1. (25 points) Suppose the array A contains [8, 3, 4, 9, 1]. (a) Please display the contents of the array before and after each call to merge and count the number of comparisons used. (b) Please do the same for the bottom-up mergesort.

Answer: (a) mergesort([8, 3, 4, 9, 1]) = merge(mergesort([8, 3]), mergesort([4, 9, 1])) = merge([3, 8], [1, 4, 9]) = [1, 3, 4, 8, 9] using 4 comparisons; mergesort([8, 3]) = merge([8], [3]) = [3, 8] using 1 comparison; mergesort([4, 9, 1]) = merge([4], mergesort([9, 1])) = merge([4], [1, 9]) = [1, 4, 9] using 2 comparisons; mergesort([9, 1]) = merge([9], [1]) = [1, 9] using 1 comparison;
The total number of comparisons is 8. (A graphical illustration is more intuitive.)
(b) After two ones merge into a two: [8, 3, 4, 9, 1] => [3, 8, 4, 9, 1] (2 comparisons) After two twos merge into a four: [3, 8, 4, 9, 1] => [3, 4, 8, 9, 1] (3 comparisons) The final merge of [3, 4, 8, 9] and [1] produces [1, 3, 4, 8, 9] (1 comparison). The total number of comparisons is 6.

2. (25 points) The function F(n) will compute the n\textsuperscript{th} Fibonacci number for n >= 0:

```
int F(int n) { if (n == 0 || n == 1) return 1; else return F(n-1)+F(n-2); }
```

Please define the number of “+” operations used to compute F(n) as a function A(n) and prove formally that A(n) = F(n) – 1 for n >= 0.

Answer: A(n) can be recursively defined as:

\[ A(0) = A(1) = 0 \] (because F(0) and F(1) doesn’t use “+”).
\[ A(n) = A(n-1)+A(n-2) + 1 \] for n > 1.
To show A(n) = F(n) – 1, we use mathematical induction.
Base cases: n = 0 or 1. A(0) = A(1) = 0 = F(0) – 1 = F(1) – 1.
Induction hypotheses: A(n−1) = F(n−1) – 1 and A(n−2) = F(n−2) – 1 for n > 1.
Inductive case: A(n) = A(n−1) + A(n−2) + 1 \quad // definition of A(n)
\quad = F(n−1) − 1 + F(n−2) – 1 + 1 \quad // induction hypotheses.
\quad = F(n−1) + F(n−2) − 1
\quad = F(n) – 1 \quad // definition of F(n)
3. (25 points) A compact data structure for a polynomial \( f(x) \), where the number of non-zero coefficients is \( k \), is to use two arrays: \( C[0..(k-1)] \) stores non-zero coefficients and \( D[0..(k-1)] \) stores the corresponding degrees of these non-zero coefficients. For example, if \( f(x) = 5x^{100} + 3x^{20} - 2x^4 \), then \( k = 3 \), \( C = [5, 3, -2] \) and \( D = [100, 20, 4] \). Given \( C[] \) and \( D[] \) for any \( k \), design an efficient algorithm 

\[
\text{double eval(double e, int k, double[]} C, \text{int[]} D),
\]

to compute \( f(e) \) for any given number \( e \), such that the number of multiplications is minimized. Please analyze the complexity of your algorithm.

Answer: We will use fastpower(x, n) in eval:

\[
\text{double eval(double e, int k, double[]} C, \text{int[]} D) \{
\text{double s = 0; D[k] = 0; for (int i = 0; i<k; i++) s = (s + C[i])*fastpower(e, D[i] - D[i+1]); return s;}
\}
\]

\[
\text{double fastpower(double e, int n) \{ if (n == 0) return 1.0; if (n == 1) return e; double x = fastpower(e, (n>>1)); if ((n & 1) == 0) return x*x; return x*x*e;}
\}
\]

The number of multiplications is \( O(k + \sum_{i=0}^{k-1} (\log_2(D[i] - D[i+1]))) \).

For \( f(x) = 5x^{100} + 3x^{20} - 2x^4 \), the number of multiplications is \( 3+7+4+2 = 16 \).

4. (25 points) We like to sort \( n \) student records by their grades for a course. If the grades are A, B or C, what is the most efficient sorting algorithm for this job? If the students are stored in an array, what is the most efficient in-place sorting algorithm for this job? Please provide the algorithms in details.

Answer: Suppose the students records are stored in the array \( S[0...n-1] \). The solution below takes \( O(n) \) time and works for both cases. \( S[0..a] \) are ‘A’; \( S[c..n-1] \) are ‘C’.

\[
\text{Sortbyscore(S, n) \{ int a = -1, b=0, c = n; while (b<c) switch(score(S[b])) \{ case ‘A’: a++; swap(S, a, b); b++; break; case ‘B’: b++; break; case ‘C’: c--; swap(S, b, c); break; \}}}
\]