Problem 1 (8 points). Implement the binary search algorithm and analyze binary search algorithm using one of the methods (substitution method, iteration method, recursion-tree method, or master method) (starter code **binarySearch.cpp** is given).

```
int binarySearch(int* A, int p, int r, int key) {
   //p = low r = high
   if(r >= p)
   {

       int mid = r+ (p-1)/2;
       if(A[mid] == key)
       {
           return mid;
       }
       else if(A[mid] > key)
       {
           return binarySearch(A, p, mid-1, key);
       }
       else if(A[mid] < key)
       {
           return binarySearch(A, mid+1, p, key);
       }
   }

   return -1;
```

Binary search recurrence relation: $T(n) = T(\frac{n}{2}) + 1$

Using the master method

$T(n) = aT(\frac{n}{b}) + f(n)$ $\quad a=1 \quad b=2 \quad f(n) = c$, a constant

$\therefore \log_b a = \log_2 1 = 0$

$n^0 = 1 \quad (1)(\log_2 n) = \log_2(n) \quad f(n) = c = \Theta(n^{\log_b a})$.

$T(n) = \Theta(n^{\log_b a} \log_2 n) = \boxed{\Theta(\log_2 n)}$

Problem 2 (8 points). Implement the merge sort (starter code **mergeSort.cpp** is given).

```
void merge(int* A, int p, int q, int r) {
   //p = left q = mid r = right
   int leftArrSize = q - p + 1;
   int rightArrSize = r-q;

   int *leftArr = new int[leftArrSize];
   int *rightArr = new int[rightArrSize];
   for(int i = 0; i < leftArrSize; i++)
   {
       leftArr[i] = A[p + i];
   }
   for(int i = 0; i < rightArrSize; i++)
   {
       rightArr[i] = A[q + 1 + i];
   }

   int indexLeftArr = 0;
   int indexRightArr = 0;
   int indexMergeArr = p;

   while(indexLeftArr < leftArrSize && indexRightArr < rightArrSize)
   {
       if(leftArr[indexLeftArr] >= rightArr[indexRightArr])
       {
           A[indexMergeArr] = leftArr[indexLeftArr];
           indexLeftArr++;
       }
       else
       {
           A[indexMergeArr] = rightArr[indexRightArr];
           indexRightArr++;
       }

       indexMergeArr++;
   }
   while(indexLeftArr < leftArrSize)
   {
       A[indexMergeArr] = leftArr[indexLeftArr];
       indexLeftArr++;
```

```
        indexMergeArr++;
    }
    while(indexRightArr < rightArrSize)
    {
        A[indexMergeArr] = rightArr[indexRightArr];
        indexRightArr++;
        indexMergeArr++;
    }
    delete[] leftArr;
    delete[] rightArr;
}


void mergeSort(int* A, int p, int r) {
    if(p < r)
    {
        int mid = + ((p + r) / 2);

        mergeSort(A, p, mid);
        mergeSort(A, mid+1, r);
        merge(A, p, mid, r);
    }


}
```