Given an array of costs, how do we pick the optimal buy and sell positions?

Example data:

```
A = [87, 86, 84, 92, 91, 89, 93, 94, 88, 93]
```

Strategy 1

```
def findMax(A):
    bestProfit = -MAX_INT
    bestBuyDate = None
    bestSellDate = None
    for i = 1 to n:
        for j = (i + 1) to n:
            if A[j] - A[i] > bestProfit:
                bestBuyDate = i
                bestSellDate = j
                bestProfit = A[j] - A[i]
    return (bestBuyDate, bestSellDate)
```

Sketch table for storing all bestProfit values:

Strategy 2

```
B = [-1, -2, 8, -1, -2, 4, 1, -6, 5]
```

```
def findMaxCrossingSubarray(B, low, mid, high):
  # check first half
  left-sum =
  sum =
  for i=mid down to low:
    sum = sum + B[i]
    if sum > left-sum:
      left-sum = sum
      max-left = i

  # check second half
  right-sum =
  sum =
  for j = (mid+1) to high:
    sum = sum + B[j]
    if sum > right-sum:
      right-sum = sum
      max-right = j
   return (max-left, max-right, left-sum + right-sum)
```

Runtime of this helper function if size of B is n?

```
def findMaxSubarray(B, low, h):

  # base case
  if low == h:
    return (low, h, B[low])

  else:
    mid = (low + h) // 2
    Llow, Lhigh, Lsum = findMaxSubarray(B, low, mid)
    Rlow, Rhigh, Rsum = findMaxSubarray(B, mid+1, high)
    Clow, Chigh, Csum = findMaxCrossingSubarray(B, low, mid, high)

    # case 1: max is entirely on left
    if (               ) >= (                ) and (                ) >= (                ):
      return

    # case 2: max is entirely on right
    elif (               ) >= (                ) and (                ) >= (                ):
      return

    # case 3: max is split
    else:
      return
```