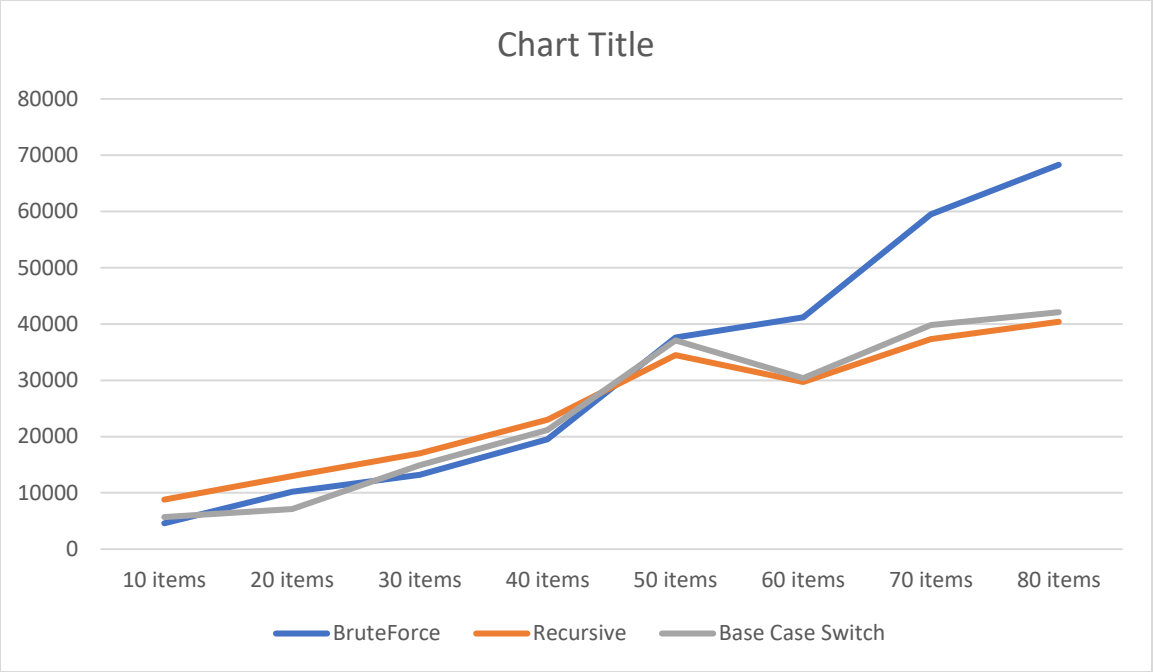
# Chapter 4 Programming

## 4.1-3:

**As far as I could tell, when I added the bruteforce to the recursive under a length threshold, the threshold of which one was higher by 2 than before it was added.**



```java
public int[] findMaxSubarray (int[] A, int low, int high) {

    if (A.length <= 50) {
        return bruteForceMax(A);
    }

    int mid = 0;
    if (high == low) {
        int[] array = {low, high, A[low]};
        return array;
    }
    else {
        mid = (low+high)/2;
        int[] left = findMaxSubarray(A, low, mid);
        int[] right = findMaxSubarray(A, mid+1, high);
        int[] cross = findMaxCrossingSubarray(A, low, mid, high);
        if (left[2] >= right[2] && left[2] >= cross[2]) {
            return left;
        }
        else if (right[2] >= left[2] && right[2] >= cross[2]) {
            return right;
        }
        else {
            return cross;
        }
    }
}
```

All of this works the same as the books pseudocode, with the addition of the first if-statement. It allows to switch to Brute force under a threshold of items in the array

```
60    public int[] bruteForceMax (int[] A) {
61
62        int[] maxSub = new int[3];
63        maxSub[2] = Integer.MIN_VALUE;
64        for (int i = 0; i < A.length; i++) {
65            int sum = 0;
66            for (int j = i; j < A.length; j++) {
67                sum += A[j];
68                if (maxSub[2] < sum) {
69
70                    maxSub[0] = i;
71                    maxSub[1] = j;
72                    maxSub[2] = sum;
73                }
74
75            }
76        }
77        return maxSub;
78    }
79
```

this works quite simply. IT just loops for every possible value of the sum to find the largest number.

```
89    public static void main(String[] args) {
90
91        Chapter4Programming ob = new Chapter4Programming();
92        int[] testArray = makeIntArrayForMaxSubArray(80);
93
94        long startTime = System.nanoTime();
95
96        ob.bruteForceMax(testArray);
97
98        long endTime   = System.nanoTime();
99        long totalTime = endTime - startTime;
100       System.out.println(totalTime);
101
102       startTime = System.nanoTime();
103
104       ob.findMaxSubarray(testArray, 0, testArray.length-1);
105
106       endTime   = System.nanoTime();
107       totalTime = endTime - startTime;
108       System.out.println(totalTime);
109
```

I used this main method to test my time for each version of the code using the same array for each.