1. (2 points each) Determine the worst-case running times for the following functions in terms of \( n \). Specify the times as functions of \( n \), and then simplify them into a tight big-O notation. In addition, specify in three words or less what each function does (e.g., "binary search", "square root").

   a. 
   ```
   float alpha(vector X, vector Y) // Assume that the "vector" type has been defined to be a one-dimensional array
   {
     float p = 0.0F; // of floats, with a "length" data member
     int n = X.length; // reflecting the array size.
     if (n != Y.length)
       return 0.0F;
     else
       {
         for (int i = 0; i < n; i++)
           p += X[i]*Y[i];
       }
   return p;
   }
   ```

   b. 
   ```
   bool beta(int n)
   {
     int max = n/2;
     for (int i = 2; i < max; i++)
     {
       if ((n/i)*i == n)
         return false;
     }
     return true;
   }
   ```

   c. 
   ```
   void gamma(int list[], int n)
   {
     int count;
     int copy[MAX_LIST_SIZE]; // Assume that MAX_LIST_SIZE is a positive integer constant declared elsewhere.
     for (int i = 0; i < n; i++) // integer constant declared elsewhere.
     copy[i] = list[i];
     for (int j = 0; j < n; j++)
     {
       count = 0;
       for (k = 0; k < n; k++)
       {
         if (copy[k] < copy[j])
           count++;
       }
       list[count] = copy[j];
     }
   }
   ```

   d. 
   ```
   int delta(int n)
   {
     if (n <= 1)
       return 0;
     else
       return 1 + delta(n/2);
   }
   ```

   e. 
   ```
   int epsilon(int n)
   {
     if (n < 0)
       return -1*epsilon(-1*n);
     else if (n == 0)
       return 0;
     else
       return 1 + 3*n*n - 3*n + epsilon(n-1);
   }
   ```
2. Given the following C++ function, which determines the median value in a floating-point array of size $n$:

```c
float median(float A[], int n)
{
    int lessCount, equalCount;
    float middle = 0.0F;
    for (int i = 0; i < n; i++)
    {
        lessCount = 0;
        equalCount = 0;
        for (int j = 0; j < n; j++)
        {
            if (A[j] < A[i])
                lessCount++;
            else if (A[j] == A[i])
                equalCount++;
        }
        if ((lessCount <= n/2) && (n/2 <= lessCount+equalCount))
        {
            middle = A[i];
            break;
        }
    }
    return middle;
}
```

a. (3 points) Determine the big-O time complexity of this function. **Show your work.**

b. (2 points) **Outline** an algorithm that accomplishes the same result as this with $O(n \log n)$ time complexity.

---

You must provide your own solutions to these problems in a clearly presented Word document. Obtaining solutions from any outside source is considered academic misconduct. The only person with whom you may discuss these problems is the instructor.