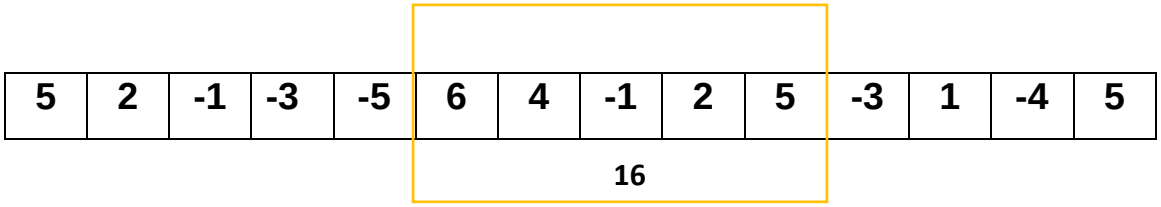
Chapter 1

Introduction

Max subarray sum problem

Given a list of n integers (positive, negative, zero). Find the subarray sum of values is maximum.

Example:



Input: vector of n real number.

Output: the maximum sum found in any contiguous subarray of the input.

Algorithm #1

```
max = -maxinteger;                          Time
for ( i = 1 ; i <= n ; i++)                  n
    for ( j = i ; j <= n ; j++)              n
        Sum = 0;
        for ( k = i; k <= j; k++)            n
            sum = sum + A [ k ];
        end for
        if ( max < sum )
            max = sum
        end if
    end for                                  _____
end for                                      T(n) = O(n^3)
```

Algorithm #2

Sum (A[i] to A[j+1]) = sum (A[i] to A[j]) + A[j+1]

```
max = -maxinteger;                                  Time
for ( i = 1 ; i <= n ; i++)                          n
    sum = 0;
    for ( j = i ; j <= n ; j++)                      n
        sum = sum + A[ j ];
        if ( max < sum )
            max = sum
        end if
    end for                                          ____________
end for                                              T(n) = O(n^2)
```

Algorithm #3

Sum (A[i] to A[j+1]) = sum (A[i] to A[j]) + A[j+1]

```
accumArray [ 0 ] = 0;
for ( i = 1 ; i <= n ; i++)
    accumArray[ i ] = accumArray[ i -1 ] + A[ i ];
end for
max = 0;
for ( i = 1 ; i <= n ; i++)
    for ( j = i ; j <= n ; j++)
        sum = accumarray[ j ] - accumArrayA[ i-1 ];
        if ( max < sum )
            max = sum
        end if
    end for
end for
```

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

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

Algorithm #4

Recursive function maxSum (L, U)

```
maxSum ( L, U )
        if ( L > U )
            return 0;
        if ( L == U )
            return max ( 0, A[ L ] );
        m = ( L + U ) /2;
        // find max crossing to left
        sum = 0;
        maxToLeft = 0;
        for ( i = m ; i >= L ; i--)
            sum = sum + A[ i ];
            maxToLeft = max ( maxToLeft, sum );
        end for
        // find max crossing to right
        sum = 0;
        maxToRight = 0;
        for ( i = m+1 ; i <= U ; i++)
            sum = sum + A[ i ];
            maxToRight = max ( maxToRight, sum );
        end for
        maxCrossing  = maxToLeft + maxToRight;
        maxInA = maxSum( L, m);
        maxInB = maxSum( m+1, U );
        return ( max ( maxCrossing, maxInA, maxInB);
```

Time

$$T(n) = \begin{cases} c & n = 1 \\ 2\,T(n/2) + n & n > 1 \end{cases}$$

$T(n) = 2\,T(n/2) + n$

$T(n/2) = 2\,T(n/4) + n/2$

$T(n) = 2\,[\,2\,T(n/4) + n/2\,] + n$

$T(n) = 2^2\,T(n/2^2) + 2n$

$T(n/4) = 2\,T(n/8) + n/4$

$T(n) = 2^2\,[\,2\,T(n/8) + n/4\,] + 2n$

$T(n) = 2^3\,T(n/2^3) + 3n$

…

$T(n) = 2^k\,T(n/2^k) + k\,n$

Let $2^k = n$ ➔ $k = \log n$

$T(n) = n\,T(1) + n \log n$

$T(n) = c\,n + n \log n$

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

Algorithm #5

	5	**2**	**-1**	**-3**	**-5**	**6**	**4**	**-1**	**2**	**5**	**-3**	**1**	**-4**	**5**
maxEndHer	**5**	**7**	**6**	**3**	**0**	**6**	**10**	**9**	**11**	**16**	**13**	**14**	**10**	**15**
maxSoFar	**5**	**7**	**7**	**7**	**7**	**7**	**10**	**10**	**11**	**16**	**16**	**16**	**16**	**16**

```
maxEndHer = 0;
maxSoFar = 0;
for ( i = 1; i <= n ; i++)
        maxEndHer = max ( 0, maxEndHer + A[ i ] );
        maxSoFar = max ( maxSoFar, maxEndHer );
end for
```

T(n) = O(n)

Problem: Chose the smallest k^{th} elements

Solution:

Sort and Scan **(Quick Sort)**

Problem: Element uniqueness problem

Given a list of n numbers $a_1, a_2, a_3, \ldots, a_n$

Find if there is a pair i, j such that $a_i = a_j$

Solution:

Sort and Scan

$T(n) = O(n\log n) + n = O(n\log n)$

Problem: Element uniqueness problem

Given a list of n numbers $a_1, a_2, a_3, \ldots, a_n$ $\quad 0 <= a_i <= 10^5$

Find if there is a pair i, j such that $a_i = a_j$

Solution #1:

We reserve an array with 10^5 **integers**

Input	**5**	**7**	**6**	**3**	**0**	**6**	**5**	**9**	**11**	**16**	**3**	**5**	**10**	**…**

index	0	1	2	3	4	5	6	7	8	9	10	11	12	…
Repeation	**1**	**0**	**0**	**2**	**0**	**3**	**2**	**1**	**0**	**1**	**1**	**1**	**0**	**…**

T(n) = O(n)

Extra Space = 10^5 unit of integer = 10^5 * 4 bytes

≈ 400 Kbyte

Solution #2:

because we need just the number of elements, where $a_i = a_j$, $i \neq j$

We reserve an array with 10^5 **bytes** and then let every **byte** = 0, When we have element, we change the **byte** from zero to 1, and if we have 1 then $a_i = a_j$

T(n) = O(n)

Extra Space = 10^5 unit of bytes = 10^5 bytes

≈ 100 Kbyte

Solution #3:

because we need just the number of elements, where $a_i = a_j$, $i \neq j$

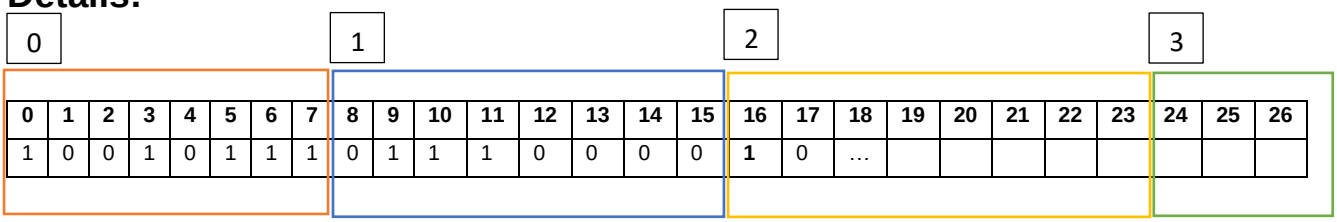
We reserve an array with 10^5 **bit** and then let every **bit** = 0, When we have element, we change the **bit** from zero to 1, and if we have 1 then $a_i = a_j$

T(n) = O(n)

Extra Space = 10^5 bits ≈ 12500 bytes

≈ 12.5 Kbyte

Details:




```
byteNumber = n / 8
bitNumber = n % 8
    count  = 0;
    for ( i = 0, i <= n ; i++)
        byteNumber = A[ i ] / 8
        bitNumber =( 7 - A[ i ] % 8 )
        if ( (A[ byteNumber ] & (1 << bitNumber )) == 0) then
            A[ byteNumber ] = A[ byteNumber ] | ( 1 <<  bitNumber );
        else
            count++;
        end if
    end for
```

Problem:

Given a list of numbers

$a_1, a_2, a_3, \ldots a_n$, a_i is between 1 and k

Output: array b where $b_1 <= b_2 <= b_3 <= \ldots <= b_n$

Solution:

```
for ( i = 1 ; i <= k; i++)
        c[ I ] = 0;

for ( i = 1; i <= n; i++)
        c[ A[ I ] ] ++;
count = 1;
for ( i = 1; i <= k; i++)
        for ( j = 1; j <= c[ I ] ; j++)
                b[ count++ ] = I;
```

T(n) = O(n)