CS 5592: Design and Analysis of Algorithms
Homework 1
Author: Hayden McParlane

(3) Given a sequence of n real numbers stored in an array, A[1], A[2], A[3], ..., A[n], we wish to find two numbers A[i] and A[j], where i < j, such that A[i] ≤ A[j] and their sum is the largest. Please design a divide-and-conquer algorithm to solve the above problem. Please analyze the complexity of your algorithm. The complexity of your algorithm must be $\mathcal{O}(n \cdot lgn)$ or better.

Pseudocode

Algorithm 1 Find max two such that:

$$i < j$$

and

$$A[i] \leq A[j]$$

```
1: function FindMax(A, p, r)
2:     if p equals r then return (−∞, −∞)
3:     else
4:         mid ← ⌊(p + r)/2⌋
5:         leftSolution ← FindMax(A, p, mid)
6:         rightSolution ← FindMax(A, mid + 1, r)
7:         crossingSolution ← FindCrossingMax(A, p, mid, r)
8:         return Max(leftSolution, crossingSolution, rightSolution)
9:     end if
10: end function
```

Algorithm 2 Perform work to actually find the max two for a given p and r.

```
1: function FindCrossingMax(A, p, mid, r)
2:     i ← mid
3:     j ← mid + 1
4:     max ← A[i] + A[j]
5:     for k from mid down to p do
6:         if max < A[k] + A[j] then
7:             i ← k
8:             max ← A[k] + A[j]
9:         end if
10:    end for
11:    for k from mid + 1 up to r do
12:        if max < A[k] + A[i] then
13:            j ← k
14:            max ← A[k] + A[i]
15:        end if
16:    end for
17:    return (i, j)
18: end function
```

Complexity Analysis

We can think of this algorithm in terms of three parts: left call, right call and a function that finds the answer which crosses over the from the left to the right. With that in mind, the time complexity of this algorithm can be modeled using the following equation:

$$\mathrm{T}(n) = \mathrm{T}(\lfloor n/2 \rfloor) + \mathrm{T}(\lceil n/2 \rceil) + \textsc{f}(n) + \mathcal{O}(1)$$

Recognizing that the time complexity is monotonically increasing and is bounded by the functions and not the constant $\mathcal{O}(1)$, we can eliminate the floor and ceiling functions and simplify the constant time element as follows:

$$\begin{aligned}\mathrm{T}(n) &= \mathrm{T}(n/2) + \mathrm{T}(n/2) + n + 1 \\ &= 2 \cdot \mathrm{T}(n/2) + n + 1\end{aligned}$$

Using the master method it is evident that

$$\begin{aligned}a &= 2 \\ b &= 2 \\ \textsc{f}(n) &= n\end{aligned}$$

Therefore

$$\begin{aligned}\textsc{f}(n) &= \theta(n^{log_b a}) \\ &= \theta(n^{log_2 2}) \\ &= \theta(n^1) \\ &= \theta(n)\end{aligned}$$

Implying that we should use rule 2 of the master method. Using rule two, we determine the following:

$$\begin{aligned}\mathrm{T}(n) &= \theta(n^{log_b a} \cdot log_2 n) \\ &= \theta(n \cdot log_2 n)\end{aligned}$$

(4) In a given sequence of n numbers in an array, A[1], A[2], A[3], ..., A[n], a number may occur multiple times. A number is called the dominating number if it occurs strictly more than n/2 times. A sequence may have no dominating number. You may compare two numbers, A[i] and A[j], $1 \leq i < j \leq n$, to see if A[i] equals A[j]. However, we assume this comparison does not tell which one is smaller in case they are not identical. Please use this kind of comparison among the numbers in array A and counting to determine if array A has a dominating number. You can expect to know which number is smaller or larger if the two numbers in comparison are not both from array A. Your algorithm must use divide-and-conquer and have $\mathcal{O}(n \cdot lgn)$ complexity.

Algorithm 3 Algorithm to solve dominating number problem using the divide-and-conquer approach.

```
1: function FindDominating(A, p, r)
2:     if p equals r then
3:         count ← 1
4:
5:         return count and
6:     else
7:         threshold ← ⌈n/2⌉
8:         mid ← ⌊(p + r)/2⌋
9:         left ← FindDominating(A, p, mid)
10:        right ← FindDominating(A, mid + 1, r)
11:
12:
13:        return
14:    end if
15: end function
```

(5) Given a sequence of n real numbers, A[1], A[2], A[3], ..., A[n], we wish to find a subsequence from A[i] to A[j], $1 \leq i \leq j \leq n$ such that the sum of all the numbers in this subsequence is maximized. Please design a $\mathcal{O}(n \cdot lgn$ divide-and-conquer algorithm to solve this problem. Note that some numbers may be negative and the value m may be negative. You need to present good pseudocode.

Algorithm 4 Divide-and-conquer algorithm that locates the subsequence of A having the greatest sum.

```
1: function MaxSubsequence(A, p, r)
2:     if p equals r then
3:         return p, r and A[p]
4:     else
5:         mid ← ⌊(p + r)/2⌋
6:         left ← MaxSubsequence(A, p, mid)
7:         right ← MaxSubsequence(A, mid + 1, r)
8:         crossing ← MaxCrossingSubsequence(A, p, mid, r)
9:         return max(left, crossing, right)
10:    end if
11: end function
```

Algorithm 5 Function that calculates the maximum subsequence that crosses the left and right barrier in the recursive function above providing total coverage of the algorithm above.

```
1:  function MaxCrossingSubsequence(A, p, mid, r)
2:      i ← mid
3:      j ← mid + 1
4:      max ← A[i] + A[j]
5:      sequenceSum ← max
6:
7:      for k from mid - 1 to p do
8:          sequenceSum ← sequenceSum + A[k]
9:          if sequenceSum > max then
10:             max ← sequenceSum
11:             i ← k
12:         end if
13:     end for
14:
15:     for k from mid + 2 to r do
16:         sequenceSum ← sequenceSum + A[k]
17:         if sequenceSum > max then
18:             max ← sequenceSum
19:             j ← k
20:         end if
21:     end for
22:
23:     return i, j and max
24: end function
```