1.a.

```
Algorithm copyArr(arr, s, e)
   Input: original array to copy and s and e index range to be copied both inclusive
   Ouput: copies array from index s to e from original array

   length <-- e - s + 1
   newArr <- new arr[length]
   pos <-- 0
   while s <= e do
      newArr[pos] = arr[s]
      s++
      pos++
   return newArr

Algorithn insertionSort(arr,s,e)
   Input: Array to be sorted and its start and end index
   Output: Array sorted from index s to e, both index inclusive

   for i <-- s + 1 to e do
      j <-- i
      temp <-- arr[j]
      while j > s and arr[j -1] > temp do
         arr[j] <-- arr[j-1]
         j--
      arr[j] <-- temp
   return arr

Array merge(arr,s,m,e)
   Input: Original array, s , m, e define start, middle and end index for sub partition array
   Output: Array sorted from index s to e, both index inclusive

   arr1 <-- copyArr(arr,s,m)
   arr2 <-- copyArr(arr,m+1,e)

   arr1Length <-- m - s + 1
   arr2Length <-- e - m


   arr1Cursor <-- 0
   arr2Cursor <-- 0
   pos <-- s
```

```
  while arr1Cursor < arr1Length and arr2Cursor < arr2Length do
    if arr1[arr1Cursor] <= arr2[arr2Cursor] then
      arr[pos] = arr1[arr1Cursor]
      arr1Cursor++
    else
      arr[pos] = arr2[arr2Cursor]
      arr2Cursor++
    pos++

  if arr1Cursor = arr1Length then
    while arr2Cursor < arr2Length do
      arr[pos] = arr2[arr2Cursor]
      arr2Cursor++
      pos++
  if arr2Cursor = arr2Length then
    while arr1Cursor < arr1Length do
      arr[pos] = arr2[arr1Cursor]
      arr1Cursor++
      pos++

  return arr


Algorithm mergeSortPlus(arr,s,e)
  Input: Array to be sorted and its start and end index
  Output: Sorted Array

  if(s=e) then return arr
  m <-- floor((s+e)/2)
  mergeSort(arr,s,m)
  mergeSort(arr,m+1,e)
  merge(arr,s,m,e)
  return arr
```

1.b.

```
public class MergeSortHybrid {
  public static int[] getArrayCopy(int[] arr,int s, int e){
    int length = e - s + 1;
    int[] newArray = new int[length];
    System.arraycopy(arr,s,newArray,0,length);
```

```
        return newArray;
    }

    public static int[] insertionSort(int[] arr,int s, int e){
//        System.out.println("insertion sort triggerred");
        for(int i=s+1 ; i<=e ; i++){
            int j = i;
            int temp = arr[j];
            while(j > s && arr[j-1] > temp){
                arr[j] = arr[j-1];
                j--;
            }
            arr[j] = temp;
        }
        return arr;
    }

    public static int[] merge(int[] arr, int s, int m, int e){

        int arrayALength = m - s + 1;
        int arrayBLength = e - m;

        int[] arrA = getArrayCopy(arr,s,m);
        int[] arrB = getArrayCopy(arr,m+1,e);
        int cursorA = 0;
        int cursorB = 0;

        int pos = s;

        while(cursorA < arrayALength && cursorB < arrayBLength){
            if(arrA[cursorA] > arrB[cursorB]){
                arr[pos] = arrB[cursorB];
                cursorB++;
            }else{
                arr[pos] = arrA[cursorA];
                cursorA++;
            }
            pos++;
        }

        if(cursorA == arrayALength){
            System.arraycopy(arrB,cursorB,arr, pos, arrayBLength - cursorB );
        }
        if(cursorB == arrayBLength){
```

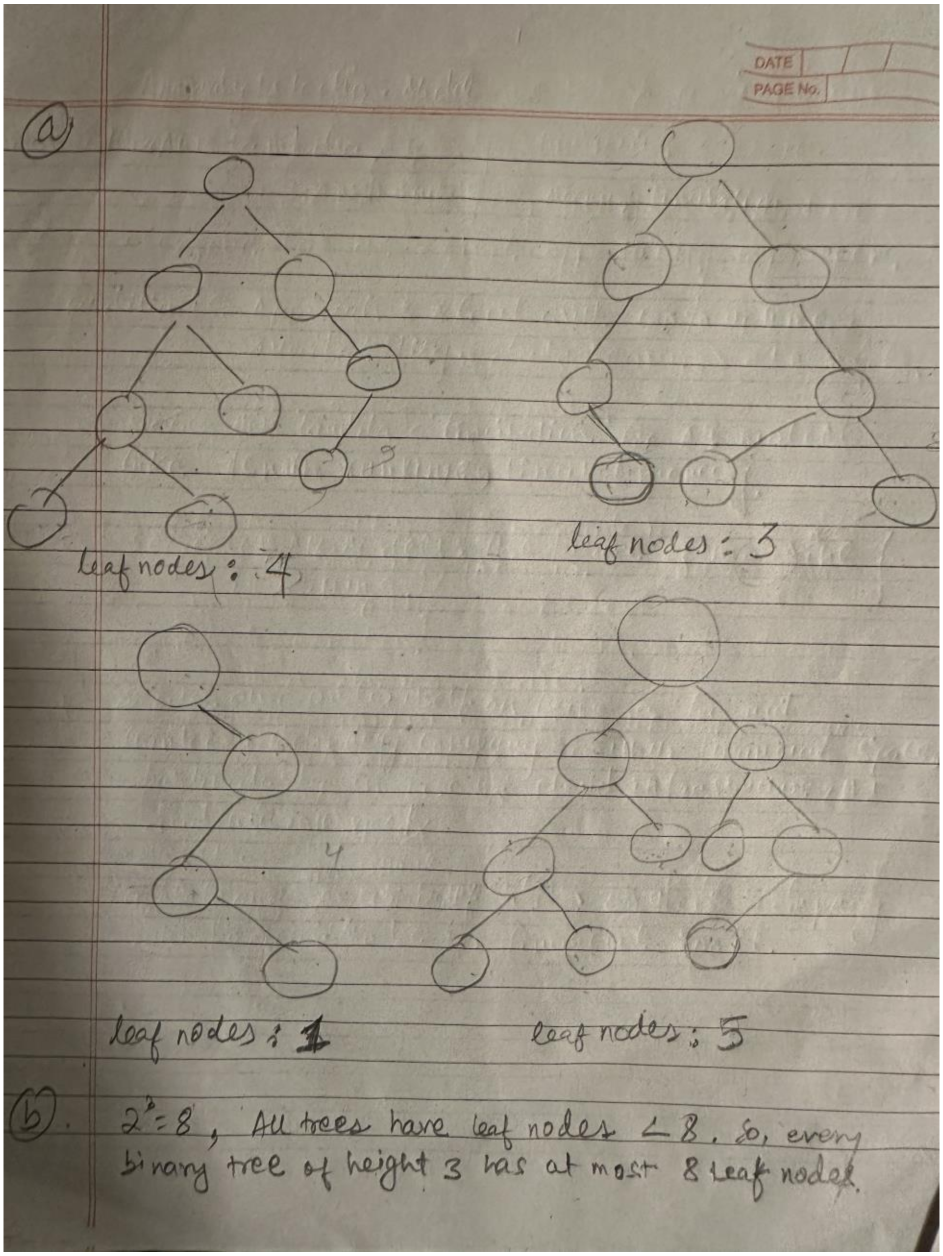
```
        System.arraycopy(arrA,cursorA,arr, pos, arrayALength - cursorA );
      }

      return arr;
   }

   public static int[] mergeSortPlus(int[] arr,int s,int e){
      if(s==e) return arr;
      if(e-s+1 <= 20) return insertionSort(arr,s,e);
      int m = (s+e)/2;
      mergeSortPlus(arr,s,m);
      mergeSortPlus(arr,m+1,e);
      merge(arr,s,m,e);
      return arr;
   }


   public static int[] mergeSort(int[] arr,int s,int e){
      if(s==e) return arr;
      int m = (s+e)/2;
      mergeSort(arr,s,m);
      mergeSort(arr,m+1,e);
      merge(arr,s,m,e);
      return arr;
   }


   public static void main(String[] args){
     int[] arr = new
int[]{77,78,99,1001,2003,45,67,89,0,1,45,33,56,78,99,10001,12345,67,32,22,11,76,89,9
6,108,120,2000,2001,2020,0,45,66,78,56,23,111,121,144,256,7,51};
     System.out.println(Arrays.toString(insertionSort(arr,0,arr.length-1)));
     System.out.println(Arrays.toString(mergeSort(arr,0,arr.length-1)));
     System.out.println(Arrays.toString(mergeSortPlus(arr,0,arr.length-1)));
   }
}
```

1.c.

The MergeSortPlus runs faster.

I tested using 500 arrays inputs of size 10000. I believe the results are very conclusive because as we know the time complexity of insertion sort is $O(n^2)$, due to which it is very slow for large datasets. But in case of merge sort the time complexity is $O(n*logn)$, which means it really shines when there is a huge dataset. But the performance difference is not seen as much in small datasets. So combining both of these can serve as a powerful sorting algorithm that provides the best of both worlds.

2.



2.c. By observation we can say that a binary tree of height "n" will have at most "2^n" leaf nodes.

3.

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class PowerSet {
   public static <T> List<Set<T>> powerSet(List<T> x){
      List<Set<T>> result = new ArrayList<>();
      Set<T> temp = new HashSet<>();
      result.add(temp); // adding empty set

      while(!x.isEmpty()){
         T removedItem = x.removeFirst();
         for(Set<T> set : new ArrayList<>(result)){
            result.add(new HashSet<>(){{
               addAll(set);
               add(removedItem);
            }});
         }
      }

      return result;
   }

   public static void main(String[] args){
      System.out.println(powerSet(new ArrayList<>(){{
         add(1);
         add(3);
         add(2);
      }}));
   }
}
```

4.

```
public class PowerSet {
   public static <T> List<Set<T>> powerSet(List<T> x){
      List<Set<T>> result = new ArrayList<>();
      Set<T> temp = new HashSet<>();
```

```
        result.add(temp); // adding empty set

        while(!x.isEmpty()){
            T removedItem = x.removeFirst();
            for(Set<T> set : new ArrayList<>(result)){
                result.add(new HashSet<>(){{
                    addAll(set);
                    add(removedItem);
                }});
            }
        }

        return result;
    }

    public static Set<Integer> subsetWithSum(List<Integer> list, int k){
        if(k==0) return new HashSet<>();
        for(Set<Integer> lst : powerSet(list)){
            Optional<Integer> total = lst.stream().reduce(Integer::sum);
            if(total.isPresent() && total.get() == k){
                return lst;
            }
        }
        return null;
    }
}
```