1. We have seen the following two algorithms to build a maximum heap from an arbitrary array of numbers.

**BUILD-HEAP**
1. for \( i \leftarrow 2 \) to \( \text{array-size} \)
2. while \( (i > 1 \text{ and } A[i] > A\left\lfloor \frac{i}{2} \right\rfloor) \)
3. exchange \( A[i] \leftrightarrow A\left\lfloor \frac{i}{2} \right\rfloor \)
4. \( i \leftarrow \left\lfloor \frac{i}{2} \right\rfloor \)

**BUILD-HEAP’**
1. for \( i \leftarrow \left\lceil \text{HEAP SIZE} \right\rceil \) downto 1
2. **HEAPIFY** \((A, i)\)

**HEAPIFY** \((A, i)\)
1. \( l \leftarrow 2i \)
2. \( r \leftarrow 2i + 1 \)
3. if \( (r \leq \text{HEAP SIZE}(A)) \)
4. then if \( (A[r] > A[l]) \)
5. then \( \text{largest} \leftarrow r \)
6. else \( \text{largest} \leftarrow l \)
7. else if \( (l \leq \text{HEAP SIZE}(A)) \)
8. then \( \text{largest} \leftarrow l \)
9. else \( \text{largest} \leftarrow i \)
10. if \( ((\text{largest} \neq i) \text{ and } (A[\text{largest}] > A[i])) \)
11. then exchange \( A[i] \leftrightarrow A[\text{largest}] \)
12. **HEAPIFY** \((A, \text{largest})\)

Show the heaps resulting from running the two algorithms **BUILDHEAP** and **BUILDHEAP’** on the same array \( A = \{1, 4, 3, 2, 5, 6, 7\} \). Are the final heaps the same or different?

2. Consider a binary minimum heap with 31 distinct elements stored in the standard array form in an array \( A[1\ldots31] \). Consider the element \( A[5] \) in the array. What are the possible legal values for its rank?

[NOTE: The rank of an element in an \( n \) element set can take any value in the range 1, \ldots, \( n \), with the minimum element having rank 1 and the maximum element having rank \( n \)]

3. Consider an array \( A \) of integers with \( n \) elements. It is given that for \( 2 \leq i \leq n \), if \( i \) is even then \( A[i] > A[k] \), for \( 1 \leq k < i \) and if \( i \) is odd, then \( A[i] < A[k] \) for \( 1 \leq k < i \). Give an optimal algorithm to sort this array \( A \) of \( n \) elements and analyse the running time of your algorithm.

[NOTE: Since this is a special type of array, the usual lower bound of \( \Theta(n \log n) \) need not apply]
4. Consider the following algorithm. You are given \( k \) arrays with \( n \) elements each.

- Build a minimum binary heap on each of the \( k \) arrays independently. Call these heaps \( H_1, \ldots, H_k \).
- Now build a minimum heap \( H_0 \) consisting of the \( k \) elements which are the minima of each of the previously build heaps. This can be done by taking a new heap and inserting the minimum elements from \( H_1, \ldots, H_k \) taken out of their respective heaps using an extract-min operation in each.
- Now repeatedly remove the minimum from the heap \( H_0 \) but instead of restoring the heap property among the remaining elements, replace the removed root element by the next smallest element from its original heap, using an extract min there. That is if the current minimum element of \( H_0 \) came from \( H_i \) where \( 1 \leq i \leq k \), then replace it with the next smallest value from \( H_i \) taken out by using extract min.
- The heap property of \( H_0 \) is restored using the heap-key-increase routine, since the key value replacing the extracted one is from the same original heap and having larger value.
- Repeat this until all the original heaps become empty. Subsequently empty the heap \( H_0 \) using repeated operations of extract-min. Write all these elements in order into an array \( A \) of length \( kn \) to get a sorted array.

Analyze the running time of this algorithm and express it as a function of \( n \) and \( k \) asymptotically.