Time complexity cont.

Development of an algorithm for finding max sub array.

```
{ -2,11,-4,13,-5,2 } => Result = 20
{ 1,2,-5,4,7,-2 } => Result = 11
{ 1,5,-3,4,-2,1 } => Result = 7
```

Time complexity cont.

Finding max sub array using Brute Force algorithm.

```
public int maxSubArray(int[] a) {
    int maxTop = 0;                                 1
    for (int i = 0; i < a.length; i++)              n+1
        for (int j = i; j < a.length; j++) {        n(n+1)/2+n
            int top = 0;                            n(n+1)/2
            for (int k = i; k <= j; k++)            n(n+1)(2n+1)/12+3n(n+1)/4
                top += a[k];                        n(n+1)(2n+1)/12+n(n+1)/4
            if (top > maxTop) {                     n(n+1)/2
                maxTop = top;                       ...
            }
        }
    return maxTop;
}
```

T(N)=n^3/3+4n^2+4n... => O(n^3)

Time complexity cont.

Finding max sub array using improved algorithm.

```
public int maxSubArray(int[] a) {
    int maxTop = 0;                                   1
    for (int i = 0; i < a.length; i++)                n+1
        int top = 0;                                  n
        for (int j = i; j < a.length; j++) {          n(n+1)/2+n
            top += a[j];                              n(n+1)/2
            if (top > maxTop) {                       n(n+1)/2
                maxTop = top;                         ...
            }
        }
    return maxTop;                                    1
}
```

$T(N)=3n^2+6n+3 \Rightarrow O(n^2)$

Time complexity cont.

Finding max sub array using linear algorithm.

```
public int maxSubArray(int[] a) {
    int maxTop = 0;                                   1
    int top = 0;                                      1
    int i = 0;                                        1
    for (int j = 0; j < a.length; j++) {              n+1
        top += a[j];                                  n
        if (top > maxTop) {                           n
            maxTop = top;                             ...
        } else
            if (top < 0) {
                i = j + 1;
                top = 0;
            }
    }
    return maxTop;                                    1
}
```

T(N)=7n+5 => O(n)

Time complexity using Sigma (Math) notation.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{2n^3+3n^2+n}{6} = \frac{n(2n+1)(n+1)}{6}$$

$$\sum_{i=1}^{\log n} n = n \log n$$

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1-a} \text{ for } 0 < a < 1$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1} \text{ for } a \neq 1$$

$$\sum_{i=1}^{n} \frac{1}{2^i} = 1 - \frac{1}{2^n}$$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$$\sum_{i=0}^{\log n} 2^i = 2^{\log n + 1} - 1 = 2n - 1$$

$$\sum_{i=1}^{n} \frac{i}{2^i} = 2 - \frac{n+2}{2^n}$$

Time complexity for recursive codes.

```
public int recursiveFunction(int n) {
    if (n <= 0)
        return 1;
    else
        return 1 + recursiveFunction(n-1);
}
```

T(n) = a + T(n – 1)

⇩

T(n) = n * a + T(0) = n * a + b = O(n)

where a, b are some constant.

Time complexity for recursive codes.

```
public int recursiveFunction(int n) {
    if (n <= 0)
        return 1;
    else
        return 1 + recursiveFunction(n-5);
}
```

T(n) = a + T(n – 5)

T(n) = ceil(n / 5) * a + T(k) = ceil(n / 5) * a + b = O(n)

where a, b are some constant and k <= 0.

Time complexity for recursive codes.

```java
public int recursiveFunction(int n) {
    if (n <= 0)
        return 1;
    else
        return 1 + recursiveFunction(n/2);
}
```

```
T(n) = a + T(n / 2)
```

```
T(n) = log2(n) * a + T(0) = log2(n) * a + b = O(log n)
```

```
where a, b are some constant.
```

Time complexity for recursive codes.

```
public int recursiveFunction(int n, int m, int o) {
    if (n <= 0)
        printf(“%d, %d\n”, m, o);
    else {
        recursiveFunction(n – 1, m + 1, o);
        recursiveFunction(n – 1, m, o + 1);
    }
}
```

```
T(n) = a + 2 * T(n - 1)
```

```
T(n) = a + 2a + 4a + … + 2^(n-1) * a + T(0) * 2^n
     = a * 2^n – a + b * 2^n
     = (a + b) * 2^n – a
     = O(2^n)

where a, b are some constant.
```

Time complexity for recursive codes.

```
public int recursiveFunction(int n) {
    for (int i = 0; i < n; i +=2)
        printf(“hi”);
    if (n <= 0)
        return 1;
    else
        return 1 + recursiveFunction(n – 5);
}
```

```
T(n) = n / 2 + T(n - 5)
```

⇩

```
T(n) = ceil(n / 5) * n / 2 + T(k)
     = ceil(n / 5) * n / 2 + b = O(n^2)
```

```
where a, b are some constant and k <= 0.
```