## Bagian 1. Pengetahuan Dasar Algoritma MCSS

1.MCSS adalah algoritma yang digunakan untuk mencari hasil penjumlahan terbesar dalam sebuah urutan angka.Masukan dari algoritma tersebut berupa bilangan real sembarang dan harus memiliki tipe data yang sama sehingga bilangan tersebut dapat dibadingkan satu sama lain.Sedangkan keluaran algoritma MCSS adalah hasil jumlah urutan bilanaggan yang memiliki nilai terbesar.

2.Keluaranya adalah 7,Karena barisan mcssnya adalah 4,-2,-1,6 yang mana barisan tersebut merupakan urutan barisan dengan jumlah terbesar dibanding dengan urutan lainya.

3.Mungkin,Salah satu contoh kasusnya adalah 1,-1,-2,2,-5

4.Tidak mungkin,karena mcss memilih nilai awal/empty dengan nilai 0(nol) sebagai mcss maximal pada nilai awal,sehingga tidak mungkin menghasilkan negatif

## Bagian 2.Experimen kompleksitas MCSS

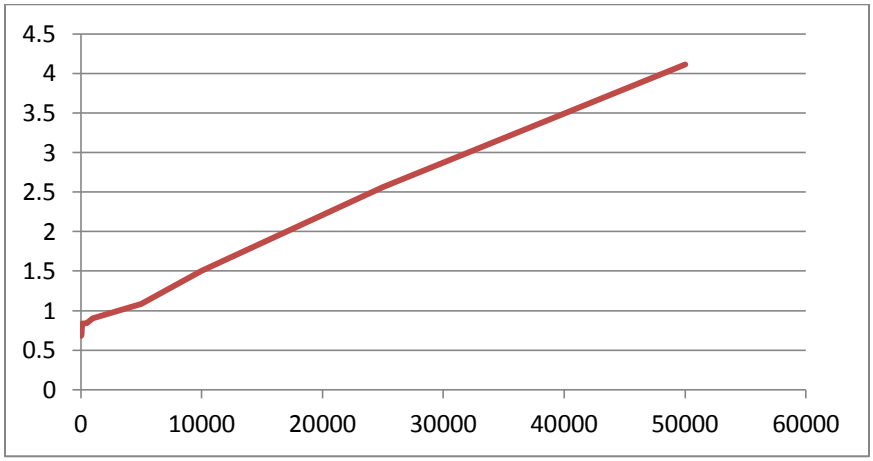
1.
```
public static int max3(int a, int b, int c) {

        if(a >=b && b<=c){
                return a;
        }

        else if(b>=c && b>= a){
                return b;
        }

        else{
                return c;
        }
}
```

2.Operating System:Windows 7 Home Basic 64-bit
Intel Platinum 1.30GHZ(2 CPU)
RAM :4GB

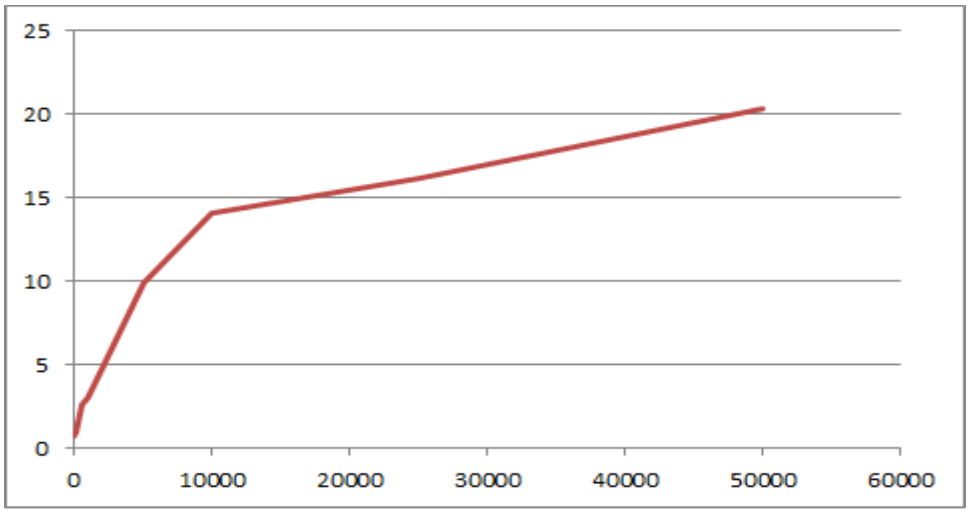
3.

| Ukuran Masukan | Running Time(ms) | | | |
|---|---|---|---|---|
| | Algoritma Linear | Algoritma NLogN | Algoritma Quadratic | Algoritma Cubic |
| 10 | 0.676572 | 0.713677 | 0.70793 | 0.827361 |
| 50 | 0.693941 | 0.774466 | 0.948938 | 2.119716 |
| 100 | 0.835255 | 0.920517 | 1.129726 | 5.577577 |
| 500 | 0.841571 | 2.575238 | 6.970195 | 94.894603 |
| 1000 | 0.899991 | 3.04576 | 8.836493 | 688.851714 |
| 5000 | 1.086305 | 9.9654377 | 60.509411 | 90647.63527 |
| 10000 | 1.503933 | 14.067495 | 158.82398 | N/A |
| 25000 | 2.561817 | 16.209316 | 776.208494 | N/A |
| 50000 | 4.117065 | 20.3662696 | 2930.966725 | N/A |

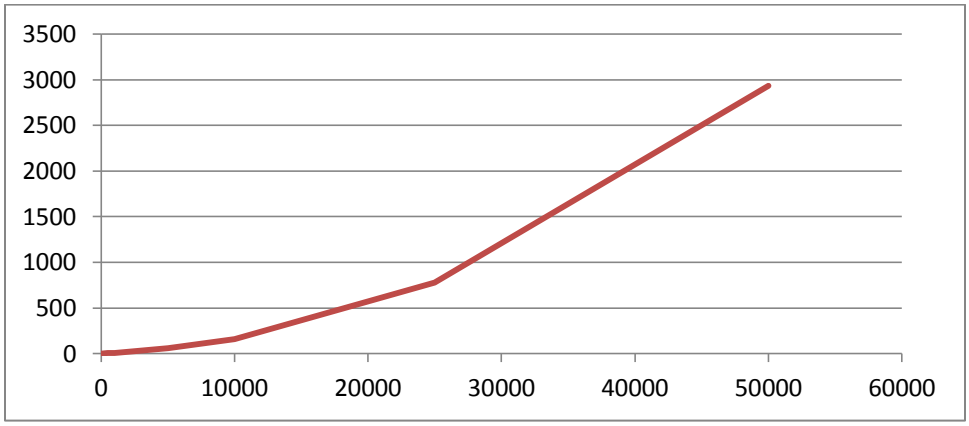
4. Grafik untuk Algoritma Linear



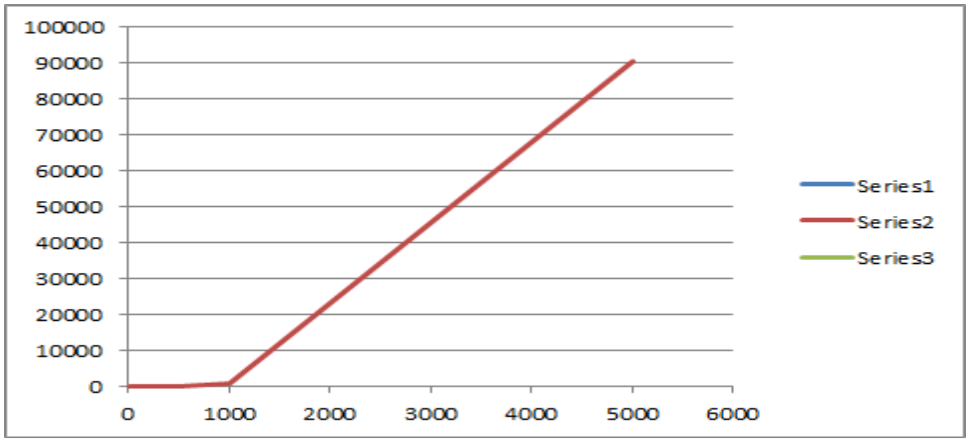

Grafik Algoritma NLog N



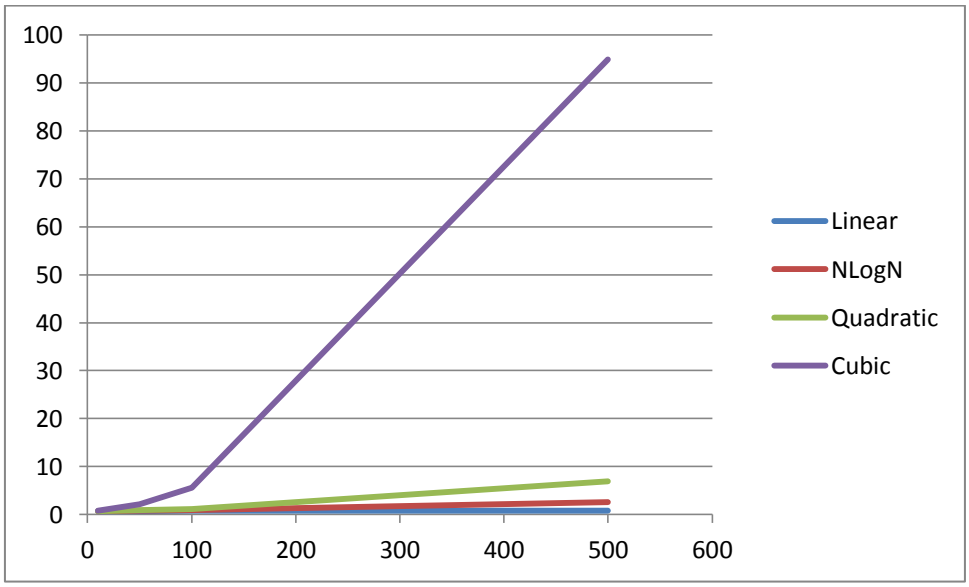
Grafik Algoritma Kuadratik

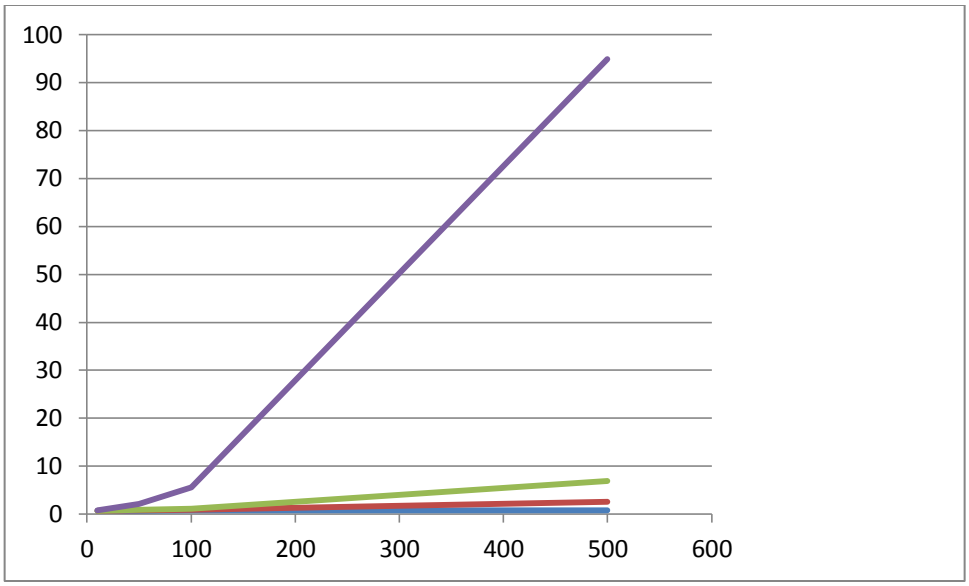



Grafik Algoritma Kubik




5.






## Bagian 3 Pertanyaan dan Kesimpulan

1.Ada yaitu algoritma Quadratic dan Algoritma NLogN,Artinya algortima Quadratic lebih cepat dibanding NLogN ketika jumlah input kecil(±10) akan tetapi ketika jumlah inputan besar maka algoritma NLogN lebih cepat dibanding quadratic

2.
Linear O(N)

| | | |
|---|---|---|
| N=10000 | Actual Time=1.503933 | |
| N=50000 | Prediction = 5(10000N)=5*1.50=7.50 | Actual Time=4.117065 |

Quadratic O(N)

| | | |
|---|---|---|
| N=10000 | Actual Time=158.82 | |
| N=50000 | Prediction = 5(10000N)^2=25*158.82=3970.5 | Actual Time=2930.96725 |

3.Dari Grafik dapat disimpulkan bahwa algoritma yang paling cepat adalah Linear O(N).Dalam jumlah input yang lebih sedikit algoritma quadratic terkadang lebih cepat dibanding NLogN akan tetapi dalam kasus jumlah input yang besar NlogN jauh lebih cepat dari quadratic.Untuk menghitung kecepatan suatu algortima dapat dihitung dengan prediction time. Actual Time dan Prediction tidak mungkin memiliki nilai yang sama persis.Nilai Actual Time bisa lebih cepat atau lebih lambat dari Prediciton time.Hal ini dikarenakan banyaknya faktor yang mempengaruhi misalnya Memori sebuah computer.

4.Ya,Pastinya saya akan memilih algoritma dengan efisiensi yang lebih baik.Karena dengan efisiensi yang baik maka kerja dari computer tidak akan terlalu berat sehingga tidak akan menimbulkan banyak resiko,misalnya error karena memori overload.

5.Ya,karena secanggih  apapun computer akan tetap lebih cepat menggunakan algoritma yang lebih efisien daripada menggunakan algoritma tanpa ada efiseinsi.sehingga algoritma yang baik akan membutuhkan waktu yang sedikit untuk berjalan dan akan memudahkan user mengunakan progam tersebut baik ketika running program atau pun memhami kode program(reusable).

## Bagian 4 Pertanyaan Tambahan

1.

```
 import java.io.BufferedReader;

import java.io.IOException;
import java.io.InputStreamReader;
public class java2{
public static void main(String[] args) throws IOException{

        BufferedReader buffReader = new BufferedReader(new InputStreamReader(System.in));
                int numberData = Integer.parseInt(buffReader.readLine());
                int[] data = new int[numberData];

                        for (int i = 0; i < numberData; i++) {
                                data[i] = Integer.parseInt(buffReader.readLine());
                }

                int hasil = mcssSpecial(data);
                System.out.println(hasil);


}
```

```
	public static int mcssSpecial(int[] input) {
		int max=0; int sum=0;
			for(int i=0;i<input.length;i++){
				if(input[i]>0){
					sum += input[i];
					max=sum;
				}
			}
		return max;
	}
}
```

Kompleksitas Program adalah O(N)

2.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class java2{

public static void main(String[] args) throws IOException{

	BufferedReader buffReader = new BufferedReader(new InputStreamReader(System.in));
	int numberData = Integer.parseInt(buffReader.readLine());
	int[] data = new int[numberData];

		for (int i = 0; i < numberData; i++) {
			data[i] = Integer.parseInt(buffReader.readLine());
		}

	int hasil = mcssSpecial(data);
	System.out.println(hasil);


}
```

```
	public static int mcssSpecial(int[] input) {
		int max=0; int sum=1;
			for(int i=0;i<input.length;i++){
				If(input[i] !=0){
					sum *=input[i];
				}
				if(max < sum){
					max=sum;
				}
				else if(sum<0){
					sum=1;
				}
			}
	return max;
	}
}
```

Kompleksitas Program adalah O(N)