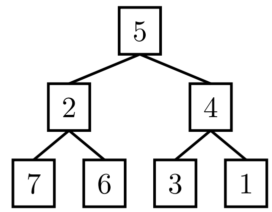
Given the array [5, 2, 1, 7, 6, 3, 4], and the construction rules that

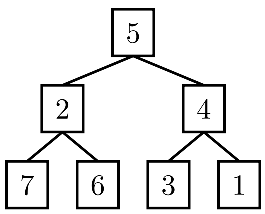
- left child of a node at index $i$ is at index $2i + 1$
- right child of a node at index $i$ is at index $2i + 2$
- parent of a node at index $i$ is at index $\lfloor \frac{i-1}{2} \rfloor$

we can build a binary heap as follows:

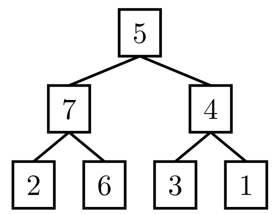


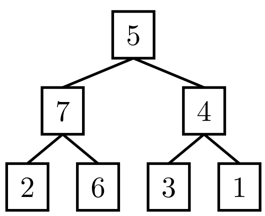

Node index at $\lfloor \frac{6-1}{2} \rfloor = 2$
Compare the 1 with 3 and 4
Swap 1 and 4
$\rightarrow$
[5,2,4,7,6,3,1]



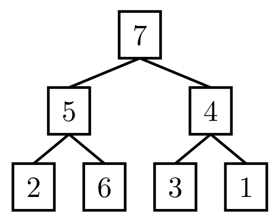



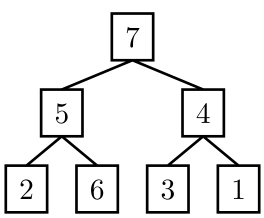

Node index at $2 - 1 = 1$
Compare the 2 with 7 and 6
Swap 2 and 7
$\rightarrow$
[5,7,4,2,6,3,1]



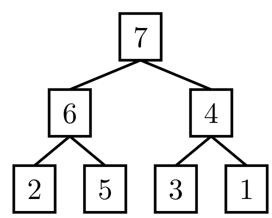




Node index at $1 - 1 = 0$
Compare the 5 with 7 and 4
Swap 5 and 7
$\rightarrow$
[7,5,4,2,6,3,1]



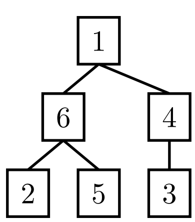




Recusive call on node index 1
Compare the 5 with 2 and 6
Swap 5 and 6
$\rightarrow$
[7,6,4,2,5,3,1]

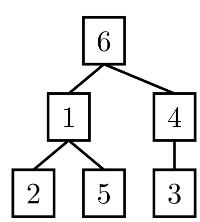



Extraction 1

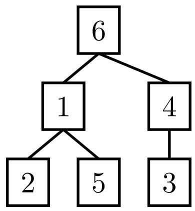
- swap 7 and 1
- then select the subarray from index 0 to n - 2
- and recusively heapify the root node:



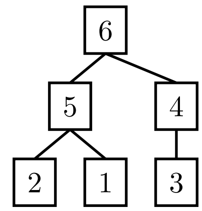

Recusive call on node 0
Compare the 1 with 6 and 4
Swap 1 and 6
$\rightarrow$
[6,1,4,2,5,3,7]



continue the Extraction 1:

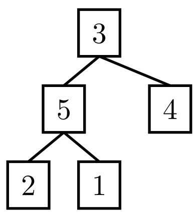


Recusive call on node 1
Compare the 1 with 2 and 5
Swap 1 and 5
→
[6,5,4,2,1,3,7]

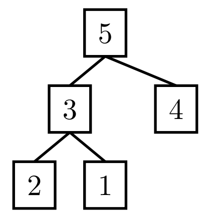


Extraction 2

- swap 6 and 3
- then select the subarray from index 0 to n - 3
- and recusively heapify the root node:



Recusive call on node 0
Compare 3 with 5 and 4
Swap 3 and 5
→
[5,3,4,2,1,6,7]

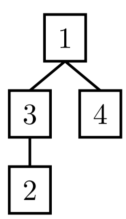


Recusive call on node 1

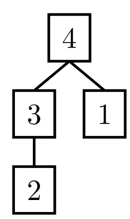
- compare 3 with 2 and 1
- no swap needed

Extraction 3

- swap 5 and 1
- the select the subarray from index 0 to n - 4
- and recusively heapify the root node:

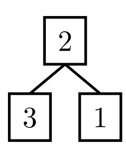


Recusive call on node 0
Compare 1 with 3 and 4
Swap 1 and 4
→
[4,3,1,2,5,6,7]

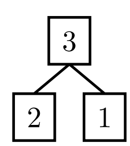


Extraction 4

- swap 4 and 2
- then select the subarray from index 0 to n - 5
- and recusively heapify the root node:

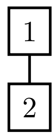


Recusive call on node 0
Compare 2 with 3 and 1
Swap 2 and 3
→
[3,2,1,4,5,6,7]

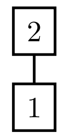


Extraction 5

- swap 3 and 1
- then select the subarray from index 0 to n - 6
- and recusively heapify the root node:

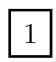


Recusive call on node 0
Compare 1 with 2
Swap 1 and 2
→
[2,1,3,4,5,6,7]



Extraction 6

- swap 2 and 1
- then select the subarray from index 0 to n - 7 = 1
- since there is only one element, the subarray is trivially heapified



→ [1,2,3,4,5,6,7]

Extration 7

- Get the last element 1

The order of the elements after extraction is [7, 6, 5, 4, 3, 2, 1] if we do not remove the last element, but take subarray of the first n-1 elements, we can get the original array sorted in place

## Solution to (c)

Proof by induction:

- Denote the array as $v[]$ and the length of the array as $n$. The $i$th element of the array is $v[i]$

Base case: $\text{iteration} = 1$

- Since the array was max heapfied before sort starts, the first element is the largest element.
- By the process of extraction, the largest element is swap with the last element (index = i - 1) of the array, therefore the last 1 element is trivially sorted.
- the MAX_HEAPIFY is called on subarray from index 0 to n - 2, and ensure that the first n-1 elements are max heapified.

Induction step:

- Assume that, after $\text{iteration} = k$, the first n-k elements are max heapified and last k elements are sorted in asceding order where $v[i] \leq v[j]\ i = 0, 1, ..., n - k - 1 \wedge j = n - k, ..., n - 1$
- Proceeding to $i = k + 1$, the first element $v[0]$ is the largest element in the first n-k elements, and is swapped with the last element of the first n-k subarray. Note that $v[0] \leq v[n - k]$. Therefore, last k+1 elements are still sorted in ascending order.
- The MAX_HEAPIFY is called on the first n-k-1 element , and ensure that the first n-k-1 elements are max heapified.
- By the induction hypothesis, the property holds for $\text{interation} = k + 1$ whenever it holds for $\text{interation} = k$. Since the property holds for $\text{interation} = 1$, the property holds for all $\text{interation} \in \mathbb{Z}+$.

## Solution to (d)

- The possible value for index 0 is 1, since the first element in a max-heapified array is the always the largest element in the array. In this case, 7
- The minimum possible value pair for node index 1 and 2 are (3, 6),(4, 6),(5,6), since the order matters for the pair, the number of possible pairs is 6.
- if the value pair for node index 1 and 2 are (3, 6), the possible leaves for 3 is (1,2) and possible leaves for 6 is (4,5). Therefore, the number of possible leaves is ${}_2P_2 \cdot {}_2P_2 = 4$
- if the value pair for node index 1 and 2 are (4, 6), the possible leaves for 4 are 1,2,3 and possible leaves for 6 are 1,2,3,5. Therefore, the number of possible leaves is ${}_3P_2 \cdot {}_{(4-2)}P_2 = 12$
- if the value pair for node index 1 and 2 are (5, 6), the possible leaves for 5 are 1,2,3,4 and possible leaves for 6 are 1,2,3,4. Therefore, the number of possible leaves is ${}_4P_2 \cdot {}_{(4-2)}P_2 = 24$
- The total number of possible heap order array is $2 \cdot (4 + 12 + 24) = 80$
- The probability is $\frac{80}{7!} = \frac{80}{5040} = \frac{1}{63}$

# Question 2

## Solution to (a)

Denote the number of nodes in a tree of height $H$ at level $l$ as $N_H(l)$ and $l > H \vee l < 0 \Rightarrow N_H(l) = 0$, it is also assumed that $N_H(0) = 1$
Consider a specific height $h$, the number of nodes in a tree of height $h$ at level $l$ can be calculated as follows:

$$\begin{aligned}
N_h(l) &= N_{h-1}(l) + N_{h-1}(l-1) \\
&= N_{h-2}(l) + N_{h-2}(l-1) + N_{h-2}(l-1) + N_{h-2}(l-2) \\
&= N_{h-2}(l) + 2N_{h-2}(l-1) + N_{h-2}(l-2) \\
&= N_{h-3}(l) + N_{h-3}(l-1) + 2(N_{h-3}(l-1) + N_{h-3}(l-2)) + N_{h-3}(l-2) + N_{h-3}(l-3) \\
&= N_{h-3}(l) + 3(N_{h-3}(l-1)) + 3(N_{h-3}(l-2)) + N_{h-3}(l-3) \\
&= \cdots
\end{aligned}$$

Let $k \in \mathbb{Z}^+$, Here we suppose that

$$N_{h(l)} = \sum_{i=0}^{k} \binom{h}{i} N_{h-k}(l-i)$$

Prove by induction:
Base case: $k = 1$

$$\begin{aligned}
\sum_{i=0}^{1} \binom{1}{i} N_{h-1}(l-i) &= \binom{1}{0} N_{h-1}(l) + \binom{1}{1} N_{h-1}(l-1) \\
&= N_{h-1}(l) + N_{h-1}(l-1)
\end{aligned}$$

Induction step: Assume that the formula holds for $k = m$, then

$$\begin{aligned}
\sum_{i=0}^{m} \binom{m}{i} N_{k-m}(l-i) &= \sum_{i=0}^{m} \binom{m}{i} (N_{k-m-1}(l-i) + N_{k-m-1}(l-i-1)) \\
&= \sum_{i=0}^{m} \binom{m}{i} N_{k-m-1}(l-i) + \sum_{j=1}^{m+1} \binom{m}{j-1} N_{k-m-1}(l-j) \\
&= N_{k-m-1}(l) + \sum_{i=1}^{m} \binom{m}{i} N_{k-m-1}(l-i) + \sum_{j=1}^{m+1} \binom{m}{j-1} N_{k-m-1}(l-j) + N_{k-m-1}(l-(m+1)) \\
&= N_{k-m-1}(l) + \sum_{i=1}^{m} \left( \binom{m}{i} + \binom{m}{i-1} \right) N_{k-m-1}(l-i) + N_{k-m-1}(l-(m+1)) \\
&= N_{k-m-1}(l) + \sum_{i=1}^{m} \left( \binom{m+1}{i} \right) N_{k-m-1}(l-i) + N_{k-m-1}(l-(m+1)) \\
&= \sum_{i=0}^{m+1} \binom{m+1}{i} N_{k-m-1}(l-i)
\end{aligned}$$

Therefore, the formula holds for $k = m + 1$ whenever it holds for $k = m$. Since the formula holds for $k = 1$, the formula holds for all $k \in \mathbb{Z}^+$
Note that the recusrion stops when $h - k = 0$ and by assumption,

$$l > h - k = 0 \text{ or } l < 0 \Rightarrow N_0(l) = 0$$

Therefore any i such that $l - i < 0$ or $l - i > h - k = 0$ will have $N_{h-k}(l - i) = 0$

$$\begin{aligned} N_{h(l)} &= \sum_{i=0}^{h} \binom{h}{i} N_{h-k}(l - i) \\ &= \binom{h}{l} N_{h-k}(0) + \sum^{i \neq l} 0 \text{ By assumption above} \\ &= \binom{h}{l} \end{aligned}$$

## Solution to (b)

```
import math

def merge(p, q):

    n1 = len(p)
    n2 = len(q)

    if n1 == 0:
        return q

    if n2 == 0:
        return p


    r = []

    if p[0] > q[0]:
        p, q = q, p
        n1, n2 = n2, n1

    h1 = math.floor(math.log2(n1))
    h2 = math.floor(math.log2(n2))

    max_h = max(h1, h2)

    o1 = 0
    o2 = 0
    for i in range(0, max_h+2):
        sz1 = math.comb(max_h, i) if i <= max_h else 0
        sz2 = math.comb(max_h, i-1) if i >= 1 else 0


        for j in range(0, sz2):
            if o2 + j >= n2:
                r.append(math.inf)
            else:
                r.append(q[o2 + j])

        for j in range(0, sz1):
            if o1 + j >= n1:
                r.append(math.inf)
            else:
                r.append(p[o1 + j])

        o1 += sz1
        o2 += sz2


    return r

def get_number_ranges(array):
    if not array:
        return []

    ranges = []
    start = 0
    current = array[0]
```

```
    for i, num in enumerate(array[1:], 1):
        if num != current or num == 0:
            ranges.append((current,start, i - 1))
            current = num
            start = i
    # Append the last group
    ranges.append((current, start, len(array) - 1))

    return ranges

def get_children_indices(p, index):
    # determine the level of the node i
    max_h = math.floor(math.log2(len(p)))
    start = 0
    end = 0
    last_level = [max_h]*max_h
    cumsum = 0
    for i in range(0, max_h+1):
        level_n = math.comb(max_h, i)
        cumsum += level_n
        end = start + level_n
        new_level = []
        j = 0
        for edge in last_level:
            new_edge = edge - j - 1
            if new_edge <= 0:
                j = 0
                new_level.append(0)
                continue
            new_level.extend([new_edge]*new_edge)
            j += 1

        if start <= index and index < end:
            remain = end - index - 1
            ranges = get_number_ranges(last_level)
            r = ranges[index - start]
            if r[0] == 0:
                return []
            else :
                if r[1] == r[2]:
                    return [r[1]+cumsum]
                return [r[1]+cumsum, r[2]+cumsum]

        last_level = new_level
        start = end
    pass

    return None  # Parent not found


def heapify(p, index):
    children = get_children_indices(p, index)

    # get the minimum child and its index and swap with the parent
    min_child = math.inf
    min_child_index = -1
    for i in children:
        if p[i] < min_child:
```

```
            min_child = p[i]
            min_child_index = i

    if min_child < p[index]:
        p[index], p[min_child_index] = p[min_child_index], p[index]
        heapify(p, min_child_index)


def delete_min(p):

    p[0], p[-1] = p[-1], p[0]
    p[-1] = math.inf

    heapify(p, 0)

    pass

def sift_up(p):
    max_h = math.floor(math.log2(len(p)))
    arr = []
    cumsum = 0
    for i in range(0, max_h+1):
        arr.append(cumsum)
        cumsum += math.comb(max_h, i)

    arr = arr[::-1]
    for i in range(1, len(arr)):
        last = arr[i-1]
        parent = arr[i]

        if p[last] < p[parent]:
            p[last], p[parent] = p[parent], p[last]
            heapify(p, parent)
    pass

def insert(p, x):
    p[-1] = x
    sift_up(p)

arr1 =[7,12,8,13]
arr2 =[3,5,4,9]
merged = merge(arr1, arr2)
print("Merged:", merged)

delete_min(arr1)
print("Deleted min:", arr1)


insert(arr1, 6)
print("Insert", arr1)
```

# Question 3

## Solution to (a)

index the root of up tree with $-1$ to avoid confusion if using 0-based index

$$[-1, 4, 0, 6, 3, 0, -1, 8, -1, 2]$$

## Solution to (b)

1. UNION(0, 8) compares the rank of the two trees, rank(Node 0) = 2, rank(Node 8) = 1, we attach Node 8 to Node 0

$$[-1, 4, 0, 6, 3, 0, -1, 8, 0, 2]$$

2. UNION(0, 6), rank(Node 0) = 2, rank(Node 6) = 3, we attach Node 0 to Node 6, So the final array becomes

$$[6, 4, 0, 6, 3, 0, -1, 8, 0, 2]$$

## Solution to (c)

- Node 4′s Parent: 3
- Node 3′s Parent: 6
- Node 6′s Parent: $-1$ (Root)
- Node 4: Set parent to Node 6.
- Node 3: Already points to Node 6 (no change needed).
- The final array is

$$[6, 4, 0, 6, 6, 0, -1, 8, 0, 2]$$