GTU CSE 222/505 – Data Structures And Algorithms

Homework 2

HATİCE SEVRA GENÇ

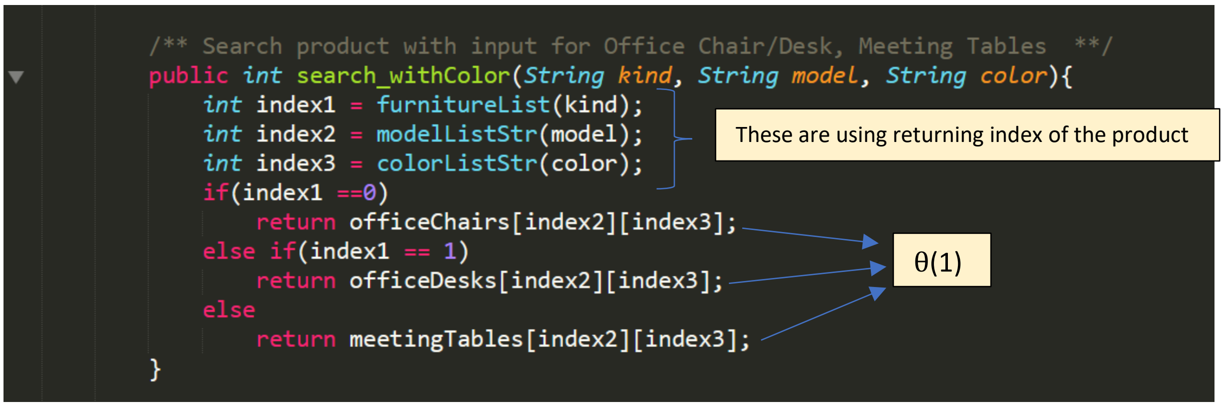
1801042611

PART 1

1.1 Search Product

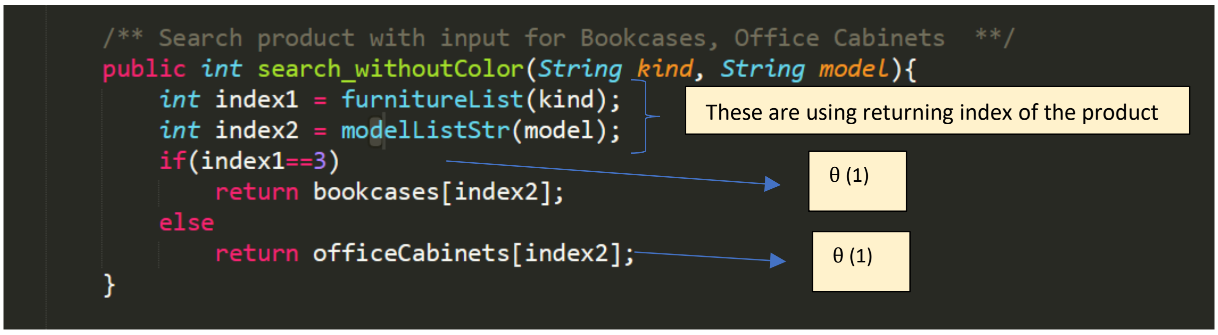
Both functions are running with same steps. And their time complexity is

T(n) = O(1)



```
/** Search product with input for Office Chair/Desk, Meeting Tables  **/
public int search_withColor(String kind, String model, String color){
    int index1 = furnitureList(kind);
    int index2 = modelListStr(model);
    int index3 = colorListStr(color);
    if(index1 ==0)
        return officeChairs[index2][index3];
    else if(index1 == 1)
        return officeDesks[index2][index3];
    else
        return meetingTables[index2][index3];
}
```

These are using returning index of the product
θ(1)



```
/** Search product with input for Bookcases, Office Cabinets  **/
public int search_withoutColor(String kind, String model){
    int index1 = furnitureList(kind);
    int index2 = modelListStr(model);
    if(index1==3)
        return bookcases[index2];
    else
        return officeCabinets[index2];
}
```

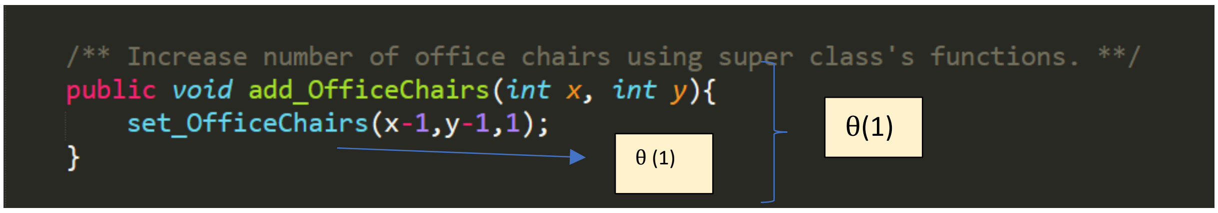
These are using returning index of the product
θ (1)
θ (1)

1.2 Add/Remove Product

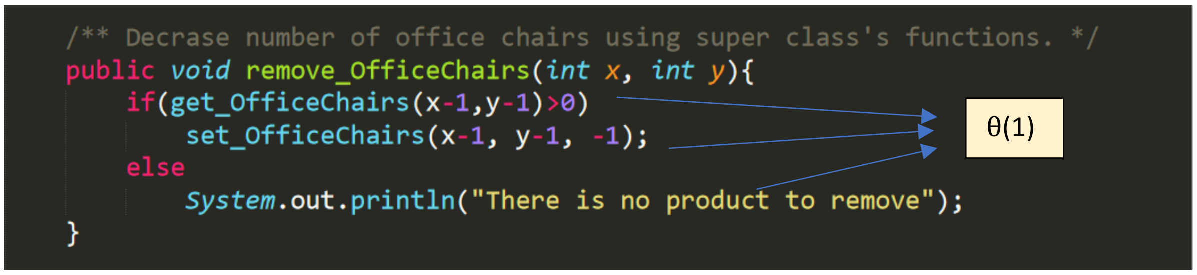
In the first assignment, I made the add/remove product operation by having the employee select the index from the printed lists. Therefore, my add / remove functions are too short.

And total complexity of each function is :

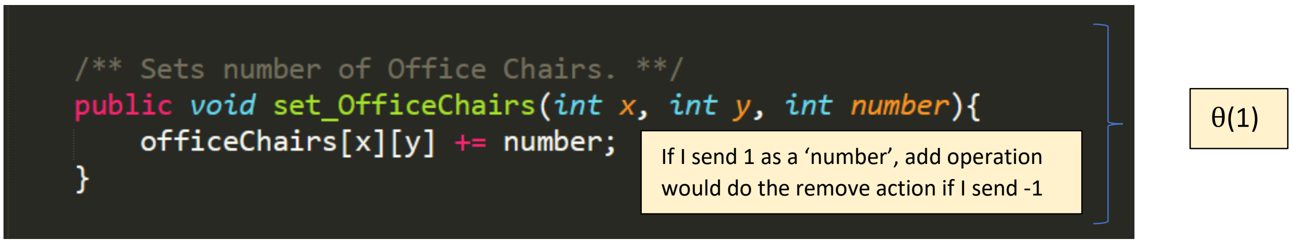
T(n) = O(1)



```
/** Increase number of office chairs using super class's functions. **/
public void add_OfficeChairs(int x, int y){
    set_OfficeChairs(x-1,y-1,1);
}
```



```java
/** Decrase number of office chairs using super class's functions. */
public void remove_OfficeChairs(int x, int y){
    if(get_OfficeChairs(x-1,y-1)>0)
        set_OfficeChairs(x-1, y-1, -1);
    else
        System.out.println("There is no product to remove");
}
```

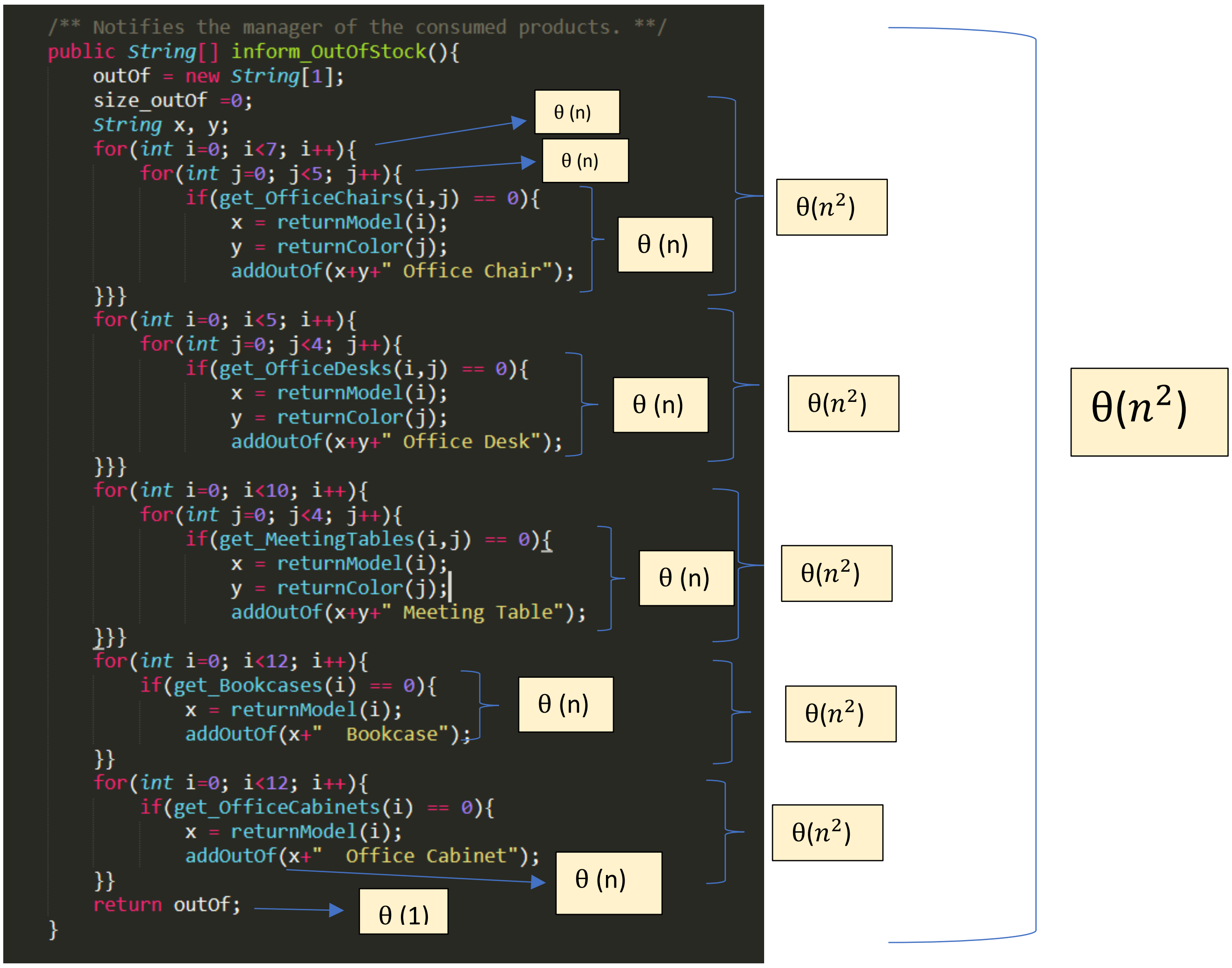


```java
/** Sets number of Office Chairs. **/
public void set_OfficeChairs(int x, int y, int number){
    officeChairs[x][y] += number;
}
```

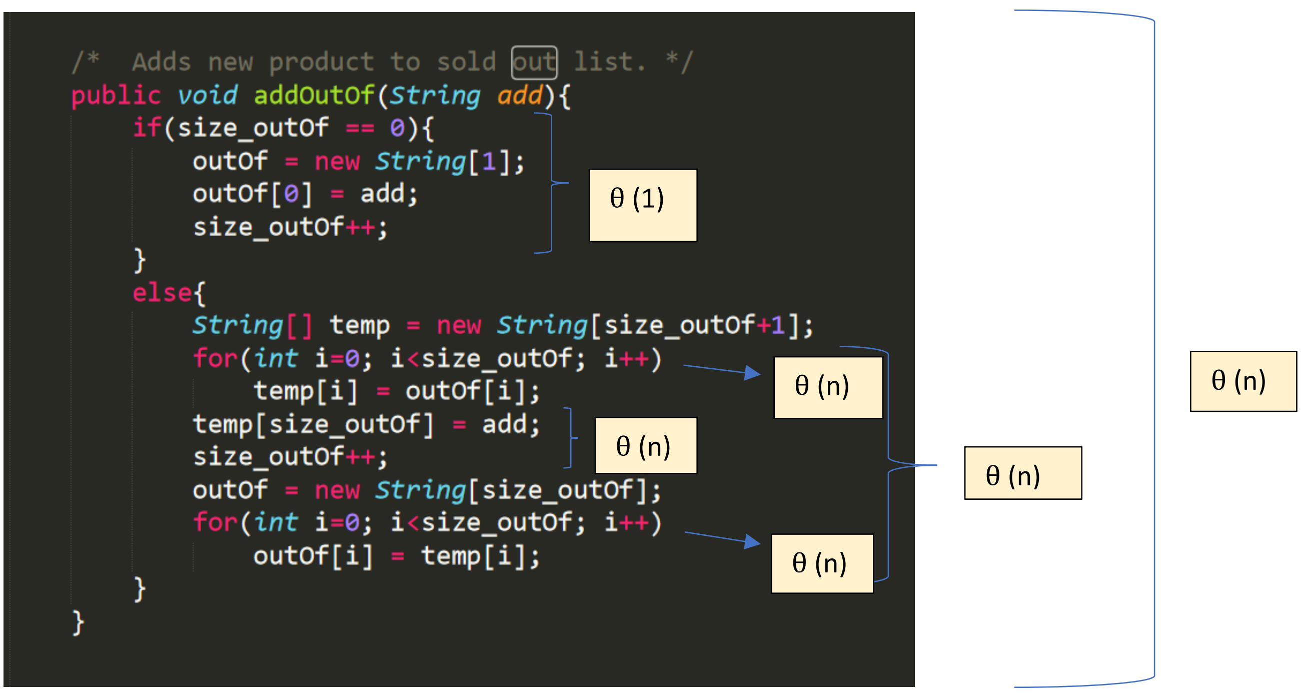
If I send 1 as a 'number', add operation would do the remove action if I send -1

1.3 Querying the Products that Need to be Supplied

'inform_OutOfStock' function is detect the product to need to be supplied. And hold the details of the product (model and color). 'addOutOf' function is adds to product to the list.



```java
/** Notifies the manager of the consumed products. **/
public String[] inform_OutOfStock(){
    outOf = new String[1];
    size_outOf =0;
    String x, y;
    for(int i=0; i<7; i++){
        for(int j=0; j<5; j++){
            if(get_OfficeChairs(i,j) == 0){
                x = returnModel(i);
                y = returnColor(j);
                addOutOf(x+y+" Office Chair");
    }}}
    for(int i=0; i<5; i++){
        for(int j=0; j<4; j++){
            if(get_OfficeDesks(i,j) == 0){
                x = returnModel(i);
                y = returnColor(j);
                addOutOf(x+y+" Office Desk");
    }}}
    for(int i=0; i<10; i++){
        for(int j=0; j<4; j++){
            if(get_MeetingTables(i,j) == 0){
                x = returnModel(i);
                y = returnColor(j);
                addOutOf(x+y+" Meeting Table");
    }}}
    for(int i=0; i<12; i++){
        if(get_Bookcases(i) == 0){
            x = returnModel(i);
            addOutOf(x+"  Bookcase");
    }}
    for(int i=0; i<12; i++){
        if(get_OfficeCabinets(i) == 0){
            x = returnModel(i);
            addOutOf(x+"  Office Cabinet");
    }}
    return outOf;
}
```



```
/*  Adds new product to sold out list. */
public void addOutOf(String add){
    if(size_outOf == 0){
        outOf = new String[1];
        outOf[0] = add;
        size_outOf++;
    }
    else{
        String[] temp = new String[size_outOf+1];
        for(int i=0; i<size_outOf; i++)
            temp[i] = outOf[i];
        temp[size_outOf] = add;
        size_outOf++;
        outOf = new String[size_outOf];
        for(int i=0; i<size_outOf; i++)
            outOf[i] = temp[i];
    }
}
```

PART 2

2.1 Explain why it is meaningless to say: “The running time of algorithm A is at least O(n^2)”

Because Big-Oh is the ‘worst case’ notation. In other words, it is the upper-bound of the function. So here ‘n’ gets the maximum value it can get in the code. For example, if it is in the loop, it goes until the end of the loop.

2.2 Let f(n) and g(n) be non-decreasing and non-negative functions. Prove or disprove that: max(f (n), g(n)) = Θ(f(n) + g(n)).

if T1(N) = O(f(N)) and T2(N) = O(g(N)) then T1(N) +T2(N) = max(O(f(N)), O(g(N))

Because sum of different O notations are equal to the highest order element. For example O(n) + $O(n^2)$ = $O(n^2)$. We always ignore lower term elements and constants.

2.3 Are the following true? Prove your answer.

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

In big-oh notation we are trying to obtain the simpliest notation. Therefore we ignore constants and lower order terms. We just use highest term. In this example 2^{n+1} is the same with 2^n.

[Answer: TRUE]

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

This example is different than the first one. There is no constant here. These two are written in base 2. But that doesn't mean they are same number. The first expression is actually base 4 ($2^2 = 4$), not 2. So the equation is wrong.

[Answer: FALSE]

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

In this example, exponantial expressions n^2 and n^4 is different. There is no constans. There is no equavalency between them. It means This equation is not true.

By the rule, theta notation is true for big-oh notation. But vice versa is not correct. $O(n^2) * \theta(n^2) = O(n^4)$ this equation disproves the given equation.

[Answer: FALSE]

PART 3

3.1 List the following functions according to their order of growth by explaining your assertions.

$n^{1.01}$, $n\log^2 n$, 2^n, √n, $(\log n)^3$, $n2^n$, 3^n, 2^{n+1}, $5^{\log_2 n}$, logn

1) $n\,log^2 n$ - $(logn)^3$ - $5^{\log_2 n}$ - $logn$ (Logarithmic)
2) 2^n - $n2^n$ - 3^n - 2^{n+1} (Exponential)
3) $n^{1.01}$ - $\sqrt{n}$ (Polynomials)

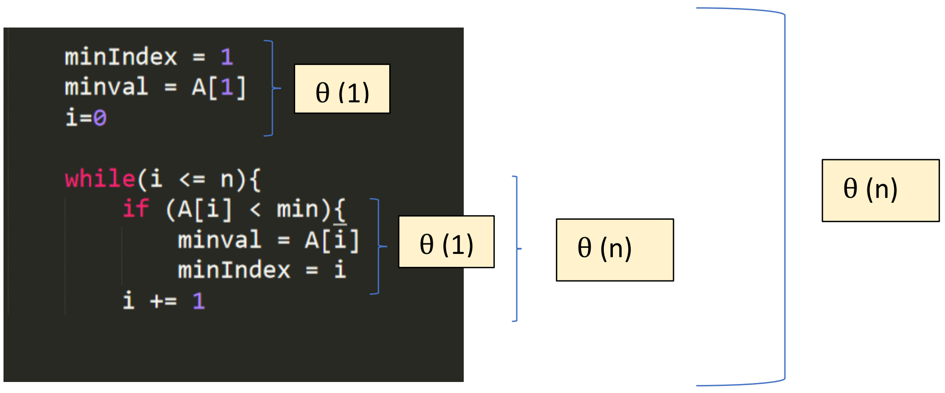
- We always ignore constants in the notation. So we can accept → $2^{n+1} = 2^n$
- If we compare exponential numbers, we can see → $3^n > n2^n > 2^n = 2^{n+1}$
- When we analyze logarithmic expressions, we can easily say → $(logn)^3 > logn$
- When we compare $(logn)^3$ and $n\,log^2 n$ → $n\,log^2 n > (logn)^3$
- If we compare n base expressions → $n^{1.01} > \sqrt{n}$
- $5^{\log_2 n}$ is the expression that has the highest value

Answer:

$\boldsymbol{5^{\log_2 n} > 3^n > n2^n > 2^n = 2^{n+1} > n^{1.01} > \sqrt{n} > n\,log^2 n > (logn)^3 >> logn}$

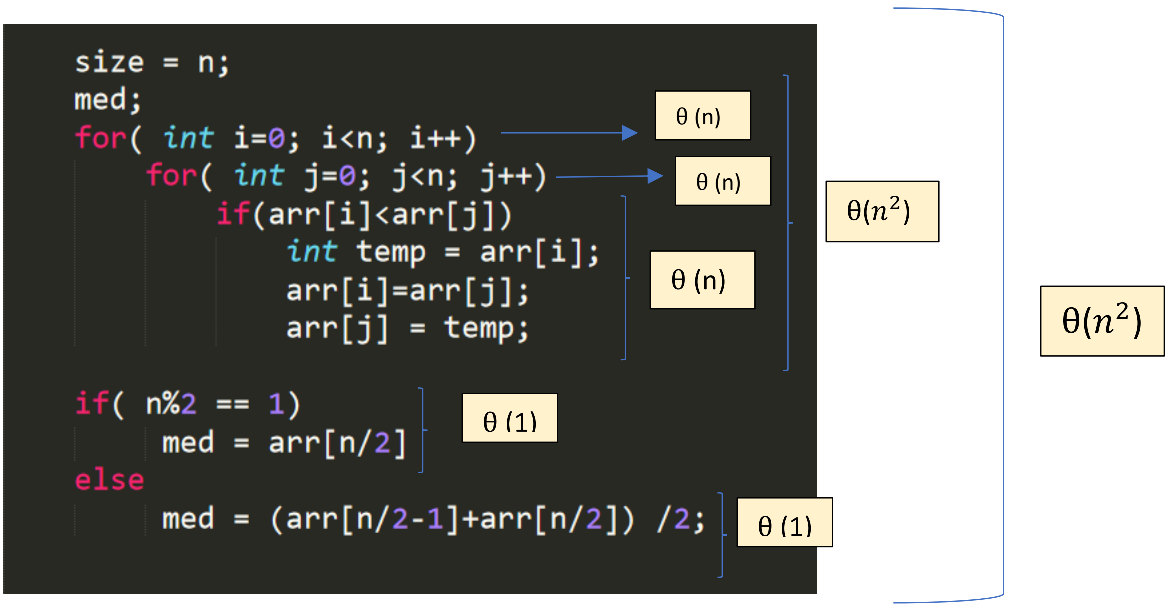
PART 4

4.1 Find the minimum-valued item



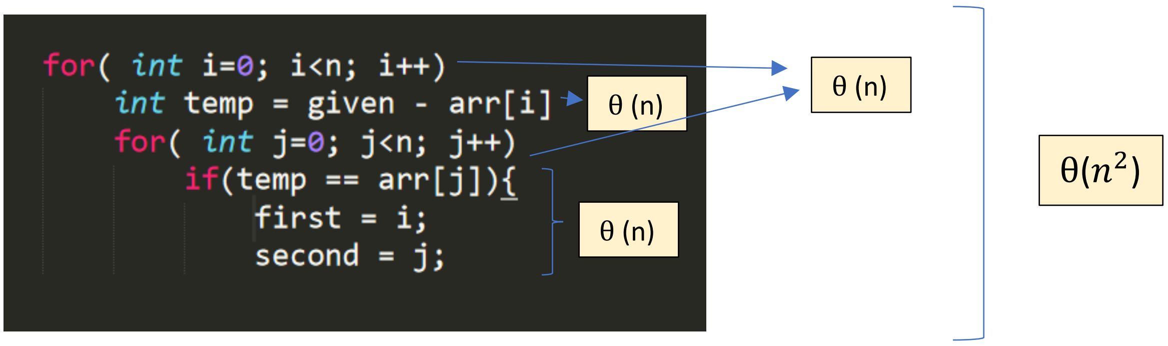
```
minIndex = 1
minval = A[1]
i=0

while(i <= n){
    if (A[i] < min){
        minval = A[i]
        minIndex = i
    i += 1
```

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



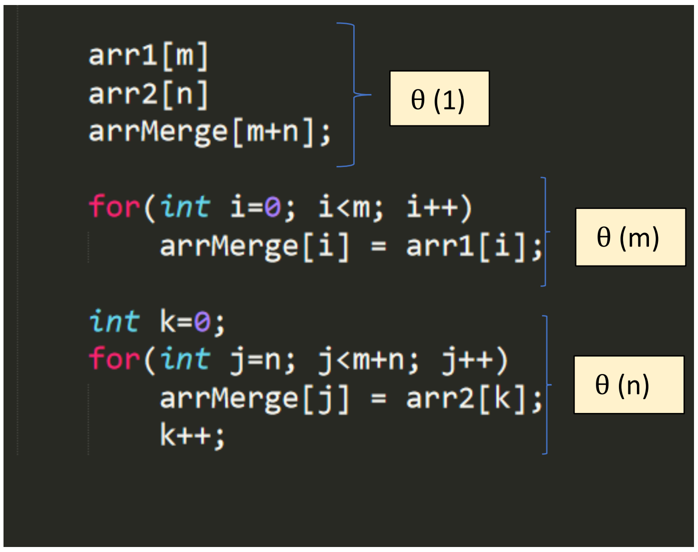
```
size = n;
med;
for( int i=0; i<n; i++)
    for( int j=0; j<n; j++)
        if(arr[i]<arr[j])
            int temp = arr[i];
            arr[i]=arr[j];
            arr[j] = temp;

if( n%2 == 1)
    med = arr[n/2]
else
    med = (arr[n/2-1]+arr[n/2]) /2;
```

4.3 Find two elements whose sum is equal to a given value



```
for( int i=0; i<n; i++)
    int temp = given - arr[i]
    for( int j=0; j<n; j++)
        if(temp == arr[j]){
            first = i;
            second = j;
```

4.4 Assume there are two ordered array list of n elements. Merge these two lists to get a single list in increasing order.



```
arr1[m]
arr2[n]
arrMerge[m+n];

for(int i=0; i<m; i++)
    arrMerge[i] = arr1[i];

int k=0;
for(int j=n; j<m+n; j++)
    arrMerge[j] = arr2[k];
    k++;
```

θ (1)

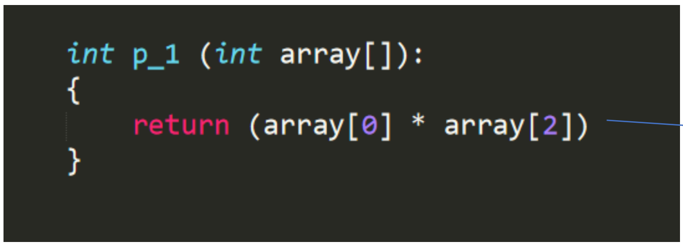
θ (m)

θ (n)

θ (m) = θ (n) = θ (m+n)

PART 5

5.1



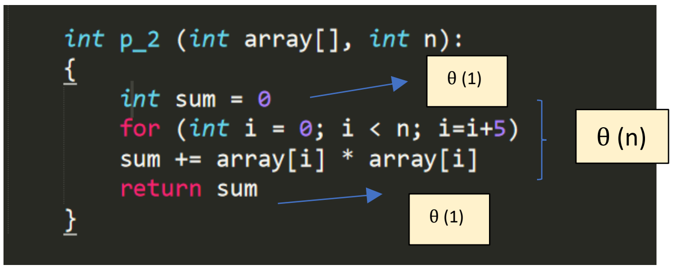
```
int p_1 (int array[]):
{
    return (array[0] * array[2])
}
```

θ (1)

Time – θ (1)

Space – θ (1)

5.2



```
int p_2 (int array[], int n):
{
    int sum = 0
    for (int i = 0; i < n; i=i+5)
    sum += array[i] * array[i]
    return sum
}
```

θ (1)

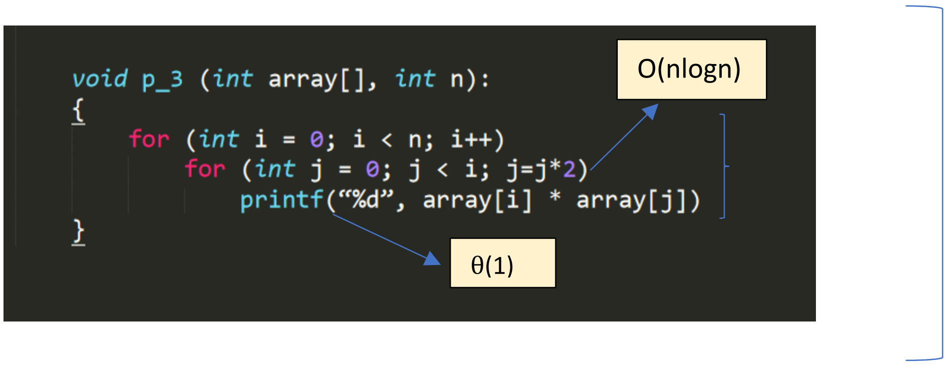
θ (n)

θ (1)

Time – θ (n)

Space – θ (1)

5.3

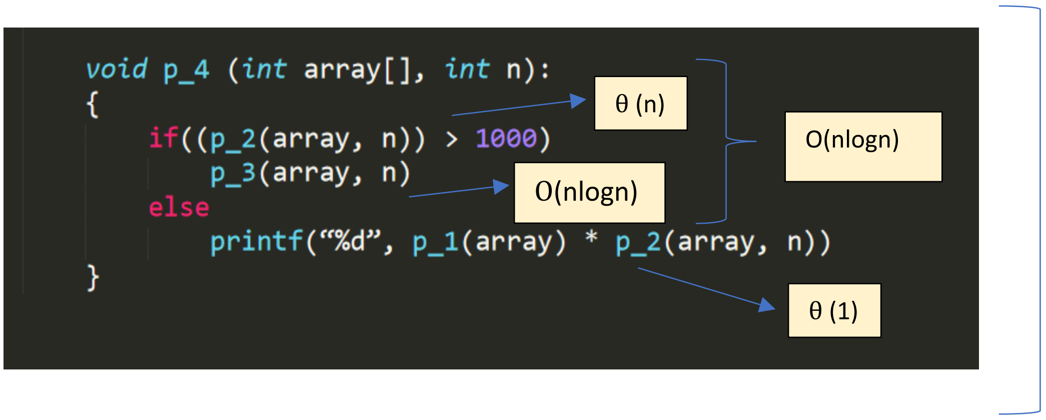


```
void p_3 (int array[], int n):
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < i; j=j*2)
            printf(“%d”, array[i] * array[j])
}
```

Time – O(nlogn)

Space – θ (1)

5.4



```
void p_4 (int array[], int n):
{
    if((p_2(array, n)) > 1000)
        p_3(array, n)
    else
        printf(“%d”, p_1(array) * p_2(array, n))
}
```

Time - O(nlogn)

Space – θ (1)