1)

a) $\log_2 n^2 + 1 = 2\log_2 n + 1$

$2\log_2 n + 1 \le cn$ for $c=2$, $n>0$

$\log_2 n^2 + 1 = O(n)$

b) $n^2+n \geq cn^2$ for $c=1$, $n>0$

So, $\sqrt{n(n+1)} \geq cn \rightarrow \sqrt{n(n+1)} = \Omega(n)$

c) $n^{n-1} \le cn^n$ for $c=1$, $n>1$

So, $n^{n-1} = O(n^n)$

$n^{n-1} \geq cn^{n-2}$ for $c=1$, $n>1$

So, $n^{n-1} = \Omega(n^n)$

$n^{n-1} = O(n^n) = \Omega(n^n) = \Theta(n^n)$

2) $8^{\log_2 n} = n^3$

$\lim\limits_{n\to\infty} \frac{10^n}{2^n} = \infty$, so $10^n > 2^n$

$\lim\limits_{n\to\infty} \frac{2^n}{n^3} = \lim\limits_{n\to\infty} \frac{2.2^{n-1}}{3n^2} = \lim\limits_{n\to\infty} \frac{4.2^{n-2}}{6n} = \lim\limits_{n\to\infty} \frac{8\,2^{n-3}}{6} = \infty$, so $2^n > n^3$

$\lim\limits_{n\to\infty} \frac{n^3}{n^2\log n} = \lim\limits_{n\to\infty} \frac{n}{\log n} = \lim\limits_{n\to\infty} \frac{1}{\frac{1}{\ln(10).n}} = \lim\limits_{n\to\infty} \ln(10).n = \infty$, so $n^3 > n^2\log n$

$\lim\limits_{n\to\infty} \frac{n^2\log n}{n^2} = \lim\limits_{n\to\infty} \log n = \infty$, so $n^2\log n > n^2$

$\lim\limits_{n\to\infty} \frac{n^2}{\sqrt{n}} = \infty$, so $n^2 > \sqrt{n}$

$\lim\limits_{n\to\infty} \frac{\sqrt{n}}{\log n} = \lim\limits_{n\to\infty} \frac{\frac{1}{2\sqrt{n}}}{\frac{1}{n\ln(10)}} = \frac{n\ln(10)}{2\sqrt{n}} = \infty$, so $\sqrt{n} > \log n$

3)

a) The values of i is: 2, 3, 7, 43, 1807. So the time complexity of the function is $O(\log_2^n)$. The growth rate of $\log_2^n$ is always bigger than the growth rate of the function.

b) There is no best-worst case for this function because if-else block does not terminate the function earlier or later than normal. The time complexity of this function depends on only sizeofArray. So, the asymptotic notation is $\Theta(n)$. Θ notation is used because both the upper and lower limit is linear.

c) The time complexity of this function does not depends on anything. So, the asymptotic notation of this function is $\Theta(1)$. Θ notation is used because both the upper and lower limit is constant.

d) The time complexity of this function depends on n. So, the asymptotic notation of this function is $\Theta(n)$. Θ is used because both limit linear.

e) The asymptotic notation of inner loop $\Theta(1)$, for best case, $\Theta(\log_2^n)$ for worst case. So, the time complexity of inner loop is $O(\log_2^n)$. O notation is used because the time complexity can be constant or linear. The asymptotic notation of outer loop is $\Theta(n)$. Because both limit of outer loop is linear. So, the asymptotic notation of the function is $O(n\log_2^n)$.

f) The asymptotic notation of this function is $\Theta(1)$ for best case, $O(n\log_2^n)$ for worst case. So, the time complexity of this function $O(n\log_2^n)$ in general.

CS CamScanner

g) The asymptotic notation of inner loop is $\Theta(n)$ because its time complexity is linear. The asymptotic notation of outer loop is $\Theta(\log_2^n)$. Because its time complexity is logarithmic.

h) The asymptotic notation of this function is same as the function above.

i) The function calls itself n times. So the time complexity of this function is n. Asymptotic notation is $\Theta(n)$.

J) The function calls itself n times. Every call while loop will run. The time complexity of the best case of the while loop is constant time, worst case is linear. So the asymptotic notation of while loop of best, worst and general case is: $\Theta(1)$, $\Theta(n)$, $O(n)$. So the time complexity of the function is quadratic. So, asymptotic notation is $O(n^2)$.

4)

a) The asymptotic running time of algorithm A is at most $O(n^2)$.

b) I) $2^{n+1} \leq c2^n$ this condition is met for $c > 2$ for $n > 0$. So, $2^{n+1} = O(2^n)$. $2^{n+1} \geq c2^n$ this condition is met for $c < \frac{1}{2}$ for $n > 0$. So, $2^{n+1} = \Omega(2^n)$. Both omega and big O notations are met so $2^{n+1} = \Theta(2^n)$

II) $2^{2n} \leq c2^n$ there is no constant c which is met this condition. So, $2^{2n} \neq O(2^n)$. So big O notation is not met. For this reason, $2^{2n} \neq \Theta(2^n)$

III) Let $F(n) = 2n$, so asymptotic running time of $F(n)$ at most $O(n^2)$ at least $\Omega(n)$. Let $g(n) = n^2$. $2n \cdot n^2 = 2n^3$. The asymptotic running time of $2n^3$ cannot be $\Theta(n^4)$ because the lower limit of $2n^3$ is $\Omega(n^3)$.

CamScanner

5) $T(n) = 2T(n/2) + n, \; T(1) = 1$

$$T(n) = 4T(n/4) + 3n \qquad T(n/2) = 2T(n/4) + n$$
$$T(n) = 8T(n/8) + 7n \qquad T(n/4) = 2T(n/8) + n$$
$$T(n) = 16T(n/16) + 15n \qquad T(n/8) = 2T(n/16) + n$$

k times

$$T(n) = 2^k T(n/2^k) + (2^k - 1)n$$

$$T(n/2^k) = T(1)$$
$$n = 2^k \rightarrow \log_2^n = k$$
$$T(n) = nT(1) + (\log_2^n - 1)n$$
$$= n(\log_2^n \cancel{-1} \cancel{+1}) = n\log_2^n$$
$$n\log_2^n = \Theta(n\log_2^n)$$

b) $T(n) = 2T(n-1) + 1, \; T(0) = 0$

$$T(n-1) = 2T(n-2) + 1$$
$$T(n-2) = 2T(n-3) + 1$$
$$T(n-3) = 2T(n-4) + 1$$

k times

$$T(n) = 4T(n-2) + 3$$
$$T(n) = 8T(n-3) + 7$$
$$T(n) = 2^k T(n-k) + (2^k - 1)$$

$$T(n-k) = T(0)$$
$$n = k$$
$$T(n) = \cancel{2^n T(0)} + 2^n - 1$$
$$= 2^n - 1 = \Theta(2^n)$$

6) The time complexity of the function is $\Theta(n^2)$. While outer loop runs n times, inner loops runs on avarage $\frac{n}{2}$ times. So $T(n) = \frac{n^2}{2} = \Theta(n^2)$

7) $T(n) = T(n-1) + G(n) + 1, \; T(0) = 1, \; G(0) = 1$

$$G(n) = G(n-1) + 1$$
$$\vdots$$
$$G(n) = G(n-k) + k$$
$$G(n-k) = G(0) \rightarrow n = k \rightarrow G(n) = n$$

$$T(n-1) = T(n-2) + n - 1 + 1$$
$$T(n-2) = T(n-3) + n - 2 + 1$$

$$T(n) = T(n-1) + n + 1$$
$$T(n) = T(n-2) + n + n + 1 = T(n-2) + 2n + 1$$
$$T(n) = T(n-3) + n - 1 + 2n + 1 = T(n-3) + 3n$$
$$T(n) = T(n-k) + kn$$
$$T(0) = T(n-k) \rightarrow n = k$$
$$T(n) = 1 + n^2 = \Theta(n^2)$$

3) What is the time complexity of the following programs? Use most appropriate asymptotic notation. Explain by giving details.

a)

```
int p_1 ( int my_array[]){
        for(int i=2; i<=n; i++){  Loop runs at most logn times. So, O(logn)
                if(i%2==0){ Q(1)
                        count++; Q(1)
                } else{ Q(1)
                        i=(i-1)i; Q(1)
                }
        }
}
```

b)

```
int p_2 (int my_array[]){
        first_element = my_array[0]; Q(1)
        second_element = my_array[0]; Q(1)
        for(int i=0; i<sizeofArray; i++){ Loop runs sizeofArray times. So, Q(sizeofArray)
                if(my_array[i]<first_element){  Does not terminate function earlier or later than normal.
                        second_element=first_element; Q(1)
                        first_element=my_array[i]; Q(1)
                }else if(my_array[i]<second_element){ Does not terminate function earlier or later than normal.
                        if(my_array[i]!= first_element){ Q(1)
                                second_element= my_array[i]; Q(1)
                        }
                }
}
```

CS CamScanner

c)

```
int p_3 (int array[]) {
        return array[0] * array[2]; Q(1)
}
```

d)

```
int p_4(int array[], int n) {
        Int sum = 0 Q(1)
        for (int i = 0; i < n; i=i+5) Loop runs n times. So, Q(n)
                sum += array[i] * array[i]; Q(1)
        return sum; Q(1)
}
```

e)

```
void p_5 (int array[], int n){
        for (int i = 0; i < n; i++) Loop runs n times. So, Q(n)
                for (int j = 1; j < i; j=j*2) Loops runs for best case, logn times for worst case. So, O(logn)
                        printf("%d", array[i] * array[j]); Q(1)
}
```

f)

```
int p_6(int array[], int n) {
        If (p_4(array, n)) > 1000) Worst case condition.
                p_5(array, n) O(nlogn)
        else printf("%d", p_3(array) * p_4(array, n)) Best case condition. Q(1)
}
```

g)

```
int  p_7( int n ){
        int i = n;  Q(1)
        while (i > 0) { Loop runs logn times. Q(logn)
                for (int j = 0; j < n; j++) Loop runs n times. Q(n)
                        System.out.println("*"); Q(1)
                i = i / 2; Q(1)
        }
}
```

h)

```
int  p_8( int n ){
        while (n > 0) { Runs logn times. Q(logn)
                for (int j = 0; j < n; j++) Runs on avarage n/2. Q(n)
                        System.out.println("*"); Q(1)
                n = n / 2; Q(1)
        }
}
```

CamScanner

i)

```
int p_9(n){
        if (n = 0) Q(1)
                return 1 Q(1)
        else Q(1)
                return n * p_9(n-1) Runs t times. Q(n)
}
```

j)

```
int p_10 (int A[ ], int n) {
        if (n == 1) Q(1)
                return; Q(1)
        p_10 (A, n - 1); Runs n times. Q(n)
        j = n - 1;
        while (j > 0 and A[j] < A[j - 1]) {  Runs 1 times for best case: Q(1), runs n-1 times for worst case: Q(n)
                SWAP(A[j], A[j - 1]); I assumed it as 1 operation.
                j = j - 1; Q(1)
        }
}
```

CS CamScanner

```java
package cayci_hw2;

public class test {
    public static void iterativeSearch(int[] numbers, int sum){
        for(int i = 0; i < numbers.length-1; i++){
            for(int j = i+1; j < numbers.length; j++){
                if(numbers[i] + numbers[j] == sum){
                    System.out.println(numbers[i] + ", " + numbers[j]);
                }
            }
        }
    }
    public static void recursiveSearch(int[] numbers, int sum, int first, int next){
        if(numbers[first] + numbers[next] == sum){
            System.out.println(numbers[first] + ", " + numbers[next]);
        }
        if(first == numbers.length - 2)
            return;
        if(next < numbers.length - 1){
            recursiveSearch(numbers, sum, first, next+1);
            return;
        }
        recursiveSearch(numbers, sum, first+1, first+2);
    }
    public static void main(String[] args) throws DoesNotContainException{
        int[] num = new int[125];
        int[] num2 = new int[25];
        int[] num3 = new int[5];
        int sum = 1;
        long start;
        start = System.nanoTime();
        recursiveSearch(num3, sum, 0, 1);
        System.out.println("Recursive with 5 inputs: " + (System.nanoTime() - start));
        start = System.nanoTime();
        recursiveSearch(num2, sum, 0, 1);
        System.out.println("Recursive with 25 inputs: " + (System.nanoTime() - start));
        start = System.nanoTime();
        recursiveSearch(num, sum, 0, 1);
        System.out.println("Recursive with 125 inputs: " + (System.nanoTime() - start));
        start = System.nanoTime();
        iterativeSearch(num3, sum);
        System.out.println("Iterative with 5 inputs: " + (System.nanoTime() - start));
        start = System.nanoTime();
        iterativeSearch(num2, sum);
        System.out.println("Iterative with 25 inputs: " + (System.nanoTime() - start));
        start = System.nanoTime();
        iterativeSearch(num, sum);
        System.out.println("Iterative with 125 inputs: " + (System.nanoTime() - start));
    }
}
```

CamScanner

```
Recursive with 5 inputs: 3900
Recursive with 25 inputs: 28500
Recursive with 125 inputs: 925200
Iterative with 5 inputs: 2900
Iterative with 25 inputs: 5300
Iterative with 125 inputs: 160100
```

CamScanner