PART 1:

1) public int searchProducts(Branch branch, ProductClass product, int number, int number){
int index = -1;
if(product instanceof Chairs){ Θ(1)
if(...) Θ(1)
throw new ---- Θ(1) → This is throw so I think it does not be best case so I did not calculate
for(int i=0; i<branch.getChairs().getUsed(); i++){ T(n)
if(index = branch.getChairs().contains(product)) != -1){ Θ(1)
return index; Θ(1)
}
}
}

public int contains(Object o)
T3(n) → for(int i=0; i<getUsed(); i++)
Θ(1) if(array[i].equals(o))
return i; Θ(1)
}
}

T5(n) = Θ(n) → to return true it reach end of for. so → Θ(n)

T3B(n) = Θ(1)
T3W(n) = Θ(n) } T3(n) = O(n)

public boolean equals(Object o)
Container<T> temp = ((Container<T>)o);
T5(n) for(int i=0; i<getUsed(); i++)
T1B(n) = Θ(1)
T1W(n) = Θ(n) } T1(n) = O(n)
Θ(1) if(array[i] != temp.array[i])
return false; Θ(1)
}
}
return true; Θ(1)
}

T(n) = T1(n) & T3(n) & T5(n) + Θ(1)
T(n) = O(n) & O(n) & Θ(n) + Θ(1)
T(n) = O(n³)

2) public void addProducts(ProductClass product, int number, int number){
if(product instanceof Chairs){
if(...) Θ(1)
throw new ---- Θ(1) → This is throw exception so I did not calculate it.
getBranch().getBookcases()[6].addElement(new Chairs(number, number));
}

$T(n) = T_{getBranch}(n) + T_{getChairs}(n) + T_{addElement}(n)$

$O(1)$ $O(1)$

$T(n) = O(1) + O(1) + T_{addElement}(\max(T_{if}, T_{else}))$

```
public void addElement(T element){
  if(isEmpty()){                                   T_if(n) = O(1)
    setUsed(used+1); O(1)                          T_else(n) = O(n)
    T[] temp=(T[]) new Object[used];               T_addElement(n) = O(n)
    temp[0]=element; O(1)                          T(n) = O(n)//
    array=temp; O(1)
  else{ setUsed(used+1); O(1)
      for(int i=0; i<getUsed()-1; i++){ O(n)
          temp[k]=array[i];
          k++;
      }
    temp[used-1]=element; O(1)
    array=temp; O(1)
  }
}
public void removeProducts(ProductClass product, int number, int number2){
  if(product instanceof Chairs){
    if(---)
      throw new ---
    getBranch().getChairs().removeElement(new Chairs(number,number1));
  }
}
```

$T(n) = T_{getBranch}(n) + T_{getChairs}(n) + T_{removeElement}(n)$

$T(n) = O(1) + O(1) + T_{removeElement}(n)$

```
public void removeElement(T element){
    boolean flag=true;
    if(element==null || contains(element)==-1)   T1(n)
        flag=false;                              O(1)
    for(int i=0; i<getUsed() && flag; ++i,)      T2(n)
        if(at(i).equals(element))                O(n) → equals
            this.array[i]=at(getUsed()-1);       O(1)
        } flag=false;                            O(1)
    }
    setUsed(getUsed()-1);
}
```

T3(n)

$T_{1B}(n) = \Theta(1)$

$T_{1W}(n) = \Theta(n)$

$T_1(n) = O(n)$

$T_{2B}(n) = \Theta(1)$

$T_{2W}(n) = \Theta(n)$

$T_2(n) = O(n)$

$T_3(n) = T_2(n) * T_{equals}(n)$

$T_3(n) = O(n) * \Theta(n)$

$T_3(n) = O(n^2)$

$T_{removeElement}(n) = T_1(n) + T_2(n)$

$O(n) + O(n^2)$

$T_{removeElement}(n) = O(n^2)$

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

3) `public void addProduct(Branch branch, Product product, int number, int number)`

```
if(product instanceof Chairs)
    branch.getChairs().addElement(new Chairs(number, number));
```

$T(n) = T_{getChairs}(n) + T_{addElement}(n)$

$\Theta(1)$

$T_{IF}(n) = \Theta(1)$

$T_{else}(n) = \Theta(n)$

$T_{addElement} = O(n)$

$T(n) = \Theta(1) + O(n)$

$T(n) = O(n)$

PART 23

a) T(n) is a running time, $f(n) = O(n^2)$ statement $\Rightarrow$ T(n) is an upper bound of f(n). since f(n) could be any function smaller than n^2 including constant function so we can rephase the statement as
"The running time of algorithm A is max be $O(n^2)$ or less than $O(n^2)$"

b) when we make addition in two function we take maximum of two function. so

$\theta(f(n)+g(n)) = \theta \max(f(n), g(n))$ $\Rightarrow$ θ says it is equal. not less or more I mean.

$\Downarrow$

$\max(f(n), g(n)) = \max(f(n) + g(n))$ $\Rightarrow$ they are equal so it is true.

c) I.

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

it is known that if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \in R$, then $f(n) = \theta(g(n))$

so $2^{n+1} = \theta(2^n)$ // it is true.

II.

$$\lim_{n\to\infty} \frac{2^{2n}}{2^n} = \lim_{n\to\infty} \frac{2^n \cdot 2^n}{2^n} = \lim_{n\to\infty} 2^n = \infty \notin R$$ then $f(n) \neq \theta(g(n))$

so $2^{2n} = \theta(2^n)$ so. steatement is false, $2^{2n} \neq \theta(2^n)$ //

III. $f(n) = O(n^2)$ $\rightarrow$ it can be n^2 or less than n^2, examples 1, n, etc.
$g(n) = \theta(n^2)$ $\rightarrow$ it can be only n^2, so
when we product each other it can be n^4 or less than n^4 because of $O(n^2)$.

$f(n) * g(n) = O(n^4)$ //

PART 3: $n^{1.01}, n\log^2 n, 2^n, \sqrt{n}, (\log n)^3, n.2^n, 3^n, 2^{n+1}, 5^{\log_2 n}, \log n$

$n^{1.01}$ $n\log^2 n$ (takes log of two sides)

$1.01\log n$ $\log n + \log\log^2 n$ (Let $\log n$ be x)

$1.01x$ $x + \log x^2$

$1.01x$ $x + 2\log x$ (let's remove x of two sides)

$0.01x$ $2\log x$ (when we throw coefficient)

x $\log x$

$n > \log n$ so, $n^{1.01} > n\log^2 n$

2^n $n.2^n$

$\lim_{n\to\infty} \frac{n.2^n}{2^n} = n$ so $n.2^n > 2^n$

2^{n+1} 2^n

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

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

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

$n\log^2 n$ $\log^3 n$ (when we divide $\log^2 n$ of two sides)

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

$5^{\log_2 n}$ 3^n (takes log of two sides)

$\log 5^{\log_2 n}$ $\log 3^n$

$\log n \log 5$ $n.\log 3$ (when we remove coefficient)

$\log n < n$ so $3^n > 5^{\log_2 n}$

$5^{\log_2 n}$ $n.2^n$ (takes log of two sides)

$\log n \log 5$ $\log n + n\log 2$ (when remove coefficient)

$\log n < n$ (max) so, $n.2^n > 5^{\log_2 n}$

$\log n$ $\sqrt{n}$ (takes log of two sides)

$\log\log n$ $\frac{1}{2}\log n$

$\log\log n < \log n$ so, $\sqrt{n} > \log n$

$5^{\log_2 n}$ $\quad 2^n$ (tobas log two sides) $\quad 2^n > n$

$\log n \log 5 \quad n \log 2$

$\log n \log 5 \quad n \log 2$

$\log n < n$ so, $2^n > 5^{\log_2 n}$

$5^{\log_2 n} \quad n \log^2 n$

$\log 5 \log n \quad \log n + \log_2 \log^2 n \quad (\log n = x)$

$\log 5 \cdot x > x + 2 \log x$, so $n \log^2 n < 5^{\log_2 n}$

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

$1,01 \log n \quad \log 5 \log n$

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

$3^n \quad n 2^n$

$\log 3^n \quad \log n + n \log 2$

$n \log 3 > \log n + n \log 2$ so, $3^n > n \cdot 2^n$

$3^n > n \cdot 2^n > 2^{n+1} > 2^n > 5^{\log_2 n} > n^{1,01} > n \log^2 n > \sqrt{n} > (\log n)^3 > \log n$

PART 4 :

```
public int findmin(int[] array){
1) for (i=0 i<array.length<i++){ θ(n)
      if (arr[i] <min) (θ(1)
   }        min = array[i]; θ(1)
   return min; θ(1)
```

int min = array[0]

$T(n) = \theta(n) + \theta(1) + \theta(1) = T(n) - \theta(n)$

2) void findnumber(int arr[]){
int counter1=0, counter2=0; boolean control1=false, control2=false;
int number1=0, number2=0;
if(n%2==1){
for(int i=0; i<arr.length; i++){ $T_1(n)$ → $T_{1B}(n)=\Theta(1)$, $T_{1W}(n)=\Theta(n)$, $T_1(n)=O(n)$
counter1=0;
counter2=0;
for(int j=0; j<arr.length; j++){ $T_2(n)$ → $T_2(n)=\Theta(n)$
if(arr[i]>arr[j]){ Θ(1)
counter1++; Θ(1)
}
else if(arr[i]<arr[j]){ Θ(1)
counter2++; Θ(1)
}
}
if(counter1==counter2){ Θ(1)
System.out.println("%d", arr[i]); Θ(1)
} break;
} }
else if(n%2==0){
for(int i=0; i<arr.length; i++){ $T_3(n)$ → $T_{3B}(n)=\Theta(1)$, $T_{3W}(n)=\Theta(n)$, $T_3(n)=O(n)$
counter1=0;
counter2=0;
for(int j=0; j<arr.length; j++){ $T_4(n)$ $T_4(n)=\Theta(n)$
if(arr[i]>arr[j]){ Θ(1)
counter1++; Θ(1)
}
else if(arr[i]<arr[j]){ Θ(1)
counter2++; Θ(1)
} }
Θ(1) → if(counter1-counter2==1){control1=true; number1=arr[i];}
Θ(1) → if(counter2-counter1==1) control2=true; number2=arr[i];
Θ(1) → if(control1==true && control2==true){
Θ(1) → System.out.println("%d",(number1+number2)/2);
break;
} } }

There is no best worst case for T_2 it has to be reach end of for;

$T_{IF\,B}(n)=O(n)$ → $\Theta(1)+O(n)$
$T_{IF\,W}(n)=O(n^2)$ → $\Theta(n)+O(n)$

$T_{IF}(n)=O(n)+\Theta(n)$
$T_{IF}(n)=O(n^2)$

$T_{ELSEIF\,B}(n)=O(n)$ → $\Theta(1)+\Theta(n)$
$T_{ELSEIF\,W}(n)=O(n^2)$ → $\Theta(n)+\Theta(n)$

$T_{ELSEIF}(n)=O(n^2)$ //

$T(n)=\max(T_{IF}(n), T_{ELSEIF}(n))$
$T(n)=O(n^2)$ //

$T_{IF}(n)=O(n^2)$
$T_{ELSEIF}(n)=O(n^2)$

3) static int[] findtwoelement (int Array[], int sum)

```
int i,j;   int arr[]=new int[2];
Arrays.sort(Array);      TArray.sort()(n) ∈ O(n log n)
i=0
j= array.length-1;
while (i<j){ O(1)  T2
    if(Array[i]+Array[j]==sum){ O(1)
        arr[0]=i  arr[1]=j; } O(1)
    else if( Array[i]+Array[j]<sum) (O(1)
        i++  O(1)
    else{
        j++  O(1)
    }
}
return arr; O(1)
}
```

T2(n) ⇒ T20(n) = O(1) first

T2w(n) = O(n) last

T20(n) = O(n)

T(n) = n log n + n

T(n) = O(n log n)//

4)

4)

```
public static int[] mergeArrays(int arr1[], int arr2[]){
int[] arr3 = new int[arr1.length + arr2.length];
int i=0, j=0, k=0;
while (i<arr1.length && j<arr2.length){ ← T1
    if(arr1[i] < arr2[j]) O(1)
        arr3[k++] = arr1[i++] O(1)
    else
        arr3[k++] = arr2[j++] O(1)
}

while (i<arr1.length){ ← T2
    arr3[k++] = arr1[i++]; O(1)
}

while (j<arr2.length){ ← T3
    arr3[k++] = arr2[j++]; O(1)
}
return arr3; O(1)
}
```

$T_n = T_{1(n)} + T_{2(n)} + T_{3(n)}$

$T_{1B}(n) = O(n)$, $T_{2B}(n) = O(1)$, $T_{3B}(n) = O(1)$ → $O(n)$

$T_{1ww}(n) = O(n^2)$ $\quad T_{2w} = O(n)$ $\quad T_{3ww}(n) = O(n)$

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

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

PART 5:

a)
```
int p_1(int array[]){
    return (array[0] * array[2]) θ(1)
}
```
$T(n) = \theta(1)$

Space complexity = $\theta(1)$ since it does not take up extra space.

b)
```
int p_2(int array[], int n){
    int sum=0 θ(1)
    for(int i=0; i<n; i=i+5) ⟶ T1(n) = θ(n/5) = θ(n)
        sum += (array[i] * array[i]) θ(1) T2(n) = θ(1)
    return sum θ(1)
}
```
$T(n) = \theta(n)$

Space complexity = $\theta(1)$ since it does not take up extra space.

$T(n) = T_1(n) + T_2(n)$

$T(n) = \theta(n) + \theta(1)$

$T(n) = \theta(n)$

c)
```
void p_3(int array[], int n){
    for(int i=0; i<n; i++) T1(n) ⟶ θ(n)
        for(int j=1; j<i; j=j*2) T2(n) ⟶ θ(logn)
            printf("%d", array[i] * array[j]) T3(n) ⟶ θ(1)
}
```
$T(n) = \theta(n) * \theta(\log n)$

$T(n) = \theta(n \log n)$

Space complexity = $\theta(1)$ since it does not take up extra space.

d)
```
void p_4(int array[], int n){
    if(p_2(array, n) > 1000) T1(n)
        p_3(array, n) T2(n)
    else
        printf("%d", p_1(array) * p_2(array, n)) T3(n) = θ(1) * θ(n)
}
```
$T(n) = \max(T_2(n), T_3(n)) + T_1(n)$

$T_1(n) = \theta(n)$

$T_2(n) = \theta(n \log n)$

$T_3(n) = \theta(n)$

$T(n) = \max(\theta(n \log n), \theta(n)) + \theta(n)$ Space complexity = $\theta(1)$

$T(n) = \theta(n \log n) + \theta(n) \rightarrow T(n) = \theta(n \log n)$