Part1.

1. Searching a product.

```
public void searchProduct(String productName){
    for(int i=0; i<getNumberOfBranches(); i++){
        for(int j=0; j<branches.at(i).getProducts().size(); j++){
            if(branches.at(i).getProducts().at(j).getName().equals(productName)){
                System.out.println(branches.at(i));
                System.out.println(branches.at(i).getProducts().at(j));
            }
        }
    }
}
```

Searching for a product varies depending on how many branches are affiliated with the company and the number of products within these

branches, the time complexity varies.

So the T(n) = $\Theta(n^2)$

2. Add/Remove Product

```
public void addProduct(String name,String model,String color,int stockNumber,int productID, int branchID){
    Product product = getProduct(productID,branchID);
    if(product != null && product.getProductID() == productID) product.setNumber(getProduct(productID,branchID).getNumber()+stockNumber);
    else getBranch(branchID).getProducts().insert(new Product(name,model,color,stockNumber,productID));
}
```

If a product is to be added, it is first questioned whether it is available in the store ($\Theta(n^2)$) to be added from that product. if any, only the number varies ($\Theta(1)$). If the product is not available in that store, the product is added to the store ($\Theta(n)$)

So T(n) = $\Theta(n^2) + \Theta(n) + \Theta(n^1) = \Theta(n^2)$

```
public void deleteProduct(int productID,int number,int employeeID,int branchID) throws Exception{
    if(getProduct(branchID,productID).getNumber() >= number){
        getProduct(branchID,productID).setNumber(getProduct(branchID,productID).getNumber()-number);
        getEmployee(employeeID).addMessage(String.format("%d product deleted from product number %d(%s Branch)",number,productID,getBranch(branchID).getBranchName()));
    }
    else{
        throw new Exception("\nThere is no such Product!\n");
    }

}
```

If a product is to be deleted from the store, it is first checked to see if that product is available in the store $\Theta(n^2)$. if not, this is the best possibility and the exception is thrown $\Theta(1)$. If the product is in the store, the product is deleted for the requested number of times. $\Theta(n^2)$

So T(n) = $O(n^2)$

3. Querying the products that need to be supplied.

```
public void showMessages(){
    System.out.println("\n\t\tMessages to Administrator\n");

    for(int i=0; i<SystemMessages.size(); i++){
        System.out.println("-> "+SystemMessages.at(i));
    }
}
```

The number of products to be supplied depends on the number of messages entered into the system.

So T(n) = Θ(n)

Part 2

a. Explain why it is meaningless to say: "The running time of algorithm A is at least $O(n^2)$".

Big-O notation is the worst case scenario, meaning that A algorithm's runtime will be n^2 or faster. If the runtime is at least $O(n^2)$ this means that the fastest runtime of algorithm A can have at least $O(n^2)$, means that can be anything slower than $O(n^2)$. So 'at least" is redundant at this sentence.

b. Let $f(n)$ and $g(n)$ be non-decreasing and non-negative functions. Prove or disprove that: $\max(f(n), g(n)) = \Theta(f(n)+g(n))$

$$c_1(f(n)+g(n)) \leq \max(f(n), g(n)) \leq c_2(f(n)+g(n))$$

$$\max(f(n), g(n)) \leq 1(f(n)+g(n))$$

$$\max(f(n), g(n)) \geq \frac{1}{2}(f(n)+g(n))$$

true for $c_1 \leq \frac{1}{2}$ and $c_2 \geq 1$

C.

I. $2^{n+1} = \Theta(2^n)$

$f(n) = 2^{n+1}$

$g(n) = 2^n$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \in R$ then $f(n) = \Theta(g(n))$ so this is true.

$\lim_{n\to\infty} \frac{2^{n+1}}{2^n} = \lim_{n\to\infty} \frac{2^n \cdot 2}{2^n} = 2$

II. $2^{2n} = \Theta(2^n)$

$f(n) = 2^{2n}$

$g(n) = 2^n$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n 2^n}{2^n} = \infty$

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ then $f(n) = \Theta(g(n))$ so this is not true

III. Let $f(n) = O(n^2)$ and $g(n) = \Theta(n^2)$ Prove or disprove that $f(n) * g(n) = \Theta(n^4)$

$O(n^2) \Rightarrow 0 \le f(n) \le c_0 n^2$

$\times \; c_1 n^2 \le g(n) \le c_2 \cdot n^2$ } Multiply

$\leftarrow 0 \le f(n) \cdot g(n) \le c_0 c_2 \cdot n^4$

this is $O(n^4)$

$O(n^4) = f(n) \cdot g(n) \Rightarrow c_3 n^4 \le f(n) \cdot g(n) \le c_4 \cdot n^4$

$\Theta(n^4) = f(n) \cdot g(n) \Rightarrow 0 \le f(n) \cdot g(n) \le c_5 \cdot n^4$

Part 3:

$n^{1,01}$ $n\log^2 n$ 2^n $\sqrt{n}$ $(\log n)^3$ $n2^n$ 3^n

2^{n+1} $5^{\log_2^n}$ $\log n$

First let's group the functions

linear	exponential	logarithmic	root
$n^{1,01}$	2^n	$n.\log^2 n$	$\sqrt{n}$
$n^{\log_2^5}$	$n.2^n$	$\log^3 n$	
	3^n	$\log n$	
	2^{n+1}		

Comparing growth rate of linear functions

Let $f(n) = n^{1,01}$ and $g(n) = n^{\log_2^5}$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^{1,01}}{n^{\log_2^5}} = \frac{n^{1,01}}{n^{1,01} \cdot n^{\log_2^5 - 1,01}} = \frac{1}{n^{\log_2^5 - 1,01}} = \frac{1}{\infty} = 0$$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ so $g(n)$ grows faster.

$n^{\log_2^5} > n^{1,01}$

Comparing growth rate of exponantral function.

Let $f(n) = 2^n$ $g(n) = n.2^n$ $h(n) = 3^n$ $k(n) = 2^{n+1}$

$f(n)$ and $g(n)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{2^n}{n.2^n} = \frac{1}{n} = \frac{1}{\infty} = 0$$

so $g(n)$ grows faster than $f(n)$

$n.2^n > 2^n$

h(n) and k(n)

$$\lim_{n\to\infty} \frac{h(n)}{k(n)} = \frac{3^n}{2^{n+1}} = \frac{3^n}{2^n \cdot 2} = \left(\frac{3}{2}\right)^n \cdot \frac{1}{2} = \infty$$

so h(n) grows faster than k(n)

$$3^n > 2^{n+1}$$

g(n) and k(n)

$$\lim_{n\to\infty} \frac{g(n)}{k(n)} = \frac{n \cdot 2^n}{3^n} = \frac{n}{\frac{1}{\left(\frac{2}{3}\right)^n}} = \frac{\infty}{\infty}$$

Let's apply the L'hospital rule.

$$\frac{1}{\left(\frac{3}{2}\right)^n \cdot \ln\left(\frac{3}{2}\right)} = \frac{1}{\infty} = 0 \quad \text{so}$$

k(n) grows faster than g(n)

$$3^n > n \cdot 2^n$$

$$\boxed{3^n > n \cdot 2^n > 2^{n+1} > 2^n}$$

Comparing growth rate of logarithmic functions

Let's $f(n) = n.\log^2 n$ $g(n) = \log^3 n$ $h(n) = \log n$

$f(n)$ and $g(n)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n.\log^2 n}{\log^3 n} = \frac{n.}{\log(n)} = \frac{\infty}{\infty}$$ Let's apply L'hospital rule.

$$\frac{1}{\frac{1}{\log(n).\ln 10}} = \log(n).\ln 10 = \infty \quad \text{so}$$

$f(n)$ grows faster than $g(n)$

$$n.\log^2(n) > \log^3(n) > \log n$$

We know that grow rates are

$$1 < \log n < n < n\log n < n^2 < n^3 < 2^n < n!$$

$$3^n > n.2^n > 2^{n+1} > 2^n > 5^{\log_2^n} > n^{1,01} > n.\log^2 n$$

$$> \sqrt{n} > \log^3 n > \log n$$

Answer

Part 4:

a. Find the minumum-valued item

```
int min = arr[0]
for(int i=0 ; i< n ; ++i ){ // Θ(n)
        if( min > arr[i]) // Θ(1)
            min = arr[i]; // Θ(1)
}
```

$T(n) = \Theta(n) * \Theta(1) = \boxed{\Theta(n)}$

b. Find the median item. Consider each element one by one and check whether it is the median.

```
int temp;
for(int i=0 i<n-1 ; ++i ) { Θ(n-1) → Θ(n)
    for(int j=0 ; j<n-i ; ++j ){ // Θ(n-i-1) → Θ(n)
        if (arr[j] > arr[j+1]) { // Θ(1)
            temp = arr[j]; // Θ(1)
            arr[j] = arr[j+i]; // Θ(1)
            arr[j+1] = temp; // Θ(1)
        }
    }
}
if (n % 2)
    return (arr[n/2] + arr[n/2 -1]) /2;
return arr[n/2];
```

$T(n) \quad \Theta(n) * \Theta(n) = \boxed{\Theta(n^2)}$

c. Find two elements whose sum is equal to given value.

```
int sum = 10;
for( int i=0 ; i<n ; ++i ) { T_best(n) = Θ(1)  T_worst = Θ(n)
    for( int j=0 ; j<n ; ++j ) { T_best(n) = Θ(1)  T_worst(n) = Θ(n)
        if ( arr[i] + arr[j] == sum && i != j ) // Θ(1)
            return true; // Θ(1)
```

$T_w(n) = \Theta(n^2)$

$T_b(n) = \Theta(1)$

$$T(n) = O(n^2)$$

d. Assume there are two ordered arrglist of n elements. Merge these in increasing order.

```
int i=0, j=0, k=0;
for(  ; i<n && j<n ; ++k) { →   T_best(n) = Θ(n)   T_worst(n) = Θ(2n)
    if ( list1[i] < list2[j] ) → Θ(1)
        list3[k] = list1[i++] ; → Θ(1)
    else
        list3[k] = list2[j++] ; → Θ(1)
}
for( ; i<n ; ++k) → Θ(n)
    list3[k] = list1[i++] ; → Θ(1)
for( ; j<n ; ++k) → Θ(n)
    list3[k] = list2[j++] ; → Θ(1)
```

(2n → constant is not significant)

$$T(n) = \Theta(n) + \Theta(n) + \Theta(n) = \Theta(n)$$

Part 5:

a)

```
int p-1 (int array [])
    return (array [0] * array [2])
}
```

→ takes constant time, no loop

$T(n) = \Theta(1)$ $\quad$ $S(n) = O(1)$

b)

```
int p-2 (int array[], int n) {
    int sum = 0
    for (int i=0; i<n; i = i+5) → Θ(n/5)
        sum += array[i] * array[i] → Θ(1)
    return sum
}
```

$\Theta(n/5)$ → there is no meaning $\frac{1}{5}$ so

$T(n) = \Theta(n)$ $\quad$ $S(n) = O(1)$

c)

```
void p-3 (int array[], int n) {
    for(int i=0 ; i<n ; i++) → Θ(n)
        for(int j=0; j<i ; j*=2) → Θ(logn)
            printf ("%d", array [i]* array [j]) → Θ(1)
}
```

$T(n) = \Theta(n \cdot \log n)$

$S(n) = O(n)$

d)

```
void p-4 (int array[], int n) {

    if ( p-2 (array, n) > 1000)  → Θ(n)
        p-3 (array, n)  →  Θ(n log n)

    else
        printf ("%d", p-1(array) * p-2 (array, n)); → Θ(n)

}
```

$T(n) = \Theta(n \cdot \log n)$

$S(n) = O(n)$