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

MPSS = 0.8

Dividing into subarrays

[10, -12, 5.2, 3.6, -8]

[5.2, 3.6, -8]

$MPSS_L = 5.2$ $MPSS_R = -8$

$MPSS_{mid} = ?$

$MPSS_{mid}$ requires that you sum both left and right sides at the same loop

5.2 3.6 -8

3.6 + (-8) = -4.4

5.2 + (-4.4) = 0.8 < 3.6 ✓

Negative, but another value in array may make it positive.

$MSS_{mid}$ checks right and left in different loops.

3.6 -8 = -4.4 X

5.2 3.6 = 8.8

8.8 > 3.6 → 3.6 ? X

How does $MPSS_{middle}$ work?

Requires elements from left and right

[A B C D E]

SSL [A, A+B] SSR [D, D+E]

sort both arrays ...

We don't know the exact range of $MPSS_{mid}$, and because of explanation above we can't use the normal $MSS_{mid}$ method.

[subsequence left] C [subsequence right]

This satisfies the condition of checking left and right sides at the same time to find MPSS.

[ _ _ _ _ _ ] change range based on evaluation

evaluate sum

mid + left + right

↳ ≤ 0 move i subsequence negative, can't use

↳ < min set new min, move j new MPSS found, shift j in case a smaller one is found.

↳ > min move j subsequence sum > min, ignore and check next

MPSS( a )

Base case ] runtime is constant $\Theta(1)$

mid = ~~~ $\Theta(1)$

mpssLeft = MPSS(~~~)
mpssRight = MPSS(~~~) → works on array size n/2 until size 1
$n + \frac{1}{2}n + \frac{1}{4}n + \cdots + 1 = 1\log n$
rest of code ran $2\log n$ times

ssL[] = ~~~ $\Theta(1)$
ssR[] = ~~~ $\Theta(1)$

for( pos=1; pos <= mid; pos++) ] for loop runs n/2 times, but fills two arrays of size n/2
{ fill ssL
{ fill ssR
$2(n/2) = n$ runtime

Arrays.sort(ssL) ] Was used to compact code.
Arrays.sort(ssR) ] A search shows that Arrays.sort() is a modified version of mergeSort = $O(n\log n)$ twice

*If I'm not allowed to use this then I would implement mergesort or quicksort.

min = ~~~ $\Theta(1)$
int i = 0 $\Theta(1)$
int j = ssR.length-1 $\Theta(1)$ ← The way my code is set up, ssR >= ssL for size.

while (~~~) ] Loop breaks when one of the arrays (ssL, ssR) is exhausted.
{ check sums of ssL and ssR indices, set min.
Runtime: $\frac{n}{2} = n$ (ignore constants)

mpssMid = min;

return min(mpssLeft, mpssRight, mpssMid); → $\Theta(1)$

Runtime: $(n + n\log n + n)(2\log n)$ → runtime in parenthesis is ran $2\log n$ times due to recursive calls in mpssLeft and mpssRight.

$2n\log n + \cancel{2}n\log^2 n + 2n\log n$

$\Theta = n\log^2 n$