1. Consider the following recurrence (for mergesort) for the case when $n$ is a power of 2.

$$T(n) = \begin{cases} 2T(n/2) + n, & n > 2 \\ 1, & n = 2. \end{cases}$$

Prove by induction on $n$ that the solution is as follows, and determine the constant $A$.

$$T(n) \leq An \log n, \quad n \geq 2$$
2. The following algorithm uses a divide-and-conquer strategy to find the maximum and minimum elements in an array of \( n \) elements. The initial call is find(A,0,n-1,MIN,MAX).

```c
void find(dtype a[], int i, int j, dtype& min, dtype& max)
{dtype min1, max1, min2, max2; //compute min and max of a[i..j]
  if (i==j) min=max=a[i]; //The case of n=1
  else if (i==j-1){ //The case of n=2
    if (a[i]<a[i+1]) {min=a[i]; max=a[i+1]}; else {min=a[i+1]; max=a[i]};
  else{ //The case of n>2:
    int m=(i+j)/2;
    find(a,i,m, min1,max1);
    find(a,m+1,j, min2,max2);
    if (min1<min2) min=min1; else min=min2;
    if (max1<max2) max=max2; else max=max1;
  }
}
```

(a) Let \( f(n) \) be the worst-case number of key comparisons used by this algorithm for an array of size \( n \). (A key comparison involves two elements of the array. They are the comparisons in lines 3,8,9 marked by asterisk.) Write a recurrence for \( f(n) \), assuming \( n \) is a power of 2. Solve by repeated substitutions.

(b) Write a recurrence for \( f(n) \) for the general case where \( n \) is any integer, \( n \geq 1 \). (Don’t solve it.)
3. Consider the following version of the bubble-sort algorithm. The algorithm first bubbles up the smallest element to end up in $a[0]$, then finds the smallest of the remaining $n - 1$ elements to end up in $a[1]$, and so on.

```c
void bubble (dtype a[], int n)
{
  int i,j;
  for (i=0; i <= n-2; i++)
    //Find the smallest of a[i..n-1] and bubble up to position a[i].
    for (j=n-1; j > i; j--)
      if (a[j]<a[j-1]) SWAP(a[j],a[j-1]);
}
```

(a) Analyze the worst-case time complexity of this algorithm, $T(n)$.

(b) Rewrite this version of bubble-sort algorithm using recursion. (Note that your recursive algorithm must make exactly the same sequence of compare/swap operations as the above algorithm.)
4. Recall the HEAP algorithm PUSHDOWN(A, r, n), where r is index of the root and n is the number of nodes. This algorithm assumes the left and right subtrees of r already satisfy the heap property, and pushes the element at the root down as far as it needs to go.

(a) Write the pseudocode for a recursive PUSHDOWN(A, r, n).

(b) What is the time complexity of this algorithm? Explain.
5. (a) Starting with an empty Binary Search Tree (BST), insert the following sequence of elements. Show the final tree.

50, 35, 75, 20, 80, 60, 40, 10, 55, 65, 70

(b) A binary tree is said to be height balanced if the heights the left and right subtrees of every node differ by at most one. Is the final tree height balanced? If not, explain which node is unbalanced.