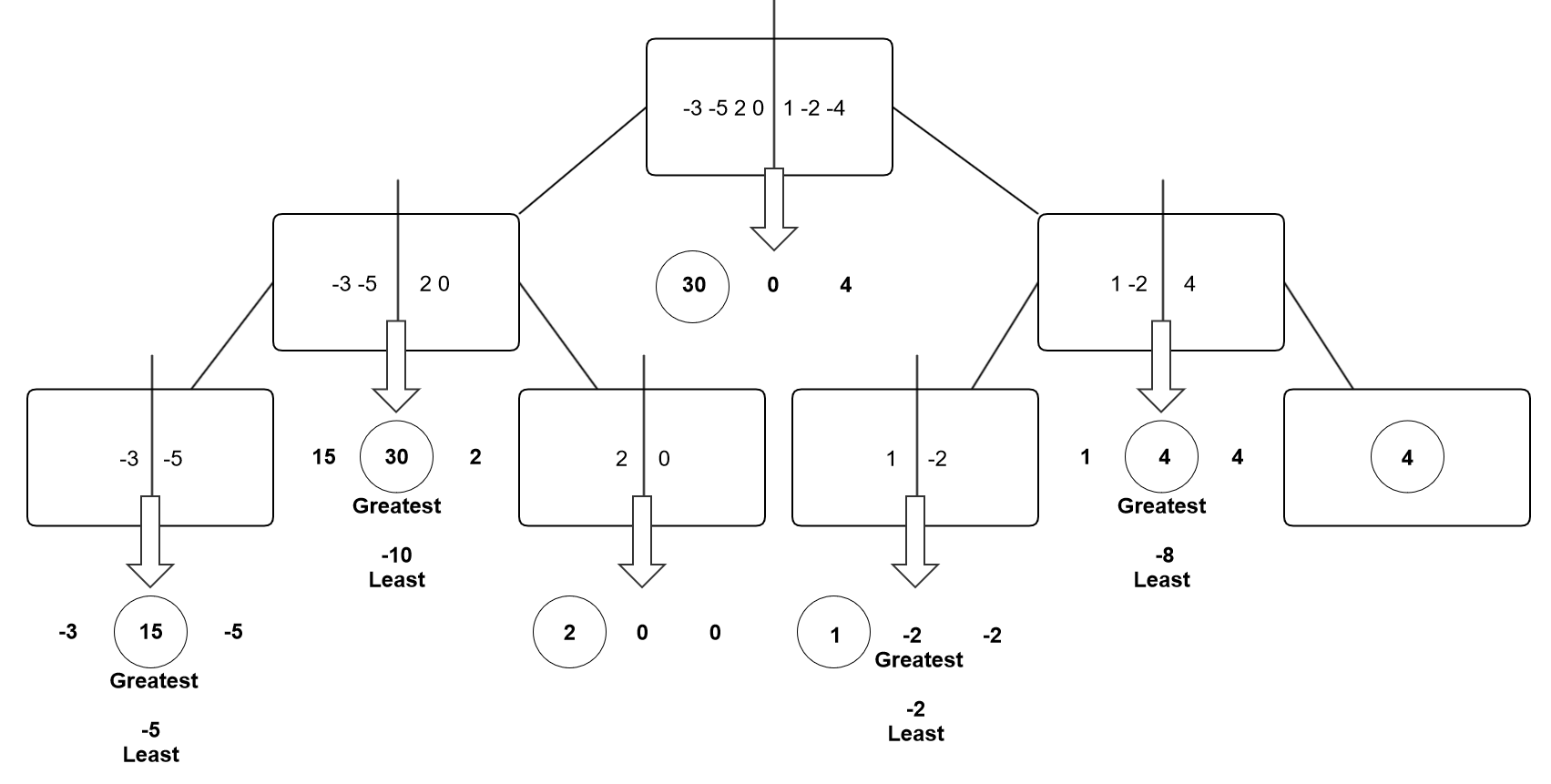
The algorithm, using the divide and conquer method, starts by splitting up the set in half, creating a left side that starts at index 0 and ends at the middle index, ((start + end)/2), and a right side that starts at index mid + 1 and ends at index end. The largest product is then found for the left and right sides recursively, until the base case is met in which the start and end indexes are equal, (when there is only one element in the subset). Along with this, the largest middle product is also found, which includes elements from both the left and right side. After these three largest products are found, they are compared to each other to check for the greatest value of the three, and that value is the returned as largest product of that current thread. This largest value is returned to the caller, in this case, the left or right sides of the previous threads, and the value is propagated back up the stack, all the way up to the first recursion call. The final compare is then made between the left, right and middle sets and the final largest product is returned for the whole set.

In order to process the middle largest product, which contains both elements from the left and right sides of the subsets, the algorithm starts in the middle of both the left and right sides, and iterates over the respective values, looking for the greatest and least products of both sides. Since negative numbers can exist within the set, the least products are found as well as the product of an even number of negative numbers can potentially be greater than the product of two positive numbers. In order to know which one is greater, the two outcomes, (the positive greatest product and the negative greatest product), are compared to each other and the greater of the two is set as the greatest middle largest product. A short circuit is included if any zeros are found within the set, as any product multiplied by zero will always be reduced to 0. Therefore, zeros cannot exist in the largest middle set, unless 0 itself is the greatest product of the whole set.



In calculating the time complexity of the algorithm, I found that the extra cases, for negative numbers and zeros simply created a greater constant value when compared to the largest sum problem done in class. Since constants are not included in O notation, the time complexity for this method is also O(n log n).