Nhan V. , SID#01777 1388
CECS328 - Lab#5

Why it is not easy to compute $MPSS_{middle}$?

Explaination: Different from Max Subsequent Sum, there is a chance that we will get a Subsequent Sum that is negative, which mean it will be smaller than other positive sum, something we don't want. Further, we cannot change the value of those negative number in the array or we will get a different result from what we expect.

Let take a look at the provided array

mpss=-2 mpss=-4 mpss=-1 mpss=-2 mpss=-2 mpss=3.2 mpss=6.8 mpss=-1.2

A = [2, -3, 1, 4, -6, 10, -12, 5.2, 3.6, -8]

mpss=4 mpss=-8

We can see that there are multiple negative Sum in the middle which is not what we want.

(5) * From the recursive method to find mpss left and mpss right

$$T_1(n) = 2T\left(\frac{n}{2}\right)$$

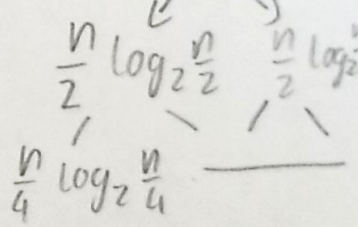
* From the method to find the mpss middle, which contains the calculation of middle sums and iteration through the entire arrays of those middle

$$T_2(n) = \underbrace{O(n\log n)}_{\text{The fastest sorting (quick sort)}} + \underbrace{\Theta(n)}_{\text{Find the Sum in the middle}} = O(n\log_2 n)$$

$$\Rightarrow T_{total} = 2T\left(\frac{n}{2}\right) + n\log_2 n$$

Step	Size of Prob
0	n
1	$\frac{n}{2}$
2	$\frac{n}{4}$
1	1
K	1

Tree



$$1 = \frac{n}{2^k} \Rightarrow k = \log_2 n$$

$$\Rightarrow T(n) = n\log_2 n + \frac{n}{2}\log_2\frac{n}{2} + \frac{n}{4}\log_2\frac{n}{4} + \ldots + \frac{n}{2^{k-1}}\log_2\frac{n}{2^{k-1}} + \Theta(1)2^k$$

$$\Rightarrow T(n) = \sum_{i=0}^{k-1} \frac{n}{2^i} \cdot \log_2 \frac{n}{2^i} + 2^k$$

$$= n \sum_{i=0}^{k-1} \frac{1}{2^i} \log_2 \frac{n}{2^i} + 2^k$$

$$= n \sum_{i=0}^{k-1} \frac{1}{2^i} \left(\log_2 n - \log_2 \frac{1}{2^i} \right) + 2^k$$

$$= n \left(\sum_{i=0}^{k-1} \frac{1}{2^i} \log_2 n - \sum_{i=0}^{k-1} \log_2 \frac{1}{2^i} \right) + 2^k$$

$$= n \left(\log_2 n \sum_{i=0}^{k-1} \frac{1}{2^i} - \sum_{i=0}^{k-1} (-i) \right) + 2^k$$

$$= n \left[\log_2 n \left(\frac{\left(\frac{1}{2}\right)^k - 1}{\frac{1}{2} - 1} \right) + \frac{(k-1)(k)}{2} \right] + 2^k$$

$$= n \log_2 n + n \log_2^2 n + n$$

$$\approx \boxed{O(n \log_2^2 n)}$$

(6) Similar the the MSS, we still calculate the Sum from both sub array. However, the way we iterate the S_L and S_R is what make the MPSS middle algorithm works. The sum will keep increment it left value until it is positive. Hence, if all the sums (S) are negative, we know that the MPSS middle will return positive infinity for Smin which mean MPSS is not in the middle. On the other hand, Smin only increment when S is positive and S after has to be less than Smin which is the smallest positive of the sum of S_L and S_R.

```
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.*;

public class main {
   // Driver
   public static void main(String[] args) {
       System.out.println("-----Default Testing-----");
       double[] a = {2,-3,1,4,-6,10,-12,5.2,3.6,-8};
       System.out.println("MPSS of the default array: "+ MPSS(a) +
"\n");
       System.out.println("-----Random Array Testing-----");
       int min = -1000;
       int max = 1000;
       System.out.println("\n-----User Interface-----\n");
       boolean con = true;
       while (con) {
           System.out.println("Please enter a positive interger or -1
to exits:");
           Scanner scan = new Scanner(System.in);
           String n = scan.nextLine();
           if (n.equals("-1")) {
               break;
           }
           while (!n.matches("\\d+")) {
               System.out.println("Your input is not an approriate
integer, Please try again:");
               n = scan.nextLine();
           }
           double[] array = new double[Integer.parseInt(n)];
           for(int i=0;i<array.length;i++) {
               double newNum = (Math.random() * (max - min + 1) +
min);
               array[i]= BigDecimal.valueOf(newNum).setScale(2,
RoundingMode.HALF_UP).doubleValue();
           }

           System.out.println("Generated Array: " +
Arrays.toString(array));
           System.out.println("MPSS of the random array:
"+MPSS(array));

       }
   }
```

```
    //The main function of MPSS
    public static double MPSS(double[] a){
        //System.out.println(MPSSMiddle(a));
        return Math.min(MPSSMiddle(a),MPSSLR(a));
    }

    // Modified MSS to make it compare two min value instead of two
max value.
    // If a value is negative, return positive infinity which will
exclude that value from the computation.
    public static double MPSSLR(double[] a){
        if (a.length==1){
            if(a[0]>0){
                return a[0];
            }
            else{
                return Double.POSITIVE_INFINITY;
            }

        }

        else{
            double MPSSL=MPSSLR(Arrays.copyOfRange(a,0,a.length/2));
            double
MPSSR=MPSSLR(Arrays.copyOfRange(a,a.length/2,a.length));
            return Math.min(MPSSL, MPSSR);
        }
    }

    //Implement function to find the MPSS middle
    public static double MPSSMiddle(double[] a){
        //System.out.println(Arrays.toString(a));
        double sum=0;
        ArrayList<Double> SL=new ArrayList<>();
        ArrayList<Double> SR=new ArrayList<>();
        int n=a.length-1;
        for(int i=n/2;i>=0;i--){
            sum+=a[i];
            SL.add(BigDecimal.valueOf(sum).setScale(2,
RoundingMode.HALF_UP).doubleValue());
        }
        sum=0;
        for(int i=(n/2)+1;i<a.length;i++){
            sum+=a[i];
```

```
            SR.add(BigDecimal.valueOf(sum).setScale(2,
RoundingMode.HALF_UP).doubleValue());
        }
        double[] Sl=SL.stream().mapToDouble(i->i).toArray();
        double[] Sr=SR.stream().mapToDouble(i->i).toArray();
        Arrays.sort(Sl);
        Arrays.sort(Sr);
        for(int i=0;i<Sr.length/2;i++){
            double temp= Sr[i];
            Sr[i]=Sr[Sr.length-i-1];
            Sr[Sr.length-i-1]=temp;
        }
        //System.out.println(Arrays.toString(Sl));
        //System.out.println(Arrays.toString(Sr));
        int i=0;
        int j=0;
        double Smin=999999999;// Positive infinity
        while(i<Sl.length){
            //System.out.println(i);
            double S=Sl[i]+Sr[j];
            //System.out.println(S+" | "+Smin);
            if(S<=0){
                i++;
            }
            else if(S<Smin){
                Smin=S;
                j++;
            }
            else if(S>Smin){
                j++;
            }
            if(i>Sl.length-1 || j>Sr.length-1){
                break;
            }

        }

        return BigDecimal.valueOf(Smin).setScale(2,
RoundingMode.HALF_UP).doubleValue();
    }
}
```