1

a. Merge Sort

[9,7,4,1] [8,3,6,7]

[9,7][4,1] [8,3][6,7]

[9][7][4][1][8][3][6][7]

[7,9] [1,4] [3,8] [6,7]

[1,4,7,9] [3,6,7,8]

[1,3,4,6,7,7,8,9]

b. Quick Sort

[9,7,4,1,8,3,6,7] pivot = 1

[1 | 7,4,9,8,3,6,7] pivot = 8

[1,7,4,7,6,3 | 8,9] pivot = 4

[1,3,4,6,7,7,8,9]

[7,4,9,8,3,6,7]

[7,4,7,8,3,6,9]

[7,4,7,6,3,8,9]

c. Selection Sort

[9,7,4,1,8,3,6,7]

[1,7,4,9,8,3,6,7]

[1,3,4,9,8,7,6,7]

[1,3,4,6,8,7,9,7]

[1,3,4,6,7,8,9,7]

[1,3,4,6,7,7,9,8]

[1,3,4,6,7,7,8,9]

2 a {9,8,7,6,3,2,0}

1) When the pivot is the first element, it is the largest element in the subarray because the array is in reverse-sorted order which results in the most unbalanced partitions possible. The depth of this recursion becomes 'n' and at each level, we do O(n) to partition the array Overall complexity = $O(n^2)$

2) When the pivot is the median value, the array is divided into two roughly equal halves at each step of the recursion The division ensures the depth of the recursion is 'log(n)' and each level of recursion does 'O(n)' work to partition the array. Overall complexity: O(n log n)

b)

```
public static void mysterySort(int arr[], int n)
        {
        boolean sorted = false;

        while (!sorted){
            sorted = true;
            int temp = 0;

            for (int i=1; i<=n-2; i=i+2) {
                if (arr[i] > arr[i+1]) {
                    temp = arr[i];
                    arr[i] = arr[i+1];
                    arr[i+1] = temp;
                    sorted = false;
                }
            }

            for (int i=0; i<=n-2; i=i+2){
                if (arr[i] > arr[i+1]){
                    temp = arr[i];
                    arr[i] = arr[i+1];
                    arr[i+1] = temp;
                    sorted = false;
                }
            }
        }

        return;
    }
```

O(n)

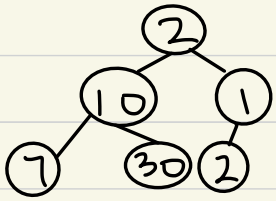
O(n/2)

O(n)

O(n/2)

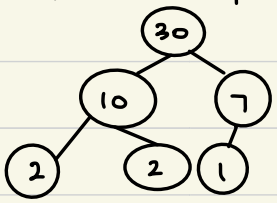
Tightest bound = $O(n^2)$ (worst case)

3 [2,10,1,7,30,2]



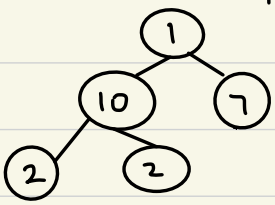
build max heap →

[30,10,7,2,2,1]



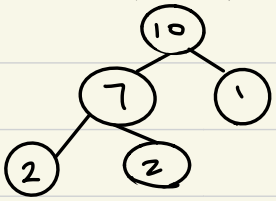
→

→ [1,10,7,2,2,30]



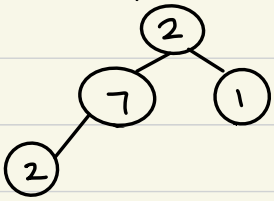
heapify →

[10,7,1,2,2,30]



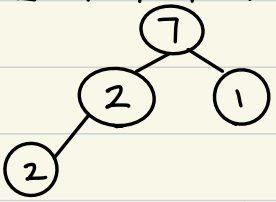
→

→ [2,7,1,2,10,30]



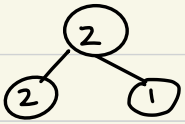
heapify →

[7,2,1,2,10,30]



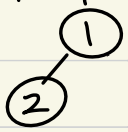
→

→ [2,2,1,7,10,30]



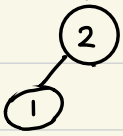
→

[1,2,2,7,10,30]



heapify →

→ [2,1,2,7,10,30]



→

[1,2,2,7,10,30]

→ 1

sorted array = {1,2,2,7,10,30]
using heapsort

4

```
public String sortByFrequencyAscending(String s)
    // create a dictionary to store the frecuency of each character
    frequencyDict = {}

    //count the frequency of each character in the string
    for each character in s
        if character in frecuencyDict
            frequencyDict[character] += 1
        else
            frequencyDict[character] = 1

    //Custom sorting function: sort by frequency, then alphabetically
    public char customSort(char1, char2)
        if frequencyDict[char1] == frequencyDict[char2]
            return char1 < char2  // sort alphabetically
        else
            return frequencyDict[char1] < frequencyDict[char2] //sort
                                                                by freq

    // Create a list of unique characters and sort it using customfucti
    uniqueChars = list of keys from frequencyDict
    sort uniqueChars using customSort

    // Build result string
    result = ""
    for each character in uniqueChars
        for i = 1 to frequencyDict[character]
            result += character
    return result
```