Arrays

Definition of an Array:

List of elements of the same type placed in a contiguous memory location.

Creation of an Array:

```
dataType arrayName[] = new dataType[size];
```

Length of array:

```
length = array_name.length;
```

Passing array as argument:

func declaration: ps `void fn_name (int array_name[])`

```
{
---
}
```

fn call: `fn_name (array_name);`

Linear search

```
public static Array{
    public static int linearSearch (int a[], int key)
    {
        for (int i=0; i<a.length; i++) {
            if (a[] == key)
                return i;
        }
    }
    return -1;
}
```

Time complexity : O(n)

Binary Search

- Array should be sorted.

Code –

```
public static int binary_search (int A[], int key)
{
    int start = 0, end = A.length - 1;
    while (start <= end) {
        int mid = (start + end) / 2;
        if (A[mid] == key) {
            return mid;
        }
        if (A[mid] < key) {
            start = mid + 1;
        }
        else {
            end = mid - 1;
        }
    }
    return -1;
}
```

Time Complexity : $O(\log_2 n)$

Reverse an array:

```
public static void reverse (int A[]) {
    int start = 0, end = A.length - 1;
    while (start < end) {
        int temp = A[start];
        A[start] = A[end];
        A[end] = temp;
        start ++;
        end --;
    }
}
```

Pairs in an array:

```
public static void pairs (int A[]) {
    for (int i=0; i<A.length; i++) {
        int curr = A[i];
        for (int j = i+1; j<A.length; j++) {
            System.out.print ("(" + curr + "," + A[j] + ") ");
        }
        System.out.println();
    }
}
```

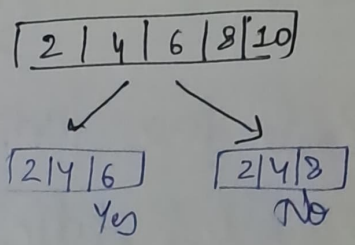
Time complexity: $O(n^2)$

2	4	6	8	10

Pairs: (2,4) (2,6) (2,8) (2,10) (4,6) (4,8) (4,10) (6,8) (6,10) (8,10)

Total pairs = $\frac{n(n-1)}{2}$

Print SubArrays



a continous part of array

Total no of subarrays: $\frac{n(n+1)}{2}$

code-

```
public static void printSubArrays (int A[]) {
    for (int i=0; i<A.length; i++) {
//start     int start=i;
        for (int j=i; j<A.length; j++) {
//end         int end=j;
            for (int k=start; k<=end; k++) {
//print           System.out.print(A[k] + " ");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

Sum of Subarrays

Brute force Method

```
public static void maxSubarray(int A[]) {

    int currSum = 0;
    int maxSum = Integer.MIN_VALUE;

    for(int i=0; i<A.length; i++) {

        for(int j=i; j<A.length; j++) {
            currSum = 0;
            for(int k=i; k<=j; k++) {

                currSum += A[k];
            }
            if(maxSum < currSum)
                maxSum = currSum;

        }
    }
}
```

Time complexity: $O(n^3)$

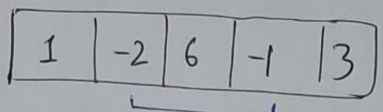
Very bad

Sum of Subarrays (II)

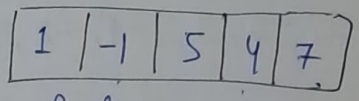
20/9/23

Prefix Method

In this we have a new array : Prefix array
That stores sum of previous indexes at that index



-2+6+-1 = 3



Prefix array

↓

prefix[end] - prefix[start-1]

= 4 - 1 = 3

```
public static void prefixSum (int A[]) {
    int currSum = 0;
    int maxSum = Integer.MIN_VALUE;
    int prefix [] = new int [A.length];
    prefix [0] = A[0];
    for (int i=1; i<A.length; i++) {
        prefix [i] = prefix [i-1] + A[i];
    }
    for (int i=0; i<A.length; i++) {
        for (int j=i; j<A.length; j++) {
            currSum = 0;
            currSum = start = 0 ?
            currSum = i = 0 ? prefix[j] : prefix[j] - prefix[i-1];
            if (maxSum < currSum)
            { maxSum = currSum; }
        }
    }
}
```

// calculate prefix array

Time complexity : $O(n^2)$

Max Subarray Sum (III)

KADANE'S Algorithm

* Do consider -ve sum as Zero in prefix method

eg:

-2, -3, 4, -1, -2, 1, 5, -3

CSum	0	0	4	3	1	0	1	6	3
MSum	0	0	4	4	4	4	4	6	6

Code-

```
public static void kadanes(int A[]) {
    int mS = Integer.MIN_VALUE;
    int cS = 0;

    for (int i=0; i<A.length; i++) {
        cS = cS + A[i];
        if (cS<0) {
            cS=0;
        }
        mS = Math.max(cS,mS);
    }
}
```

Time complexity: O(n)

Trapping Rainwater

Points -

* min no of bars > 2
* Asc / Des → no water is trapped
* $\text{trapped water} = (\text{water level} - \text{bar height}) \times \text{width}$

  ↳ $\text{water level} = \min(\text{max left}, \text{max Right})$

This is done with help of Auxiliary arrays (helping arrays)

1st) Generate 2 arrays maxleft & max Right, these will store max left & maxRight of that index respectively.

2nd) loop -

waterLevel = min (max left, max Right)

trapped water = waterlevel - bar height

PTO for CODE

Code -

```
public static int trapping (int A[]) {
    int n = A.length;
    int maxleft[] = new int [n];
    max_left[0] = A[0];
    for (int i=0 ; i<n ; i++) {
      maxleft[i] = Math.max (A[i], maxleft[i-1]);
    }
    int maxRight[] = new int [n];
    maxRight[n-1] = A[n-1];
    for [int i=n-1 ; i>=0; i--) {
      maxRight[i] = Math.max (A[i], maxRight[i+1]);
    }
    int trap_water =0;
    for (int i=0; i<=n ; i++) {
      int water_level = Math.min (maxleft[i], maxRight[i]);
      trapwater += water_level - A[i];
    }
    return trap_water;
}
```

Time Complexity: O(n)